WEBSITE FINGERPRINTING ATTACKS

By

WEIQI CUI

Bachelor of Science in Computer Science
Dalian University of Technology
Dalian, Liaoning
China
2014

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 2019

WEBSITE FINGERPRINTING ATTACKS

Thesis Approved:

Dr. Eric Chan-Tin

Dissertation Advisor

Dr. Nohpill Park

Dr. Christopher Crick

Dr. Yanmin Gong

Name: WEIQI CUI

Date of Degree: May, 2019

Title of Study: WEBSITE FINGERPRINTING ATTACKS

Major Field: COMPUTER SCIENCE

Abstract: Most privacy-conscious users utilize HTTPS and an anonymity network such as Tor to mask source and destination IP addresses. It has been shown that encrypted and anonymized network traffic traces can still leak information through a type of attack called a website fingerprinting (WF) attack. The adversary records the network traffic and is only able to observe the number of incoming and outgoing messages, the size of each message, and the time difference between messages. In previous work, the effectiveness of website fingerprinting has been shown to have an accuracy of over 90% when using Tor as the anonymity network. Thus, an Internet Service Provider can successfully identify the websites its users are visiting.

Mitigations to these attacks are using cover/decoy network traffic to add noise, padding to ensure all the network packets are the same size, and introducing network delays to confuse an adversary. Although these mitigations have been shown to be effective, reducing the accuracy to 10%, the overhead is very high. The latency overhead is above 100% and the bandwidth overhead is at least 40%. We introduce a new realistic cover traffic algorithm, based on a user's previous network traffic, to mitigate website fingerprinting attacks. In simulations, our algorithm reduces the accuracy of attacks to 14% with zero latency overhead and about 20% bandwidth overhead. In real-world experiments, our algorithms reduces the accuracy of attacks to 16% with only 20% bandwidth overhead.

One main concern about website fingerprinting is its practicality. The common assumption in previous work is that a victim is visiting one website at a time and has access to the complete network trace of that website. However, this is not realistic. In our work, we aim to reduce the distance between the lab experiments with the realistic conditions. We propose a new algorithm based on Hidden Markov Model to deal with situations when the victim visits one website after another. After that, we employ deep learning algorithm to handle the situations when the captured traces are not perfect, such as partial traces, two-page traces or traces with background noise.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

Anonymous communication's goal is to hide the relationship and communication contents among different parties. Once two parties establish an anonymous communication between them, the contents are encrypted and routing information is hidden, thus masking the source and destination IP addresses from third parties. Tor Tor (2017); Dingledine et al. (2004) is one of the most popular low-latency anonymity-providing network. It is used by millions of people daily Portal (2017). Tor protects users' privacy through a telescoping three-hop circuit and encrypting the network traffic using onion routing. Although Tor and many other privacy-enhancing technologies such as HTTPS proxy hide the communication contents and network layer contents, the network traffic itself may leak information such as packet size, inter-packet timing information, and direction of the packets (from server to client or other way around).

A website fingerprinting (WF) attack is one where an attacker identifies a user's web browsing information by merely observing that user's network traffic. The attacker is not attempting to break the encryption algorithm or the anonymity protocol. The only information available to the attacker is the metadata information such as packet size, the timing information between packets, and the direction of the packet. The success of this attack is measured by the number of websites correctly identified.The accuracy has been shown to be around 90%, thus violating any privacy offered by HTTPS and anonymity services like Tor.

1.1 Website Fingerprinting Defense

Various defenses against website fingerprinting attacks Cai et al. (2012); Wang et al. (2014); Nithyanand et al. (2014); Cai et al. (2014b,a) have been proposed. The defenses include padding so that every packet has the same size, cover traffic to generate enough noise to fool the adversary, or introducing network delays between network packets. Although they have been shown to be effective, the overhead introduced by these defenses is very high. The latency overhead is above 100% and the bandwidth overhead is from 50% to over 100%.

We developed a new cover traffic algorithm that generates just enough noise to mitigate website fingerprinting attacks. Our algorithm also has zero latency overhead and lower bandwidth overhead than current schemes. Our algorithm generates "realistic" cover traffic; it collects the network traffic from a user, then uses that historical network traffic data as training set to feed the cover traffic generation algorithm. The generated noise thus will look exactly like a website that a user has previously visited. This prevents website fingerprinting attacks and introduces little bandwidth overhead.

1.2 Website Fingerprinting Assumptions

The first WF work was proposed by Hintz Hintz (2003). Researchers extracted important features from the metadata and applied a variety of machine learning algorithms to classify websites. Wang et.al Wang et al. (2014) utilized k-NN with a new weight system, Panchenko et.al Panchenko et al. (2016) proposed the cumulative features with Support Vector Machine(SVM) and re-evaluated the state-of-art techniques at a internet scale. They showed high accuracy in predicting websites, however one main concern in WF attack is its practicality which was detailed discussed in Juarez et al. (2014). The previous WF work made limiting assumptions 1) Closed world: the WF attack is tested in a fixed set of monitored websites; 2) Browsing behaviour: the

victim visits one website at a time; 3) the adversary can record the whole network traffic trace from the beginning to the end for a website [1].

The first assumption is addressed in recent research work Lu et al. (2010); Panchenko et al. (2011); Gong et al. (2012); Wang and Goldberg (2013); Cai et al. (2012); Wang et al. (2014); Cai et al. (2014b); Panchenko et al. (2016); Oh et al. (2017a) where they deployed WF test under the open world model. The second assumption has been explored in Wang and Goldberg (2016). They tried to find the end of the first website and the start of the second website(the split point) when the clients visits the website one after another. They achieved the accuracy of 92% in finding the split point when there is a time gap between the first and second website. However, their accuracy falls 66% and 32% to when the time gap is zero and negative. Even though they didn't test the accuracy of the split traces, the low accuracy in finding the split point will definitely have a negative impact in website predicting. Another limitation of Wang and Goldberg (2016) is that it could not directly predict the websites in the network traffic trace recorded. It attempted to find the split point first and used previous WF approaches for predicting the websites, which leads to a higher cost in time.

1.2.0.1 Traditional Machine Learning

We propose a new algorithm Cui et al. (2019) based on machine learning to deal with assumptions mentioned above. In the second assumption when the user visits more than one websites at a time, we consider the case when the victim visits two pages one after the other (continuous visits). We propose a new algorithm based on Hidden Markov Model to split and detect traces with two continuous pages. We show that our algorithm gives a higher accuracy in finding the split point in two continuous websites (80% compared to 63% in previous work). This is also the first time that

[1]Note that we used trace, network trace, website, and webpage interchangeably

the accuracy in directly predicting websites in continuous traffic traces is tested and shown.

1.2.0.2 Deep Learning

Recently deep learning has been shown outperforming traditional machine learning techniques in many areas such as image recognition He et al. (2016), speech recognition Song (2015). Besides, DL doesn't require manually feature extraction. The effectiveness of using DL in WF attack has been explored in Rimmer et al. (2017); Sirinam et al. (2018); Oh et al. (2017b). However, they all built and evaluated on integrate one-page traces. Our goal is to explore whether we can use deep learning to handle the situations when the captured traces are not perfect, such as partial traces, two-page traces or traces with background noise. In this study, we won't distinguish between continuous or overlapped traces. We emphasize that the goal of this study is to improve the performance of WF attack in the real world instead of building a better DL model. The contribution of this work are summarized as follows.

- We propose a CNN-based classification to distinguish between one-page and multiple page traces. From the best of our knowledge, Wang et.al Wang and Goldberg (2016) is the only one that investigate this before with k-NN based method. We compared our work with them on the same dataset and found that our model achieves an accuracy of 85% over their 68.6%.

- We evaluate the DL model on partial traces and improve the performance on traces missing the head part by adding the head detection. With 10% of packets missing in the beginning of the trace, the accuracy is increased from 22.12% to 86.35%.

- We developed the method to predict both websites in a two-page traces. We first time show the TPR, FPR in the open-world on two-page trace prediction.

We employed Ren-yi entropy in the open-world evaluation and proved that it's a better algorithm to calculate the prediction confidence from probability distribution obtained than Shannon entropy. Besides the packet size, we found that timestamp is an important feature for the second website prediction and the accuracy is improved 14% by adding the sequence of packet timestamp to the training model.

- For traces with background noise, we found that 10 noise packets per second has the ability to fully disturbing the DL model which causes more than 70% reduction in accuracy.

- To investigate in a more realistic setting, we collect the largest testing dataset with multi-page traces which is based on 118 monitored websites and $17,700$ unmonitored websites. We also collect the corresponding training dataset with balanced amount of monitored and unmonitored traces. We first time launch binary classification with DL algorithm in WF attack and proved the effectiveness of this approach in two-page prediction.

The rest of the document is organized as follows. In chapter II, we review the background of website fingerprinting. In chapter III we introduce our WF mitigation algorithm. In chapter IV, we revisit assumptions in WF attack and propose two new algorithms to deal with the situations without these assumptions. We employ deep learning to handle more complicated situation and outlined the work in chapter V.

CHAPTER II

BACKGROUND

2.1 Website Fingerprinting Attack Procedures

Website fingerprinting has been shown to be a serious threat against privacy mechanisms for anonymous web browsing. Researchers have proposed different scenarios for website fingerprinting. The attack and resulting experiment vary from each other; however, they all follow similar steps. A website fingerprinting attack and analysis can be divided into six steps: 1) collect data, 2) extract features from data, 3) select algorithm, 4) build model based on 1) to 3), 5) evaluate real network traffic trace, and 6) evaluate results. Figure 2.1 shows an illustration of all the steps of a website fingerprinting attack. The last right-most block contains the measurements to evaluate the effectiveness of an attack.

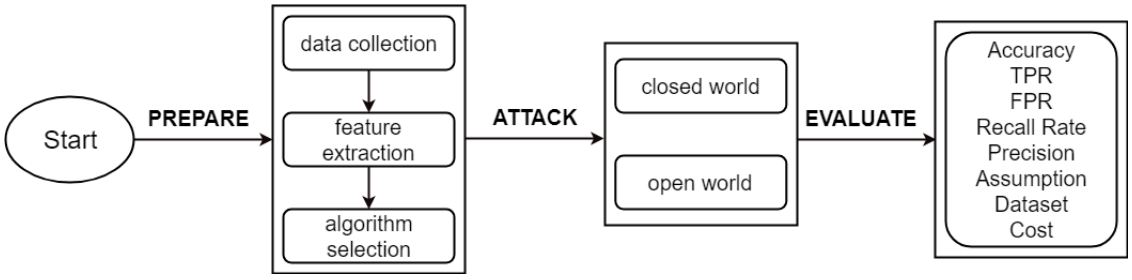

Figure 2.1: Steps of launching and evaluating a website fingerprinting attack.

When setting up an experiment for a website fingerprinting attack, the first step is to perform data collection. A network traffic recording tool such as wireshark or tcpdump is used. Before running any scripts to automatically collect data, the configuration of the browser should be set to match the assumptions, such as disabling

all plug-ins to avoid background noise and clearing the browser cache. The automated script will then visit websites in a certain order. The time taken to collect data depends on the number of instances recorded for each website and the size of the website list. Features extracted from the recorded network traffic traces will be used for training. Each network trace is composed of a list of features. The features can be treated as attributes in a machine learning context. A classification algorithm is applied to these features to build the attack model. Different websites correspond to different classes. Different network traffic traces are then collected to evaluate the performance of the model. A 10-fold cross validation is often employed to reduce the bias in the evaluation process.

2.2 WF traces

A WF trace is a record of network traffic packets during users' visits to the websites. So, one WF trace could contain only one visit to a website or multiple visits to websites like we open several websites nearly at the same time. Each WF trace, in our experiments, consists of a sequence of pairs of packet size and its timestamp: $< timestamp : \pm packet\_size >$ where the sign indicates the direction of the packets, positive as incoming and negative as outgoing.

2.3 Classification

Previous work Hintz (2003); Sun et al. (2002); Bissias et al. (2006); Liberatore and Levine (2006); Lu et al. (2010); Herrmann et al. (2009); Panchenko et al. (2011); Gong et al. (2012); Wang and Goldberg (2013); Cai et al. (2012); Wang et al. (2014); Panchenko et al. (2016); Wang and Goldberg (2016); Spreitzer et al. (2016); Hayes and Danezis (2016) have achieved a classification accuracy of around 90% in both the open and closed world settings. A closed world is where the set of training packet traces are the same as the testing set. An open world setting is where there is a small

set of monitored/sensitive packet traces among a larger set; the goal is to detect if a packet trace belongs to one of these monitored websites.

To perform classification, various features have been used such as number of outgoing and incoming packets, total size of incoming and outgoing packets, and cumulative size of packets. If the Tor network is used, some features that have also been considered include the Tor cells before and after. Various algorithms have also been used such as k-nearest neighbors (K-NN), support vector machine (SVM), random decision forests, edit distances, Jacard index, and Naive Bayes.

2.4 Threat Model

The threat model is a local adversary that can see all the network traffic from a user. The adversary cannot decrypt the contents of the network packets but can observe the metadata such as packet sizes, direction of the packets, and the timings of packets. The adversary can also look at the IP headers to determine the source and destination IP addresses and port numbers. The victim is using an anonymous service such as a VPN or Tor Tor (2017). Figure 2.2 illustrates the model and shows where the adversary is located. The goal for the adversary is to guess the website or webpage from only the encrypted network packet trace.

2.5 Closed World and Open World

The WF attack experiments can be built based on two different scenarios: closed world and open world. The closed world model is used when complete information is available. The assumption in a closed world model is that an attacker knows the metadata information for a list of websites. The website visited by a victim is in the list known by the attacker. It is a strong assumption which is used to simplify the threat model, implementation of the experiment, and evaluation of the success of the attack. Since the closed world scenario is the more basic model, most research

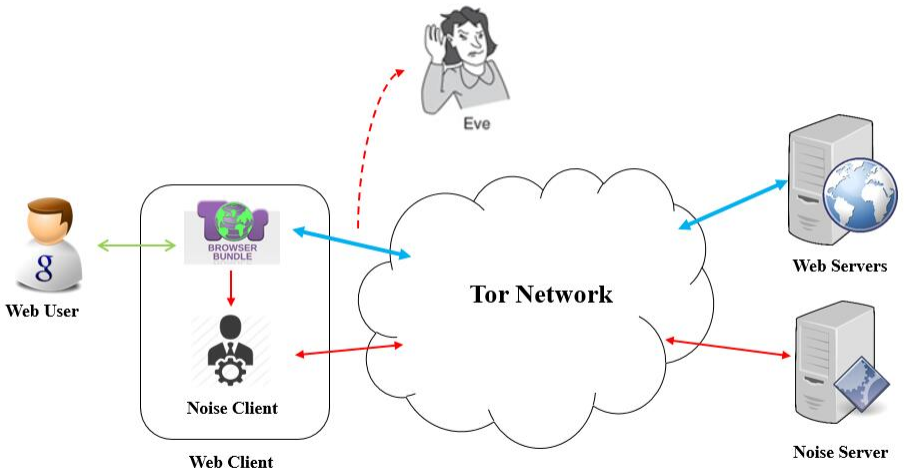

Figure 2.2: Our system model and experimental setup.

work Hintz (2003); Sun et al. (2002); Bissias et al. (2006); Liberatore and Levine (2006); Herrmann et al. (2009); Panchenko et al. (2011); Wang and Goldberg (2013); Cai et al. (2012); Juarez et al. (2014); Cai et al. (2014b,a); Spreitzer et al. (2016) include an analysis of results in this closed world model.

In an open world model, a website being fingerprinted can be either from the list or not in the list. The attacker keeps track of a small list of monitored websites. Once a website fingerprint is obtained, the attacker attempts to determine if that website is part of the list of monitored websites or not. More recent research work Lu et al. (2010); Panchenko et al. (2011); Gong et al. (2012); Wang and Goldberg (2013); Cai et al. (2012); Wang et al. (2014); Juarez et al. (2014); Cai et al. (2014b); Panchenko et al. (2016); Wang and Goldberg (2016); Oh et al. (2017a) deployed their website fingerprinting experiments under the open world model and identified whether a website is from the list of monitored sites.

CHAPTER III

REALISTIC COVER TRAFFIC TO MITIGATE WEBSITE FINGERPRINTING ATTACKS

3.1 Proposed Noise Algorithm

3.1.1 Overview

Our proposed algorithm to generate cover traffic is novel since it generates realistic noise rather than random noise or random padding. The noise generated is learned from the network traffic generated by the user's webbrowser. The information recorded is the network traffic trace without the payload contents: each incoming and outgoing packet's size, and the time interval between packets and "train" of packets. A train is a set of incoming packets with size of MTU (Maximum Transmission Unit) with the last packet size less than MTU. Usually an outgoing web request is following by one or more trains of incoming packets. Replaying this recorded network traffic will simulate that user's browsing habit. Our hypothesis is that if the cover traffic generated is similar to what the user usually does, this will provide a better noise in preventing website fingerprinting and also reduce the bandwidth overhead since this would be traffic that the user usually generates anyway. It has already been shown that if a client visits several webpages at the same time Wang and Goldberg (2016), then it is hard for an adversary to identify the webpage visited.

Instead of replaying the web requests to the actual servers which would use up resources on these servers, we set up our own simple webserver. Our algorithm can be implemented as a plugin for Firefox (Tor Browser Bundle). It will send a web

request padded to a certain packet size to our webserver through the Tor network. The request will contain the total size of data that the server has to send back and the time that the data should be sent. Both the client plugin and webserver do not have to send any content; only pad the packets to the specified packet size. The generated network traffic will be transferred over the Tor network; a local adversary will not be able to determine which packet is noise.

The algorithm will first record traffic of a web page, then parses the recorded traffic trace. Packets are put into two sets based on whether they are incoming packets or outgoing packets. For each set, packets are organized by trains of packets. For each train, the size and timestamp of each packet is recorded. Trains of packets are listed in order by the timestamp of the first packet in the train.

Figure 3.2 shows an example sample of a recorded network traffic. The only information recorded is the relative time between packets and the packet sizes. Both incoming and outgoing packets are recorded. The format of the sample shown in the figure is as follows: $< timestamp >: \pm < packet\_size >$. The actual time is not relevant; only the time difference between two packets is used. This is the time difference between the current packet's timestamp and the next packet's timestamp. The packet size is the TCP-level packet size. A red packet size indicates an outgoing packet.

3.1.2 Implementation Details

The cover traffic needs to have two features: 1) it should customize total packet size so that the cover traffic can be controlled, 2) it should have similar packet distribution with the real web traffic so that it cannot be filtered out easily. Modern web pages usually contain multiple resources, such as html text, CSS files, javascript files, and images. Modern web browsers support multiple parallel connections to a website. From a typical web page network traffic, we can see that a web browser sends multiple

requests to download multiple resources in parallel. Considering these web page features, we separate web page traffic into segments based on requests. We define two parameters: 1) *MaxNumberOfRequest* and 2) *MinTimediffBtwRequests*. *MaxNumberOfRequest* denotes the maximum number of web requests and *MinTimediffBtwRequests* denotes the minimum time interval between two web requests. Combining these two parameters, we can separate a network traffic trace into request segments. We scan the recorded traffic trace, find two consecutive outgoing packets which have a time interval greater than *MinTimediffBtwRequests*, then use these two packets as the delimiter of two request segments. If we get a higher number of request segments than *MaxNumberOfRequest*, the algorithm will adjust the requested segment time interval threshold accordingly to get exactly *MaxNumberOfRequest* number of request segments. To further control the distribution of incoming packets, we separate incoming packets into segments inside the request segment. We define two other parameters: 1) *MaxNumberOfResponse* and *MinTimediffBtwResponses*. These two parameters work the same as for outgoing packets.

To control the **overall** cover traffic size, we define another parameter *CoverTrafficLoadRatio*. The total cover traffic size will be the total packet size of network traffic multiplied by *CoverTrafficLoadRatio*. Since we separate the network traffic trace into segments, the size of cover traffic segments will be the size of corresponding traffic segments multiplied by *CoverTrafficLoadRatio*. This is equivalent to s from Section 3.1.1.

So far, we have parsed the recorded real web traffic, our next step is to generate the cover traffic. To do that, we have a cover traffic client agent running on the user side. This client agent connects to a cover traffic server, sends requests to the server and receives responses from the server. Figure 3.1 shows how our cover traffic is generated. The first step we need to do is to parse the recorded real traffic trace. As we can see from the sample traffic trace, the only information recorded is the time of

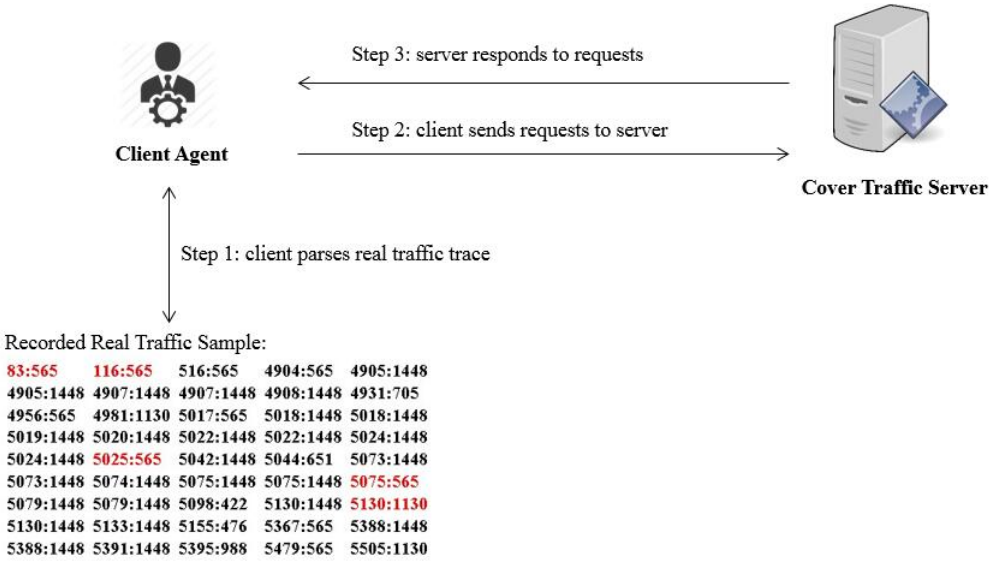

Figure 3.1: Cover traffic client and server.

packets and the packet sizes. Both incoming and outgoing packets are recorded. The format of the sample shown in the figure is as follows: ¡timestamp¿:¡packet size¿. We use negative packet sizes for outgoing packets and positive packet sizes for incoming packets.

The content of a request contains the following information: relative time to send back segments of responses and total packet size of each segment. Table 3.1 shows a list of requests sent to the cover traffic server. These requests are generated by parsing one recorded web traffic network trace. Each cover traffic request will be sent at a certain time ("Time to Send") and will be of certain size ("Total Size"). The request content shows the number of responses to be sent back by the server, along with the time and packet size of each response. Since all requests and responses are encrypted, no actual content is sent; the content of both the request and response can be filled with random data.

From Table 3.1, the client agent sends a total of nine requests to the cover traffic server. The first request is sent at time 0 (relative time), the size of the request is 703 bytes and the request content is "Response=0:676,354:18,756:91". We add padding

Cover Traffic Request	Time to Send (ms)	Total Size (bytes)	Request Content
1	0	703	Response=0:676,354:18,756:91;
2	7,320	4,149	Response=0:784,872:325;
3	9,190	8,087	Response=0:325,569:1045,1065:645,2000:325,6109:211;
4	19,196	1,0628	Response=0:217,380:108,657:942,1101:4581,1444:942,1920:8442;
5	27,094	703	Response=0:676,312:109;
6	29,434	148	Response=0:676;
7	30,395	358	Response=0:18;
8	31,166	197	Response=0:91;
9	38,185	543	Response=0:108;

Table 3.1: Cover traffic requests based on one recorded real web traffic trace. The request content contains the number of responses to be sent from the cover traffic server and is in the format $< relativetime >: \pm < packet\_size >$.

83:565 116:565 516:565 4904:565 4905:1448
4905:1448 4907:1448 4907:1448 4908:1448 4931:705
4956:565 4981:1130 5017:565 5018:1448 5018:1448
5019:1448 5020:1448 5022:1448 5022:1448 5024:1448
5024:1448 5025:565 5042:1448 5044:651 5073:1448
5073:1448 5074:1448 5075:1448 5075:1448 5075:565
5079:1448 5079:1448 5098:422 5130:1448 5130:1130
5130:1448 5133:1448 5155:476 5367:565 5388:1448
5388:1448 5391:1448 5395:988 5479:565 5505:1130

Figure 3.2: Sample of recorded network traffic. The format is ¡timestamp¿:¡packet size¿. Red indicates outgoing packets.

to the request content if its size is less then the expected request size. When the cover traffic server receives this request, it will send back 676 bytes of data at time 0. The time is relative to when the server receives the request. At time 0, that means the server just received the request. At time 354 ms, the server sends a response packet of size 18 bytes and at time 756 ms, the server sends a packet response of size 91 bytes. The contents of the response packet the server sends back to the client are filled with random characters. At time $7,320$ ms, the client agent sends the second request to the server.

3.1.3 Example

Taking the packet traces from Figure 3.2 as the example, the following two tables are built. Table 3.2 shows the parsed outgoing packets set, denoted as TS_out. Table 3.3 shows the parsed incoming packets set, denoted as TS_in. Most webbrowsing network traces have a higher number of incoming packets than outgoing packets. Moreover,

Traffic Train ID	Time and Packets
1	83:565
2	116:565
3	5025:565
4	5075:565
5	5130:1130
6	5560:565

Table 3.2: The parsed outgoing traffic train set.

the size of the incoming packets is higher than outgoing packets, which are usually web requests for a URL resource (such as jpg, html, etc...). This is typical of web traffic and is reflected in the tables. Generating noisy cover traffic works as follows.

1. Randomly select one traffic train from TS_{out} and TS_{in} each, denoted as T_{out} and T_{in} respectively. The total size of T_{out} is S_{out} and the total size of T_{in} is S_{in}.

2. Construct a cover traffic request with size S_{out}.

3. Send this request to the noise server.

4. The server will reply back with data of size S_{in} in WT_{in} milliseconds. WT_{in} is the time difference between the first packet of T_{in} and the first packet of the next traffic train after T_{in} in the incoming traffic train set.

5. Wait for some time WT_{out}, where WT_{out} is the time difference between the first packet of the chosen outgoing traffic train T_{out} and the first packet of the next outgoing traffic train.

6. Repeat steps 1 to 5 until the total incoming traffic from the noise server is equal to the size of all the incoming packets of the recorded traffic trace.

As an example, let's suppose outgoing traffic train 3 is selected from TS_{out} and incoming traffic train 8 is selected from TS_{in}. Our algorithm will create a new cover

Traffic Train ID	Time and Packets
1	516:565
2	4904:565 4905:1448 4905:1448 4907:1448 4907:1448 4908:1448 4931:705
3	4956:565
4	4981:1130
5	5017:565 5018:1448 5018:1448 5019:1448 5020:1448 5022:1448 5022:1448 5024:1448 5024:1448
6	5042:1448 5044:651
7	5073:1448 5073:1448 5074:1448 5075:1448 5075:1448 5079:1448 5079:1448 5098:422
8	5130:1448 5130:1448 5133:1448 5155:476
9	5367:565
10	5388:1448 5388:1448 5391:1448 5395:988
11	5479:565
12	5505:1130

Table 3.3: The parsed incoming traffic train set.

traffic request to send to the noise server. The request will ask the server to send back data of size $1448 + 1448 + 1448 + 476 = 4820$ bytes with a time of $5367 - 5130 = 237$ milliseconds. The request contains only total size of data to be sent and the time. For example, the server only needs to send padded data with size 4820. The lower level network interface will determine how to send each packet – if the MTU is 1448, packet size will be $1448 + 1448 + 1448 + 476$. The outgoing packet will be of size 565 bytes. Since the actual contents of the packet is small, the rest of the packet is padded. To simplify the example, we ignore packet headers. When the noise server receive this cover traffic request, it will send back data of size 4820 bytes and sleep for 237 milliseconds before responding to next request. At the same time, the client side waits for 55 milliseconds, which is the time difference between the first packets of the outgoing traffic train 4 and outgoing traffic train 5 from Table 3.2. This process is repeated until the sum of all the incoming packet sizes from the server is equal to the recorded traffic trace. The reason for waiting on both client and server sides is to ensure that the generated noise traffic is well distributed to look more realistic.

This generated cover traffic can achieve better performance in terms of obfuscating the overall traffic collected by a website fingerprinting attacker.

The user can choose as a parameter, the size of the cover traffic s. Since the cover traffic mimics the websites that the user has previously visited, the bandwidth used will be doubled. To minimize the bandwidth overhead, each train size could be reduced by a factor of s. If the factor s is 0.5, the incoming train will thus have a total packet size of $0.5 \times S_{in}$ in time WT_{in}. This reduces the bandwidth overhead and generates fewer packets.

We emphasize that the cover traffic is only between the client's webbrowser and the cover traffic webserver through Tor. The only data sent are padded data so that the packets are of a certain pre-determined size. The cover traffic generated will look realistic as it is traffic that was generated by the user. This recorded network traffic is only stored locally on the browser.

We expect our algorithm to effectively mitigate website fingerprinting attack since it has already been shown that cover traffic is effective. We expect that our algorithm will have lower bandwidth overhead since the amount of noise generated can be modified. Moreover, there is no extra latency added as no padding or network delay is introduced. Our algorithm only generates cover traffic to another website.

3.2 Experimental Setup

3.2.1 Simulation

We utilized the dataset provided by Panchenko et al. (2016), which consists of $1,125$ webpages and 40 instances of each webpage. Each instance contains the timestamp of each packet along with the packet sizes (negative packet sizes indicate outgoing packet). We implemented the noise generation algorithm described in Section 3.1.

Due to the new data generated by our noise generation algorithm, we could not re-use the authors Panchenko et al. (2016) SVM algorithm. Instead we use the stan-

dard Weka wek (2019) tool and experimented with different classification algorithms: Support Vector Machine (SVM), decision tree (REPTree in Weka), neural network (Multi-layer perceptron), linear regression, and random forest.

The six classification features used in our experiments are similar to those used in Panchenko et al. (2016). The first four features are: total size of outgoing packets, total size of incoming packets, total number of outgoing packets, and total number of incoming packets. The remaining two features are the sampled cumulative representation of packet size. There are two ways to calculate the cumulative packet size: c is the cumulative size of packets size where an outgoing packet has a negative packet size and a is the cumulative size of packets size where both outgoing and incoming packet sizes are denoted as positive numbers. The number of samples used n can be varied and will be taken at equidistant points in the packet trace. For example, if there are 75 packets and $n = 100$, a sample is taken every 0.75 packet. To determine the packet size of the 0.75th packet, the linear interpolation is calculated. If the 0th packet size was 10 and the 1st packet size was 20, the cumulative packet size for 0 is 10 and the cumulative packet size for 1 is $20 + 10 = 30$. The 0.75th packet size is thus $(0.75 * (30 - 10)) + 10 = 25$.

We compared our proposed cover traffic algorithm with the basic cover traffic scheme. The latter works as follows. When a user visits a website, the basic scheme will randomly pick another website to also visit. As shown by Wang and Goldberg (2016), having two simultaneous website visits significantly lowers website fingerprinting accuracy.

The original dataset contained $1,125$ webpages, many from the same website. We filtered out webpages of the same website and used 91 websites as our base training dataset [1]. The dataset Panchenko et al. (2016) contained timestamps and packet sizes. Merging the original website packet trace with the noise packet trace is relative

[1]Note that since each webpage is a unique website, we used webpage and website interchangeably from now on.

straightforward. Since there are 40 instances of each website, we randomly picked one instance as the noise data to merge with the original packet trace.

We considered two different basic cover traffic algorithms. The first one always picks the same webpage (but possibly different instances). The second one randomly picks from a set of 10 webpages different from the 91 previously selected. The second case provides a more diverse set of webpages and noise to be added.

Our noise generation algorithm "learning" dataset consists of a further 10 webpages where the packet traces are recorded. For each of the original 91 webpages, we ran our algorithm to generate one packet trace of noise and merge that trace with the original webpage packet trace.

3.2.2 Real Experiment

We collected network traffic data for the top 100 web sites listed from Wikipedia on April 17, 2017. After we removed duplicates (e.g. google.com and google.co.uk) and adult websites, we were left with 75 websites. For each website, we recorded 20 instances of network traffic through Tor without noise and 20 instances of network traffic through Tor including traffic from the noise server. Each network traffic trace's duration was two minutes. We used Wireshark version 1.6.7 to capture packets at the TCP level and TorBrowser version 6.5.2 as the web browser. The computers used were Dell Optiplex with Intel i5 and 4GB of RAM. We note that Wireshark cannot differentiate whether a packet is from the noise server or from the webserver. All the traffic looks like it originates from the Tor network. The following steps outline how we record the network traffic of a website.

1. Launch Wireshark to record network traffic

2. Launch TorBrowser and visit a webpage

3. Launch cover traffic client agent (only for the experiments with noise)

4. Wait 2 minutes

5. Save network traffic to file

6. Shut down Wireshark and TorBrowser

The cover traffic server is deployed on a different machine from the client. The cover traffic client agent runs Tor version 0.2.2.35; all the cover traffic are forwarded to the noise server through the Tor client. The noise traffic consists of the the network traffic trace of the top 10 web pages; in reality, this would be the traffic trace of websites visited by the user. Each time the cover traffic client needs to generate noise, it will randomly pick on these ten traces and sends requests and receives responses from the noise server as described in Section 3.1.2. In our experiments, we set *MinTimediffBtwRequests* to 500 milliseconds, *MaxNumberOfRequest* to 10, *MaxNumberOfResponse* to 10 and *MinTimediffBtwResponses* to 200 milliseconds. Figure 2.2 illustrates how our experiments are set up.

3.3 Evaluation

3.3.1 Simulation Results

Figure 3.3 shows the classification accuracy for varying amount of noise added to original traces. Figure 3.4 shows the bandwidth overhead in % of the extra netwok traffic generated. The two basic cover traffic algorithms are indicated by $k = 1$ for adding the same one website as noise each time and by $k = 10$ for randomly adding one of ten websites as noise. The x-axis indicates the amount of noise s added. When $s = 1.0$, this means the whole packet trace is added as noise. When $s = 0.5$, only half of the packet trace is added as noise, that is, every other packet is added as noise to preserve the time intervals. For the basic cover traffic cases ($k = 1$ and $k = 10$), we are "simulating" the noise generated; in a real-world setting, this would be hard to achieve without controlling the server – in this case, the browser could

send random packets. We show different values of s to compare with our algorithm. As more noise is added (s increases), the accuracy decreases, as expected. Similarly, the bandwidth overhead also increases as more noise is added. Our proposed noise generation algorithm achieves the same accuracy regardless of the amount of noise; this is because we are generating realistic noise that can more effectively hide a user's real traffic rather than generating random noise. Our algorithm's bandwidth overhead is the same as the basic cases. However, even with $s = 0.25$, the overhead is 20% and the accuracy is 14%. Since our proposed algorithm generates random packet traces based on real recorded network traffic, we ran our experiments five times; the graphs show the average of the five experiments. For these experiments, the training dataset used in the Random Forest classification algorithm is the original 91 webpages and the testing dataset is the new webpages with noise added.

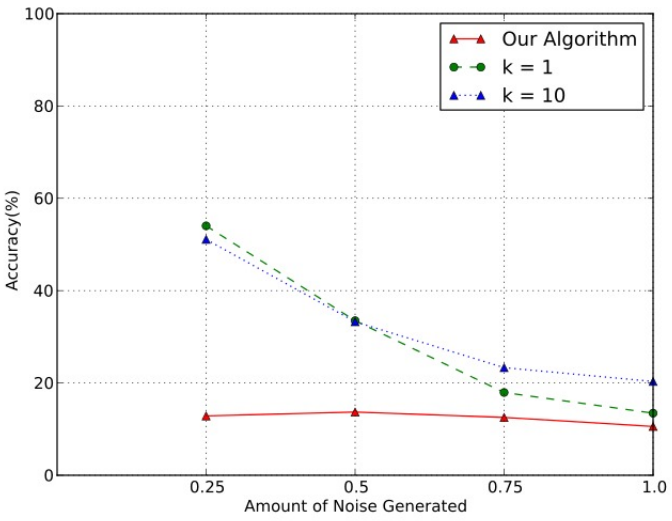

Figure 3.3: The accuracy using the Random Forest algorithm when introducing different kinds of cover traffic.

Intuitively, accuracy should decrease as more noise is added. However, in Figure 3.3, we found that in our algorithm, $s = 0.25$ has a lower accuracy than $s = 0.5$. We hypothesized that this could be due to the features being considered. Recall that two of the five features are the total number of incoming packets and the to-

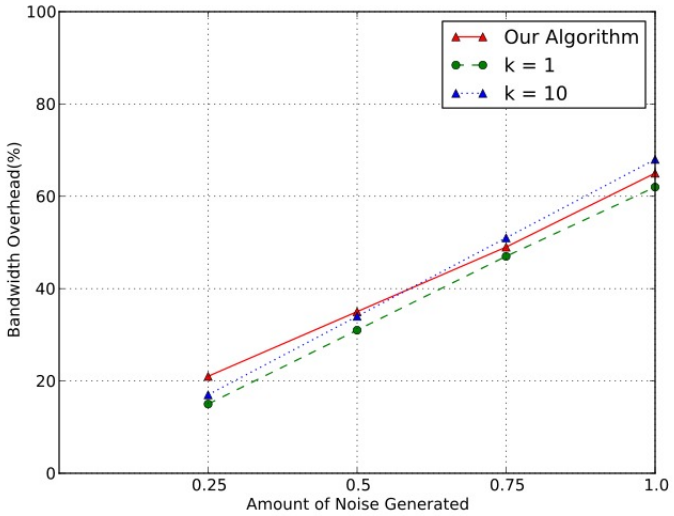

Figure 3.4: The bandwidth overhead when introducing different kinds of cover traffic.

tal size of incoming packets. When s is lower, the number of incoming packets is lower. To verify our hypothesis, we re-ran our algorithms without considering these two features of incoming packets. Figure 3.5 shows the result. It can be seen that without these two features, accuracy decreases as noise generated increases, which is expected. This shows that the attributes for total size of incoming packets and total number of incoming packets help the website fingerprinting adversary in successfully identifying the correct website (increase in accuracy). Without these two features, our proposed algorithm performs even better as the accuracy is reduced to under 10% when $s = 1.0$. Previous work Wang and Goldberg (2016) has shown that the number of incoming packets is one of the most useful attributes in classification for website fingerprinting. We also considered not including the incoming packet features; the results are shown in Figure 3.6. The results are expected as well but the change in accuracy is not as obvious as Figure 3.5 since the number of outgoing packets is about the same regardless of the value of s.

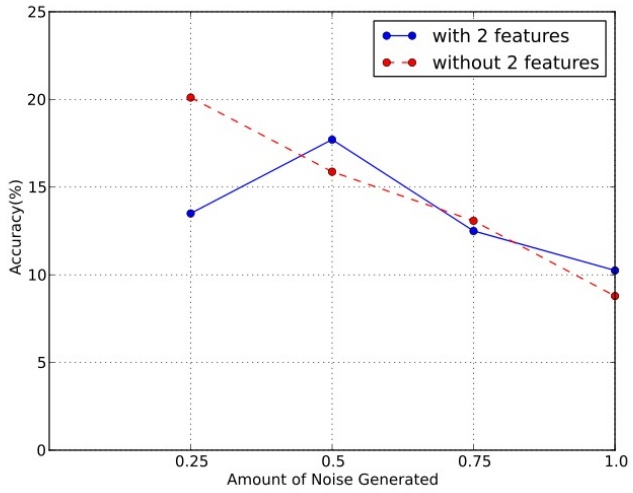

Figure 3.5: The accuracy using the Random Forest algorithm when considering the incoming packet features or not.

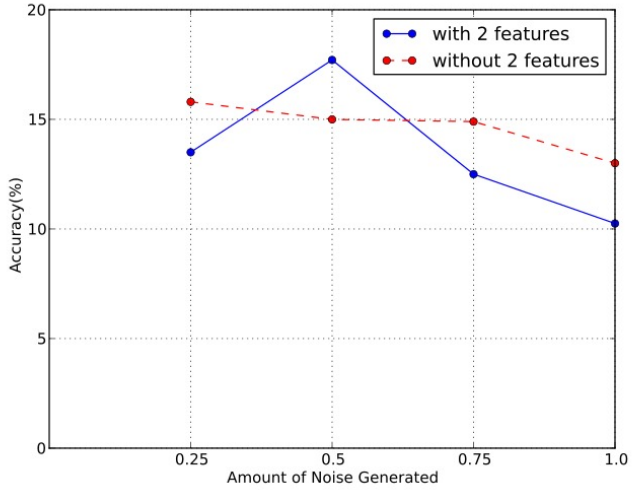

Figure 3.6: The accuracy using the Random Forest algorithm when considering the outgoing packet features or not.

3.3.2 Real-World Experiment Results

We deployed our real-world experiments from May 13 to August 10. We visited 75 websites and collected data for 20 instances of each of the 75 websites. This was our training dataset. We then repeated the experiments for the same websites and number of instances for each website with cover traffic generated by the noise server. This made up our testing dataset. Similar to the simulations, we used the Random Forest classification algorithm to perform the website prediction. Figure 3.7 shows the accuracy of website fingerprinting. When no cover traffic is generated, the accuracy is 81.59%, which is close to the 90% obtained in previous research. However, when generating 10% cover traffic, the accuracy decreases to 17.81% which shows that our cover traffic algorithm is significantly impacting the adversary's ability to perform a website fingerprinting attack. When generating 20% and 30% cover traffic, the accuracy obtained is 16.37% and 10.72% respectively. We also ran the k-nearest neighbors classifier as this was used in previous research. The accuracy obtained is similar to that obtained when using the Random Forest algorithm.

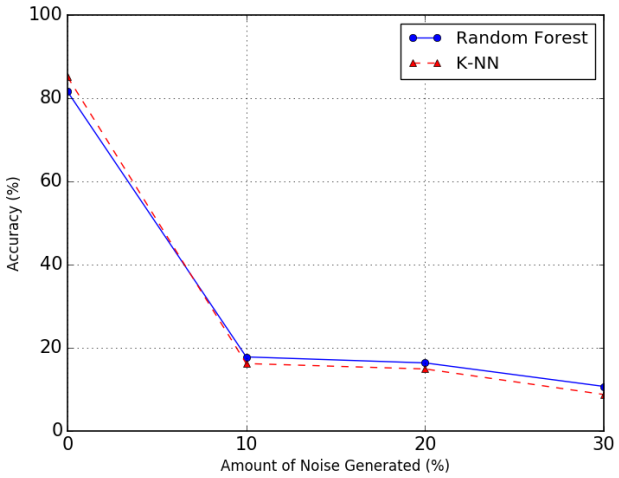

Figure 3.7: The accuracy of the website fingerprinting attack for real-world experiments with varying amounts of noise generated. Classifier used was Random Forest.

Table 3.4 shows a comparison of our proposed algorithm with existing mitigation

Mitigation	Accuracy (%)	Latency Overhead (%)	Bandwidth Overhead(%)
No Defense	91%	0%	0%
CS-BuFLO Cai et al. (2014a)	22%	173%	130%
Tamaraw Wang et al. (2014)	10%	200%	38%
WTF-PAD Juarez et al. (2016)	15%	0%	54%
Our Algorithm (simulation)	14%	0%	20%
Our Algorithm (experiment)	16%	0%	20%

Table 3.4: Comparison of our algorithm's accuracy and overhead with previous mitigation schemes. We showed the lowest accuracy numbers for the other schemes, regardless of algorithms used. The table is based from Juarez et al. (2016).

techniques. Our algorithm has comparable accuracy with the other schemes, zero latency overhead, and lower bandwidth overhead. The table shows the lowest accuracy (best-case for the mitigation) regardless of the classification algorithm used. Our algorithm has zero latency overhead since we are only introducing cover traffic. No padding or delays are introduced.

3.4 Related Work

It has been shown that analyzing encrypted network traffic can reveal the websites and webpages visited Hintz (2003); Sun et al. (2002); Bissias et al. (2006); Liberatore and Levine (2006); Lu et al. (2010); Herrmann et al. (2009); Panchenko et al. (2011); Gong et al. (2012); Wang and Goldberg (2013); Cai et al. (2012); Wang et al. (2014); Nithyanand et al. (2014); Juarez et al. (2014); Cai et al. (2014b); Panchenko et al. (2016); Wang and Goldberg (2016); Spreitzer et al. (2016); Hayes and Danezis (2016). Since the payload is encrypted, only the metadata is available such as packet sizes, number of packets, direction of packets, and time interval between packets. A training dataset is built. Then, given a network traffic trace, machine learning techniques are used to predict the website visited. Previous results have shown that websites can be recognized with a high accuracy. More recent research results have looked at anonymized network traces such as using Tor instead of a simple HTTPS proxy. Although initial results showed that Tor provided adequate protection against website fingerprinting, more advanced data parsing techniques show that websites can be

recognized with a fairly high accuracy even when the website trace is over Tor. The consequences of website fingerprinting is censorship or prosecution by the government if the user visits a forbidden website. It has been argued Perry (2011) that website fingerprinting is not a practical attack due to the large number of webpages and the false positive would be high. Website fingerprinting attacks have also been extended to identify the webbrowser used Yu and Chan-Tin (2014), which could lead to user identification and linking as most users utilize a unique webbrowser (based on fonts installed, languages, plugins, etc...) Eckersley (2010); pan (2017).

Website fingerprinting is one type of network traffic analysis. There has been other work on network traffic analysis Miller et al. (2014) and traffic analysis resistant protocols Le Blond et al. (2013); Dyer et al. (2012); Mittal et al. (2011). Network traffic analysis is usually performed for censorship Tschantz et al. (2016). Various techniques to avoid censorship have been proposed, using traffic morphing Wright et al. (2009a) to disguise the network traffic as VoIP Houmansadr et al. (2013b); Moghaddam et al. (2012) or using other covert channels Fifield et al. (2012); Wang et al. (2012); Holowczak and Houmansadr (2015). It has, however, been shown that it is still possible to see through this obfuscation Wang et al. (2015); Houmansadr et al. (2013a); Geddes et al. (2013).

Using cover/dummy/fake traffic to mask a user's activities has been proposed before Diaz and Preneel (2004). It has been shown that this mechanism can be countered or the cover traffic removed Mallesh and Wright (2007); Simon Oya and Pérez-González (2014) to reveal the user's activities. Cover traffic is useful to mask real web search queries by performing many other unrelated and random search queries. Cover traffic can also be used to make network traffic analysis harder by adding unrelated network-level packets. Our algorithm generates realistic cover traffic making it harder for website fingerprinting attacks to accurately guess the website from the observed packet trace. Another scheme, Track Me Not Howe and Nissenbaum (2008), focused

on web search queries and generating fake web searches, but Peddinti and Saxena (2010) has shown that web search queries obfuscation can still be analyzed.

Various website fingerprinting defenses have been proposed Panchenko et al. (2011); Cai et al. (2012, 2014b); Nithyanand et al. (2014); Cai et al. (2014a); Wang et al. (2014); Juarez et al. (2016). They all make use of some sort of padding, delaying sending of packets, or adding cover traffic. Many of these defenses have high latency and/or bandwidth overhead and have been shown to be somewhat effective in mitigating website fingerprinting attacks. Our proposed cover traffic defense has zero latency overhead and lower bandwidth overhead while maintaining a high level of effectiveness.

Traffic morphing Wright et al. (2009b) is another possible defense against website fingerprinting attacks. It attempts to modify the shape and patterns of network traffic such that it looks different. For example, Stegotorus Weinberg et al. (2012) attemps to make Tor network traffic look like HTTPS. Similarly, Moghaddam et al. (2012); Houmansadr et al. (2013b) attempt to morph Tor traffic to look like VoIP traffic so that network traffic analysis or deep packet inspection will not allow Tor traffic to be blocked or identified; VoIP traffic is usually allowed. However, Houmansadr et al. (2013a); Geddes et al. (2013) have shown that these traffic morphing schemes can be circumvented. We are not proposing to modify network traffic patterns. Our algorithm generates realistic cover traffic to mask the original packet trace.

3.5 Summary

We showed that our proposed cover traffic (noise generation) algorithm mitigates website fingerprinting attacks as effectively as current existing schemes. However, the bandwidth overhead is only 20% for simulation and 10% for real-world experiments, much lower than existing schemes. The latency overhead is also 0%. Our algorithm can also be configured to utilize different amounts of bandwidth.

Recording user's browsing session: We emphasize that the user's webbrowsing session only need to be recorded locally. This information is not shared. Moreover, only the packet sizes and number of packets are recorded. The server and actual contents are not recorded. The information sent to the noise server is only the number of packets to send back and their size. The contents of the packets are random, not actual contents. Since the packets are encrypted, an adversary cannot determine that these packets are cover traffic. Since no actual contents are sent, our scheme does not leak any information to an eavesdropper.

Using a dedicated noise server: The cover traffic could be sent to real webservers; however, this would put extra strain on these servers. We, thus, decided to use a dedicated noise server. Removing the noise server will mean only outgoing cover traffic can be sent which could be filtered out by an adversary. Multiple noise servers, such as using Amazon cloud, could be use if this scheme is deployed. Since Tor is used, it cannot be determined whether the user is contacting a noise server.

We plan to expand this work in considering more webpages for both the training dataset and our learning algorithm. A more detailed study on the different classification algorithms and parameters used will also be performed. Further improvements to our algorithm can be made, such as, if a user has multiple tabs open at the same time, no noise is needed. This would reduce the bandwidth overhead.

CHAPTER IV

REVISITING ASSUMPTIONS FOR WEBSITE FINGERPRINTING ATTACKS

4.1 Background

- **Definitions**. We first define some terms we use throughout the paper.
  - Continuous Trace. When a trace consists of two pages, and the second page starts when the first page ends, we call it a continuous trace. It has the same meaning as when the two pages are separated with zero-time.
  - Split Point. When a trace is composed of two pages, the first step is to separate them before further detecting. The point where the second page starts and the first page ends is the split point.
- **Dataset**. Based on the foreground dataset of RND-WWW from Panchenko et al. (2016), our experiments in Section 4.2 randomly pick 100 website records which contain 40 instances for each website from the original dataset. Each instance is a trace containing the timestamped incoming and outgoing packets' size in chronological sequence. Incoming packets are marked with a positive sign, while outgoing packets are marked with a negative sign.
- **Hidden Markov Model**. The Hidden Markov Model (HMM) is a Markov process with unobserved states. It is a statistical tool to model sequences that can be characterized by a process from a generated observable sequence Blunsom (2004). Based on some training data, the HMM generates the probabilities of the states in the dataset. The parameters of a HMM are of two types: transition

probabilities and emission probabilities. The transition probability indicates the probability that a state changes to another state and the emission probability is the probability of an observation within a state. The transition matrix and emission matrix store the transition probability and emission probability of each state respectively.

- **Classification of single-page and two-page traces**. An approach was developed to distinguish traces between one-page trace and two-page traces in Wang and Goldberg (2016). The authors employed k-NN binary classification and trained on two classes: a class of two-page traces (a network trace consisting of two webpages), and a class of single-page traces (a network trace consisting of only one webpage). The classification accuracy is 97%. Based on their results, we assume it is capable to identify a trace with single page or two pages.

4.2 Analysis of Continuous Traces

Notation	Definition
n_{wb}	the number of websites
n_{unique}	the number of packets with unique sizes in all website traces
$n_{p_{As}}$	the number of a packet size p in website A $\{start\}$ state
$n_{total_{As}}$	the total number of packets in website A $\{start\}$ $state$
s_{block}	the size of each block
p_{final}	a matrix of the probabilities of each packet belonging to each class/website
l_{trace}	the length of a trace

Table 4.1: Notations used in our algorithm.

In this section, we introduce our algorithm based on Hidden Markov Model to detect two continuous websites with zero-time separated. We describe the details of the algorithm first, followed by the experiments and evaluations of this algorithm. The notations used in the algorithm are introduced in Table 4.1.

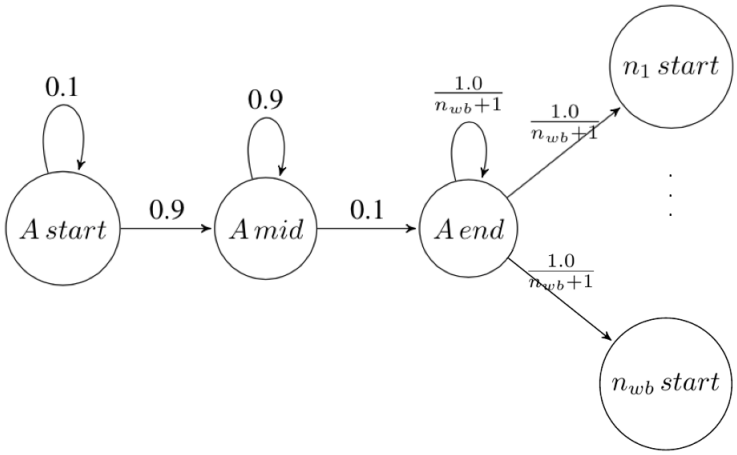

Figure 4.1: State transition of website A.

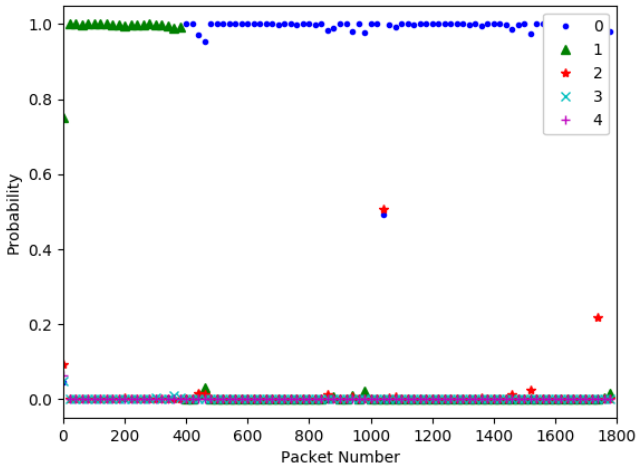

Figure 4.2: Probability of each packet belonging to each website(the sum of three states in each website) obtained from the HMM model (print every 20 point for clarity, best viewed in color).

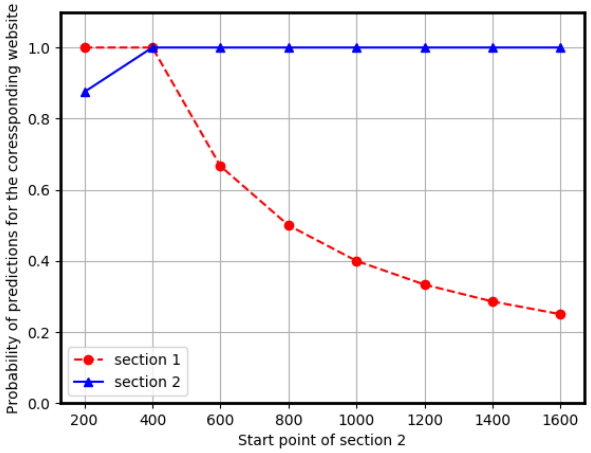

Figure 4.3: Probability of predictions for the corresponding website (probability of website1 belonging to section1 and probability of website0 belonging to section2) when moving the split point between section 1 and section 2 from left to right. See Figure 4.2 for the actual predictions.

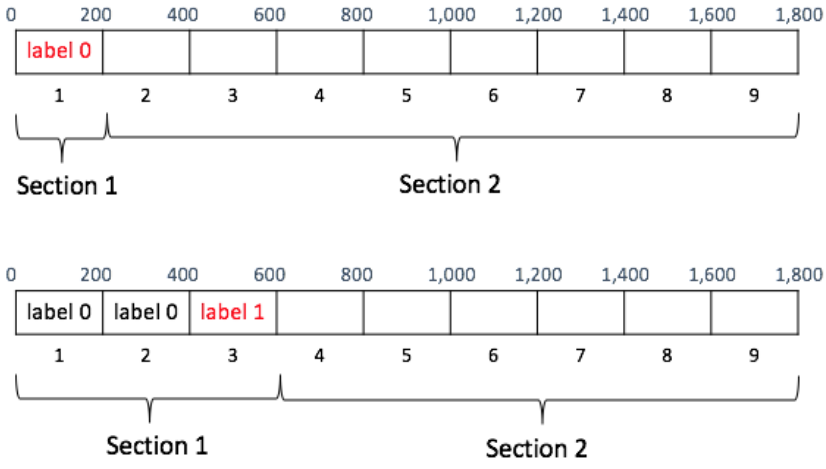

Figure 4.4: Example when labeling block 1 and block 3.

4.2.1 Algorithm description

Our proposed algorithm can be divided into two steps: 1) Apply Hidden Markov Model to get the probability matrix, 2) Label each block based on the probability matrix and pick split point based on labels.

Step 1: Apply Hidden Markov Model to network traffic trace and obtain the probability of each class (website) that each packet belongs to (probability matrix).

To form states, we split each packet trace into three parts: 1) *start* which is first 20 packets in the network traffic trace, 2) *middle* which is the collection of packets between *start* and *end*, and 3) *end* which is last 20 packets in the network traffic trace. We then build our transition and emission matrices. The dimension of the transition matrix would be $(3 \times n_{wb})^2$.

We use website A as an example. Assume the length of a trace is l_{trace}; this is the number of packets in website A. Figure 4.1 shows the state transition within website A. For a packet from website A, it has $\frac{20}{l_{trace}}$ probability to belong to A's start state, $1 - \frac{40}{l_{trace}}$ probability to belong to A's middle state and $\frac{20}{l_{trace}}$ probability to belong to A's end state. If a packet in A is in the *start* state, then for the next packet it has a $\frac{20/l_{trace}}{1-(40/l_{trace})+(20/l_{trace})}$ probability to stay in its current state and $\frac{1-(40/l_{trace})}{1-(40/l_{trace})+(20/l_{trace})}$ probability to change to the *middle* state. From analysis of the dataset, we find that $\frac{20}{l_{trace}} = 0.9$ or 9%, thus we set $\frac{20/l_{trace}}{1-(40/l_{trace})+(20/l_{trace})}$ as 10% and $\frac{1-(40/l_{trace})}{1-(40/l_{trace})+(20/l_{trace})}$ as 90%. In a similar way, if A is currently in the *middle* state, then the next packet could stay in the *middle* state with 90% probability or change to the *end* state with a 10% probability. When the packet is in the *end* state, we set that A has an equal probability of $\frac{1.0}{n_{wb}+1}$ to stay in the *end* state or change to any other website's *start* state. For the emission matrix, the dimension is $(3 \times n_{wb}) \times n_{unique}$ where n_{unique} is the number of unique length of packets in all website traces, where 3 indicates the three states (*start*, *middle*, and *end*) for each website. For a packet size P in website A in the *start* state, $n_{p_{As}}$, the total number of packets in A *start* state is $n_{total_{As}}$. The emission

probability for P in A's start state would then be $\frac{n_{p_{As}}}{n_{total_{As}}}$. After applying forward and backward propagation, we obtain the probabilities of each packet belonging to each class/website as p_{final}. For simplicity, we show the prediction for five websites in Figure 4.2. The split point happens at packet number 389 and the network traffic trace is composed of website 1 followed by website 0. The different colors of the lines indicate different websites. A website's different states are shown in the same color. From the figure, we can see that the algorithm predicts website 1 with a probability higher than any other website until about the split point where the algorithm predicts website 0 with the highest probability.

Step 2: Find the split point.

The split point needs to be identified automatically. The main idea of this step is to traverse each packet in a trace from left to right as a split point and measure the probability of the website in two sections, section 1 and section 2, split by the split point. Based on the example in Figure 4.2, Figure 4.3 shows the trend of the ratio in section 1 and section 2. By moving the split point from left to right, the probability of predicted website (website 1) in section 1 drops while the probability of predicted website (website 0) increases in section 2. The split point 400 occurs at the intersection of the two lines in Figure 4.3.

Instead of analyzing each packet, we decide to extract features from blocks to reduce the processing time. We divide the whole trace into several blocks with multiple packets; the size of each block is s_{block}. Assume the length of a trace is l_{trace}, then a trace is split into $n_{block} = l_{trace}/s_{block}$ blocks. We name the block from left to right as block 1, block 2, ..., block n. And we assume the split point is at the end of one of the blocks.

First, we label each block to indicate whether the block is before or after the split point. When labeling block m, we consider block 1 to block m to be *section 1* and block $m+1$ to block n as *section 2*. The block m is labeled based on the ratio r_{s1}

of the number of packets belonging to website X in *section 1* and the ratio r_{s2} of the number of packets belonging to website Y in *section 2*, where X and Y indicate the website with the highest ratio in *section 1* and *section 2* respectively. We set a ratio bottom line t_{block} for r_{s1} and r_{s2}. When labeling the block, we use 0 to represent the block is before the split point and 1 to represent the block is after the split point. If $r_{s1} > t_{block}$, which indicates more than t_{block} section1 is composed of website X , and label the block as 0. If $r_{s1} < t_{block}$ and $r_{s2} > t_{block}$, label the block as 1. If $r_{s1} < t_{block}$ and $r_{s2} < t_{block}$, it indicates that this block does not provide valid information, then this block won't be labeled and recorded. When labeling every block, record the end point of each block as a block index into *point_list*.

We use the example in Figure 4.2 to illustrate the algorithm behind the labeling process. Figure 4.3 and Figure 4.4 are based on this example. The trace contains $1,800$ packets and is divided into 9 blocks; the size of each block is 200 packets. The split point between section 1 and section 2 moved from 200 to 1600.

We only list the process when labeling block 1 and block 3 as an example. For a packet, we use the website with the highest probability as the prediction for the packet. In this example we set t_{block} as 95%. When labeling block 1, section 1 contains block 1 and section 2 is from block 2 to block 9; split point is 200. From Figure 4.2 we can see that the predicted website in section 1 is website 1 and from Figure 4.3, the confidence of this prediction is close to 1, which means, r_{s1} is close to 100% thus $r_{s1} > 95\%$, then we label block 1 as 0, meaning block 1 is before the split point. For labeling block 3, section 1 is composed of block 1, block 2 and block 3, and section 2 is from block 4 to block 9. For the first 600 packets in section 1, the probability of website 1 is 67% which is less than 95%, then we label block 3 as 1 representing block 3 is after the split point. Under the perfect condition, the split point should be at the point when r_{s1} and r_{s2} meet which is in between 0 and 1 in the label list.

After labeling each block, we pick the split point based on the sequence of labels.

The expected list of labels is $\{0, 0, ..., 0, 1, 1, ..., 1\}$. However, when labeling block n, if none of the highest ratio in section 1 and section 2 achieve the threshold t_{block}, the label of this block will be missed. We propose an algorithm to find the split point and is able to handle all these situations. The two main cases are classified by whether 1 is in the *label_list*.

- **label_list contains 1s.** If $label\_list = (0, 0...0, 1, 1...1)$, that is the format we expect. 0 represents the block is before the split point and 1 is the opposite. Then the split point is after the last block labeled with 0.

  If $label\_list = (1, 1, 1...1, 0, 0.., 0, 1, 1, ..., 1)$, it shows that there is some noise at the beginning of the trace as well as some at the end of the trace. However it does not affect the process to find the split point since we only focus on the changes in the trace. We will still assume the split point is after the block with last 0.

- **label_list contains 0s only.** The algorithm will check if enough information is obtained first before making the decision. If blocks are continuously labeled from the first to last block, then we assume that pattern for the probabilities of the first website is clear and return the point after the last labeled block as the split point. This means that the last block is section 2. Otherwise, the backup algorithm will be applied.

The pseudo code of the algorithm is outlined in Algorithm 1. *label_list* contains the labels of blocks and *point_list* is composed of the index of the corresponding block. The condition in the algorithm is based on the sequence in the *label_list*. The comments on the right describe the specific situation when a *label_list* is in that condition. When no points satisfy the conditions or there is only one element in the *point_list*, we need a backup algorithm to pick the split point.

The main idea of the backup algorithm is to find the split point when the average of the highest ratio of predicted websites in section 1 and section 2 is higher than

Algorithm 1 Main Algorithm to Find Split Point

```
1: procedure GET_CHANGEPOINT(point_list, label_list)
2:     if point_list is not empty then
3:         if 1 is in label_list then
4:             if label_list[0] is 0 then                    ▷
   label_list = (0, 0...0, 1, 1...1)
5:                 Return point after last 0
6:             else     ▷ label_list = (1, 1...1, 0, 0...0, 1, 1...1)
7:                 if label_list[0] is 1 then
8:                     Return point after last 0
9:                 else
10:                    Return backup algorithm
11:                end if
12:            end if
13:        else                          ▷ label_list contains 0 only
14:            if length(point_list) is 1 then
15:                Return backup algorithm
16:            end if
17:            if point_list[0] is s_block and point_list =
   (1 s_block, 2 s_block, ..., n s_block) then
18:                Return point_list[n − 1]       ▷ return last
   point in the point_list
19:            else
20:                Return point_list[0]
21:            end if
22:        end if
23:    else                                  ▷ point_list is empty
24:        Return backup algorithm
25:    end if
26: end procedure
```

any other point. Assume that $point\_list = (1\ s_{block}, 2\ s_{block}, ..., n\ s_{block})$, for split point $i - s_{block}$, where $i = 1, ..., n$. The two sections split by this point i are called section 1 and section 2. We denote the percentage of packets belonging in section 1 as $r1_i$ (that is, these packets are correctly marked in the correct section) and the percentage of packets belonging in section 2 as $r2_i$. The average ratio at point $i - s_{block}$ is $avg(r1_i, r2_i)$ – the point with the highest average ratio among all points in the *point_list* is considered as the split point. The backup algorithm is rarely called in our simulations.

4.2.2 Results for Finding Split Point

The values of s_{block} and t_{block} are selected as 200 and 95%. We used the dataset foreground RND_WWW and CUMUL features from Panchenko et al. (2016) and randomly picked 100 distinct websites with 40 instances each from the dataset. For each website, 20 instances are applied in training and the other 20 are used for testing. Training dataset is then composed of 100 websites with 20 instances each. In order to simulate the process of visiting one website after another, we picked two websites randomly from the testing set 200 times and concatenated their network traffic trace to form the test set. Since each website for the testing set has 20 instances, there are $4,000$ traces in total in the testing dataset.

We removed the packets with size of Maximum Transmission Unit (MTU) to improve the accuracy. We also consider another threshold in addition to p_{final}, that is, if the highest probability of a packet belonging to every class is lower than the threshold $t_{original}$, where $t_{original}$ is set to 0.8 in the experiment, then that packet will be ignored. Thus we only consider the predictions with high probability. We found that by removing MTU and adding this new threshold value, the accuracy is increased.

We analyze the accuracy of the split point from two metrics: 1) the related

deviance of the predicted split point from the real split point and 2) the accuracy of the split point. The related deviance is calculated by $abs(predicted\_point - real\_point)/l_{trace}$. The lowest average related deviance we obtained when testing on 10 and 100 websites are 0.154 and 0.16 respectively, which means the performance of the algorithm is stable when increasing the number of websites. If the predicted split point is before the real split point, the first website loses partial data at the end, and the second website receives extra data at the beginning. Figure 4.5 shows the decrease in prediction accuracy under a closed world setting. The test dataset is composed of 12 parts; each part contains 100 websites and 20 instances of each website. The first 6 parts consist of cutting 30%/20%/10% traces at the beginning or end of each trace, The second 6 parts are composed of adding 30%/20%/10% traces at the beginning or end of each trace. It shows the effect on prediction accuracy for the first and second website in a continuous trace when the predicted split point is before or after the the real split point. For two continuous websites, the error on the split point has a bigger effect on the second website. A 0.16 related deviation means the average split point is between 0.84 to 1.16 on the x-axis in Fig 4.5. In this area, it can detect the first website with a decreased accuracy of 15%. Since the original accuracy in detecting one website with website fingerprinting using the k-NN algorithm is about 90% (from figure 4.3), the accuracy to predict the first website is thus around $90\% - 15\% = 75\%$. However, for the second website, the accuracy is lower. To calculate the accuracy of the split point, we consider that the prediction is correct if the block/point prediction is closest to the real split point. The number of points/blocks is decided by l_{trace}/s_{block}, where s_{block} is selected as 200. For example, if the length of the continuous traces is $3,000$ with the split point at packet number 425, and $point\_list = \{200, 400, 600, 800, ..., 2800\}$, we consider the prediction is correct if the predicted split point is 400. Among $4,000$ test traces, $3,200$ of them are predicted with the correct split point. The split point accuracy is thus 80%.

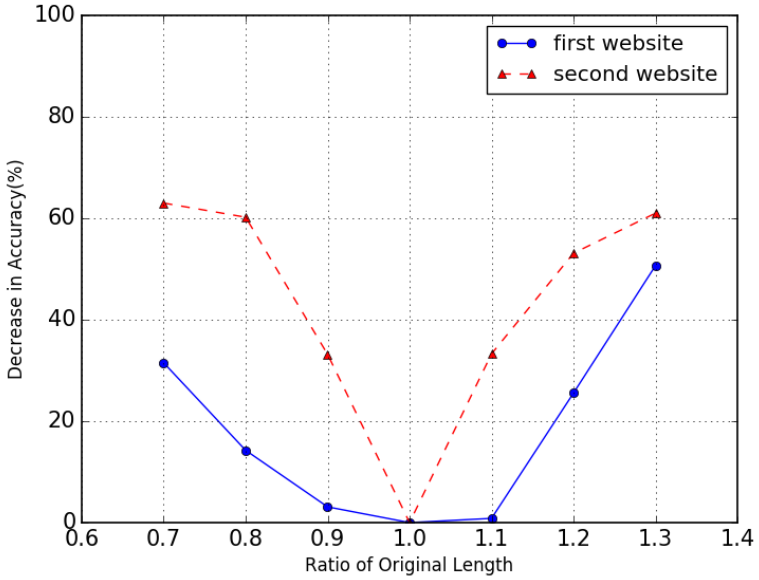

Figure 4.5: Decrease on prediction accuracy of the first and second website when the predicted split point is not accurate.

4.2.3 Results for Website Prediction

The ultimate goal of WF attack is to predict visited websites. The advantage of this algorithm is that it can detect the website directly after finding the split point. We still use $t_{original}$ to filter packets first and assume the packet belongs to website A if A has the highest probability among all websites (known from p_{final}). Then we calculate the percentage of each website before and after the split point. The website with the highest percentage is the predicted website. We trained on 100 websites and tested on $4,000$ instances. When considering websites with the top three highest probabilities, the accuracy for the first website is 70.2% and the accuracy for the second website is 69.2%.

4.3 Summary

In summary, our "splitting" algorithm has three distinct advantages over Wang and Goldberg (2016). First, it doesn't require new training data, only based on original

website traces. Second, it has a higher accuracy of 80% in detecting split point compared to 63%. Third, Wang and Goldberg (2016) didn't predict websites for real after finding the split point. From their description, they will reuse previous WF attack approach on two split sections. However, in our algorithm, websites can be predicted directly after finding the split point from the probability matrix and predicted split point.

CHAPTER V

MORE REALISTIC WEBSITE FINGERPRINTING USING DEEP LEARNING

5.1 Background

In this study, we mainly focus on two major types of neural networks that previous work has shown to be promising for WF attacks: 1) CNN, a convolutional neural network, and 2) LSTM, a recurrent neural network.

- **Convolutional Neural Network (CNN).** In Deep Learning, a CNN is an algorithm consists of a group of convolutional layers: an input layer, hidden layers and an output layer. Each hidden neuron only connected to a restricted region called receptive field. And these receptive fields partially overlap with neighborhoods. Thus CNNs can go deeper with less parameters when doing feature extraction. CNN may have pooling layers to downsample from prior layer. Such as max pooling, it uses the maximum value of a group of neurons to do the downsampling. Due to the advantage of its good performance in classification and its efficiencies on pre-processing, CNNs has been applied to process image recognition and classification, object detection etc.

- **Long-short term Memory network (LSTM).** A LSTM, a special kind architecture of a recurrent neural network (RNN), is developed to enhance the ability of deal with long-term dependencies problem of RNN. Because of the design of a LSTM, it can process data based on time series very well. In our experiments, WF traces are time series data. Hence, LSTMs can be used to

analyse WF traces.

5.2 Methodology

Current WF techniques are usually developed under strong assumptions that each traffic trace corresponds to an entire single webpage. However, the captured trace may contain 1) one or more webpages, 2) a partial webpage, or 3) webpage with background noise. In this section, we provide a detailed outline of our DL based methodology on WF attacks with these realistic conditions.

5.2.1 Motivation

A systematic exploration of DL algorithms applied to WF attacks is provided in Rimmer et al. (2017). However, their experiments are still based on the assumption that each trace contains an entire single webpage. Using traditional machine learning algorithms, Wang and Goldberg (2016) classified network traces into either single-page (corresponding to one web page) or two-page traces. The two-page traces could further be split into zero-time (continuous traces), negative-time (overlapping traces), and positive-time (non-overlapping traces) traces. For zero-time and negative-time separated two-page traces, the accuracy to find the start of the second web page known as *split-point* was only 66% and 32% respectively. The authors didn't provide further analysis after this, but the low accuracy in finding split-point will certainly impact the final website prediction in a negative way. We aim to use DL methodology to reduce the distance between research/laboratory work and the real world. Thus, we assume collected traces may contain one or two webpages and also evaluate our methodology under conditions of traces of partial webpages and traces composed of webpages with background noise. We emphasize that the goal of this study is to improve the performance of WF attack in the real world instead of building a better DL model when compared to previous work Rimmer et al. (2017); Sirinam et al. (2018)

5.2.2 Features

The two types of features we used in the evaluation are listed as follows. Each of them can be represented in a time sequence, and we treat these two types of features as two different feature dimensions.

1. Packet-size. To increase the speed of convergence of gradient decent, we normalize the data first. We divide each packet by the MTU (Maximum Transmission Unit) which results in each packet size in the range of $[-1, 1]$. We keep only 2 decimal digits in the packet-size sequence.

2. Timestamp. We use inter-packets time interval instead of the absolute time, that is the difference between each packet time-stamp and the trace's start time.

5.2.3 Classification of Traces

We attempt to distinguish between single-page traces and two-page traces first, since approaches applied to these two types of traces are different in website detection.

1. One-page traces. This is the optimal situation when a victim visits one website at a time and the attacker captures the whole trace of a single webpage. In addition to evaluating traces with entire single webpage, we consider two more situations: 1) when an attacker fails to capture whole traces that results in missing the beginning (head) of trace or the end (tail) of traces; we classify this as a partial trace, and 2) when there exists some brief background noise in the network; for example, a process sending a keep-alive message while the user is visiting a webpage.

2. Two-page traces. This situation occurs when a victim visits one website after another. There are three different cases based on the time gap between the two pages: 1) negative time gap (overlapped traces): visiting the second website

while the first one is still downloading, 2) zero time gap (continuous trace): visit the second website right after the first one, 3) positive time gap: visit the second website after visiting the first one. In Wang and Goldberg (2016)'s work, after applying their time splitting method, they are able to split two-page trace into single trace in case 3 with an accuracy of 88%. We will focus on the first two cases in this study and aim to identify both pages under these situations.

One metric of DL in model training is that it can eliminate the need for feature engineering such as feature design and feature tuning. Rimmer et al. (2017) proposed to automate feature extraction through DL and the results showed that DL can automate the feature engineering process and effectively create WF classifiers. Thus, we use raw data as training input. For each trace, we keep the first N packets. For traces with fewer than N packets, we pad 0 at the end of these traces. A trace is composed of a sequence of "$timestamp$: $packetsize$" sorted by time. We will perform a CNN-based supervised binary classification on this problem. The algorithm takes traces as input and return a binary result 1 or 0 to indicate whether the trace corresponds to single-page or two-page.

5.2.4 One-page Trace

5.2.4.1 Scenario

Current deep learning techniques assumed single complete webpage traces. We argue that even though a victim visits one webpage at a time, an adversary may not be able to capture the whole webpage trace. For example, if a victim closed the webpage before it finished downloading, then the captured trace will miss the tail part. A trace may also lose the head part (beginning) if an attacker started to collect packets after the victim visited the webpage.

We thus have three scenarios: 1) one-page trace missing tail part, 2) one-page trace missing head part, and 3) whole one-page trace. Since 3) has been well evaluated

in previous research work Rimmer et al. (2017); Sirinam et al. (2018), we focus on scenarios 1) and 2), that is, partial traces. We won't explore the traces losing both head and tail in this study.

5.2.4.2 Head Detection Method

In closed-world settings, we performed a CNN-based classification training. The model takes the first N packets of traces as input and return the predicted website it belongs to. For traces with fewer than N packets, 0s are padded to the end of the trace. After hyperparameter tuning, we tested the model on 1) and 2).

As expected, the prediction accuracy drops in cases 1) and 2) compared to 3). We aim to distinguish between scenarios 1) and 2) first and apply different models on different cases. We apply a head detection first which is a binary classifier based on k-NN and adopt first k packets of a trace as features to recognize whether a trace has the head part or not. For traces classified into 1), we reuse the basic model that takes first N packets of traces as input. For traces classified into 2), we use the last N packets of traces. The training model for this case is also built on traces with last N packets instead of first N packets. For traces with length less than N, 0s will be padded to the head of the trace. We will outline approaches used in open-world settings in Section 5.2.5 for models trained on first and last N packets of traces. We will show the efficiency of this approach in section 5.3.

5.2.5 Two-page Trace

In section 5.2.3, we showed that a two-page trace consists of two webpage traces that are separated with positive, zero, and negative time gap. In this section, we aim to predict two pages directly from the traces. Figure 5.1 shows an example of a two-page trace with overlapping parts (negative-time separated). This trace was obtained when a victim visited website2 while website1 was still downloading. The overlapped

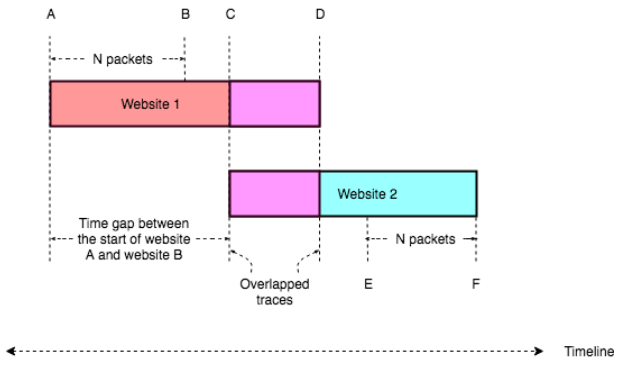

Figure 5.1: Two-page traces

traces from C to D in the figure contains the mix of packets from the tail of website1 and the head of website2. Instead of using the whole trace, we extract information from first N packets and last N packets. In an ideal situation, the first N packets of the trace from point A to point B come from website1 and the last N packets from point E to point F are from website2. In order to avoid including the overlapping part, N should be chosen as low as possible. However, in one-page traces, increasing N improves the model performance. The value of N thus needs to balance these two aspects.

The intuition of this algorithm is from Rimmer et al. (2017) where they showed that LSTM can classify traffic traces based solely on their first 150 Tor cells with 1000 instances of each website and achieve an accuracy of more than 90%. This indicates that deep learning method can predict websites based on a small fraction of the network trace with enough webpage instances. We expect that the tail of the trace also provides enough information for website detection. We note that two webpages visited simultaneously causing a nearly 100% overlap is beyond our consideration.

In a closed-world evaluation, we construct two classification DL models training on the first N and last N packets to predict the first and second website in traces respectively. For the open world, we use the following solutions to distinguish between monitored and unmonitored websites.

- Prediction Confidence Approach. We reuse the model trained in the closed-world settings and use an indicator of the prediction confidence. For an input trace, if the calculated confidence is higher than a certain threshold, we consider to trust the model and classify the trace as a monitored website. Otherwise, the trace is classified as an unmonitored website. By varying the threshold, we are able to balance the True Positive Rate and False Positive Rate. The two indicators we evaluate are Shannon entropy and Renyi entropy.

  1. Shannon Entropy:

  $$H(X) = -\sum_{i=1}^{N} p_i log_2 p_i \tag{5.1}$$

  where p_i is the probability of the test trace belonging to class i. The entropy achieves maximum value $log(n)$ if p_i comes from a uniform distribution that all outcomes are equally likely.

  2. Renyi Entropy:

  $$H_\alpha(X) = \frac{1}{1-\alpha} log(\sum_{i=1}^{N} p_i^\alpha) \tag{5.2}$$

  It has similar properties as the Shannon entropy: it is additive and has maximum as $log2(n)$ for $p_i = 1/n$ Maszczyk and Wlodzislaw (2008). The difference is that the additional parameter α controls its sensitivity to the probability distribution. Renyi entropy increasingly weighs more on events of highest probabilities as α approaches infinity and considers nearly all possible events as α approaches 0.

  We use $H(X)/log(n)$ as a threshold to limit the value between 0 and 1. The higher the entropy value, the less confidence the classifier gets. Hence, we classify the trace as a monitored website if the corresponding entropy is less than the threshold or as an unmonitored website otherwise.

- Binary Classification. This method is widely used in WF attack with classic

machine learning algorithm but haven't been evaluated in DL approach as far as we know. The training set is composed of $n_{mweb} \times n_{inst} + n_{unweb}$ where n_{mweb} is the number of monitored websites, n_{inst} is the number of instances of each monitored website and n_{unweb} is the number of unmonitored websites. We balance the amount of $n_{mweb} \times n_{inst}$ and n_{unweb} to be equal in the training set. The unmonitored websites are randomly chosen. For an input trace, the classifier will return a binary decision (0 or 1) to indicate whether it belongs to a monitored website or an unmonitored website.

All the models mentioned above are built on packet-size. We are wondering if adopting the sequence of timestamp has a positive effect on classifying. Hence, we add corresponding timestamp of each packet as another dimension's features to the training data besides the packet size sequence and evaluate the classifier performance again.

5.2.6 Noise

In this study, we evaluate how our DL model is robust to noise. We add noise packets to the original traffic every n seconds to simulate traces with noise. By varying n, we can see how much noise our model is able to deal with.

5.3 Simulation

The goal of the simulation is to verify the methodology proposed in Section 5.2. From the simulation, we are also able to compare performance among different solutions.

5.3.1 Dataset

In our simulation, we adopt the subset of dataset collected in Rimmer et al. (2017) for the experiments. We use the TCP layer packets directly. In the closed world, we use Alexa top 100 as monitored website domain and there are 93 out of 100 websites

qualified with at least 1280 (1024 instances for training and 256 instances for testing) valid instances each in the dataset. For the open world, besides the closed-world dataset, we randomly picked 119040 unmonitored websites with one instance each. The training data in the open world is formed with $93 \times 1024 = 95232$ monitored website traces with or without 95232 unmonitored website traces. The test data then consists of the rest in the dataset, which is 93×256 monitored website traces plus 23808 unmonitored website traces.

5.3.2 Deep Learning Model

5.3.2.1 Implementation

We use Python deep learning libraries Keras Chollet et al. (2015) as the frontend and Tensorflow Abadi et al. (2015) as backend in the implementation. The performance of various DL algorithms for WF was shown in Rimmer et al. (2017). Among CNN, LSTM and SDAE (Stacked Denoising AutoEncoder), CNN has the least time cost with the best accuracy in the closed-world evaluation and has closed TPR (80.11%) and FPR (10.53%) with SDAE (TPR:80.25%, FPR 9.11%) which performs best in the open world. Hence, we choose CNN as our main DL algorithm. Rimmer et al.Rimmer et al. (2017) provided us their source code to reproduce their experiments and results. We adopted their LSTM and CNN model to compare with our CNN model. We built the model for one-page trace first before dealing with other scenario (one-page partial trace, two-page trace, trace with noise). We followed the deep learning techniques in Rimmer et al. (2017); Sirinam et al. (2018); Oh et al. (2017b) to improve our model. Figure 5.2 shows the structure of our CNN model. The basic block consists of one convectional layer with activation function, followed by max pooling and dropout layer. In Figure 5.2, we mark the difference between our model and Rimmer et al. (2017)'s model in color. First, we normalize the data and add a dropout layer right after this to minimize overfitting. Instead of ReLU (Rectified

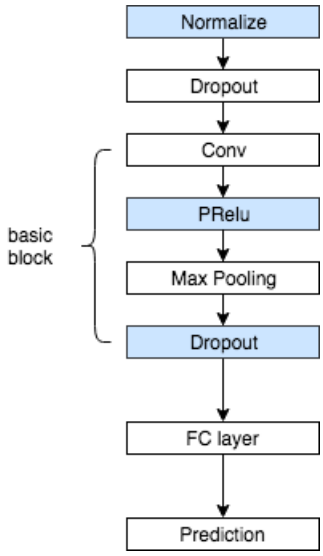

Figure 5.2: CNN model structure. The blue boxes are additions to our model when compared to previously proposed models.

Linear Unit), PReLU (Parametric ReLU) is chosen as the activation function in our model since PReLU doesn't map all negative values to zero like ReLU and performs better from our testing. We considered using doubled convolutional layer in a basic block which is inspired by Oh et al. (2017b); however the performance turned out similar but with higher time cost and more complicated network due to the doubled convolutional layer in each block.

5.3.2.2 Hyperparameter Tuning

Hyperparameter tuning is a basic process in machine learning classification. It is a process to improve the model performance as well as balancing the trade-off between the performance and overfitting. The amount of hyperparameter and required training data in deep learning are considerably greater than classic machine learning algorithms. Hence, it is difficult for us to conduct an exhaustive search due to the limitation of computational resources. Besides, our goal is to demonstrate the effectiveness of our method instead of building a perfect deep learning classifier.

Yosinski et al. (2014) pointed out the transferability property in CNN. They

Table 5.1: Hyperparameter Tuning for CNN.

Hyperparameter	Search Space	Value
Input Length	[1000...5000]	3000
Learning Rate	[0.001...0.05]	0.023
Batch Size	[64...256]	512
Training Epochs	[10...50]	45
Dropout rate	[0...0.5]	0.18
Activation	[ReLU, PReLU]	PReLU
Optimizer	[Adam, RMSProp]	RMSProp
Pooling Layers	[Average, Max]	Max
Filter	[16...128]	[32,256]
Kernel Size	[1...20]	15

showed that "the transferability of features decreases as the distance between the base task and target task increases, but that transferring features even from distant tasks can be better than using random features". Thus, instead of using random features first, we initialize our model with their parameters and get a rough domain of the search space.

We use Talos built in Keras automating the process of hyperatmeter tuning. We perform a two-step selection of hyperparameters for our model. First, we select top-n best parameters for one layer based on loss function from top to the bottom and use them as the initial parameters for the optimization in the next layer. When all layers are set, we select the best combination of hyperparameters. All models in this paper use this same tuning steps. The search space and final value of the best parameters for one single trace are listed in Table 5.1.

5.3.3 Classification of Traces

We performed a CNN-based binary classification to distinguish between one-page and two-page traces as depicted in Section 5.2.3. We picked $N = 3,000$ and used signed packet size as features. The model trained on one-page trace is labeled as 0 and the

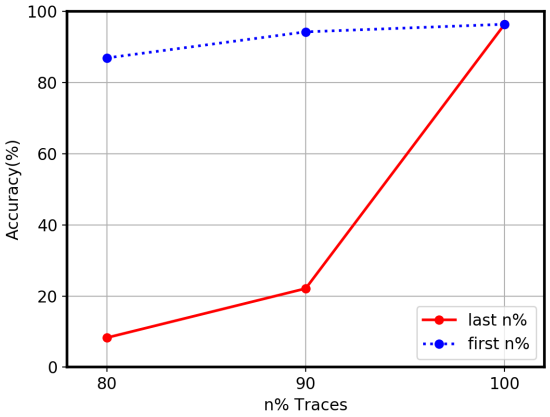

Figure 5.3: Accuracy for partial traces using k-NN algorithm in the closed-world evaluation.

two-page trace is labeled as 1. Since we did not limit the domain of the one-page trace to monitored website domain, we use the same model for the closed-world and open-world evaluation in this step. The model achieves an accuracy of 85%.

Wang Wang and Goldberg (2016) also developed an approach based on k-NN to solve this problem. We strictly followed their published code, and evaluated on our dataset to make a fair comparison. Their method achieves an accuracy of 68.60% from this evaluation.

5.3.4 Closed-world Evaluation

We evaluate the performance of our model and Rimmer et al. (2017)'s model on single traces first. We trained and tested on the same dataset (closed-world data in Section 5.3.1) for both models. The results show that our model achieves an accuracy of 97% compared to 95% for Rimmer et al. (2017). Thus, we use our CNN model evaluation in this section.

5.3.4.1 One-page Trace

We removed the head (beginning) or the tail (end) of a trace to simulate partial traces, as mentioned in Section 5.2.4, to create the test data. We also varied the size of the partial traces by changing the percentage of missing packets from the traces. Figure 5.3 shows the accuracy when testing on traces with 10% or 20% missing packets on the head or tail of the trace. When $n = 100\%$, it indicates that an attacker captures the entire trace. Last $n\%$ traces in the figure corresponds to capturing the last $n\%$ packets (or having $(100 - n\%)$ packets missing at the beginning). The figure demonstrates how the classification accuracy decreases as n decreases. When missing the last 20% of packets, the classification accuracy decreases to 86.93% from 97%. When missing the first 20% of packets, the accuracy decreases to 8.28%. It has already been shown that the head part (beginning) of a trace carries more information. We introduced our method to deal with this situation in Section 5.2.4, and we now present the results of our approach. The k-NN model we used to detect whether the trace missing head (label 0) or tail (label 1) takes first 10(after tuning from $10, 20, 30, 50$) packets' sizes of a trace as features. It achieves an accuracy of 94.95% in classifying. For traces missing head, we trained another model the same as the previous CNN model except based on last 3000 packets.

Figure 5.4 shows the accuracy of our method compared to the original model. Although the accuracy is lower when $n = 100$, the accuracy is respectively 86.35% and 71.55% when 10% and 20% of the packets are missing at the beginning. This is compared to 22.12% and 8.28% when using the original model. This shows that the last $n\%$ packets also carry information to identify the website and the poor performance previously is due to the model used.

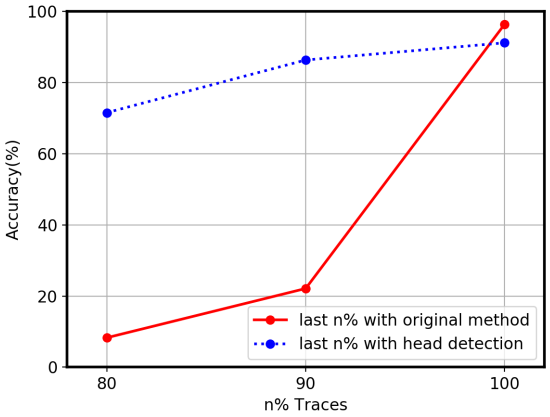

Figure 5.4: Head detection method VS original method for last $n\%$ traces in the closed-world evaluation.

5.3.4.2 Two-page Traces

We have described our method to predict each webpage in the two-page traces in Section 5.2.5. We want N to be as large as 3000 where the model performs best, however it will have a higher probability to include packets from overlapped traces. After balancing the model performance and avoiding overlapped traces, we set N to 200. Rimmer et al. (2017) showed that LSTM has a great performance on one-page trace classification based solely on 150 Tor cells, so we will take LSTM as a comparison in two-page traces evaluation.

Test data. To form the test data two-page traces, we merge two single traces from the test dataset (see Section 5.3.1) and sort by time, as shown in Figure 5.1. We merge-insert $website2$ to $website1$ from point c. After obtaining the timestamp $time_c$ of the packet from $website1$ at point c and the start timestamp $time_{start\_w2}$ of $website2$, we reset each packet's timestamp in $website2$ as $timestamp - time_{start\_w2} + time_c$, and then sort packets in $website1$ and $website2$ based on the new timestamp.

Results. We reproduce Rimmer et al. (2017)'s LSTM model and select the length of trace to 200 to make a fair comparison with CNN. By increasing the percentage p of overlapped traces (the length of the overlapped traces is equal to $p * l_{first}$, where

l_{first} is the length of the first website), we are able to test if our model is robust to large amount of overlapped packets. When $p = 1$, $website1$ and $website2$ are fully overlapped, we won't consider this case in our evaluation (see section 5.2.5). When $p = 0$, it's corresponding to a continuous visit that $website2$ is visited right after $website1$ finishes loading. The results are depicted in figure 5.5 for LSTM and CNN model applied to predict first and second website. As expected, prediction on the first website is more accurate than the second one. However, the model performance on the second website is still acceptable with 70% compared to Wang and Goldberg (2016)'s work (34% accuracy to find the split point between $website1$ and $website2$). We varied p from 0 to 0.8 and found that the model performance is stable and isn't impacted by the amount of overlapped traces. It shows that N is small enough to avoid overlapped traces while maintaining an efficient model. The results also demonstrate that CNN performs better than LSTM on both website prediction considering prediction accuracy. We attempt to improve the performance of the model on the second website by adding the timestamp as another dimension of the feature. Instead of using the raw timestamp, we calculate the relative value which is represented by $(timestamp - t_{start})$, where t_{start} is the start timestamp of the trace. We successfully improve the accuracy for the second website from 72.35% to 81.61% when testing on traces with 50% overlapping through this new model.

5.3.4.3 Noise

Our goal is to identify the number of noise packets per second which is necessary for the accuracy to decrease. The attacker in this case captured traces with random noise in it without knowledge of the noise.

- **Test data.** Assume that the start timestamp of a trace is t_{start} and the end is t_{end}. To insert random noise m packets/second (p/s), we draw m numbers $m_1, m_2..., m_j, ...m_m$ from $[0, 1]$ uniformly first. Then use $t_{start} + i + m_j$ where

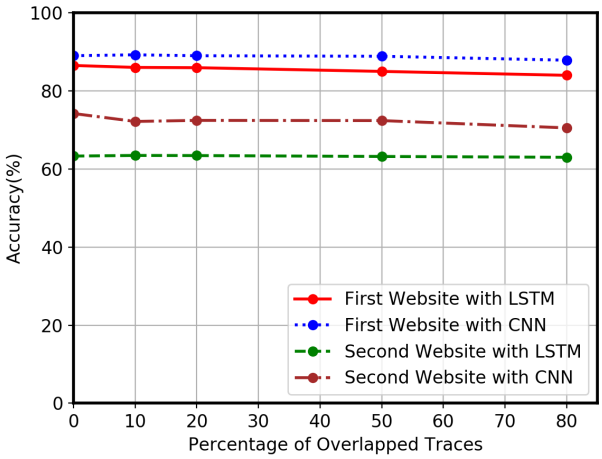

Figure 5.5: Closed-world evaluation on two-page traces (zero time and negative time separated).

Noise	Accuracy	Decrease in accuracy	Overhead
10p/s	25.71%	71.29%	56.4%
5p/s	54.97%	42.03%	28.2%
2p/s	80.81%	16.19%	14.5%
1p/s	89.89%	7.11%	5.2%

Table 5.2: Decrease in accuracy when random noise is added.

i is from $0, 1, ..., (t_{end} - t_{start})$ as the noise packet's timestamp. For each i, j is from $1, 2, ..., m$ Then we merged the generated noise packets with the original trace to get the test data. The noise is thus randomly spread out in the whole trace. It is easy to understand that with the same number of noise packets, the more their timestamp compact together, the less they impact the predicting performance. They only affect a certain part of the trace and information from unaffected parts are still valuable. We attempt to insert the noise constantly crossing the whole trace in order to get the conservative prediction accuracy.

- **Results.** We listed the noise frequency, the predicting accuracy, the decrease in accuracy compared to traces without noise, and the noise overhead in Table 5.2. The overhead is calculated by the n_{noise}/n_{packet}, where n_{noise} is the number of

noise packets and n_{packet} is the number of packets from original traces. The accuracy drops fast at first and starts to fall slower from $5p/s$. Like in traditional machine learning algorithm, noise also has a strong effect in disturbing the DL model.

5.3.5 Open-world Evaluation

5.3.5.1 One-page Trace

The open-world evaluation of state-of-art work Rimmer et al. (2017); Oh et al. (2017b) uses the model built on monitored websites with a threshold which could measure the prediction confidence. The main theory here is that if the model has a strong confidence in prediction, then the input trace belongs to the monitored website, otherwise it belongs to the unmonitored website. However, from our evaluation, we found that binary classification (mentioned in Section 5.1 with $N = 3,000$) has a better performance on the whole traces which is a guarantee for partial traces testing, with the Area Under Curve (AUC) of 97.5% compared to 91% of method with prediction confidence. Hence, we pick binary classification with CNN (CNN has many advantages than LSTM when N is not limited) as the open-world evaluation method for this scenario. We also apply the head detection first and trained two models as in the closed-world evaluation.

Since the amount of monitored and unmonitored instances are controlled to be same in the test dataset, the accuracy of the model is valuable and can be calculated as $(1 + TPR - FPR)/2$. We select the threshold as 0.5 from our validation and draw Figure 5.6 to present the performance of the model on partial traces. The results show that the model consistently performs best on first n% packets at around 94% TPR with 4% FPR. For the last n% where traces miss head part, the accuracy tends to slightly decrease with the reduction of n. However, the results are still promising compared to the model without head detection applied where the accuracy falls to

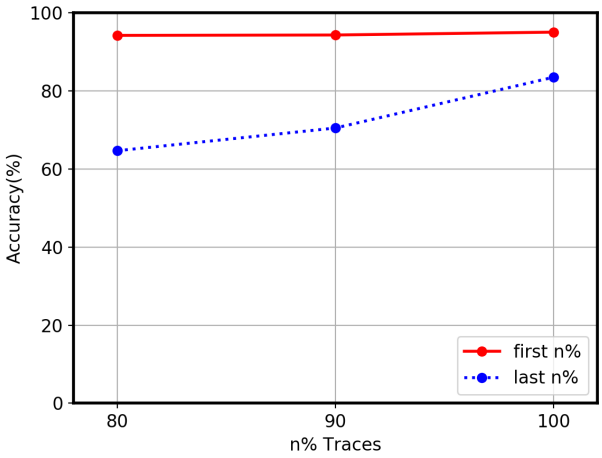

Figure 5.6: Open-world evaluation on partial one-page traces with binary classification.

20% for last 90% packets.

5.3.5.2 Two-page Traces

We have outlined the method used in open-world evaluation for two-page traces in Section 5.2.5 and will discuss the performance of corresponding methods in this section. The test data is formed following the pattern in two-page traces in closed-world evaluation (Section 5.3.4.2). The number of traces from the monitored and unmonitored websites are the same.

CNN and LSTM. Similar to the closed-world evaluation, we set N to 200 and pad 0s to traces with length less than 200. We use the prediction confidence with Shannon entropy to test the performance of CNN in this scenario and LSTM as a comparison. The accuracy is defined the same as in Section 5.3.5.1 which is based on TPR and FPR. It makes it simpler to demonstrate the results across different percentage of overlapped traces together in this way. As shown in Figure 5.7, LSTM and CNN perform with similar efficiency on the first website with an accuracy around 78%. However, CNN performs significantly better than LSTM on the second website on whichever percentage of overlapped traces. Figure 5.8 and Figure 5.9 shows the

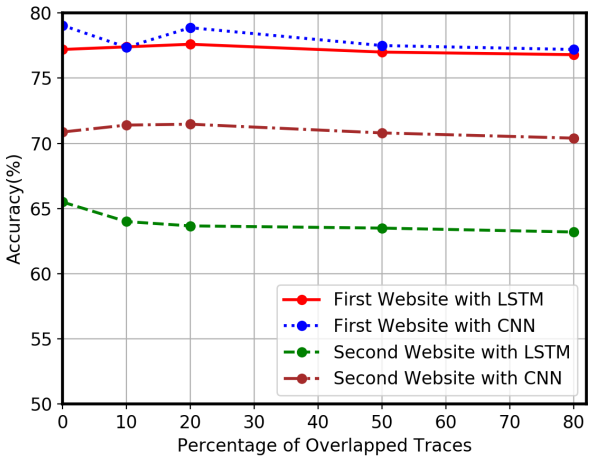

Figure 5.7: Open-world evaluation on first and second website in two-page traces

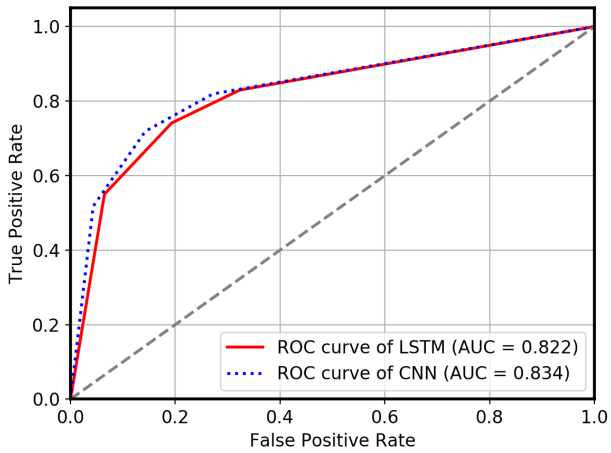

Figure 5.8: ROC curve on predicting first website with 10% overlapping traces.

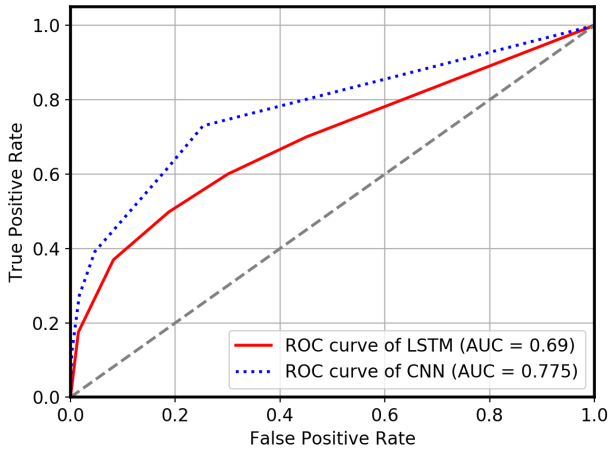

Figure 5.9: ROC curve on predicting second website with 10% overlapping traces.

ROC curves when the first and second website have 10% overlapping traces and show the same results.

Binary Classification and Prediction Confidence with Shannon Entropy. Binary classification is another widely used method in WF open-world evaluation. However, to the best of our knowledge, it has not been applied in DL algorithms. We use the previously-built CNN as the basic model to launch this classification. We assess the prediction accuracy for first and second website in Figure 5.10 and Figure 5.11 and compare the accuracy with the prediction confidence, using the Shannon entropy, method. The thresholds used in binary classification and Shannon entropy for the first website are $0.5, 0.07$ and $0.5, 0.28$ for the second website. We observe that the binary classification outperforms the prediction confidence with Shannon entropy significantly when predicting the first website. The accuracy is increased from 80% to 95% and remain the same regardless of the different amount of overlapped traces. It also slightly improves the performance on the second website. Figure 5.12 plots the ROC curve of these two methods applied on the first website with 10% overlapped traces. The AUC for the binary classification is 97.2%.

Improvement on the Second Website. When considering TPR and FPR, the prediction on the first website in two-page traces has met our expectation. We

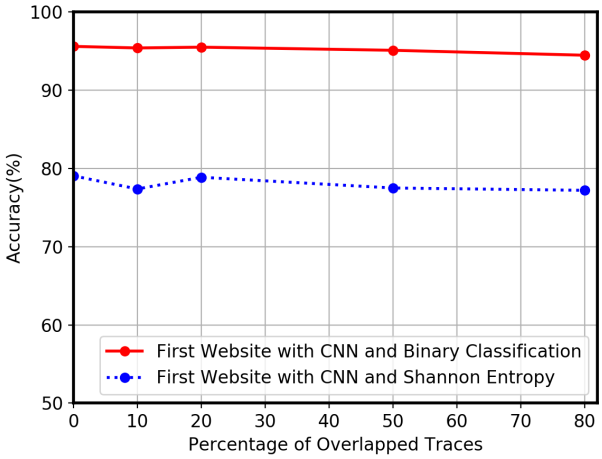

Figure 5.10: Open-world evaluation on first website with binary classification and Shannon entropy.

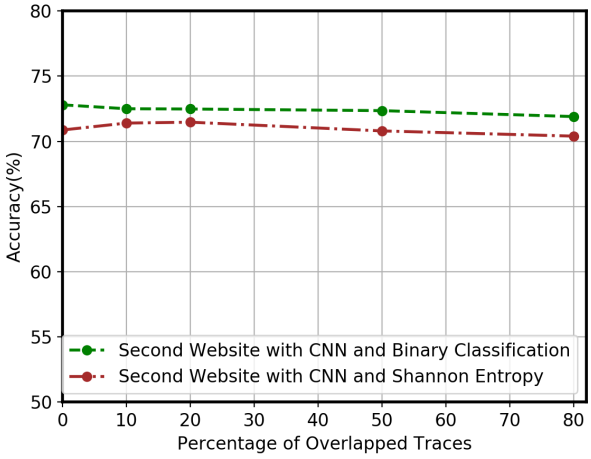

Figure 5.11: Open-world evaluation on second website with binary classification and Shannon entropy.

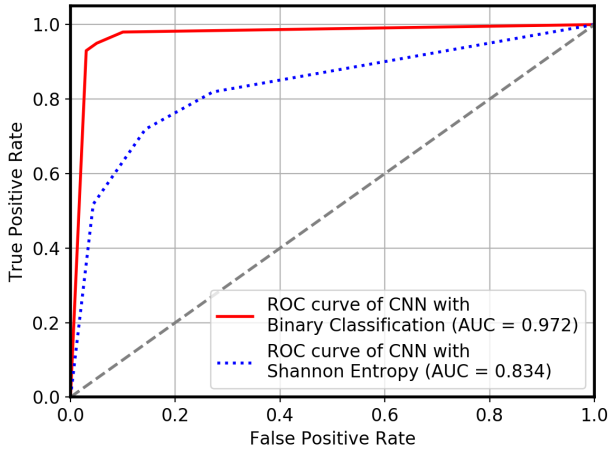

Figure 5.12: ROC curve of binary classification and Shannon Entropy in the open-world evaluation on the first website with 10% overlapping traces.

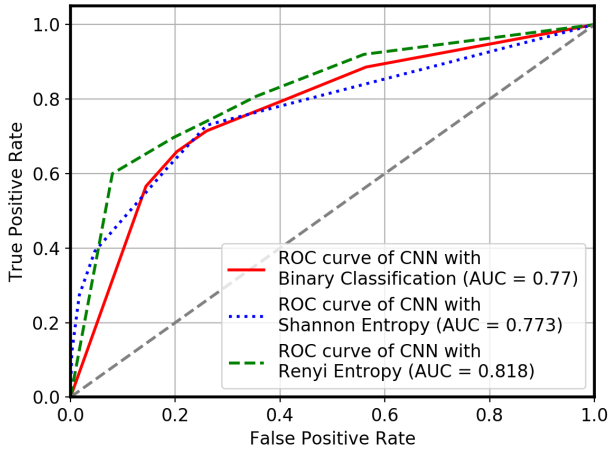

Figure 5.13: ROC curve of binary classification, Shannon Entropy and Renyi Entropy in the open-world evaluation on the second website with 10% overlapping traces.

wonder if it's possible to improve the accuracy on the second website. We didn't find any improvement by adding the timestamp to the model in this scenario. We tried to make progress from the prediction confidence method with only training on monitored websites. We employed a new indicator called *Renyi Entropy*. We've introduced this in Section 5.2.5 and mentioned that the similarities and differences between this entropy with Shannon entropy are that it has similar properties but has an additional parameter α to control weights on events with different probabilities. After tuning, we set $\alpha = 0.5$ and run the experiment with *Renyi Entropy* on the second website prediction. We present the comparison of the ROC curve between the three discussed approaches in Figure 5.13 under the condition of 10% overlapped traces. From the results, we can see that this approach is better than both binary classification and Shannon entropy. To show whether the results are significant, we employed McNemar's test. We compared Renyi Entropy (test 1) and Shannon Entropy (test 2). If the trace is classified correctly, we call it positive, otherwise negative. Assuming b is the number of test cases that are positive in test 1 and negative in test 2, and c is the opposite. The test statistic is calculated by $(b-c)^2/(b+c)$ and has a chi-squared distribution with 1 degree of freedom. P-value is $0.0139 < 0.05$, which shows the significance of our improvement.

Evaluation on Time Gap. We draw figure 5.14 to illustrate the average time gap on two-page traces with $0-80\%$ overlapping. In fact, in the real world experiment, it is very difficult to control the percentage of overlapped traces when collecting two-page traces. A more straightforward and meaningful way is to set the visiting time gap between the first and second website. Besides, the length of certain percentage of overlapped traces is decided by the length of the original trace, if the original trace is longer than $2,000$ packets, and we only take the first 200 packets as training features, then the accuracy will stay the same until 90% overlapped traces. To make it consistent with our real world experiment and obtain a clear insight about how

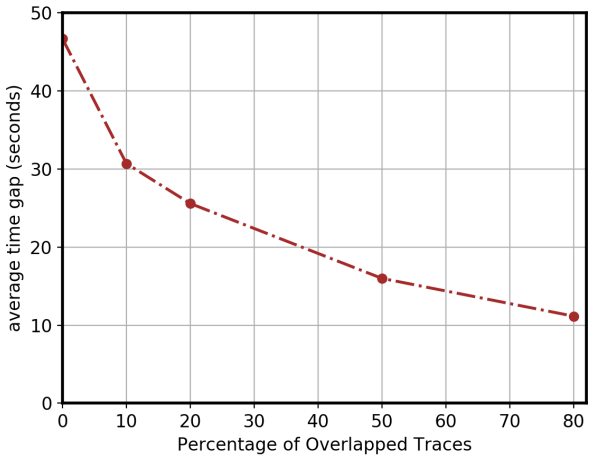

Figure 5.14: Average time gap between the first and second website on different percentage of overlapped traces.

the accuracy is impacted by the time gap, we evaluate our model with various time gaps between two websites and outline the results for the first and second websites in Figure 5.15 and Figure 5.16.

We observe that binary classification still performs better than Shannon entropy for the first website. However, the prediction confidence with Shannon entropy is relatively stable with the change of time gap. Since the binary classification on the first website has achieved an accuracy more than 90%, we won't evaluate adding timestamp as a second dimension and Renyi entropy here. For the second website, we found that the efficiency of binary classification falls significantly compared to the evaluation based on the percentage of overlapped traces. However, when adding the timestamp for each packet besides the packet size to the binary classification model, it outperforms all the other three methods. Also, for the prediction confidence approach, the indicator of Renyi entropy is along with the previous performance that exceeds Shannon entropy. We add timestamp as a new dimension of features to the prediction confidence approach as well, but it doesn't improve the predicting accuracy.

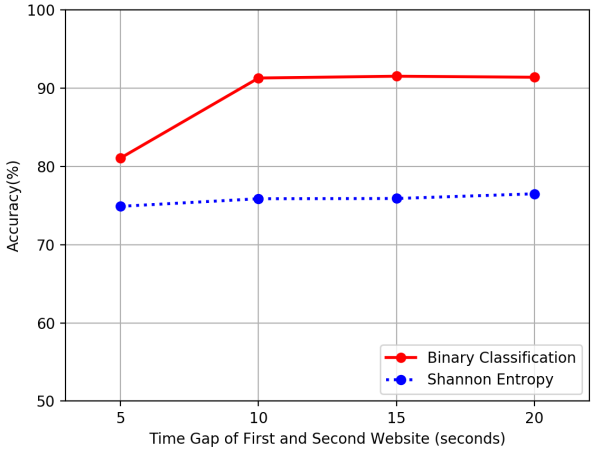

Figure 5.15: Open-world evaluation on first website in two-page traces with time gaps.

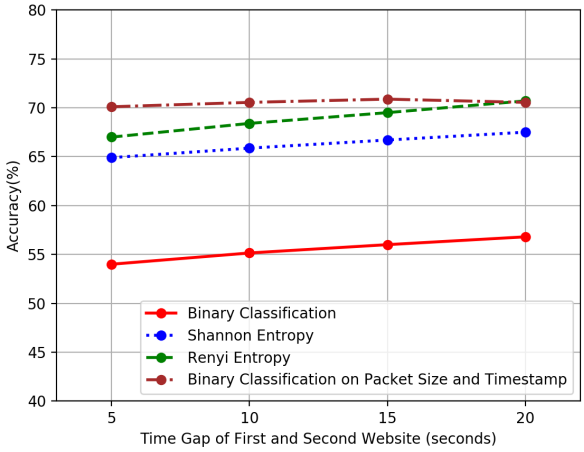

Figure 5.16: Open-world evaluation on second website in two-page traces with time gaps.

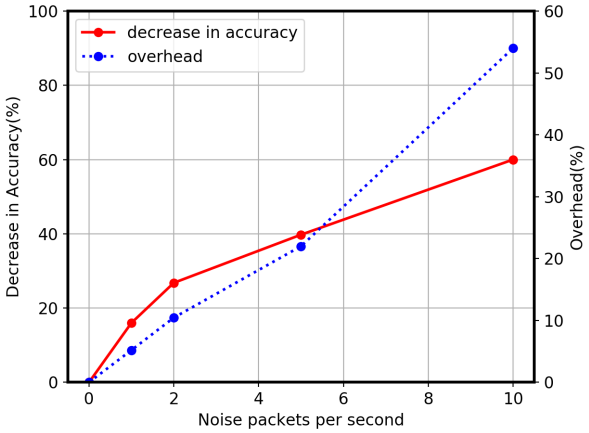

Figure 5.17: Open-world evaluation of accuracy and overhead on traces with noise.

5.3.5.3 Noise

We evaluate our trained model described in Section 5.3.5.1 with noise traces. It is a binary classification model building on CNN using packet size as input. Figure 5.17 shows how the accuracy is affected by the increase of noise packets per second(p/s) and the corresponding overhead. The overhead grows linearly while the decrease in accuracy comes quickly at first and becomes relatively slow after $2p/s$. When noise is increased to $10p/s$, the accuracy of $95\% - 60\% = 35\%$ becomes less than 50% which is an accuracy gained from a random decision in distinguishing monitored and unmonitored website. It indicates that this amount of noise has the ability to fully disturb the model.

5.4 Real World Experiment

In this section, we perform the two-page traces evaluation in the real world experiment.

5.4.1 Data Collection

We design our website-traffic-traces-crawler based on the tor-browser-crawler Juarez et al. (2014). By adding new features of concurrent visits, our crawler can visit 2 or more websites at the same time or with a certain time gap between visits to different websites. The crawler is built on Tor browser (version 7.0.6) and Tor process (version 0.3.1.7). We combine the crawler with the browser automation tool Selenium (version 3.6) to automatically open, load and close visits to websites. We allow 150 seconds to finish loading a webpage in each visit. For recording network traffic traces, we use a network traffic dump tool, "dumpcap". To make the crawler easier to deploy, we create a docker image with Docker 18.09.1. Then the docker image is dispatched and loaded on our 24 virtual machines. Each of our virtual machines has 6 CPUs 2.50 GHz with 6 GB of RAM. The whole data collection process runs from Jan. 25 2019 to Feb. 14 2019.

Our process of collecting webpages traces is run in batches. For monitored websites for training data, there are 100 batches. Each batch has 10 visits to every website in our monitored 118 websites list. The monitored list is filtered from Alexa top 200 websites. We eliminate the same website with different domains like google.com, goolge.hk, etc. In total, we collected $118 * 1000$ ($1,000$ visits to each website) monitored training WF traces where each trace only contains traffic data of one website. Also for unmonitored websites training data, it has 1 batch and 1 visit in a batch. That is, we collected one trace for each of $118,000$ websites that we randomly picked from Alexa website list. We randomly split half of Alexa top $400,000$ websites (excluding top 200 websites) to be list A and another half to be list B. Therefore, both lists A and B contain about $200,000$ websites each. List B is used for unmonitored test data collection.

When we collect monitored test dataset, we visited one website first then visited the second website after 10 or 20 seconds. The second website is randomly chosen

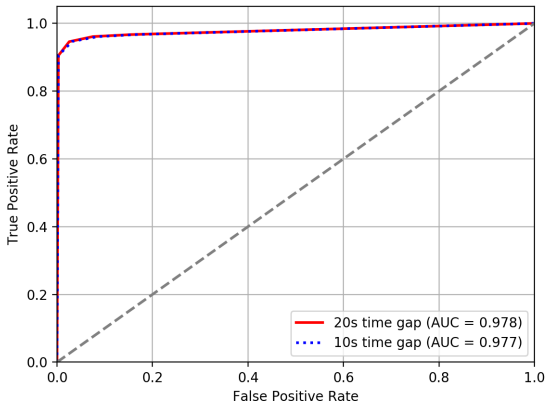

Figure 5.18: ROC curve of binary classification on first website with 10s and 20s time gap.

from the monitored 118 websites list. We repeated this process for each website in the list for 150 times. So there are $118 * 150$ test traces for 10 seconds gap and $118 * 150$ test traces for 20 seconds gap. Similar for collecting unmonitored test dataset, we put a time gap of 10 seconds and 20 seconds between the two visits. The second website here is randomly chosen from the Alexa top $17,700$ websites. These $17,700$ websites are randomly chosen from list B. For each of the $17,700$ websites picked from list B, we only did the process once. In total, we collected $17,700$ traces for 10s gap and $17,700$ traces for 20s gap.

5.4.2 Results analysis

In this section, we will present the results on predicting the first and second website from collected two-page traces. We will launch the evaluation in the open-world settings which is closer to the real world attack where there may be any possible webpages. We use our model built in Section 5.3.2 and follow the method proposed in Section 5.2.5. We pick the approaches with the best performance from the simulation. Figure 5.18 shows the ROC curve of the prediction on the first website in two-page traces using binary classification. The model achieves the best accuracy at 95%, and

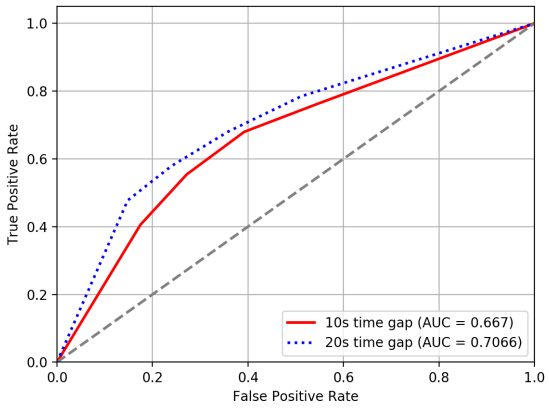

Figure 5.19: ROC curve of binary classification with packet size and timestamp on second website with 10s and 20s time gap.

there is no big difference between different time gap $10s$ and $20s$. The results are consistent with our simulation (Figure 5.12 and Figure 5.10). Figure 5.19 shows the ROC curve for WF attacks on the second website based on CNN binary classification with both packet size and timestamp features. As we can see in the figure, the results of $20s$ time gap between the start of the first and second websites tends to be slightly better for the 10 seconds gap considering both the TPR and FPR. The accuracy is similar to the simulation thus validating the efficiency of our proposed approach. The prediction accuracy for the second can't be as good as the first website for two reasons. First, the information extracted from the tail of the trace is limited compared to the beginning of the trace. The model based on first N packets in classification always outperforms the last N packets (see Figure 5.6). Second, the second website might end before the first website finished loading. In this case, even though we found the perfect split point between the first and second website, there's still nothing that can be done on the second website prediction since the second website is 100% mixed with another website traffic (shown in Cui et al. (2018)). However, DL approach shows potential in multi-page prediction with its strong ability in feature extraction based on only a small part of the whole trace.

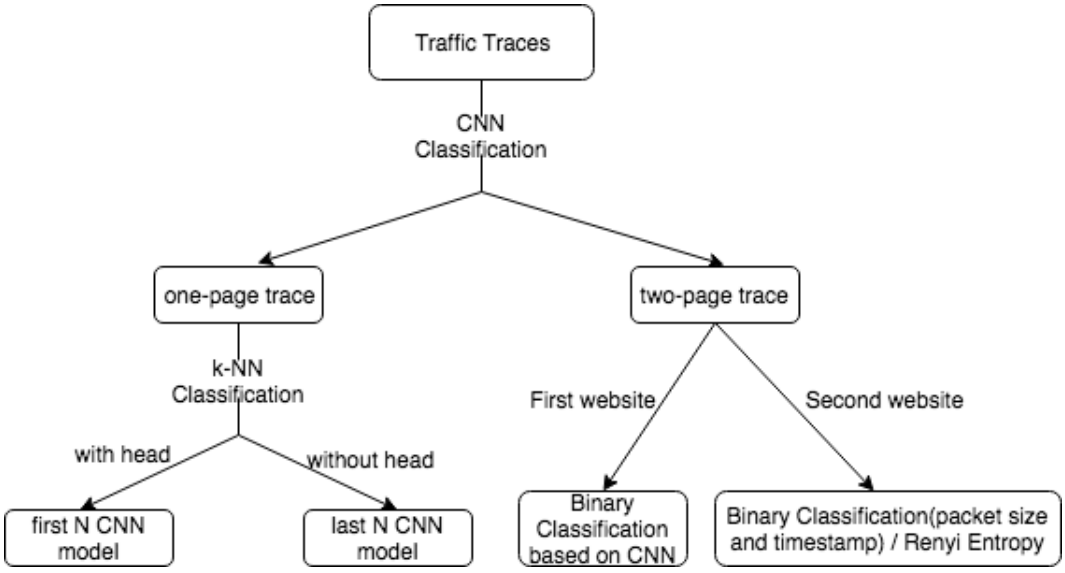

Figure 5.20: Summary of best algorithms in one-page and two-page traces prediction.

5.5 Summary

Our goal in this paper is to expand WF attack to a more realistic environment with the help of DL algorithm. We investigate the settings when the captured trace is a part of the entire traces, containing two pages or with noise in both closed-world and open-world. Figure 5.20 shows each step of one-page and two-page traces prediction with best algorithms. We use CNN classification first to distinguish between one-page and two-page traces. For one-page trace, we make decision of whether the trace contains the head part from the results of k-NN binary classification and apply corresponding CNN models to traces with and without head. For two-page traces, CNN binary classification with packet size works best on the first website, and prediction confidence approach with Renyi entropy has the similar performance as CNN binary classification with packet size and timestamp on the second website. In conclusion, DL algorithm has a great performance in partial page, multi-page prediction with its strong ability in feature extraction based on only a small part of the whole trace.

CHAPTER VI

CONCLUSION AND FUTURE WORK

In this work, we explore the mitigation and assumptions in website fingerprinting attacks. We showed the efficiency of our proposed defense algorithm. After that we address the impracticalities of website fingerprinting attacks and propose solutions to several limitations. Then we expand WF attack to a more realistic environment with the help of DL algorithm. We investigate the settings when the captured trace is a part of the entire traces, containing two pages or with noise in both closed-world and open-world.

- **Mitigation.** We showed that our proposed cover traffic (noise generation) algorithm mitigates website fingerprinting attacks as effectively as current existing schemes. However, the bandwidth overhead is only 20% for simulation and 10% for real-world experiments, much lower than existing schemes. The latency overhead is also 0%. Our algorithm can also be configured to utilize different amounts of bandwidth.

- **Splitting algorithm.** We propose a "splitting" algorithm to identify two continuous network traces with an accuracy of 80% in finding the split point of the two traces.

- **Partial trace.** We evaluate the DL model on partial traces. The accuracy drops less than 3% when missing 10% packets at the end of the trace. We improve the performance on traces missing the head part by adding the head detection. With 10% of packets missing in the beginning of the trace, the

accuracy is increased from 22.12% to 86.35%.

- **Two-page trace in general cases.** We developed the method to predict both websites in a two-page traces and verified its effectiveness in the real world experiment. The methods with the best performance from the simulation is binary classification based on CNN with packet size only for the first website prediction. With $10s$ gap between the first and second website, it achieves an accuracy of 95%in the open world evaluation with real world dataset. For the second website, the best approaches are 1) prediction confidence with Renyi-entropy and 2)binary classification with packet size and timestamp. We found that timestamp is an important feature for the second website prediction and the accuracy is improved 14% by adding the sequence of packet timestamp to the training model.

- **Noise** With the increase number of noise packet added per second, the performance of the model drops gradually and fails in classification when the number reaches to 10.

Two limitations of this work are

1. We limit the multi-page visiting case to two-page or we can only detect the first and last visited websites in a trace with more than two websites according to our methodology. Traces with three (or more) overlapped pages case are even more complicated and it's very difficult to extract valuable information from the middle webpage since it has a higher probability to mix more with other pages compared to the first and last webpage.

2. We evaluate the impact of noise in WF with DL, however didn't find an effective way to perform prediction, that is, eliminate the noise.

In multi-page identification, another approach is to find the split point of each webpage and analyse each one independently. However, it still depends on the fraction

of overlapped traces. One solution is cutting the single traces into several sections and analyse each one separately. If non-overlapped sections are more than the overlapped ones, there's still chance to make predictions. For future work, we would like to investigate more about the multi-page and more realistic situation such as noise elimination.

REFERNECES

(2017). Panopticlick. https://panopticlick.eff.org/.

(2019). Weka. `http://www.cs.waikato.ac.nz/ml/weka/`.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bissias, G. D., Liberatore, M., Jensen, D., and Levine, B. N. (2006). Privacy vulnerabilities in encrypted http streams. In *Proceedings of the 5th International Conference on Privacy Enhancing Technologies*, PET'05, pages 1–11, Berlin, Heidelberg. Springer-Verlag.

Blunsom, P. (2004). Hidden markov models. Lecture Notes in Computer Science, University of Melbourne.

Cai, X., Nithyanand, R., and Johnson, R. (2014a). Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 121–130, New York, NY, USA. ACM.

Cai, X., Nithyanand, R., Wang, T., Johnson, R., and Goldberg, I. (2014b). A systematic approach to developing and evaluating website fingerprinting defenses. In

Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, pages 227–238, New York, NY, USA. ACM.

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

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Cui, W., Chen, T., Fields, C., Chen, J., Sierra, A., and Chan-Tin, E. (2019). Revisiting assumptions for website fingerprinting attacks. In *ASIACCS '19: Proceedings of the 2019 on Asia Conference on Computer and Communications Security*, New Zeland. ACM.

Cui, W., Yu, J., Gong, Y., and Chan-Tin, E. (2018). Realistic cover traffic to mitigate website fingerprinting attacks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS) Workshop*, pages 1579–1584.

Diaz, C. and Preneel, B. (2004). Taxonomy of mixes and dummy traffic. In *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*.

Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*.

Dyer, K. P., Coull, S. E., Ristenpart, T., and Shrimpton, T. (2012). Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*.

Eckersley, P. (2010). How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10.

Fifield, D., Hardison, N., Ellithorpe, J., Stark, E., Boneh, D., Dingledine, R., and Porras, P. (2012). Evading censorship with browser-based proxies. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, PETS'12, pages 239–258, Berlin, Heidelberg. Springer-Verlag.

Geddes, J., Schuchard, M., and Hopper, N. (2013). Cover your acks: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 361–372, New York, NY, USA. ACM.

Gong, X., Borisov, N., Kiyavash, N., and Schear, N. (2012). Website detection using remote traffic analysis. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, PETS'12, pages 58–78, Berlin, Heidelberg. Springer-Verlag.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

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

Hintz, A. (2003). Fingerprinting websites using traffic analysis. In *Proceedings of the 2Nd International Conference on Privacy Enhancing Technologies*, PET'02, pages 171–178, Berlin, Heidelberg. Springer-Verlag.

Holowczak, J. and Houmansadr, A. (2015). Cachebrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 70–83, New York, NY, USA. ACM.

Houmansadr, A., Brubaker, C., and Shmatikov, V. (2013a). The parrot is dead: Observing unobservable network communications. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 65–79, Washington, DC, USA. IEEE Computer Society.

Houmansadr, A., Riedl, T. J., Borisov, N., and Singer, A. C. (2013b). I want my voice to be heard: Ip over voice-over-ip for unobservable censorship circumvention. In *NDSS*.

Howe, D. and Nissenbaum, H. (2008). Trackmenot: Resisting surveillance in web search. *On the Identity Trail: Privacy, Anonymity and Identify in a Networked Society*.

Juarez, M., Afroz, S., Acar, G., Diaz, C., and Greenstadt, R. (2014). A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 263–274, New York, NY, USA. ACM.

Juarez, M., Imani, M., Perry, M., Diaz, C., and Wright, M. (2016). Toward an efficient website fingerprinting defense. In *ESORICS*.

Le Blond, S., Choffnes, D., Zhou, W., Druschel, P., Ballani, H., and Francis, P. (2013). Towards efficient traffic-analysis resistant anonymity networks. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*.

Liberatore, M. and Levine, B. N. (2006). Inferring the source of encrypted http

connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 255–263, New York, NY, USA. ACM.

Lu, L., Chang, E.-C., and Chan, M. C. (2010). *Website Fingerprinting and Identification Using Ordered Feature Sequences*, pages 199–214. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mallesh, N. and Wright, M. (2007). Countering statistical disclosure with receiver-bound cover traffic. In Biskup, J. and Lopez, J., editors, *Proceedings of 12th European Symposium On Research In Computer Security (ESORICS 2007)*, volume 4734 of *Lecture Notes in Computer Science*, pages 547–562. Springer.

Maszczyk, T. and Wlodzislaw, D. (2008). Comparison of shannon, renyi and tsallis entropy used in decision trees. volume 5097, pages 643–651.

Miller, B., Huang, L., Joseph, A. D., and Tygar, J. D. (2014). *I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis*, pages 143–163. Springer International Publishing, Cham.

Mittal, P., Khurshid, A., Juen, J., Caesar, M., and Borisov, N. (2011). Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS 2011)*.

Moghaddam, H. M., Li, B., Derakhshani, M., and Goldberg, I. (2012). Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*.

Nithyanand, R., Cai, X., and Johnson, R. (2014). Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 131–134, New York, NY, USA. ACM.

Oh, S. E., Li, S., and Hopper, N. (2017a). Fingerprinting keywords in search queries over tor. *PoPETs*, 2017.

Oh, S. E., Sunkam, S., and Hopper, N. (2017b). p-fp: Extraction, classification, and prediction of website fingerprints with deep learning.

Panchenko, A., Lanze, F., Zinnen, A., Henze, M., Pennekamp, J., Wehrle, K., and Engel, T. (2016). Website fingerprinting at internet scale. In *Proceedings of the 23rd Internet Society (ISOC) Network and Distributed System Security Symposium (NDSS 2016)*.

Panchenko, A., Niessen, L., Zinnen, A., and Engel, T. (2011). Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, pages 103–114, New York, NY, USA. ACM.

Peddinti, S. and Saxena, N. (2010). On the privacy of web search based on query obfuscation: A case study of trackmenot. In Atallah, M. and Hopper, N., editors, *Privacy Enhancing Technologies*, volume 6205 of *Lecture Notes in Computer Science*, pages 19–37. Springer Berlin Heidelberg.

Perry, M. (2011). Experimental defense for website traffic fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting.

Portal, T. M. (2017). https://metrics.torproject.org/.

Rimmer, V., Preuveneers, D., Juárez, M., van Goethem, T., and Joosen, W. (2017). Automated feature extraction for website fingerprinting through deep learning. *CoRR*, abs/1708.06376.

Simon Oya, C. T. and Pérez-González, F. (2014). Do dummies pay off? limits of dummy traffic protection in anonymous communications. In *Proceedings of the 14th Privacy Enhancing Technologies Symposium (PETS 2014)*.

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

Song, W. (2015). End-to-end deep neural network for automatic speech recognition.

Spreitzer, R., Griesmayr, S., Korak, T., and Mangard, S. (2016). Exploiting data-usage statistics for website fingerprinting attacks on android. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, pages 49–60, New York, NY, USA. ACM.

Sun, Q., Simon, D. R., Wang, Y.-M., Russell, W., Padmanabhan, V. N., and Qiu, L. (2002). Statistical identification of encrypted web browsing traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 19–, Washington, DC, USA. IEEE Computer Society.

Tor (2017). https://www.torproject.org/.

Tschantz, M. C., Afroz, S., Anonymous, and Paxson, V. (2016). SoK: Towards Grounding Censorship Circumvention in Empiricism. *IEEE Symposium on Security and Privacy*.

Wang, L., Dyer, K. P., Akella, A., Ristenpart, T., and Shrimpton, T. (2015). Seeing through network-protocol obfuscation. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 57–69, New York, NY, USA. ACM.

Wang, Q., Gong, X., Nguyen, G. T., Houmansadr, A., and Borisov, N. (2012). Censorspoofer: Asymmetric communication using ip spoofing for censorship-resistant web browsing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 121–132, New York, NY, USA. ACM.

Wang, T., Cai, X., Nithyanand, R., Johnson, R., and Goldberg, I. (2014). Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 143–157, Berkeley, CA, USA. USENIX Association.

Wang, T. and Goldberg, I. (2013). Improved website fingerprinting on tor. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 201–212, New York, NY, USA. ACM.

Wang, T. and Goldberg, I. (2016). On realistically attacking tor with website fingerprinting. In *Privacy Enhancing Technologies Symposium (PETS)*.

Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Cheung, S., Wang, F., and Boneh, D. (2012). StegoTorus: A camouflage proxy for the Tor anonymity system. In *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*.

Wright, C., Coull, S., and Monrose, F. (2009a). Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium - NDSS '09*. IEEE.

Wright, C., Coull, S., and Monrose, F. (2009b). Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium - NDSS '09*. IEEE.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *CoRR*, abs/1411.1792.

Yu, J. and Chan-Tin, E. (2014). Identifying webbrowsers in encrypted communications. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 135–138, New York, NY, USA. ACM.

VITA

WEIQI CUI

Candidate for the Degree of

Doctor of Philosophy

Dissertation: WEBSITE FINGERPRINTING ATTACKS

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the degree of Doctor of Philosophy in Computer Science at Oklahoma State University, Stillwater, Oklahoma, in May 2019

Completed the requirements for the Bachelor of Science in Computer Science at Dalian University of Technology, Dalian, Liaoning, China, in July 2014

Experience:

1.Machine Learning Internship, Facebook, Inc., Seattle, (May 2018—August 2018)
2.Teaching Assistant, Oklahoma State University, (Aug 2014—Present)