DynaFlow: An Efficient Website Fingerprinting Defense Based on Dynamically-Adjusting Flows

David Lu
MIT PRIMES
davidboxboro@gmail.com

Sanjit Bhat
MIT PRIMES
sanjit.bhat@gmail.com

Albert Kwon
MIT
kwonal@mit.edu

Srinivas Devadas
MIT
devadas@mit.edu

ABSTRACT

Website fingerprinting attacks enable a local adversary to determine which website a Tor user visits. In recent years, several researchers have proposed defenses to counter these attacks. However, these defenses have shortcomings: many do not provide formal guarantees of security, incur high latency and bandwidth overheads, and require a frequently-updated database of website traffic patterns. In this work, we introduce a new countermeasure, DynaFlow, based on dynamically-adjusting flows to protect against website fingerprinting. DynaFlow provides a similar level of security as current state-of-the-art while being over 40% more efficient. At the same time, DynaFlow does not require a pre-established database and extends protection to dynamically-generated websites.

ACM Reference Format:
David Lu, Sanjit Bhat, Albert Kwon, and Srinivas Devadas. 2018. DynaFlow: An Efficient Website Fingerprinting Defense Based on Dynamically-Adjusting Flows. In *2018 Workshop on Privacy in the Electronic Society (WPES'18), October 15, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3267323.3268960

1 INTRODUCTION

Due to increases in mass surveillance and other attacks on privacy, many Internet users have turned to Tor [8] to protect their anonymity. Over the years, Tor has grown to over 6,000 volunteer servers and 4 million daily users [16]. At a high level, Tor protects its users' identities by routing each packet through a number of Tor servers. Each server knows only the immediate hop before and after itself, and as a result, no single server learns both the identity of the user and the destination of the packet.

Tor, however, has been known to suffer from different traffic analysis attacks [2, 5, 9, 10, 12, 17]. In particular, there have been several recent works on an attack called *website fingerprinting* (WF) that allows an adversary who observes only the connection between the user and the Tor network to identify which website the user visits. The WF adversary is a passive observer (he will not drop, modify or insert packets) who monitors the connection between a user and the Tor network, as shown in Figure 1. The adversary is interested in identifying visitors of *monitored* websites, which the adversary deems "sensitive." To carry out the attack, the adversary visits the monitored websites on his own, and possibly many unmonitored websites as well, to collect *traces*, the sequence of packets and their time stamps generated while visiting a website. He then creates a database of the traces, and proceeds to monitor and collect users' traces. Finally, he attempts to infer which website corresponds to which user trace by comparing the user traces to the database (i.e., classify the user traces into one of many classifications).

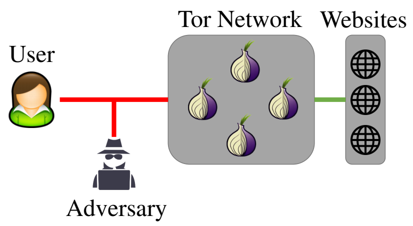

Figure 1: The Website Fingerprinting threat model.

We consider two different attacker settings.

Closed-world. In this setting, we assume that users only visit monitored websites. Here, *accuracy* measures the effectiveness of the attacker, which is simply the proportion of user traces that were classified correctly.

Open-world. In the real world, users can visit websites that the adversary does not monitor. The open-world setting aims to emulate this scenario by allowing the users to visit both monitored and unmonitored websites. The adversary must first classify each trace as either an unmonitored or monitored website, and if it is a monitored website, then also identify the specific website. The effectiveness in this setting is typically measured by *true positive rate* (TPR), the proportion of monitored traces that are classified correctly, and *false positive rate* (FPR), the proportion of unmonitored traces that are incorrectly classified as a monitored website.

2 BACKGROUND AND RELATED WORK

Some of the earlier WF defenses were designed to counter specific attacks. Although such defenses defeated older attacks, they often failed against newer ones [5, 10, 17]. Other works instead have focused on creating defenses with stronger security guarantees that protect users against much larger classes of attackers while providing bounds on the attacker's effectiveness. There are two

Table 1: Comparison of DynaFlow with prior defenses.

	Low Overheads	Strong Privacy Guarantees	No Database	High Tunability
DynaFlow	✓	✓	✓	✓
BuFLO [9]	×	×	✓	×
CS-BuFLO [3]	×	✓	✓	×
Tamaraw [4]	×	✓	✓	×
Supersequence [17]	×	✓	×	×
Glove [13]	×	✓	×	×
Walkie-Talkie [18]	✓	✓	×	✓
Decoy Pages [15]	×	×	✓	×
WTF-PAD [11]	✓	×	✓	✓
LLaMA [7]	✓	×	✓	×
ALPaCA [7]	✓	×	×	✓

high-level classes of such defenses: *supersequence* defenses and *constant-flow* defenses.

2.1 Supersequence defenses

Defenses in this class [13, 17, 18] first collect a database of traffic traces of many different websites. Next, they group the traces into one or more sets, each having a single "supersequence"; the supersequence of a set contains all traces in the set as subsequences. Then, each trace is morphed into the supersequence of the set via padding and delays between packets. Walkie-Talkie [18], for example, uses half-duplex communication and breaks each trace down into sequences of bursts, making supersequences easier to create.

Unfortunately, supersequence defenses in general are difficult to deploy in practice because they require a database of web traces that must constantly be updated as the websites change. Consequently, these defenses only apply easily to static websites that do not change over time and have difficulty protecting websites that use AJAX or JavaScript [13, 18]. Furthermore, they usually incur large overheads, often doubling the latency and bandwidth usage [13].

2.2 Constant-flow defenses

Defenses in this second class flood the network with a continuous stream of packets to prevent the adversary from identifying any meaningful patterns in the traffic [3, 4, 9]. The current best constant-flow defenses are Tamaraw [4], which sends packets at a fixed rate, and CS-BuFLO [3], which varies the rate of packet transmission.

Constant-flow defenses do not require any pre-built databases, potentially making them easier to deploy. However, these defenses incur high latency and bandwidth overheads (usually 100% or more), even for their lightest configurations. Thus, our goal in this work is to construct a defense with similar guarantees as Tamaraw and CS-BuFLO but with significantly lower overheads. Table 1 compares our defense with the top WF defenses.

3 DYNAFLOW

At a high level, DynaFlow is a WF countermeasure based on constant-flow with efficiency improvements that come from its dynamic nature and parameterizability. Unlike supersequence defenses, DynaFlow does not use a database and does not limit protections to static content. Instead, DynaFlow uses fixed burst patterns with dynamically-changing intervals between packets to hide what website a user visits. In contrast to other constant-flow defenses, DynaFlow is highly tunable; it can be configured to reduce attacker accuracies to below 10% with moderate efficiency or reduce overheads to 30-50% with moderate security. Prior constant-flow defenses do not have this feature because they incur high overheads at all security levels [3, 4, 9].

To measure the overheads of our defense, we will use the time overhead (TOH) and the bandwidth overhead (BWOH), suggested by prior work [4]. TOH (i.e., latency overhead) is defined to be the extra amount of time needed to load the defended trace over the load time without any defense. Similarly, BWOH is defined to be the number of additional bytes in the defended trace over the byte size of the original trace.

Table 2: Parameters of DynaFlow.

Parameter	Description
o	Number of outgoing packets in a burst
i	Number of incoming packets in a burst
t_i	Initial inter-packet timing
b	Number of bursts between inter-packet timing adjustments
a	Number of allowed inter-packet timing adjustments
m	Burst padding base
T	Set of possible inter-packet timings

3.1 DynaFlow design

Our defense consists of three parts: (1) burst-pattern morphing, (2) constant traffic flow with dynamically-changing intervals, and (3) padding the number of bursts. Table 2 summarizes all system parameters of our defense.

3.1.1 Burst-pattern morphing. Although existing constant-flow defenses such as CS-BuFLO handle incoming and outgoing packets independently [3, 4, 9], DynaFlow considers incoming and outgoing packets together, and morphs packets into fixed bursts. We first note that every traffic pattern can be broken into small bursts of packets consisting of o consecutive outgoing packets followed by i consecutive incoming packets [18] (from the client's point of view). In DynaFlow, we fix a particular o and i, and morph the traffic such that every burst is identical by adding dummy packets in both directions. After sweeping the parameters, we found that $o = 1$ and $i = 4$ resulted in the least overhead. That is, the traffic pattern with DynaFlow is always of this form: one outgoing packet, four incoming packets, one outgoing packet, and so on.

3.1.2 Inter-packet timing. In addition to morphing the burst patterns, we also have a constant flow of traffic. Specifically, packets are queued and sent out every t seconds according to the pre-selected burst pattern (§3.1.1). If there are no packets in the queue and we are supposed to send a packet, then DynaFlow inserts a dummy packet. Moreover, similar to CS-BuFLO [3], DynaFlow dynamically changes the value of t. However, DynaFlow allows for more adjustable tuning of t by introducing several parameters. We initially set the inter-packet interval t to be a predetermined value t_i. Then, after a fixed number of bursts b, we allow dynamic changes to the interval t. Our defense estimates the new value of t by computing a weighted sum of the average inter-packet time interval of the last 20 bursts (before morphing) and the number of packets in the queue. Intuitively, this tracks whether we are using too many dummy packets, or conversely, setting the interval too high.

For every website, we allow up to a adjustments, and during each adjustment, the client chooses t from a limited set of available

Table 3: Closed-world results against k-NN [17] and k-FP [10]. Baseline indicates the case with no defense. All values are in %.

Config.	Parameters	k-NN [17]	k-FP [10]	TOH	BWOH
Baseline	N/A	88.0	94.3	0	0
1	$o = 1$, $i = 4$, $t_i = 0.012$ $b = 160$, $a = 7$, $m = 1.2$ $T = \{0.0012, 0.005\}$	17.5	45.0	31	53
2	$o = 1$, $i = 4$, $t_i = 0.012$ $b = 80$, $a = 1$, $m = 1.2$ $T = \{0.0015\}$	6.0	18.4	38	84

intervals $T = \{t_1, \dots, t_k\}$ that best matches its traffic pattern. For the highest levels of security, we can set $|T| = 1$, $b = \infty$, or $a = 0$, which would guarantee that all traces are identical except for the total number of bursts. Conversely, by increasing $|T|$, decreasing b, or increasing a, we can trade off security for lower overheads.

3.1.3 The number of bursts. The total number of bursts (i.e., the sizes of the traces) may still leak a significant amount of information. For example, it would be easy to distinguish between a video-hosting website and a small text-based search engine. To hide this, we pad the number of bursts to $\{\lfloor m \rfloor, \lfloor m^2 \rfloor, \lfloor m^3 \rfloor, \cdots\}$ for some fixed $m > 1$ by inserting dummy bursts. By padding to a power of m, we mitigate the privacy loss by significantly limiting the number of possible traces.[1] At the same time, this padding scheme generates enough trace lengths to ensure that overheads remain relatively low.

3.1.4 Combining packets. We also introduce a small optimization to offset some of the overhead. Our defense often results in multiple outgoing requests waiting in the queue. In the case where there are two consecutive outgoing packets (before burst morphing) that total less than a single Tor packet (512 bytes), we combine them into a single packet. Small responses are also combined by the exit node. This decreases the number of packets, which mitigates the effect of delays.

4 DYNAFLOW EVALUATION

We now evaluate our defense.

4.1 Implementation and data set

To evaluate our defense, we use a data set that we collected,[2] which consists of 100 monitored pages (90 traces each) and 9000 unmonitored pages (1 trace each). The 100 monitored pages were retrieved from Alexa's list of the 100 most popular websites [1] while the next 9000 most popular pages make up the unmonitored class. We collected our traces by configuring Firefox 57.0.1 to access websites through Tor 0.2.9.9.[3] Each time we collected a trace, we changed the Tor circuit to emulate the fact that different users and the adversary will access a single website using different circuits.

Similar to prior defenses [4, 9, 13, 17, 18], we simulate our defense by replaying the collected traces through our defense to generate new traces. Our code and data set can be found on GitHub.[4]

[1]Information theoretically speaking, this would leak a logarithmic number of bits.
[2]Prior data sets do not contain enough information to simulate our defense because they excluded the size of unpadded packets used for packet combination.
[3]In an actual deployment, the users would use the Tor browser, which could yield a different result from our experiments.
[4]https://github.com/davidboxboro/DynaFlow

Table 4: Open-world results against k-NN [17] and k-FP [10]. The configurations are the same as those of Table 3. All values are in %.

Config.	k-NN [17]		k-FP [10]		TOH	BWOH
	TPR	FPR	TPR	FPR		
Baseline	84.5	2.5	86.3	1.6	0	0
1	15.4	20.6	5.0	1.6	23	59
2	5.9	69.0	4.4	40.1	28	112

4.2 Evaluation against existing attacks

We first run DynaFlow on the data set, and then run different website fingerprinting attacks on the defended data set in closed-world and open-world scenarios.

4.2.1 Closed-world defense evaluation. We evaluated the effectiveness of our defense against k-NN [17] and k-FP [10] using two different configurations.[5] The results and the configurations used are summarized in Table 3.

DynaFlow causes substantial decreases in the accuracy of both classifiers. Configuration 1 of DynaFlow decreases the accuracy of k-NN [17] from 88.0% to 17.5% and k-FP [10] from 94.3% to 45.0%. At the same time, configuration 1 has fairly low time and bandwidth overheads at 31% and 53%, respectively. Configuration 2 can lower the accuracy even further to 6.0% and 18.4%, at the cost of higher latency and bandwidth overheads of 38% and 84%.

4.2.2 Open-world defense evaluation. DynaFlow is just as effective in the open-world using the same configurations as the closed-world, as summarized in Table 4. With configuration 1, we reduce the TPR of both attacks to below 16%, with time and bandwidth overheads of 23% and 59%, respectively. Configuration 2 further lowers the TPR to below 6% for both attacks, while increasing the FPR to above 40%.

4.3 Comparison with prior work

We now compare with two prior works, CS-BuFLO [3] and Tamaraw [4], and demonstrate that DynaFlow can achieve the same level of security with lower overheads.

4.3.1 Optimal attacker strategy. We first describe the strategy of the optimal attacker. The optimal attacker wishes to learn $\Pr[\mathtt{w}|\mathtt{t}]$ for a trace $\mathtt{t}$ and website $\mathtt{w}$ (i.e., the probability that $\mathtt{w}$ is visited when $\mathtt{t}$ is observed) for all $\mathtt{w}$ and $\mathtt{t}$ so that he can maximize his chances of classifying a website correctly. We assume that the attacker knows $\Pr[\mathtt{w}]$ (i.e., how likely a website is visited) for all websites $\mathtt{w}$.

The attacker starts by collecting multiple traces per website. For a given $\mathtt{w}$ and $\mathtt{t}$, the attacker's best estimate for $\Pr[\mathtt{t}|\mathtt{w}]$ is the proportion of traces of $\mathtt{w}$ that are identical to $\mathtt{t}$. The attacker can then compute $\Pr[\mathtt{t}]$ as follows:

$$\Pr[\mathtt{t}] = \sum_{\mathtt{w}'} \Pr[\mathtt{w}', \mathtt{t}] = \sum_{\mathtt{w}'} \Pr[\mathtt{w}'] \cdot \Pr[\mathtt{t}|\mathtt{w}'] ,$$

where $\Pr[\mathtt{t}|\mathtt{w}']$ is estimated using the traces he collected. Finally, the attacker can use Bayes' theorem to estimate $\Pr[\mathtt{w}|\mathtt{t}]$ for any given $\mathtt{w}$ and $\mathtt{t}$:

$$\Pr[\mathtt{w}|\mathtt{t}] = \frac{\Pr[\mathtt{w}] \cdot \Pr[\mathtt{t}|\mathtt{w}]}{\Pr[\mathtt{t}]} = \frac{\Pr[\mathtt{w}] \cdot \Pr[\mathtt{t}|\mathtt{w}]}{\sum_{\mathtt{w}'} \Pr[\mathtt{w}'] \cdot \Pr[\mathtt{t}|\mathtt{w}']} .$$

[5]We did not compare against CUMUL [14] because prior work [10, 18] has shown that k-FP and k-NN consistently outperform CUMUL when defenses are applied.

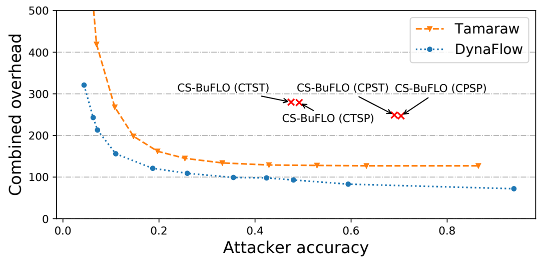

Figure 2: Sum of TOH and BWOH versus the optimal attacker's accuracy for different defenses.

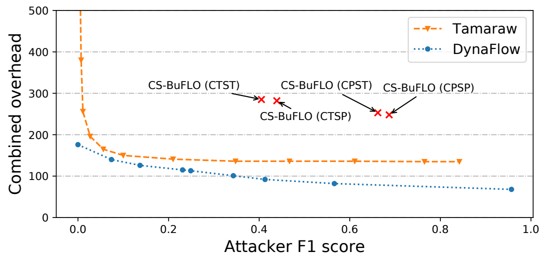

Figure 3: Sum of TOH and BWOH versus the optimal attacker's F1 score for different defenses.

Now suppose that the optimal attacker observes a trace t. To maximize his chances of guessing the correct website that corresponds to t, he should choose a website w such that Pr[w|t] is maximized.

4.3.2 Defense against optimal attacker. We now compare DynaFlow to prior works [3, 4]. For CS-BuFLO [3], we used Cherubin's simulation implementation [6]. To ensure fair comparison, we run Tamaraw [4] and CS-BuFLO [3] on our data set, sweep their parameters, and use the parameters that performed best. For Tamaraw, we outgoing and incoming packet intervals of 0.012 and 0.003, respectively, while varying the padding parameter L. For CS-BuFLO, we set the initial, minimum, and maximum values of $\rho*$ to 0.016, 0.004, and 0.128, respectively, and tested all four padding schemes (CTST, CTSP, CPST, and CPSP) described in the paper [3].

We assume that the probability that an unmonitored website is visited is 0.5, and the probability a monitored website is visited is the same across all websites (i.e., $\Pr[w_1] = \Pr[w_2]$ for all monitored websites w_1 and w_2), though this could easily be changed. For our experiments we found that $o = 1$, $i = 4$, and $t_i = 0.012$ works the best, with other parameters varying. We found the parameters by sweeping one parameter at a time while fixing the others.

Once we apply the defenses to our data set, we create an optimal attacker that performs the attack described in §4.3.1 for both closed-world and open-world settings. In the closed-world, we plot the sum of the time and bandwidth overhead for a given attacker accuracy in Figure 2. As shown, DynaFlow achieves lower overhead than Tamaraw and CS-BuFLO for all attacker accuracies. For example, DynaFlow can reduce the optimal attacker accuracy to below 50% while keeping the sum of the overheads to 93% (34% TOH and 59% BWOH). To achieve the same level of security, Tamaraw's requires 128% total overhead (46% TOH and 82% BWOH), and CS-BuFLO requires 279% overhead (127% TOH and 152% BWOH). At higher levels of security (which only our defense and Tamaraw can achieve), our defense remains more efficient. When the attacker's accuracy is 20%, our defense incurs 121% total overhead (38% TOH and 84% BWOH), while Tamaraw incurs 162% total overhead (58% TOH and 104% BWOH). At 7% optimal attacker accuracy, our defense expends 213% overhead, compared to Tamaraw's 419%. Altogether, the flexibility of DynaFlow allows for greater efficiency than Tamaraw at all security levels.

For open-world, we plot the sum of the two overheads for a given F1 score in Figure 3. F1 is the harmonic mean of precision (the number of monitored traces the adversary correctly classifies divided by the number of traces that were classifed as monitored) and recall (TPR); intuitively, F1 is a single metric that aims to capture both true positives and false positives [11]. Similar to the closed-world setting, DynaFlow achieves better overhead than Tamaraw and CS-BuFLO. To achieve an F1 of 41% (29.0% TPR and 11.4% FPR), DynaFlow incurs an overhead of 92% (26% TOH and 66% BWOH). For the same F1, Tamaraw needs 136% overhead (39% TOH and 97% BWOH) and CS-BuFLO needs 285% overhead (112% TOH and 173% BWOH). At higher F1 scores, the gaps are even larger.

5 DISCUSSION AND FUTURE WORK

In this section, we discuss possible limitations of our work and possible avenues for future work.

DynaFlow parameters. DynaFlow has several parameters, which must be tuned to network conditions and user preferences. For instance, if the user's primary concern is the latency of the connection, then we would set the inter-packet timing to be something lower. In this paper, we set the parameters by fixing all but one and doing a sweep of the one left. This may not result in optimal parameters, and it takes a while to find the parameters. In the future, we would like to find a more efficient way to determine the parameters to allow easy configuration based on the user's priorities.

DynaFlow overheads. To protect users, all WF defenses must incur overheads in the Tor network. Compared to prior defenses like Tamaraw, DynaFlow's time and bandwidth overheads are relatively low, at 30-50% for moderate levels of security. However, for high security configurations, the total overhead sum can exceed 120%, which might be too high for general purpose usage. We hope to further reduce the overheads in future work.

DynaFlow requirements. To run DynaFlow, the Tor source code must be modified to change the behavior of both the Tor client and exit node. This does make it more difficult to deploy the defense, as it requires changes in the volunteer servers. This also could negatively affect performance of exit nodes, which are already overloaded. Nevertheless, we believe that the strong security guarantees, low overheads, and lack of a database still provide significant benefits over prior defenses. Moreover, many existing defenses [3, 4, 9, 13, 17, 18] require participation from exit servers as well.

6 CONCLUSION

In this work, we propose DynaFlow, a provably-secure defense based on burst pattern morphing and dynamically-changing flows. DynaFlow lowers the overhead of prior art by over 40% while achieving similar security guarantees due to its dynamic nature and tunability. At the same time, DynaFlow is also easier to deploy than prior work: it does not need a large, up-to-date database, which allows it to extend protections to non-static content.

REFERENCES

[1] Alexa. 2017. The Top 500 Sites on the Web. https://www.alexa.com/topsites.

[2] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. 2007. Low-Resource Routing Attacks Against Tor. In *WPES*. 11–20.

[3] X. Cai, R. Nithyanand, and R. Johnson. 2014. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *WPES*. 121–130.

[4] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *ACM CCS*. 227–238.

[5] X. Cai, X. Zhang, B. Joshi, and R. Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM CCS*. 605–616.

[6] G. Cherubin. 2017. Bayes, not Naïve: Security Bounds on Website Fingerprinting Defenses. In *PETS*. 215–231.

[7] G. Cherubin, J. Hayes, and M. Juarez. 2017. Website Fingerprinting Defenses at the Application Layer. In *PETS*. 186–203.

[8] R. Dingledine, N. Mathewson, and P. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security*. 303–320.

[9] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *IEEE S&P*. 332–346.

[10] J. Hayes and G. Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *USENIX Security*. 1187–1203.

[11] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. 2016. Toward an Efficient Website Fingerprinting Defense. In *ESORICS*. 27–46.

[12] S. J. Murdoch and P. Zieliński. 2007. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *PETS*. 167–183.

[13] R. Nithyanand, X. Cai, and R. Johnson. 2014. Glove: A Bespoke Website Fingerprinting Defense. In *WPES*. 131–134.

[14] A. Panchenko, F. Lanze, A. Zinnen, and M. Henze. 2016. Website Fingerprinting at Internet Scale. In *NDSS*.

[15] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. 2011. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *WPES*. 103–114.

[16] The Tor Project. 2018. Tor Metrics Portal. https://metrics.torproject.org.

[17] T. Wang, X. Cai, R. Johnson, and I. Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security*. 143–157.

[18] T. Wang and I. Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *USENIX Security*. 1375–1390.