Delay-Conscious Defense Against Fingerprinting Attacks

Jingyuan Liang
Cleveland State University
Cleveland, Ohio, USA
Email: j.liang18@vikes.csuohio.edu

Chansu Yu
Cleveland State University
Cleveland, Ohio, USA
Email: c.yu91@csuohio.edu

Kyoungwon Suh
Illinois State University
Normal, Illinois, USA
Email: kwsuh@ilstu.edu

***Abstract*—In the past few years, many defense mechanisms have been proposed against website fingerprinting attacks. Walkie-Talkie (WT) built on top of Tor network is known to be one of the most effective and efficient defense mechanisms. However, we observed that WT significantly increases page loading times (time overhead) while adding little bandwidth overhead compared to other approaches. We analyze the cause of the increased page loading time and present a defending approach called Tail Timeout (TT), which addresses the problem, by introducing a timeout mechanism limiting the maximum time for which a pending request can block subsequent requests. Our experimental results indicate that the proposed TT defense can significantly reduce the page loading time while keeping similar defense performance in terms of true positive and true negative ratios achieved by WT.**

Keywords–Website fingerprinting; Censorship; Tor; Machine learning; User perceived experience.

I. INTRODUCTION

Internet censorship is on the rise [1]–[5]. Earlier, it was a simple surveillance of information in plaintext [6], [7]. Then, censors simply blocked the encryption layer when encryption was not widely-deployed [8]. Now, censors face the choice of what to do with the data that they cannot fully understand, and they usually cannot afford a complete block [9]–[11].

One of the popular approaches adopted by censors is known as website fingerprinting [12]–[14]. It uses features of packets in a network communication to infer the contents in the communication, or more precisely, to match the communication to one of several known feature models. Most popular features that have been exploited are: unique packet lengths, packet length frequency, packet ordering, and interpacket timing [15]. Machine learning approaches, such as *k*-Nearest Neighbors [16] can be used to conduct the fingerprinting attack. Once identified, the censor may decide to block the network connection completely. However, considering the fact that the identification of such connections is not guaranteed to be accurate, it is also possible that the censor may start deteriorating the quality of service by, for example, randomly dropping packets, or throttling the traffic [8], [17].

In this paper, we propose TT defense against website fingerprinting attacks, as an extension of WT [18] defense, which works on top of Tor [19]. WT is known to add little bandwidth overhead compared to other approaches, while keeping a moderate defending performance. We find that WT increases webpage loading time substantially, which is already high since it uses the Tor network. The page loading time may get larger unbearably if the censor decides to randomly drop packets as discussed earlier. TT defense addresses the problem by limiting the resource wait time. Our experimental study shows that WT increases page loading time by 50% or more in 22% of top 100 websites. On the other hand, in the case of TT of 1,000 msec, it occurs in only 7% of websites. Note that the defense efficiency of TT is on par with WT, but alleviates the time overhead problem.

This paper is organized as follows: Section II describes background and related work on fingerprinting attacks and the issues raised; Sections III and IV describe the TT defense methodology, implementation and setup in our experiments, respectively; Section V presents our experiment results; Section VI provides conclusions and future work.

II. BACKGROUND

To combat against website fingerprinting attacks, many efforts have been made on improving the Tor network itself or developing ones on top of Tor [20]. This is because Tor itself is already proven to be resilient to website fingerprinting attacks to some degree [21], [22]. Although Tor itself is not a complete standalone solution in combating website fingerprinting attacks, the fact that Tor network traffics are packed into fixed-length cells and sent over circuits built on-the-fly together with various control cells provides some degree of defense. Major media websites, such as BBC News, also adopted Tor, to ensure that their contents can be distributed to audiences without censorship [5]. Other defenses against website fingerprint attacks are to reschedule the packets (or other transmission units, such as Tor cells), or to insert padding units to confuse the attacker. An example of defenses using the rescheduling approach is Shmatikov's adaptive padding [23], and one using the inserting approach is Wright's traffic morphing [24]. The former approach adds a delay ("time overhead") because a unit can no longer be sent immediately once generated and needs to wait for scheduling, and the latter approach wastes network bandwidth ("bandwidth overhead") because of the padded units that do not carry useful information [25]. In addition, they only defend in a single aspect and may not be able to stand against modern attacks any more. More recent proposals for defenses against website fingerprinting attacks (BuFLO [26], Tamaraw [25], and WT [18]) use both rescheduling and padding. For example, BuFLO renders different websites produce the exact same packet sequence, defeating any possible classifier, but it increases both bandwidth and time overhead as mentioned above.

WT is known to be one of the most effective and efficient defense mechanisms. It addresses the bandwidth overhead problem by changing HTTP to a half-duplex communication protocol, in which only one party transmits data until all data

 ISBN: 978-1-61208-770-2

held on this party get processed. With this modification, as much as possible requests or responses are combined into a single burst, within which there are only transmissions in the same direction (i.e., either all requests or all responses), reducing the total number of bursts and thus leading to a smaller amount of padding units. Bandwidth overhead is less of a concern when there exists a vast amount of bandwidth available. Rather, time overhead in WT is a concern because it directly affects the quality of service and is easily misinterpreted as service unavailability.

III. Delay-Conscious Defense

WT works on top of Tor. With Tor, the adversary can only see (or infer) a sequence of fixed-length cells and their directions when users are browsing a website. The adversary calculates the fingerprints of such cell sequences for a list of popular websites and/or websites to be monitored, and stores the fingerprints in advance. When users are browsing the web, the adversary tries to match the cell sequence observed from users against the dataset of known fingerprints, attempting to identify the website users are browsing. Considering that the cells are fixed-length and encrypted, what the adversary can see are a number of alternating (as in direction) batches, and the number of cells in each batch, as fingerprints. WT combines some of the batches by delaying sending data (until the responses to the previous batch are completely received), reducing the number of batches. Furthermore, WT then adds padding data to each batch, attempting to make cell sequences more similar to each other. Since the number of batches is already reduced, and padding is needed only for each batch, the amount of padding data is limited.

The proposed mechanism, TT, is an improvement over WT. It excludes the slow resources from WT scheduling, to minimize the delay caused by such scheduling. However, the time needed for a certain resource request is not known at the time of request. What we can do is to remove it from the pool of pending requests when it has stayed there for certain tail time (t). Precisely speaking, the proposed mechanism is that, as long as there is a previous request to the server with pending response, which was sent no more than t seconds ago, no further request is sent, and they are queued to be sent later; as soon as all pending requests either get responses or become older than t seconds, all queued requests can be sent at once in one burst, then wait for the burst for their responses; and t is a parameter, to be adjusted. Once the pool of pending requests is empty, that is, requests either received a response, or have been removed because it passed t, the next batch of requests can be sent.

The number of such slow resources is usually less than 10% according to our study on top 100 websites. Considering that only a small number of requests trigger such scenarios, we expect that this change will not significantly increase the number of bursts, thus not improving the fingerprintability as demonstrated in our experimental evaluation.

A. WT and TT Implementation

We implemented WT and TT based on the Mozilla Firefox browser using an browser add-on. The originally published WT / half-duplex implementation was made by modifying the Mozilla Firefox source code. The patch no longer applies in current versions of Firefox because the relevant part in Firefox code has been rewritten; we decided to keep the Firefox platform, but reimplemented the procedure. To ensure that our solution is future-proof, we made our implementation using the webRequest WebExtensions API in Mozilla Firefox, instead of patching the codebase of Mozilla Firefox itself. We also included the support for a user-configurable tail timeout value.

We implement WT's half-duplex communication by changing the way the client's browser works. The destination server does not need to make any change to accommodate the change at the browser side; the client simply does not talk when the destination server is talking. We use the webRequest API, which allows extensions to attach event listeners to the various stages of making an HTTP request, and the event listeners receive detailed information about the request and can modify or cancel the request. To achieve the half-duplex communication, the add-on keeps track of all requests, and blocks requests from being sent unless there are not any other requests on the way according to the original WT half-duplex communication design. In order to implement the TT as described in this section, we record the timestamp for each request when a request is released from the blocked status; when the current time goes past *tail timeout* seconds after the recorded timestamp of a request, the request is no longer considered on the way even if it is, and does not block other requests anymore.

IV. Experiment Setup

In our experiment with the defense implementation described above, we accessed top 100 websites and captured traces of Tor cell sequences during website accesses. We then conducted attacks against the traces to compare the attack accuracy between scenarios with WT or TT used, or TT with different timeout values used. Further we also measured the time needed to load those websites in the WT and TT scenarios, and compared them with the loading time without defenses.

A. Data Set

We fetched Alexa's top website list [27], which was also used by Wang et. al. [18] in the WT research, as well as other researchers working on the same area [28], as of January 24, 2019, and removed duplicated subdomains (such as `tmall.com` and `login.tmall.com`) and localization domains (such as `google.com` and `google.co.in`) from the list. We access the main page using HTTP protocol (which redirects to the HTTPS version in many websites) to start the browsing session for each site.

To build the Tor cell sequence data sets, from the list after duplication removal, we use the top 100 websites as monitored websites, with each website visited 50 times. Then we take the next 5000 websites, each accessed once, as unmonitored websites. We run the open world scenario in our experiment, meaning that the attempted attack is trained on a part of monitored data and tested on mixed unmonitored and another part of monitored data; the attack is expected to identify whether the tested traces are from the monitored websites and if one trace is from a monitored website, which website it is from. In contrast, a closed world scenario is when a the trace being tested is always known to be from one of the monitored websites, and all those monitored websites have been seen in the training set by the attack. For each visit, we try one access with unmodified Firefox, another with WT and three accesses using tail timeout values of 1,000, 2,000 and 5,000 milliseconds, respectively.

 ISBN: 978-1-61208-770-2

For each access, we start a Firefox instance controlled by Selenium [29] through geckodriver [30]. The Firefox instance is configured by Selenium to use a SOCKS proxy listened by a modified Tor instance, which emits cell information. The Firefox instance also gets a WT/TT implementation extension installed (other than the access without WT/TT), with tail time value configured. At the same time, Tor cell capture and web browsing to the intended target start together. After 15 seconds, the Firefox instance is terminted and the captured cell sequence is saved to a file as a part of the data set.

Note that in [18], the collection of Tor cells during web browsing sessions with WT was done by doing TCP packet captures. They then reassemble TCP packets into TCP streams, and finally analyze TLS records and guess Tor cells based on record lengths. This is also the approach an actual attacker have to use, since all they can see are the packets, however there is no guarantee that the original cells can perfectly reconstructed. We adopted a different approach by modifying Tor and having it emit individual cell information directly. Although this approach is not practical for an attacker in real world, it would provide the most accurate cell sequences, allowing us to focus on evaluating the fingerprinting attacking and defending algorithms operating on cell sequences. Note that we are working on a higher layer before an outgoing SENDME is added and after an incoming SENDME is removed, so we do not need to think about SENDME removal. This would make attacking even easier in the experiment, and eliminate noises that can occur in real world attacking.

According to [18], there needs to be a padding step to make traffic through WT actually not fingerprintable. There were several factors to consider in the padding step, including difficulties in implementation for real time packet streams, but in consistence with the original research by Wang et. al., we only do simulated padding on the cell sequence data set built above to construct a "padded" data set. We follow the padding procedure described by Wang et. al. [18] to align the cell sequence being padded with a pre-selected decoy sequence by bursts. Then, for each burst (incoming and outgoing), if the number of cells is lower in the cell sequence to be padded than the decoy sequence, we add padding cells in the cell sequence to be padded inside the burst to make the number of cells equal to the number of cells in the burst in the decoy sequence at the same position. Specifically, we make the added cells uniformly distrubuted between the first cell and the last cell in the burst on the timeline, and for decoy sequence selection, we choose the largest sequence in the same data set for all sequences in the data set.

B. Performance Measures

To evaluate the defenses with and without the TT modification, we follow the approach taken to evaluate the original WT and measure the true positive rate (TPR), false positive rate (FPR), true negative rate (TNR) and false negative rate (FNR), on the cell sequences both before and after applying the padding step.

We run k-Nearest Neighbors (k-NN) classifier-based attack [16] on the cell sequences, which is shown in [18] by Wang et. al. to achieve much better performance compared to previous attack methodologies including SVM [12] and CUMUL [15]. There are also newer attacks, such as k-fingerprinting [31] and p-FP [28], which outperforms k-NN, while we still chose k-NN to ensure results comparable with the original WT research. During the attack, 1,225 features are extracted for each trace, and initial weights for those features are randomly chosen between 0.5∼1.5.

To measure the page loading time, independent from the 50 visits on each website collecting Tor cell sequences, we access each of the 100 websites five times and record the loading time values, and caluclate the average to build a list of page loading time for the 100 websites, for each TT configuration. For each access, we browse the website and record the timestamps of `navigationStart`, `domInteractive` and `domComplete` events in `window.performance.timing`. If the page does not finish loading within a certain time period, which is 120 sec when doing overall measurement and 300 sec when inspecting individual websites, or an error occurs during loading, the data is marked as such and may be excluded from final processing. Otherwise, we calculate (`domComplete` − `navigationStart`) as the page loading time.

V. Results and Discussion

As described above, evaluated data during experiments include time needed to load the pages, and defense effectiveness, as evidenced by attack accuracy on defensed datasets. To better discuss the reason why page loading time is improved, we further looked into two cases to identify the source of delay.

A. Page Loading Time

As described before, we measured website loading times from experiments with 70 websites selected out of the top 100 websites. The 30 websites were excluded because of unstable data and excessive amount of network errors, which usually happen on foreign websites without a good connectivity to the Internet. With each TT configuration, each site is accessed five times and the average value of the loading time is taken.

Figure 1 shows the overview of the loading times of websites before and after the TT change, and a significant improvement can be observed. More specifically, WT increases the website access time by 50% or more in 22% of the websites in our dataset. On the other hand, in the case of TT of 1,000 milliseconds, it occurs in only 7% of websites.

For comparison, we also captured data in Figure 2 where the `domInteractive` event is used in place of `domComplete`. Unlike `domComplete`, which fires when the whole page completes loading including all the resources, `domInteractive` comes earlier when the necessary information needed to complete the DOM tree is ready, which is usually just the document and synchronous scripts, but without any styling information or media resources. It is expected that the slowdown of WT compared to unmodified Firefox is less significant, since the total amount of data and requests are both lower, which can also been seen in the chart, while the use of TT still provides some speed up. WT increases the website access time by 50% or more in 12% of cases. On the other hand, in the case of TT of 1,000 milliseconds, it occurs in only 2% of websites.

Looking into the data, we specifically noticed the major improvement on some popular content-oriented websites. For example, on `cnn.com` and `nytimes.com`, as well as `sina.com.cn`, the loading times are reduced by a maximum

 ISBN: 978-1-61208-770-2

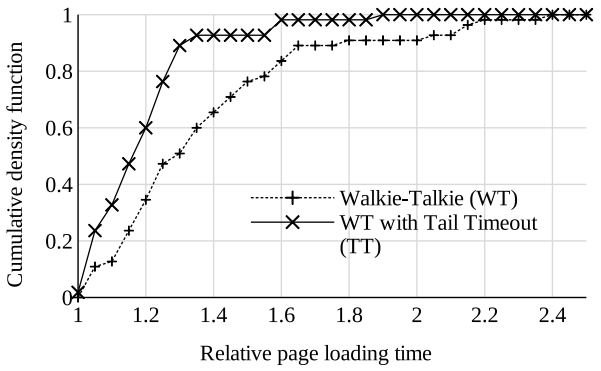


Figure 1. Page Loading Time Comparison (`domComplete` event used; page loading time values in X-axis are normalized to unmodified Firefox, i.e., loading time of 1 in X-axis means an equal loading time of the same website in unmodified Firefox, Y-axis shows the ratio of websites which finished loading by the given loading time in each scenario; timeout = 1,000 ms)

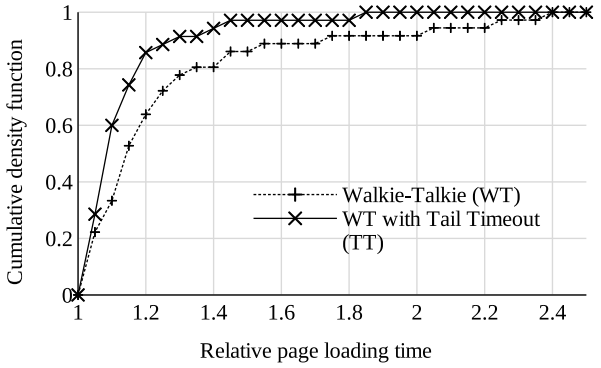


Figure 2. Page Loading Time Comparison (`domInteractive` event used; normalized to unmodified Firefox; timeout = 1,000 ms)

of 30~50%, as shown in Table I. On `nytimes.com`, the webpage could not finish loading within 300 seconds when using WT and we terminated page loading at 300 seconds. Also as expected, within the range of tail timeout values we tested, smaller value reduces page loading time.

TABLE I. LOADING TIME (`domComplete` EVENT, SECONDS) OF CERTAIN WEBSITES AT SOME DIFFERENT TAIL TIMEOUTS

Website	Unmodified	WT	TT 1,000	TT 2,000	TT 5,000
`cnn.com`	46.758	123.441	59.231	67.160	121.596
`nytimes.com`	26.262	> 300	30.335	37.370	41.306
`sina.com.cn`	35.094	75.404	42.075	46.530	66.359

B. Waterfall Charts

To further identify the source of the delay, we captured the timing information during page loading. For each request, we capture the following four timestamps during page loading.

- Requested time: the time when the browser initially wants to process the request
- Beginning time: the time when the request is released by the add-on from the "blocked" status
- Ending time: the time when the request is taken out of the "on-the-way" pool, or responded time, whichever earlier
- Responded time: the time when the response of the request is received, or the request is otherwise finished

The timing information is recorded into logging data in the WT/TT implementation add-on to the Firefox browser when requests are processed by the add-on. Between the beginning time and ending time, requests are made by the browser with its original scheduling mechanisms.

We tested `nytimes.com` and `cnn.com` for page loading, and plotted the timing data for `cnn.com` in Figure 3 and that for `nytimes.com` in Figure 4. For the sake of brevity, only some requests may be shown for multiple requests in the same half-duplex burst with similar timing characteristics, and only first few bursts are shown.

We can notice that in Figures 3a and 4a, when no TT is applied, certain requests, such as requests 7, 15 and 16 in `cnn.com` and requests 6 and 7 in `nytimes.com`, took a long time to finish, while other requests in the same burst finished within one or two seconds. The slow requests held all following requests so they cannot be sent, causing a significant delay. Recall that in Table I, smaller tail timeout values result in shorter overall page loading times. We see it here that when a long loading time happens, only a few "bottleneck" requests are blocking the whole browsing session. By using a smaller tail timeout value, the maximum delay that can be caused by a request is limited, and a shorter overall time can be expected.

We further tracked down the particular request 7 on `cnn.com`. By looking at the page source code of `cnn.com`, we noticed that request 7 was introduced in the HTML document with a `<script>` tag with the `async` attribute set. Request 7 points to https://native.sharethrough.com/assets/sfp-creative-hub-listener.js and Sharethrough is an advertisement platform. As we pointed out before, the resource fetched by the request is exactly for advertisements, and should have a relatively lower priority.

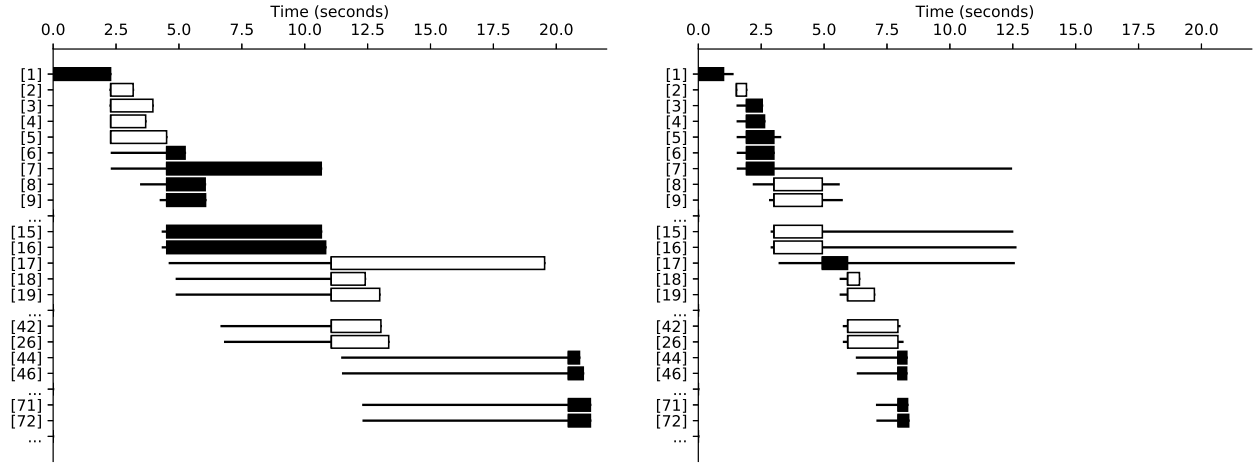
Web developers of `cnn.com` correctly specified the lower priority for these resources by using the `async` attribute, then by design, the browser tends to load these resources at a later time, and gives priority to other resources. However with WT's half-duplex communication, no other requests can be sent. This situation can be resolved only when the browser finishes the low priority request, probably after an internal timeout, or with the designed TT in the add-on, when the timeout passed, which can be earlier than the internal timeout, saving some page loading time. Similar results can be concluded from several other requests in the same shape.

In addition to the lower priority requests, there can also be requests to a slow remote server that take a relatively long time. This scenario was not observed in our experiment, but we can predict that such scenarios can happen and cause a similar delay.

C. Defense Effectiveness

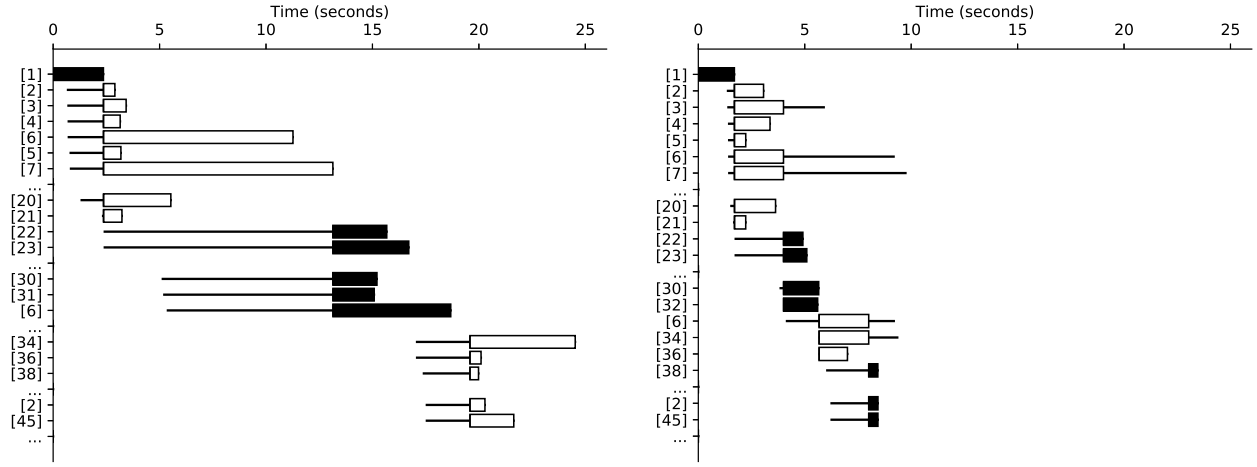
We also want to verify that TT does not increase the fingerprintability on the traffic. We follow the approach taken to evaluate the original WT, to run k-NN attacks on the cell sequences using WT and TT, with padding (actual working scenario) and without padding (as contrast), and measure the

 ISBN: 978-1-61208-770-2



(a) Original WT (b) TT of 1 second

Figure 3. Loading `cnn.com` main page with original WT and with TT of 1 second. See [32] for URLs. (The beginning and the end of a slim line denote the Requested time and Responded time, respectively. A black or white bar shows the beginning and end times of a request during the page loading process, respectively. Neighboring requests with bars in the same color (e.g., [6]∼[16]) are in the same burst. In (b), the burst containing requests [18]∼[26] is not held by requests [7], [15] and [16] for long time anymore.)



(a) Original WT (b) TT of 2 second

Figure 4. Loading `nytimes.com` main page with original WT and with TT of 2 second. See [32] for URLs. (In (b), The burst containing requests [22]∼[30] is not held by requests [6] and [7] for long time anymore.)

true positive rate (TPR), false positive rate (FPR), true negative rate (TNR) and false negative rate (FNR).

In the evaluation, we count the number of true positives (TP), false positives (FP), true negatives (TN), false negatives (FN) given by the attack on the dataset, as well as total number of actual positives (P) and actual negatives (N). We use TPR = TP / P, FPR = FP / N, TNR = TN / N, FNR = FN / P. For attempted *k*-NN attacks against data sets, as described previously, we plot the TPR in Figure 5 and the TNR in Figure 6. Note that tail timeout value is applicable to TT (without and with padding) only.

In Figure 5, it is observed that TT shows a comparable defense to WT while padding improves the defense by 15∼21% in both WT and TT. In Figure 6, it is observed TT (padding) shows 3∼5% better defense than WT (padding) while they perform worse than the Unmodified Firefox without padding, which means to block more websites unnecessarily.

Considering TT is an extension to WT, and WT has already been evaluated with other defenses and shown a decent performance, we believe that TT is providing a good website fingerprinting defense comparable to WT.

VI. Conclusion and Future Work

In this paper, we presented a new delay-conscious defense mechanism based on WT against website fingerprinting attacks. We demonstrated that WT could significantly increase the page loading time for many popular websites, such as `cnn.com` and `nytimes.com` in practice. By analyzing the websites experiencing inflated page loading time under WT, we found that the contents marked with low priority, such as online Ads

 ISBN: 978-1-61208-770-2

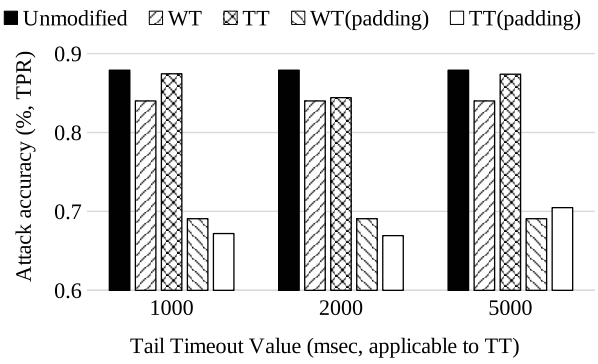


Figure 5. Attack accuracy - True Positive Rate (TPR)

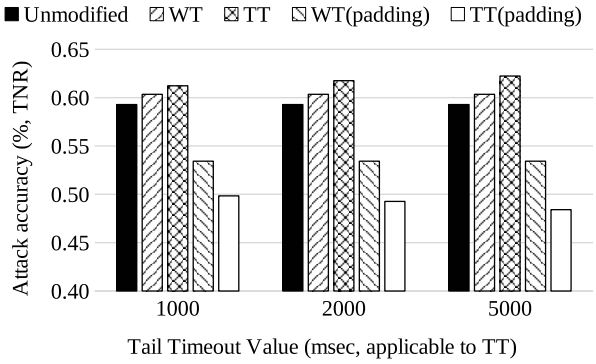


Figure 6. Attack accuracy - True Negative Rate (TNR)

could incur such delays in practice. To address the problem of increased page loading time, we proposed a TT defense as an extension of WT. Our experimental results indicate that the TT extension can significantly reduce the page loading time while keeping similar defense performance in terms of true positive and true negative ratios achieved by the original WT.

In our research, we arbitrarily selected the tail timeout values to test (1,000, 2,000, etc. milliseconds). We demonstrated that TT works with these values, while we have not identified an approach to determine the optimum values. At the same time, identifying the optimum values is currently not our goal. We are leaving this as future work, as well as investigating the trade-off between the defense effectiveness and user experience.

In the future, we also plan to try other attack models, such as naïve Bayes or the more trending deep learning based classifiers [28], [33], or on larger datasets, to evaluate the performance of TT. In addition, we plan to apply TT extension to other defense mechanisms, such as LLaMa [34] that potentially increase page load times due to the same head-of-line issue.

REFERENCES

[1] R. Deibert, J. Palfrey, R. Rohozinski, J. Zittrain, and J. G. Stein, Measuring Global Internet Filtering. MITP, 2008.

[2] C. S. Leberknight, M. Chiang, H. V. Poor, and F. Wong, "A taxonomy of internet censorship and anti-censorship," in Fifth International Conference on Fun with Algorithms, 2010.

[3] P. Winter and S. Lindskog, How the Great Firewall of China is blocking Tor. USENIX-The Advanced Computing Systems Association, 2012.

[4] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your state is not mine: a closer look at evading stateful internet censorship," in Proceedings of the 2017 Internet Measurement Conference. ACM, 2017, pp. 114–127.

[5] "Bbc news launches 'dark web' tor mirror," BBC News, Oct 2019. [Online]. Available: https://www.bbc.com/news/technology-50150981

[6] J.-P. Verkamp and M. Gupta, "Inferring mechanics of web censorship around the world." in FOCI, 2012.

[7] R. Clayton, S. J. Murdoch, and R. N. Watson, "Ignoring the great firewall of china," in International Workshop on Privacy Enhancing Technologies. Springer, 2006, pp. 20–35.

[8] S. Aryan, H. Aryan, and J. A. Halderman, "Internet censorship in iran: A first look," in Presented as part of the 3rd USENIX Workshop on Free and Open Communications on the Internet, 2013.

[9] K. Ko, H. Lee, and S. Jang, "The internet dilemma and control policy: political and economic implications of the internet in north korea," The Korean journal of defense analysis, vol. 21, no. 3, 2009, pp. 279–295.

[10] J. Holowczak and A. Houmansadr, "Cachebrowser: Bypassing chinese censorship without proxies using cached content," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015, pp. 70–83.

[11] C. Brubaker, A. Houmansadr, and V. Shmatikov, "Cloudtransport: Using cloud storage for censorship-resistant networking," in International Symposium on Privacy Enhancing Technologies Symposium. Springer, 2014, pp. 1–20.

[12] 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. ACM, 2011, pp. 103–114.

[13] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in 2012 IEEE symposium on security and privacy. IEEE, 2012, pp. 332–346.

[14] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," International Journal of Network Management, vol. 25, no. 5, 2015, pp. 355–374.

[15] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in NDSS, 2016.

[16] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in 23rd USENIX Security Symposium (USENIX Security 14), 2014, pp. 143–157.

[17] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall, "Detecting intentional packet drops on the internet via tcp/ip side channels," in International Conference on Passive and Active Network Measurement. Springer, 2014, pp. 109–118.

[18] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 1375–1390.

[19] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[20] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in European Symposium on Research in Computer Security. Springer, 2016, pp. 27–46.

[21] 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. ACM, 2013, pp. 201–212.

[22] ——, "On realistically attacking tor with website fingerprinting," Proceedings on Privacy Enhancing Technologies, vol. 2016, no. 4, 2016, pp. 21–36.

[23] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in European Symposium on Research in Computer Security. Springer, 2006, pp. 18–33.

[24] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis." in NDSS, vol. 9. Citeseer, 2009.

 ISBN: 978-1-61208-770-2

[25] 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. ACM, 2014, pp. 227–238.

[26] 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. ACM, 2012, pp. 605–616.

[27] [Online]. Available: http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[28] S. E. Oh, S. Sunkam, and N. Hopper, "p1-fp: Extraction, classification, and prediction of website fingerprints with deep learning," Proceedings on Privacy Enhancing Technologies, vol. 2019, no. 3, 2019, pp. 191–209.

[29] "Selenium - web browser automation." [Online]. Available: https://www.seleniumhq.org

[30] "mozilla/geckodriver: Webdriver for firefox." [Online]. Available: https://github.com/mozilla/geckodriver/releases

[31] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in 25th USENIX Security Symposium (USENIX Security 16), 2016, pp. 1187–1203.

[32] J. Liang, C. Yu, and K. Suh, "Delay-conscious defense against fingerprinting attacks," Cleveland State University, Tech. Rep., 2020.

[33] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2018, pp. 1928–1943.

[34] G. Cherubin, J. Hayes, and M. Juarez, "Website fingerprinting defenses at the application layer," Proceedings on Privacy Enhancing Technologies, vol. 2017, no. 2, 2017, pp. 186–203.

 ISBN: 978-1-61208-770-2