Realistic Cover Traffic to Mitigate Website Fingerprinting Attacks

Weiqi Cui, Jiangmin Yu, Yanmin Gong, Eric Chan-Tin
Computer Science Department
Oklahoma State University
Stillwater, OK 74078

***Abstract*—Website fingerprinting attacks have been shown to be able to predict the website visited even if the network connection is encrypted and anonymized. These attacks have achieved accuracies as high as 92%. 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.**

I. Introduction

Website fingerprinting violates the privacy expected from a user when she is using an anonymizing service such as a proxy or Tor [1]. The goal of website fingerprinting attacks [2] is to determine the website visited by a victim. The adversary, in this case, is usually local, for example on the same network or the Internet Service Provider, and can observe all the network traffic sent by the victim. These attacks are effective and are very accurate in successfully identifying the websites. The accuracy is over 90% even when the network traffic is encrypted or anonymized through a proxy. Since the adversary knows who the user is and can accurately guess what websites she is visiting, the user has no privacy.

Various defenses against website fingerprinting attacks [3]–[7] 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%.

Our **contribution** is 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 [1]; 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.

Table III shows a comparison of our proposed algorithm with existing mitigation techniques. Our algorithm has comparable accuracy over 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 delays are introduced.

[1]cover traffic and noise are used interchangeably

II. Related Work

It has been shown that analyzing encrypted network traffic can reveal the websites and webpages visited [2]–[6], [8]–[20]. 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 [21] 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 [22], which could lead to user identification as most users utilize a unique webbrowser [23].

Using cover/dummy/fake traffic to mask a user's activities has been proposed before [24]. It has been shown that this mechanism can be countered or the cover traffic removed [25], [26] 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

2575-8411/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDCS.2018.00175

accurately guess the website from the observed packet trace. Another scheme, Track Me Not [27], focused on web search queries and generating fake web searches, but [28] has shown that web search queries obfuscation can still be analyzed.

Various website fingerprinting defenses have been proposed [3]–[7], [13], [29], [30]. 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 effective in mitigating website fingerprinting attacks. Our proposed defense has zero latency overhead and lower bandwidth overhead while maintaining a high level of effectiveness.

III. Background

A. Website Fingerprinting

Website fingerprinting aims to determine the website visited by examining the network trace sent by a victim's webbrowser. That trace is usually encrypted and sent over a proxy or an anonymous network like Tor [1], [31] so that the network contents cannot be analyzed. The only information that can be observed are the packet sizes, the direction of the packets, the time interval between packets, and the number of packets sent and received.

A packet trace PT consists of n network packets. PT also consists of the tuple $< \Sigma_{i=0}^{n} N_i, N_s, N_c >$, where N_i is each individual network packet, N_s is the total number of packets from server to client, N_c is the total number of packets sent from the client to the server. Each network packet N_i forms the tuple $< S_i, T_i, D, M_s, M_c >$, where S_i is the size of the packet, T_i is the time interval until the next packet, D is the direction of the network packet (from client to server or from server to client), M_s is the number of packets from the server to the client, and M_c is the number of packets from the client to the server. M_s and M_c denotes each "train" of packets, that is, there are usually a few packets from client to the server, followed by several packets from server to client. On the other hand, N_c and N_s denotes the total number of packets for the whole packet trace.

B. Classification

Previous work 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.

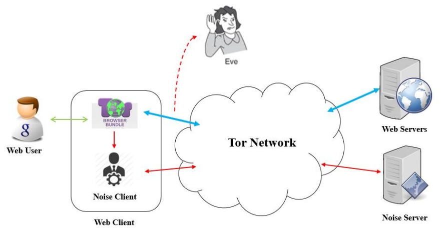

Fig. 1. Our system model and experimental setup.

In this work, we used the same features as those mentioned in [17]: cumulative size of packets sampled at regular intervals over the whole packet trace, number of incoming packets, total size of incoming packets, number of outgoing packets, and total size of outgoing packets. We also used the random decision forests as it was used by the most recent paper [20].

C. 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 [1]. Figure 1 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.

IV. Proposed Noise Algorithm

A. Overview

Our proposed algorithm to generate cover traffic is novel since it generates realistic noise rather than random noise or random padding. The intuition of this algorithm is from the results of previous work [18] which shows that it's hard to split two mixed traces of websites. 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 followed 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 [18], 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

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

Fig. 2. Sample of recorded network traffic. The format is $< timestamp >:< packetsize >$. Red indicates outgoing packets.

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

TABLE I
THE PARSED OUTGOING TRAFFIC TRAIN SET.

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 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: $< timestamp >:< packetsize >$. 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.

B. Algorithm Description

We illustrate the algorithm and explain it with an example shown in Figure 2. The following two tables are built based on this example. Table I shows the parsed outgoing packets set, denoted as TS_out. Table II 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, 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.

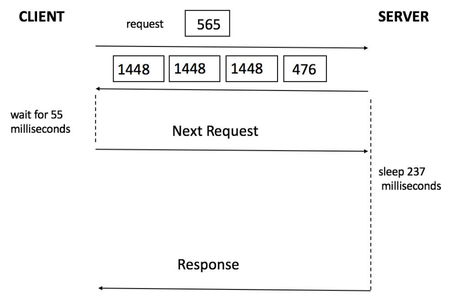

Fig. 3. An example of the algorithm, train 4 is selected from TS_{out} and train 8 is selected from TS_{in}.

As an example, let's suppose outgoing traffic train 4 is selected from TS_{out} and incoming traffic train 8 is selected from TS_{in}. Our algorithm will create a new cover 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 receives 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 I. The procedure of this example is shown in figure 3. 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

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 II
THE PARSED INCOMING TRAFFIC TRAIN SET.

Mitigation	Accuracy (%)	Latency Overhead (%)	Bandwidth Overhead(%)
No Defense	91%	0%	0%
CS-BuFLO [7]	22%	173%	130%
Tamaraw [4]	10%	200%	38%
WTF-PAD [29]	15%	0%	54%
Walkie-Talkie [30]	19% $\sim$ 44%	34%	31%
Our Algorithm	14%	0%	20%

TABLE III
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 [29].

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 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 network delay is introduced. Our algorithm only generates cover traffic to another website.

V. SIMULATED EXPERIMENT

A. Experiment Setup

We utilized the dataset from [17], which consists of $1,125$ webpages and 40 instances of each webpage. Each instance contains the timestamp of each packet with the packet sizes (negative packet sizes indicate outgoing packet). We implemented the noise generation algorithm described in Section IV.

We use the standard Weka [32] tool and experimented with the Random Forest classification algorithm. The classification features used in our experiments are the same as those used in [17]. 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 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 [18], 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 [2]. The dataset [17] contained timestamps and packet sizes. Merging the original website packet trace with the noise packet trace is relatively 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 as a comparison. 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.

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

Our noise generation algorithm "learning" dataset consists of a further 10 webpages where the packet traces are recorded. As the noise is learned form the users' network traffic, in the simulation, we ran our algorithm to generate one packet trace of noise for each of the original 91 webpages and merge that trace with the original webpage packet trace.

B. Simulation Results

The accuracy of the Random Forest classification algorithm including either features c or features a and c is 81.77% and 81.88% respectively. The first four features are always included. Adding features a, the cumulative total of the absolute value of all packet sizes, does not improve the accuracy of website fingerprinting attacks by much. The sample size n was set to the recommended 100 from [17]. We only considered set of features c from now.

Figure 4 shows the classification accuracy for varying amount of noise added to original traces. Figure 5 shows the bandwidth overhead in $\%$ of the extra network 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.

Intuitively, accuracy should decrease as more noise is added. However, in Figure 4, 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.

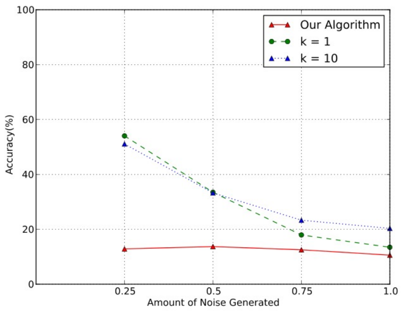

Fig. 4. The accuracy using the Random Forest algorithm when introducing different ki

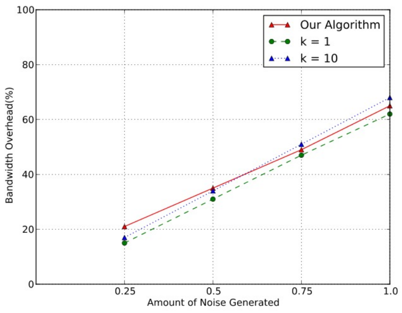

Fig. 5. The bandwidth overhead when introducing different kinds of cover traffic.

VI. Discussion and Future Work

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, much lower than existing schemes. The latency overhead is also 0%. Our algorithm can also be configured to utilize different amounts of bandwidth. We plan to run a real-world experiment in the future, where computation overhead will be calculated.

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. Another possibility that we will explore in the future is using a fixed set of m websites for the noise algorithm, where m would be a large number like $10,000$ or $100,000$. Since the noise algorithm is based on a probability distribution, an adversary is not able to guess the noise traffic and filter it out.

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.

Acknowledgment

The research results discussed in this publication were made possible in total or in part by funding through the

Health Research award for project number HR13-035, from the Oklahoma Center for the Advancement of Science and Technology.

References

[1] Tor, https://www.torproject.org/, 2017.

[2] A. Hintz, "Fingerprinting websites using traffic analysis," in *Proceedings of the 2Nd International Conference on Privacy Enhancing Technologies*, ser. PET'02. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 171–178. [Online]. Available: http://dl.acm.org/citation.cfm?id=1765299.1765312

[3] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 605–616. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382260

[4] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 143–157. [Online]. Available: http://dl.acm.org/citation.cfm?id=2671225.2671235

[5] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES '14. New York, NY, USA: ACM, 2014, pp. 131–134. [Online]. Available: http://doi.acm.org/10.1145/2665943.2665950

[6] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 227–238. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660362

[7] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES '14. New York, NY, USA: ACM, 2014, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/2665943.2665949

[8] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, ser. SP '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 19–. [Online]. Available: http://dl.acm.org/citation.cfm?id=829514.830535

[9] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted http streams," in *Proceedings of the 5th International Conference on Privacy Enhancing Technologies*, ser. PET'05. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1007/11767831_1

[10] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 255–263. [Online]. Available: http://doi.acm.org/10.1145/1180405.1180437

[11] L. Lu, E.-C. Chang, and M. C. Chan, *Website Fingerprinting and Identification Using Ordered Feature Sequences*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 199–214. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15497-3_13

[12] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, ser. CCSW '09. New York, NY, USA: ACM, 2009, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/1655008.1655013

[13] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, ser. WPES '11. New York, NY, USA: ACM, 2011, pp. 103–114. [Online]. Available: http://doi.acm.org/10.1145/2046556.2046570

[14] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis," in *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, ser. PETS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 58–78. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31680-7_4

[15] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, ser. WPES '13. New York, NY, USA: ACM, 2013, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/2517840.2517851

[16] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 263–274. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660368

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

[18] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting," in *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[19] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting data-usage statistics for website fingerprinting attacks on android," in *Proceedings of the 9th ACM Conference on Security &#38; Privacy in Wireless and Mobile Networks*, ser. WiSec '16. New York, NY, USA: ACM, 2016, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2939918.2939922

[20] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 1187–1203. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes

[21] M. Perry, "Experimental defense for website traffic fingerprinting," https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, 2011.

[22] J. Yu and E. Chan-Tin, "Identifying webbrowsers in encrypted communications," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES '14. New York, NY, USA: ACM, 2014, pp. 135–138. [Online]. Available: http://doi.acm.org/10.1145/2665943.2665968

[23] P. Eckersley, "How unique is your web browser?" in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, ser. PETS'10, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881151.1881152

[24] C. Diaz and B. Preneel, "Taxonomy of mixes and dummy traffic," in *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, August 2004.

[25] N. Mallesh and M. Wright, "Countering statistical disclosure with receiver-bound cover traffic," in *Proceedings of 12th European Symposium On Research In Computer Security (ESORICS 2007)*, ser. Lecture Notes in Computer Science, J. Biskup and J. Lopez, Eds., vol. 4734. Springer, September 2007, pp. 547–562.

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

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

[28] S. Peddinti and N. Saxena, "On the privacy of web search based on query obfuscation: A case study of trackmenot," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, M. Atallah and N. Hopper, Eds. Springer Berlin Heidelberg, 2010, vol. 6205, pp. 19–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14527-8_2

[29] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *ESORICS*, 2016.

[30] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1375–1390. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tao

[31] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[32] Weka, http://www.cs.waikato.ac.nz/ml/weka/, 2017.