Link Prefetching: A Defense Against Website Fingerprinting on Tor [*]

Vaibhav Sharma
vaibhav@umn.edu

Taejoon Byun
taejoon@umn.edu

Elaheh Ghassabani
ghass013@umn.edu

Se Eun Oh
seoh@umn.edu

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55454

ABSTRACT

We present a novel defense mechanism to protect against website fingerprinting attacks. Link prefetching has been used by website developers to improve user browsing experience by anticipating what web content the user is likely to browse to next and caching it in the browser in advance. Link prefetching causes extra upstream and downstream packets to be introduced in the network traffic for a webpage thereby altering its fingerprint. We evaluate the effect of link prefetching on classifier accuracy by alternating the prefetching browser setting between training and test sets. Our results strongly suggest that link prefetching can be used to alter webpage network fingerprints regardless of the feature sets being used by the classifiers. We then evaluate the effect of varying the number of prefetching requests and size of prefetched responses but find the Tor browser dishonors large number of prefetch requests.

Keywords

Website fingerprinting, anonymity, encrypted traffic, Tor

1. INTRODUCTION

Tor is an overlay network that provides communication anonymity and security using onion routing. An onion network encapsulates a message in layers of encryption and reroutes the messages throughout volunteering nodes, so as to hide not only the contents of the message but also the location of the origin. Although the main goal of Tor is to protect its users against *traffic analysis*, which is a form of Internet surveillance, it is still vulnerable to a targeted attack called website fingerprinting (WF). An adversary of WF attempts to infer which website a victim is visiting, and this can be a serious threat in privacy-sensitive use cases.

Website fingerprinting gained attention since the early work by Hermann et al. [18], when the authors achieved 3% of success rate using a Naive Bayes classifier. Although the success rate is too low to be practical, following works using a more sophisticated classification techniques achieved a much higher accuracy around 90% [25, 30, 10], showing that website fingerprinting can be a serious threat against the users of the anonymity network. Especially, Cai et al. [10] showed that their attack can successfully defeat existing well-known defense mechanisms such as HTTPOS [22].

Several defenses were also suggested accordingly to defend users against fingerprinting attacks. Those defense mechanisms are usually developed at the IP/TCP level by changing the pattern of the traffic whether by splitting, morphing, injecting, or padding packets [16, 31, 22, 26]. Many of the defense mechanisms, however, had been neutralized by following attacks [10]. Moreover, all of the defenses involve making changes either to the Tor protocol itself or to the browser. Using link prefetching as a defense mechanism has the distinct advantage of not requiring any changes to be made to the protocol implementation or the browser. It provides more browser-side control which allows this defense to be deployed as a browser extension.

In this work, we propose a novel technique to provide defense against WF attacks using link prefetching. Link prefetching is a HTML5 feature that gives hints to a browser the list of resources and pages to prefetch, when it can be sure about which page a user is likely to request next. When the browser encounters prefetch tags, it silently loads and caches the specified resources after the page is fully loaded. Since prefetching generates extra outgoing and incoming packets, it affects the features of website fingerprint and thus undermines the accuracy of the classifiers used in WF attack (Section 4).

We validated our approach through a series of experiments for the two research questions – (1) *does prefetching itself provide an extra degree of defense?* (2) *can prefetching be uses as a browser-side mechanism?*

The major contribution of our work is twofolds:

- We present a novel approach of using link prefetching as a defense mechanism against WF attacks.

The remaining sections of this report is organized as fol-

[*]This report is submitted as a partial fulfillment of *CSCI5271: Introduction to Security* course.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

lows: Section 2 provides a brief background about Tor anonymity network and website fingerprinting, and Section 3 mentions related works on website fingerprinting attacks, defenses and criticisms. We suggest link prefetching as a defense mechanism in Section 4 followed by experimental evaluation (Section 5). Section 6 discusses about the feasibility of the proposed defense based on experimental result, and finally conclude our report on Section 7.

2. WEBSITE FINGERPRINTING ON TOR

This section provides a background on The Onion Router (Tor) and the website fingerprinting attack which can neutralize the anonymity that Tor provides.

2.1 Tor

Tor is one of the most popular anonymizing networks, which distributes user's communications over several places on the Internet. The distribution helps user's final destination not to be linked to a single point. Tor can be considered as an overlay network on top of the internet which directs TCP streams to different proxies. In this network, packets take a random path through several relays, instead of taking a direct route. These layers cover the user path such that it will not be possible to tell where the data comes from, or where it is going. Such a pathway in Tor is called a private network pathway. The Tor client application is responsible of creating this private pathway by incrementally building a circuit of encrypted connections. The circuit is constructed hop by hop, which means each individual relay is only aware of the previous and next hop. That is to say, individual relays never can identify the complete path. In order to build circuits, Tor makes use of authenticated DH key exchange encrypted with symmetric keys [6, 12]. Figure 1 demonstrates how Tor works.

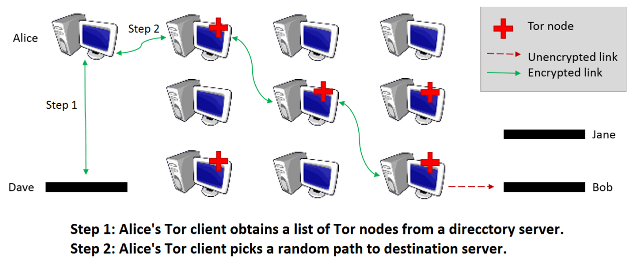

Alice
Step 2
Step 1
Dave
Tor node
Unencrypted link
Encrypted link
Jane
Bob
Step 1: Alice's Tor client obtains a list of Tor nodes from a direcctory server.
Step 2: Alice's Tor client picks a random path to destination server.

Figure 1: The Onion Router (Tor) [6]

The design described above aims to support a variety of anonymity goals. However, if an attacker can see both ends of the private pathway, Tor will fail. For example, suppose the attacker watches the Tor relay to which the user enters the network, and also watches the website she visits; in this case, the attacker can correlate volume and timing information on the two sides. To cope with this problem, Tor came up with the idea of *entry guards*, where each Tor client randomly selects a few relays as entry points for her first hop. If those relays are not observed by an attacker, the user is secure. But, needless to say, there is always a probability of losing anonymity [6].

2.2 Website Fingerprinting

Tor does not provide protection against all anonymity problems. In addition, the website fingerprinting (WF) attack is still able to bypass its privacy mechanism. In the case of Tor, WF attack would take place between the user and the Guard node, or at the Guard node itself. In general, attack models can be divided into two categories: ”closed world” and ”open world” scenarios. This section describes these attack models.

In the closed world model, the classifier can successfully recognize only web pages that it has already been trained on. Therefore, in this case, we deal with a small set of censored web pages. The open world scenario is when the adversary is capable of recognizing censored pages that might have not seen before [2].

Figure 2 illustrates the attack model of website fingerprinting. We also assume the WF adversary as local and passive, which means although the adversary is able to eavesdrop on user's traffic, he cannot add, modify, or drop it. It should also be noted that we assume the adversary is not be able to decrypt the traffic. Otherwise, he would not bother lunching a WF attack [21].

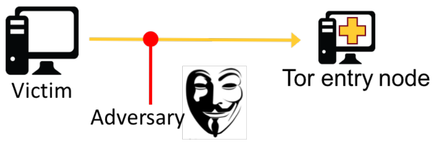


Figure 2: Website Fingerprinting Attack Model [21]

Juarez et al. in [21] divided the WF attack into two different types based on the number of the users it targets. In the first category, called *targeted*, the adversary aims to obtain the browsing activity of a specific victim. In this case, since the adversary is able to train the classifier using the information of the victim, the attack is more successful. The adversary may even have enough information and background knowledge about the user and her behavior. So, it could be possible to guess and emulate the user configuration. Therefore, the accuracy of the classifier would increase. The second type of the attack is called *non-targeted* or *dragnet surveillance*, where a set of users, instead of one, are targeted by the attacker. In this case, ISP, entry guard, or Internet exchanges serves the attacker. In other words, the attacker is able to intercept the traffic of many users by using such servers. A classifier in the non-targeted attack is trained based on a specific setting and used for all communications.

3. RELATED WORK

This section surveys attacks/ defenses on/against anonymizing systems, especially Tor.

3.1 Fingerprinting Attacks

The idea of inferring meaningful information based on sniffing and analyzing encrypted SSL packets, which is called deep packet inspection (DPI), was introduced in 1996 [29]. Later in 2002, Hintz presented the first website fingerprinting attack on an encrypted web proxy. The attack shows how DPI is powerful enough to identify a specific web site that the user is surfing [20]. Since then, a lot of research has been conducted in this area trying either to propose a more realistic WF attack, or a more effective defense mechanism against WF.

Herrmann et al. [19] pointed out that widely used anonymous networks such as Tor still failed to defend against WF attack. They achieved only a 3% success rate for 775 pages using Multinomial Naive Bayes classifier focusing only on packet sizes. The WF attacks on Tor suggested by Penchenko et al. [25] showed more reasonable accuracy with more sophisticated machine learning techniques and diverse features. Later on, based on [25]'s attack model, many works such as [30, 10] tried to build more powerful and realistic attacks that improved the accuracy rate up to around 90% in closed word setting. They did thorough inverstigation on extracting powerful features such as Tor cells and improving classification method. Especially, Cai et al. [10] had shown that their novel WF attack successfully defeated existing well-known defense mechanisms such as HTTPOS [22]. However, even though existing works have successfully improved the effectiveness of WF attacks on Tor, underlying assumptions for experiments significantly simplified real world setting. As Juarez et al. [21] pointed out that existing WF attacks are unrealistic with assumptions such as the client setting, we decided to focus on building more robust and practical defense mechanisim instead of developing advanced attack models to improve the accuracy rate.

3.2 Defense Mechanisms

Defense mechanisms are usually developed at the IP/TCP level by changing the pattern of traffic. For example, they splited packets into multiple packets, injected spurious packets into the traffic, or involved padding packets to prevent from leaking features of traffic. A study performed in [16] suggested to transmit packets at random intervals as a defense mechanism against traffic analysis. Another defense technique proposed in [31], called traffic morphing, uses some tricks to change a traffic pattern to disguise new or existing other traffic. However, since their approach still leaks the information such as packet order, attacks that work without packet size information can easily defeat their mechanism.

In [22], Luo et al. proposed novel HTTP/TCP level defense mechanisms against some analysis attacks by changing window sizes and the order of packets in the TCP stream. Besides, at the HTTP level, they tried to inject some extra data into HTTP GET headers, generate some irrelevant HTTP request, and re-order requests. The Tor community also introduced "randomized pipelining" [26] to defend the WF, by having the browser load the web content in a random order. Despite such techniques, the proposed attack in [10] managed to successfully recognize target web pages with the accuracy of 87%. Howerver, all exisitng defenses have not been neither efficient nor effective for fancy WF attacks. In contrast, our work more focuses on utilizing existing HTML systax that is link prefetching shown in Section 4, which subsequently does not accomodate extra cost.

3.3 Criticism

In the literature, some work always debate over the practical feasibility of the WF attack. This section tries to summarize issues questioning the practicality of WF.

There are many different factors that play a role in the success of WF. The main criticism is that academic papers usually oversimplify WF attack models by making some unrealistic assumptions over users' browsing habits, training and testing traces, even the version of Tor used for testing/ training. In [21], it is discussed that the studies performed in [10, 19, 25, 30, 28] simplify the problem and overestimate the adversary's capabilities.

Another criticism is that all works lunch attacks on individual pages, instead of overall websites. So, although the attack is called website fingerprinting, it is actually about webpage fingerprinting. In addition, there are some main factors, including the the hypothesis state space and the size of the instance space, that affect the accuracy of a classifier. Even, the number of training samples provided to the classifier and false positive rates matters. Since reliable feature information is constant, with the increase in the number of classification categories, the classifier eventually runs out of descriptive feature information, which causes either true positive accuracy goes down or the false positive rate goes up. A Tor blog article [2] discusses that the effects of such factors are quite observable in the papers with a sufficient world size.

Although more attention should be paid to the scenarios by which attacks are evaluated, we should not dismiss WF as a threat. However, [2] argues that, due to theoretical and practical issues, realistic WF attacks are hard to lunch on Tor. Therefore, it claims that even simple defenses could protect Tor users against WF. It seems that the Tor community believes that defense mechanisms do not need to be very complicated to be effective.

To the best of our knowledge, none of the defense techniques has investigated the effect of link pre-fecting on WF. In the next section, we will describe the concept of link prefetching. Since most of the defenses are applied at the Tor network, they are required to be acceptable by the Tor community. However, we aim to propose a defense technique which can also be used by the website owners.

3.4 Web Prefetching

A problem with the defense techniques that use packet padding or traffic morphing is that they impose extra overhead to the network, in terms of both delay and band width. This issue often makes them unpractical and inefficient for anonymizing networks. In [32], authors proposed a defense that mitigates this problem. The idea proposed there is much closer to our work; the original approach is that, instead of inserting dummy packets into the traffic, adding the packets that are predicted to be needed by the user in a short time. Of course in that way also some extra packets are added to the traffic changing the fingerprint. In addition, [32] provides a mathematical model that gives the user some hints to estimate the cost of using the defense. To predict which packets must be injected into the traffic, [32] makes use of link pre-fetching and web caching. The difference between their work and our approach is that they are still trying just to inject extra packets to the traffic, while our final goal is more than that. First, we intend to have some experimental results to see how pre-fetching affects the WF attack. In addition, we propose a new defensive mechanism based on changing the amount of pre-fetched data from time to time. In other words, [32] only injects pre-fetched data to change the traffic pattern. Therefore, it could be possible to train a classifier for the pre-fetched enabled streams as well. In contrast, we present a set of experiments that show how we can fight the WF attack by altering the amount of pre-fetched data, which will be discussed in more detail later.

Web prefetching has been widely used to reduce the latency for users. Fan et al.[13] introduced the first prefetching between caching proxies and clients. Chen et al. [11] pre-fetched web pages that will be visited near future. In addition, there have been several works using prefetching over anonymous network. Nguyen et al. [] prefetched web components of web page, which users are visiting, to solve the delay performance of Tor. They set two proxies, one sits between the client and browser, and the other sits between the exit relay and the website. Even though they successfully improved tor performance, the extra setting of proxies and the communication over them are not good in terms of security since we need to trust them. (e.g., proxies are compromised by adversaries). In contrast, we do not enforce any additional setting on tor network as well as client-side to hinder from such additional security risks.

4. LINK PREFETCHING AS A DEFENSE

A classifier used in a website fingerprinting attack can distinguish the differnce between two classes of packets when the difference is consisntent between them [9]. Thus, intuitively, a classifier would work the best if the difference in the same class is minimal and the difference among heterogeneous classes is high at the same time. However, when variance among fingerprints in the same class is high, a classifier would not be able to characterize it clearly or at least the detection rate using the classifier would be low. This is the basic idea of using prefetching as a defense mechanism; our conjecture is that we can use prefetching to manipulate the number and size of incoming and outgoing packets in order to increase the variance among packets in the same class.

This section explains the concept of link prefetching, discusses its effect on website fingerprints and argue its possible usage as a defense mechanism.

4.1 Link Prefetching

Link prefetching is a HTML syntax that gives the web browser hints about which page the user is most likely to visit in a near future [14, 15]. The pages and resources to pre-fetch are specified in the web page so that the web browser can silently load them (or pre-fetch them) after an idle time. Since the pre-fetch happens only after the page is fully loaded, it does not sacrifice the loading time of the requested web page. Moreover, it can save the loading time for the pre-fetched pages and thus improve the user experience by caching the *future* contents. It was first suggested by Mozilla Foundation in 2003 and supported by most modern browsers nowadays.

The resources to prefetch can be simply specified in *HTML* using a `link` tag [24]. For example, a `link` tag `<link rel="prefetch" href="/page2.html">` tells the browser to prefetch a HTML file named `page2.html`. Resources other than a *HTML* web page can also be pre-fetched similarly using the same syntax. There are some variations for different types of prefetching called DNS prefetching also, (specified as `<link rel="dns-prefetch" ..>`) which is supported by *Mozilla Firefox* and *Google Chrome*. Another form of expression `<link rel="prerender" ..>` also does the same job as `prefetch` in *Google Chrome* and *Microsoft Internet Explorer*.

Figure 3 illustrates how prefetching actually works in a browser (*Google Chrome*). This page is set up arbitrarily by the authors to demonstrate link prefetching, and contains a link pre-fetch tag that specifies a big image (the image on the left side labeled as *prefetch*). When the prefetching is off, this image shall be requested only when a user hovers his mouse cursor on it. However, it can be seen on the network timeline (on the right bottom) that the image is pre-fetched right after loading the page, not when the user actually requested it. This is indicated by a long blue bar on the second row for the file named "*Very-high...*", and it is long because the size of the file is relatively big (*3.5 MB*) that it took a longer time to download. Please also note that the time it took for loading the image when the user actually requested (by hovering his mouse on it) was very short, because the image had already been pre-fetched that the browser merely loaded it from the cache (as shown in the *size* column on the fourth row).

4.2 An Example of Packet Sequences

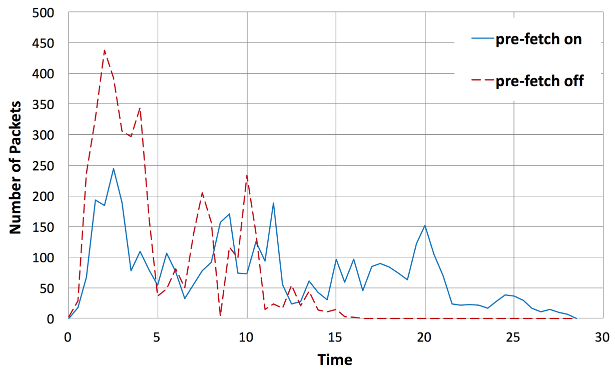
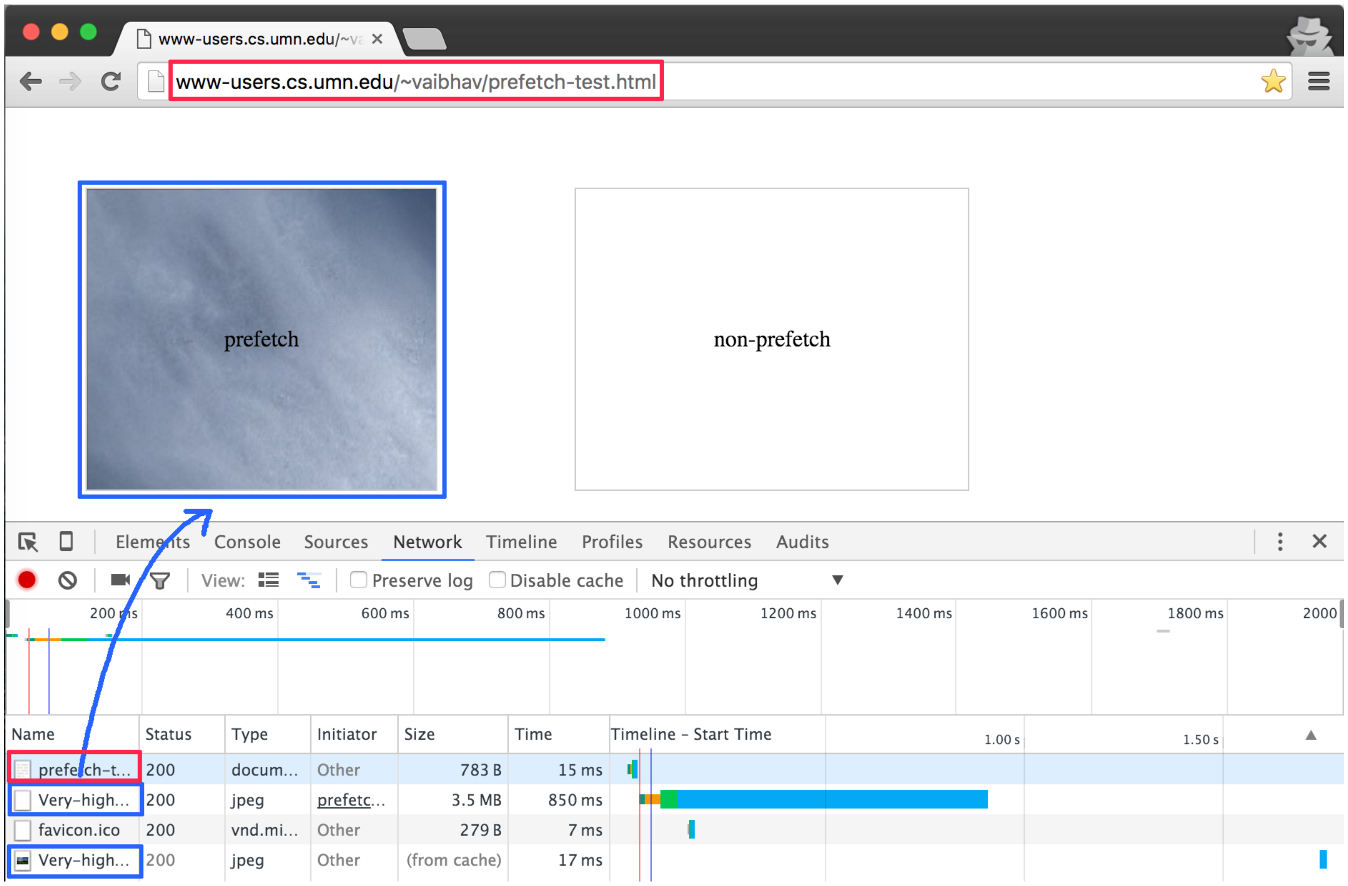


Figure 4: A website fingerprint for pre-fetch on/off cases

Figure 4 illustrates a website fingerprint in terms of inter-packet timing, where the *x-axis* corresponds to time in seconds and the *y-axis* represents the number of packets captured during a certain time interval (*500 ms*). The solid line depicts the packets captured while prefetching was enabled, and the dashed line depicts when the pre-fetch was disabled in the browser settings. Both the cases are captured for the same web page, which is the first page of `wired.com`. However, the number of incoming and outgoing packets combined are different for each case (4055 for the *off* case while 4218 for the *on* case), because the prefetch-on case obviously requests more resources for caching purpose. The elapsed time for loading the whole page was also different, but this variance is mainly due to the difference in network condition as shown in the figure that the packets for the *on* case are more scattered throughout time compared to the pre-fetch-off case. When we ignore the speed difference, we can see that the four peaks of the two lines are roughly the same that if we capture the packets multiple times for the same website, we would be able to characterize how the *fingerprint* of a specific website looks like. Please note also that inter-packet timing is only one of many features for characterizing website fingerprint.

4.3 Fingerprint Features

Since all the traffic on Tor is encrypted, fingerprinting



Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
prefetch-t...	200	docum...	Other	783 B	15 ms	
Very-high...	200	jpeg	prefetc...	3.5 MB	850 ms	
favicon.ico	200	vnd.mi...	Other	279 B	7 ms	
Very-high...	200	jpeg	Other	(from cache)	17 ms	

Figure 3: Network timeline showing pre-fetch

attacks blindly analyze a sequence of packets without knowing its contents, and extract *features* which characterizes a fingerprint. The attacker then trains his/her classifier on multiple packet sequences from a set of website of interest, for a set of features that attacker determines to use. In this subsection, we summarize the definition of the four most popular features defined by Cai et al. [9] to discuss the effect of prefetching on this features in the following subsection.

Packet Sequence: A packet sequence P can be written as $P = \langle(t_1, l_1), (t_2, l_2), ..., (t_n, l_n)\rangle$ [9], where t_i is the interpacket time between packets i and $i-1$, and l_i is the byte length of the i-th packet. We write P_t and P_l as the sequences of only the interpacket times and lengths, respectively.

Unique Packet Lengths: Unique packet length is differnet when there exists a unique packet length L in a sequence of packet lengths P_l such that it belongs to a sequence while it does not to the other. In other words:

$$(\exists L \in P_l | L \notin P'_l) \vee (\exists L \in P'_l | L \notin P_l) \qquad (1)$$

Packet Length Frequency: When $n_L(P_l)$ is the number of times packet length L appears in P_l,

$$\exists L | n_L(P_l) \neq n_L(P'_l) \wedge n_L(P_l) > 0 \wedge n_L(P'_l) > 0 \qquad (2)$$

In other words, P and P' have different packet length frequencies iff there exists a packet with length L that appears different times in the two sequences.

Packet Ordering: When M_l is the multiset of packet lengths in P_l without ordering, two packet sequences P and P' have different packet ordering iff:

$$M_l = M'_l \wedge P_l \neq P'_l \qquad (3)$$

Interpacket Timing: Interpacket timing is different if the packets of the same index from the two different sequences P and P' have different interpacket timing:

$$\exists i, 1 \leq i \leq min(|P|, |P'|) : (P_t)_i \neq (P'_t)_i \qquad (4)$$

Based on these features, Cai et al. [9] claimed that if $P \neq P'$, then one of the four features shall differ.

4.4 The Effect of Prefetching on the Features

Based on the features we summarized in Section 4.3, this subsection explain how prefetching can affect the four features. We will compare the two packet sequences P and P' for the same webpage, where P is the packet sequence without prefetching and P' is the packet sequence where prefetching was enabled (and where there is more than one pre-fetch request).

Packet Sequence: By the definition of link prefetching [15], pre-fetch is requested only after a webpage is fully loaded and after an idle time. Thus, for a packet sequence P where prefetching was disabled, another packet sequence P' with prefetching enabled would not differ except the prefix of P'. In other words, P' is a concatenation of two sequences

P and Q such that $P' = P \| Q$, where Q is the incoming and outgoing packets for pre-fetch request and download such that:

$$Q = \langle (t_{n+1}, l_{n+1}), (t_{n+2}, l_{n+2}), ..., (t_{n+m}, l_{n+m}) \rangle, m > 1 \quad (5)$$

Unique Packet Lengths: In order for the *unique packet lengths* to be different between P and P', it is sufficient to have one packet with a different length in P' that do not appear in P, or vice versa. When there are tailing packets Q for P', the likelihood is high that there is one or more packets in Q that do not exist in P. Although in some cases the multiset of packet lengths M_l' can be a subset of M_l, it is not likely considering that the prefetching resources are different ones from the resources that are already fetched – the designer of a website would not want to pre-fetch the same resource that's already been downloaded. Thus, unique packet lengths will be different between P and P'.

Packet Length Frequency: This feature differs when there exists a packet length L such that the frequencies differ between P and P', while appearing at least once in both P and P'. This is highly likely to be different considering that most the packets are transmitted with the size of MTU (maximum transmission unit), which is typically 1500 bytes. When the requested resource is bigger than 1500 bytes, which is true in most cases, the number of packts with length 1500 will be larger in P' than in P ($n_{1500}(P') > n_{1500}(P)$) and thus yield false for this feature.

Packet Ordering: P and P' obviously do not have the same packet ordering because $P_l \neq (P\|Q)_l$.

Interpacket Timing: This feature would not be affected because of the clause $min(|P|, |P'|)$ would always yield $|P|$, and $(P_t)_i$ where i is between 1 and $|P|$ would not be affected by Q.

In summary, at least one out of features is affected by link prefetching. Furthermore, if our assumptions about the prefetching packets are valid, at most three features are affected by link prefetching, which is sufficient for a classifier to put P' in a different class to P.

We show the effect of prefetching in various settings throughout a series of experiments.

5. EXPERIMENTAL EVALUATION

We present the design of two sets of experiments for evaluating the effect of link prefetching on website fingerprinting and the results from these experiments in the following subsections.

5.1 Research Question

We formulate the research questions of this project as follows:
RQ1: Does prefetching itself provide an extra degree of defense?
RQ2: Can prefetching be used as a browser-side defense mechanism?

We summarize our plans to tackle *RQ1* in Table 1. The rows represent the link prefetching setting used during the training stage by the attacker and the columns represent the link prefetching setting used during the testing stage by the attacker.

Table 1: Experiment design to answer RQ1

train\test	prefetch on	prefetch off
prefetch on	(1)	(2)
prefetch off	(3)	(4)

1. We speculate that prefetching itself might provide extra defense because of the extra packets. As per the Link Prefetching FAQ [14], Firefox prefetches web content requested by the webpage at browser idle time where browser idle time is defined as the time when the web page has finished loading. If this is implemented similarly in the Tor browser, prefetched packets requests and responses should flow into the browser after it has finished downloading resources required by the webpage. It can also be the case however, prefetching websites are more vulnerable to fingerprinting because of the extra prefetch upstream and downstream packets that creates a distinct suffix.

2. This case is unlikely to present itself in reality since the prefetching option is turned on by default in the latest version of the Tor browser as of this writing. We assume that victims will more likely be using Tor under the default setting.

3. This case is what we are most curious about, whether a victim can confuse an attacker by simply turning the prefetching option off in his/her browser. This scenario, therefore has the potential to confuse the attacker.

4. This case simulates a situation where a victim is loading websites that does not prefetch any resource. This can be thought of the most normal scenario as of now and can be used for checking the strength of our fingerprinting classifier.

For *RQ2*, we found that the performance of link prefetching as a browser-side defense mechanism depends primarily on two parameters.

1. number of prefetching requests made by the browser

2. total size of prefetched responses received by the browser

We vary the values of both these parameters and present our results in Section 5.5.3.

5.2 Experimental Design

We have completed two sets of experiments corresponding to *RQ1* and *RQ2*. Our first experiment consists of checking the accuracy of known classifiers using coarse features for the four scenarios described above. Our second experiment consists of checking the effect of randomly varying the number of prefetching requests and size of prefetched responses on the classifier accuracy.

5.3 Data Collection for RQ1 and RQ2

We created a web scraper using a Python framework called Scrapy [5] and ran on the top 10,000 most popular websites of the top 1 million as listed by Alexa [3]. On scraping through the HTML content of the top 6000 websites for link prefetching keywords such as *prerender,next,prefetch* we found 60 websites using link prefetching for caching web content in the browser. We then configured a virtual machine

to run a Tor browser crawler [1] to visit each of these 60 websites 16 times and captured all packets seen. This was done once with the *prefetch-next* option left to its default value and repeated with this option turned off. By doing this, we captured packets for link prefetching as done currently in the real world by some of the most popular websites.
When doing the data collection for *RQ2*, we uploaded a copy of the Wired homepage [7] and uploaded on our webspace [8]. We then created nine different variations of this page by changing the number of prefetching requests in this page to 10,50,100 and changing the size of prefetched response per request to 100,1 kB, 10 kB. We used the Tor browser crawler [1] to capture packets for each of these webpages 144 times. In order to have equal data for comparison for some other websites, we also captured data for two other websites which we found to be similar. We found the classifier accuracy was not affected by much with these variations of dynamic link prefetching.

5.4 Feature Set

We had the option of choosing fine-grained features or coarse grained features for our classification framework. Fine-grained features have the advantage of capturing structural information about the webpage since packet ordering sequence will depend on the sequence of requests and responses for a webpage. Fine-grained features suffer from the disadvantage of suffering from variations introduced in packet ordering sequence due to network conditions. On the other hand, coarse-grained features avoid the randomness induced by network conditions by using information for the entire session but lose out on taking advantage of some structural information contained in the packet directions and packet sizes. We decided to test the effect of prefetching using both fine-grained features which use the information about the order in which packets appear as well as coarse-grained features which only use information from the entire session of information for a webpage. The coarse-grained features used by us are listed as follows:

1. Number of incoming packets: We accumulate the total number of TCP packets which came in to the client machine from any IP address during the load of the webpage

2. Number of outgoing packets: We accumulate the total number of TCP packets going out from the client machine to any IP address

3. Total size of incoming packets: We add up the size of TCP packets coming in to the client machine from any IP address during the load of the web page

4. Total size of outgoing packets: We add up the size of TCP packets going out of the client machine to any IP address during the web page load

All four of the features described above can include noise from background connections being made by the client machine. We also observed many incoming and outgoing packets in our dataset which we were not entirely able to explain, thus contributing to noise in our dataset itself. Some of this noise may have been due to the Tor browser querying the Tor directory for a list of Tor relays, trying to connect to some Tor relays before finally choosing a entry guard.
We decided to use only one fine-grained feature after seeing a stong result report by Cai et al. [10]. This paper reports a classification accuracy of 83.7% when using the sequence of packet directions. From the packet trace, we captured packet directions and represented them using letters. We then used an implementation [27] of the edit distance algorithm [23] as the distance function for a SVM-based classification.

Table 2: Experiment results for accuracy(%) Naive Bayes classifier using coarse-grained features

train\test	prefetch on	prefetch off
prefetch on	51.48	41.35
prefetch off	42.29	56.43

5.5 Experimental Validation for RQ1

We decided to try to evaluate the effect of prefetching by training and testing classifiers on opposite values of the prefetching setting. This would allow us to observe the effect of prefetching on classifier accuracy and also allow us to confirm this effect by trying different classifiers on the same dataset. We decided to use the Naive Bayes classifier since it has been used by Hermann et al. [18] and the SVM classifier with the edit distance as the distance function since it has been used by Cai et al. [10]. We provide a brief description of these classifiers in the following sections.

5.5.1 *Naive Bayes Classifier*

The Naive Bayes classifier is a simple classifier to test and acts as a first level sanity check in many classification problems before more complex classifiers are tried out. It assumes all features used are independent which is an assumption not entirely met in our classification problem. If the number of incoming packets is zero, the total size of incoming packets will also be zero. However, such a communication session would not be terribly useful in website fingerprinting. However, if the number of incoming or outgoing packets is not zero, the total size of incoming, outgoing packets will be independent of their respective numbers.

5.5.2 *Support Vector Machine-based Classification*

SVM classifier tries to find the hyperplane that maximizes the margin to the nearest data points in the training set. In the case of SVM-based classification using coarse-grained features, we used the SVM classifier with the Radial Basis Function kernel with the value of C set to 1 and γ set to 0.07. The values of these parameters were determined empirically. In the case of SVM-based classification using the fine-grained packet direction sequence feature, we modeled the packet directions with the characeters 'b' and 'd' to represent incoming and outgoing packets respectively. We then used the SVM classifier with the Radial Basis Function with the distance function set to use edit distance between the any two strings with the value of σ set to 1/8.

5.5.3 *Results*

Results for our experiments for finding an answer to *RQ1* are presented in Table 2, 3 and 4. Table 2 clearly shows the classifier accuracy is the highest when no prefetching packets are present in the packet capture. This can be caused by the fact that while prefetching requests are created at browser idle time, coarse grained features do not take into

Table 3: Experiment results for accuracy(%) of SVM classifier using coarse-grained features

train\test	prefetch on	prefetch off
prefetch on	31.625	30.77
prefetch off	34.34	34.93

Table 4: Experiment results for accuracy(%) of SVM classifier using fine-grained features

train\test	prefetch on	prefetch off
prefetch on	27.5	20
prefetch off	20	27.5

account when browser idle time occurs. The accuracy of the classifier is highest when the attacker has trained and tested on data captured using the same setting of prefetching. The attacker sees a drop of 15% in accuracy when opposite prefetching settings are used during training and test stages. This indicates that the current state of prefetching that exists in real world popular websites has the ability to change coarse-grained features .
Table 3 shows similar results but using SVM classifier with the Radial Basis Function kernel. In this case, the attacker is able to escape some penalty to the accuracy caused by opposite prefetching settings during the training and test stage but still experiences a lower accuracy than was observed when using the Naive Bayes classifier.
Table 4 shows results when only packet direction sequence is used as a feature and all other information from the packet captures is ignored. This information itself is found enough to accurately predict atleast 27.5% of the websites in our dataset. When opposite prefetching settings during the training and test stages are used, the classifier accuracy drops to 20%. This drop indicates that fine-grained features are also affect by the presence of prefetching packets in the packet trace.
We conclude from the difference in performance of the three classifiers that using opposite prefetching setting during the training and test stages causes the performance of the classifier to drop, irrespective of the type of features used.

5.6 Experimental Validation for RQ2

We created a closed world model consisting of only two websites, the first being a copy of our Wired.com homepage [8] and second being the homepage of the website EOnline.com [4] since we found this website's homepage closely resembled the Wired.com homepage. We used the Naive Bayes classifier on the coarse-grained features extracted from traffic for these two webpages. We present our results in Table 5. In this case, we varied the number of prefetching requests to values 10, 50, 100 and the size of each prefetched response to 100, 1 KB, 10 KB.

As can be observed, the use of dynamic link prefetching did not lead to a considerable drop in classifier accuracy. A closer look at our dataset revealed one cause of this to be the difference in the features for the two websites even after prefetching was used.

We summarize the difference in the means of feature values of our copy of the dynamically prefetching Wired.com page and EOnline.com [4] in Table 6. It can be seen that there exists a large difference in every feature. We also noted that this difference is equal to almost half the feature values for our dynamically prefetching version of Wired.com [8]. We investigated further in order to get to the root cause of this issue. We even collected another dataset wherein the number of prefetching requests made for every page was 1000 and the size of the prefetched response was between 2-4 MB and created four such variations of the Wired homepage on our webspage [8]. We crawled each page 36 times in order to get 144 packet capture sessions. However, we found the mean feature values for our copy of the dynamically prefetching Wired.com page from the new packet capture to be almost same as those from the previous capture. The difference in the mean values also continued to be almost the same as shown in Table 6. On further investigation we found that the Tor browser does not honor all the link prefetching requests specified in a webpage immediately after a page load has completed. Prefetching requests continue to be sent many seconds after a page load has completed. We found this behavior to be very different from the way prefetching requests are handled in the Chrome browser [17]. Our sample Wired.com page [8] contains 1000 prefetching requests to be submitted by the browser. From the Web Inspector tool built into the Chrome browser, we can see all the prefetching requests for our sample Wired.com page being serviced during and after the page load completes whereas in the Tor browser we see only 4-6 requests being serviced. We present a comparison of the behavior observed by us in Figure 5. It can be clearly seen that a tail of prefetching traffic is created by Chrome by not by Firefox.

It is our conclusion that it is this deviation of the Tor browser from the expected behavior that causes all of our specified prefetching requests to not be serviced and prevents us from getting the packet capture containing traffic for the prefetching requests and responses.

Table 5: Experiment results for accuracy(%) of Naive Bayes classifier using coarse-grained features for two websites when dynamic link prefetching is being used by one for the websites

Accuracy\Prefetch Setting	prefetch on	prefetch off
Accuracy	95.49	96.64

Table 6: Experiment results for accuracy(%) of Naive Bayes classifier using coarse-grained features for two websites when dynamic link prefetching is being used by one for the websites

number of incoming packets	number of outgoing packets	total size of incoming packets	total size of outgoing packets
1,800	1,400	2,422,900	455,100

6. DISCUSSION

The performance of link prefetching as a defense mechanism depends on the values used by the webpage for the number of prefetching requests made and the total size of prefetched responses. A large number of outgoing requests and incoming responses would make the network prefetch-

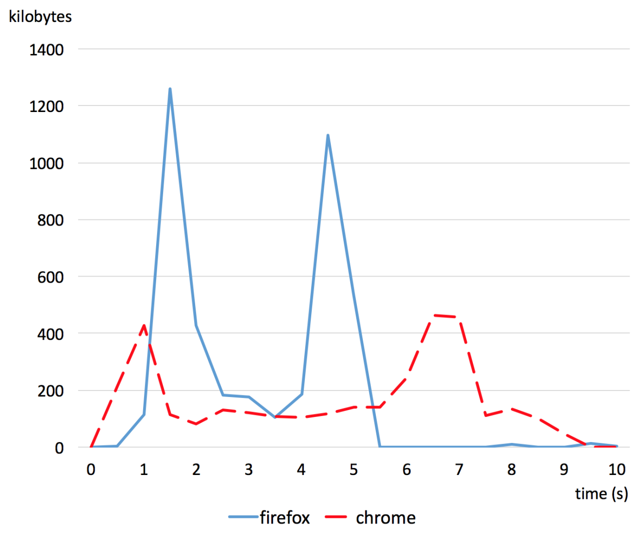


Figure 5: Comparing link prefetching behavior of Firefox and Chrome for our test webpage[8]

ing traffic indistinguishable from the normal traffic required to display the webpage. If a webpage A is to make its own network traffic fingerprint similar to another webpage B's network traffic fingerprint, assuming that webpage A currently has a smaller fingerprint than webpage B, it should create prefetching network traffic to compensate for the difference in the network traffic fingerprints of A and B. This can further be generalized to the open world model where every webpage can trigger a set of prefetching requests and responses which change its fingerprint to be more similar to a different webpage which has a larger fingerprint. This can work for fingerprints which use coarse-grained features as well as fine-grained features. e.g. a webpage A that finds another webpage B that creates K more outgoing packets resulting in an increase in the total incoming packet size of S can simply issue K prefetching requests for resources that have a total size of S. It should however be noted that webpage A would have to make requests for resources that are unlikely to be currently cached by the browser.
Some attacks may also choose to look at only the first 50 or 100 packets that show up in a packet capture for a given webpage and try to develop a fingerprint using only those packets. While the question of the number of packets to be used to create a fingerprinting is definitely interesting, it also needs to be further investigated how many prefetching request and response packets appear in such packet captures.

7. CONCLUSIONS

We proposed a new method for defending against website fingerprinting attacks on the Tor anonymity network. Link prefetching allows web pages to specify resources which are expected to be downloaded by the browser in the near future and provides web developers with an opportunity to improve user experience on their websites. We presented how link prefetching can be useful not only for improved browsing experience but also for acting as a defense mechanism against fingerprinting attacks. We showed the effect of link prefetching on different features used by fingerprinting attack classifiers. We formulated the challenge of using link prefetching as a defense mechanism in the form of two research questions and designed experiments to evaluate answers to our research questions. We create three different classifiers to evaluate the effect of the link prefetching setting on existing real world link prefetching performed by 60 of the 6000 most popular Alexa websites and found the existence of prefetching traffic to reduce the accuracy of classifiers. We then tried to evaluate the effect of changing the parameter values on the fingerprintability of a webpage but found that the Tor browser does not honor the prefetching requests specified in our test webpage. This behavior deviates from the expected behavior of downloading the prefetching requests once the webpage itself has finished downloading. Finally we discussed how a webpage can calculate the value of the number of prefetching requests and size of prefetched responses in order to disguise its fingerprint.

8. ACKNOWLEDGMENTS

This report is submitted as a partial requirement for the CSCI5271 research project. The authors acknowledge the guidance of Professor Stephen McCamant in the University of Minnesota throughout the course of this research project.

9. REFERENCES

[1] A crawler based on tor browser and selenium. https://github.com/webfp/tor-browser-crawler, 2015. Accessed: 2015-12-04.
[2] A critique of website traffic fingerprinting attacks. https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks, 2015. Accessed: 2015-11-1.
[3] Does alexa have a list of its top-ranked websites ? - alexa support. https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-, 2015. Accessed: 2015-12-14.
[4] Entertainment news. http://www.eonline.com/, 2015. Accessed: 2015-12-14.
[5] Scrapy - a fast and powerful scraping and web crawling framework. http://scrapy.org, 2015. Accessed: 2015-12-04.
[6] Tor project. https://www.torproject.org/, 2015. Accessed: 2015-12-12.
[7] Wired. http://www.wired.com, 2015. Accessed: 2015-12-04.
[8] Wired. http://www-users.cs.umn.edu/~byunx063/wired/random1.html, 2015. Accessed: 2015-12-04.
[9] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *the 2014 ACM SIGSAC Conference*, pages 227–238, New York, New York, USA, 2014. ACM Press.
[10] 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*, pages 605–616. ACM, 2012.
[11] X. Chen and X. Zhang. A popularity-based prediction model for web prefetching. *Computer*, 36:63–70+4, 2003.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[13] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance. *SIGMETRICS Perform. Eval. Rev.*, 27(1):178–187, May 1999.

[14] D. Fisher. Link prefetching FAQ. https://developer.mozilla.org/en-US/docs/Web/HTTP/Link_prefetching_FAQ, 2003.

[15] D. Fisher and G. Saksena. Link prefetching in mozilla: A server-driven approach. In *Web content caching and distribution*, pages 283–291. Springer, 2004.

[16] X. Fu, B. Graham, R. Bettati, and W. Zhao. On countermeasures to traffic analysis attacks. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 188–195. IEEE, 2003.

[17] Google. Chrome. http://www.chrome.google.com/, 2015. Accessed: 2015-12-14.

[18] 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*, CCSW '09, pages 31–42, New York, NY, USA, 2009. ACM.

[19] 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*, pages 31–42. ACM, 2009.

[20] A. Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pages 171–178. Springer, 2003.

[21] 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*, pages 263–274. ACM, 2014.

[22] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, 2011.

[23] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, Mar. 2001.

[24] M. Nottingham. Rfc5988: Web linking. Technical report, 2010.

[25] 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*, pages 103–114. ACM, 2011.

[26] M. Perry. Experimental defense for website traffic fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, 2011. Accessed: 2015-11-23.

[27] E. Polityko. Calculation of distance between strings - matlab. http://www.mathworks.com/matlabcentral/fileexchange/17585-calculation-of-distance-between-strings/content/strdist.m, 2015. Accessed: 2015-12-14.

[28] Y. Shi and K. Matsuura. Fingerprinting attack on the tor anonymity system. In *Information and Communications Security*, pages 425–438. Springer, 2009.

[29] D. Wagner, B. Schneier, et al. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, 1996.

[30] 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*, pages 201–212. ACM, 2013.

[31] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.

[32] S. Yu, T. Thapngam, S. Wei, and W. Zhou. Efficient web browsing with perfect anonymity using page prefetching. In *Algorithms and Architectures for Parallel Processing*, pages 1–12. Springer, 2010.