Improved Website Fingerprinting on Tor

Tao Wang
University of Waterloo

Ian Goldberg
University of Waterloo

Abstract

In this paper, we propose new website fingerprinting techniques that achieve a higher classification accuracy on Tor than previous works. We describe our novel methodology for gathering data on Tor that enhances the quality of classifier comparison and analysis. We offer new ways to interpret the data by using the more fundamental Tor cells as a unit of data rather than TCP/IP packets. We demonstrate an experimental method to remove Tor SENDMEs, which are control cells that provide no useful data, in order to improve accuracy. We also propose a new set of metrics to describe the similarity between two traffic instances; they are derived from observations on how a site is loaded. Using our new metrics we achieve a higher success rate than previous authors. To identify the potential power of censorship on Tor, we perform open-world experiments; we achieve a recall rate over 95% and a false positive rate under 0.06% for several potentially censored sites, which is significantly more accurate than previous techniques. In the closed-world experiments, our success rate is 90–91%, as compared to 86–87% from previous classifiers on the same data.

1 Introduction

When browsing the web, clients inadvertently reveal their destination websites to a number of routers along the way. These routers may passively observe and collect information on client behaviour (such as an ISP who may wish to sell this data to marketers), and they may aggressively attack the client's communication by censoring specific sites or certain types of behaviour [3]. Anonymous communication networks can protect the client from such threats by preserving the client's privacy. Tor is a popular anonymous communication network used by around 500,000 people per day [14]. To disassociate the client's identity from her destination, Tor routes her communication through a number of volunteer *relays* using multiple layers of encryption. The client's identity and her destination will not both be revealed to any single relay. However, an attacker can still attempt to compromise a web-browsing client's privacy by observing patterns in her sequence of packets, even assuming that the encryption leaks no extra information, using a technique known as *website fingerprinting*. A site may prove to be uniquely identifiable from the order, direction, and size of the packets used to load the site, allowing a client-side observer to identify a Tor web client's destination server, thus violating the privacy one might expect from Tor. The objective of this paper is to demonstrate that these techniques may be more powerful against Tor than previously thought. Our strong open-world experimental results demonstrate that an attacker can potentially monitor or block accesses to a specific site in Tor with very high accuracy.

To attack a Tor-using client, the attacker gathers traffic information about sites the client might visit by accessing these sites over Tor and recording the resulting packet sequences. We call the traffic record corresponding to a single visit to each site a "traffic instance". The attacker then taps the encrypted connection of the client and compares the client's traffic instance with the attacker's recorded traffic instances. In this paper, we focus on *distance-based metrics* the attacker can use to compare traffic instances. A distance-based metric is a function, which, given two traffic instances, produces a value that describes how similar those two instances are. This measure of similarity can be used in Support Vector Machines (SVMs) to classify traffic instances into classes corresponding to the site from which they were collected [2, 11]. We focus on distance-based metrics because in previous works, distance-based metrics offered better results than non-distance-based metrics [5]. They are a natural way to measure similarity, and they are easy to modify. We compare previous results using these two types of metrics in Section 2.

Tor has proven to be relatively resilient to website fingerprinting [6] as compared to other privacy technologies such as IPsec and SSH tunnels. The following factors contribute to Tor's defenses. First, Tor transfers data in units of 512 bytes, called *cells*, and Tor always pads all data transfers up to a cell boundary. This covers up identifiable packet lengths that are often unique to a given website [7]. Second, Tor clients transfer information on randomly chosen, short-lived *circuits* of three relays. Clients using different circuits may experience significantly different performance, in terms of latency, congestion, and bandwidth capacity [14]. These factors induce different sequences of packets for the same site, lowering fingerprinting accuracy. Third, Tor performs a number of background activities, such as circuit construction, circuit speed testing, flow control using SENDME packets, and so on. These activities may pollute the data if not filtered out. Finally, Tor uses pipelining and order randomization to add variance to the network traffic from a site. We give an overview of Tor, focusing on the aspects that impact website fingerprinting, in Section 3.

Our contributions

Improved data gathering. We describe how we collect our data in a more thorough manner than previous works. We take precautions to collect the data in the same way a realistic attacker would. We collect a new set of data that offers a fair comparison for classifier analysis. Our data collection methodology is described in Section 4.1.

New data processing. We make the key observation that we can exploit knowledge of Tor's inner workings in order to analyze its traffic more accurately than previous works. Instead of using TCP/IP packets, which are merely capsules used to transport Tor cells, we parse the packet data to obtain the underlying Tor cells. We are able to achieve better accuracy by using Tor cell sequences instead of TCP/IP packet sequences to train a classifier. We also attempt to identify the SENDME cells in Tor, which provide no extra information, and remove them to decrease noise. This is described in Section 4.2.

New website fingerprinting metrics. We demonstrate that we can achieve better accuracy with metrics that incorporate important observations on website fingerprinting. The observations include: dynamic content such as advertisements may cause a variation in incoming packets but not outgoing packets, and dynamic content is often loaded last in a website. These metrics are used to train a Support Vector Machine to achieve a higher accuracy. We describe the Support Vector Machine and present our new metrics in Section 5.

We conduct a series of open-world and closed-world experiments to validate our claims that these techniques achieve better accuracy, and they are presented in Section 6. We discuss our results in Section 7, and we conclude in Section 8.

2 Related Work

A number of attacks have been proposed for website fingerprinting. An attack generally consists of a way to process each training traffic instance to extract useful information, followed by a classification mechanism to identify a testing traffic instance. We outline related work on website fingerprinting attacks below. For our purposes, they can be divided into two classes: non-distance-based methods, and distance-based methods.

Defenses. We briefly note that while many interesting defenses have been suggested to counter these attacks [11, 18], they are not implemented in Tor because of their additional network load [12], which is a bottleneck in Tor [13]. Previous work has indeed shown that a number of known website fingerprinting defense techniques are either ineffective or inefficient [5].

2.1 Non-distance-based methods

In 2005, Liberatore and Levine [7] published two methods for "inferring the source of encrypted HTTP connections". The authors used the lengths of incoming and outgoing TCP/IP packets and discarded the order. Testing instances were classified with a Naïve Bayes classifier, based on packet lengths. The probability of an observed traffic instance belonging to a given class (representing one site) is computed from the product of the probabilities of the observed packet lengths occurring at their observed frequencies. The probability of a packet length occurring at a given frequency for a specific class is computed from a normal distribution determined by each training instance in that class. This method was shown to be quite effective on plain encryption (a client using a simple SSH tunnel, for example). Liberatore and Levine did not report the accuracy of their classifier on Tor.

Herrmann et al. [6] in 2009 described an improved method that gave a higher accuracy than Liberatore and Levine under comparable conditions. They described their method as an application of known text mining techniques to website fingerprinting. The classifier they used for training was the Multinomial Naïve Bayes classifier. As before, the order of packets was discarded, and only lengths and frequencies were used. The Multinomial Naïve Bayes classifier does not learn a normal distribution over possible frequencies of certain packet lengths. Instead, this frequency is used as an exponent to the relevant probability value. They applied a set of well-known text mining transformations to optimize their classifier. Herrmann et al. demonstrated that their algorithm achieves a higher success rate than Liberatore and Levine's algorithm on simple encryption. However, they also showed that its accuracy on Tor is only 3%. This indicates that website fingerprinting on Tor poses a greater challenge than plain encryption.

2.2 Distance-based methods

Besides the Naïve Bayes classifier, Liberatore and Levine also proposed using the Jaccard coefficient for classification [7]. For two sets of packet lengths, the Jaccard coefficient is a ratio of the size of their intersection to the size of their union. This coefficient was shown to be less effective than the Naïve Bayes classifier; it only considers packet lengths and does not consider the frequency in which they appear.

In 2011, Panchenko et al. [11] used a Support Vector Machine (SVM) to perform website fingerprinting specifically on onion-routing anonymity networks (such as Tor) using a variety of features. Features of two traffic instances induce a distance between them, based on how different those features are, which is used to decide whether or not these instances belong to the same site. These features include the total size of all packets in each direction, the size of the HTML document, the total number of transmitted bytes, markers for indicating direction changes in packet order, the percentage of incoming bytes, and more. Order is used in classification, unlike the non-distance-based methods described above. The authors performed separate open-world and closed-world experiments. In the open-world experiments, which simulated internet censorship, the attacker gathered a number of traffic instances of 5 specific censored sites and 1 instance each from a large set of 4000 uncensored sites; the simulated client could choose a censored site or one from another set of 1000 uncensored sites. In the closed-world experiments, the attacker gathered a number of traffic instances from a limited set of sites and the client was only allowed to choose among those sites. They achieved a recall of 73% with a false positive rate of 0.05% in the open-world model and an accuracy of 54% in the closed-world model for Alexa's top-ranked pages.

In 2012, Cai et al. [2] proposed using a different metric to achieve better results than Panchenko et al. They used the optimal string alignment distance, which is described in more detail in Section 5.2. They were able to achieve a higher accuracy of 87.3% in the closed-world model, although there are no open-world results. We supplement open-world results in this paper for our metrics and theirs. They also described how they can use Hidden Markov Models to classify web sites instead of web pages.[1]

3 Tor

In this section, we briefly describe Tor, focusing on aspects which are relevant to website fingerprinting.

Tor is a popular anonymity network, currently used by around 500,000 daily clients and carrying 2000 MB of data per second [14]. Tor consists of around 3000 volunteer *relays*, which are routers that volunteer to relay information for Tor clients. Clients build *circuits*, consisting of three relays, to communicate with destination websites. Tor uses TLS to communicate between relays. Each connection to a destination server is represented as a stream in Tor, which is multiplexed in Tor's circuits. We give more details on circuits and streams in Section 3.1.

Tor seeks to protect clients' anonymity. If website fingerprinting is accurate, however, then Tor cannot protect client anonymity against an attacker who is able to monitor the connection between the client and the first relay of the Tor circuit (called the *entry guard*). Furthermore, as Tor relays are operated by volunteers with no presumption of trust, they may act as attackers as well. Tor has implemented a *pipelining and order randomization* defense to protect itself against website fingerprinting; more details are given in Section 3.2.

3.1 Circuits and Streams

When using Tor, a client picks three relays: the entry guard, the middle relay, and the exit relay. The client constructs a circuit through these three relays, and uses that circuit for about ten minutes before switching to a new circuit. To limit the rate of deanonymization, each client keeps a list of three entry guards it will use for 30–60 days, while the middle and exit relays are chosen randomly for each circuit, weighted chiefly by bandwidth for load balancing. As relay bandwidth ranges over more than three orders of magnitude, the random selection of relays causes variance in Tor's performance, which impacts our attacker's data collection. We deal with this problem in detail in Section 4.1.

Once a circuit has opened, the client communicates through the circuit using a number of *streams*. Each stream corresponds to a separate TCP connection at the exit relay, and streams are multiplexed in a circuit. A client may open many streams to load a single site. Using

[1] We do not attempt to re-engage the problem of classifying web sites instead of web pages, and in this work, our use of the words "site" and "website" refer to a single web page.

persistent HTTP connections, more than one resource from the same server can be downloaded through each stream.

Tor uses a number of control cells to communicate commands among relays and the Tor client. These control cells may include circuit construction and closing cells, stream open and closing cells, flow control cells and cells used to transmit network information. Tor uses the SENDME control cell to perform flow control [4]. A circuit-level SENDME cell is sent every 100 cells per circuit, and a stream-level SENDME cell is sent every 50 cells per stream. While the attacker is collecting data for website fingerprinting, those cells will be included in the traffic instance. If the attacker can identify them under the encryption, SENDME cells should be removed as they provide no extra information, just as ACK packets should be removed from captures at the TCP/IP level.

3.2 Defenses

In response to the first successful website fingerprinting attack on Tor by Panchenko et al. [11], Tor developers have implemented an experimental defense against website fingerprinting [12]. This defense has three components: HTTP pipelining is enabled so that multiple requests can be made on a single stream without having to wait for each to finish; the pipeline size is randomized; and the order of requests is randomized. This defense does not significantly impact the total size of the transmission. The Tor developers did not test the effectiveness of this defense, and no additional defenses have been implemented yet. Cai et al. showed that the defense is ineffective [2] against both their strategy and that proposed by Panchenko et al. We analyze this defense on our metrics as well.

4 Data Collection and Processing

4.1 Collecting Data

Previous website fingerprinting works generally did not go into detail on the problem of data collection on Tor. Tor is a live network with ever-changing performance, and different clients may have entirely different experiences. Precautions need to be taken in order to ensure that data is collected the same way a realistic attacker would. With our new methodology, we collect a set of data for our experiments. In this section we specify exactly how we collect the data while addressing issues in circuit construction, timing, and website localization. We will later compare different website fingerprinting techniques using this data set.

4.1.1 Circuit Construction

Tor relays have a wide range of bandwidths. As of January 2013, the top 1% of relays comprised 34% of the total bandwidth of Tor and the top 20% of relays comprised 92% of the bandwidth. Bandwidth and congestion will affect the sequence of packets received by the client, so that traffic instances collected using the same circuit are more similar than those collected using different circuits. We assume that the attacker can observe the client's network traffic, but cannot control or observe which Tor circuits the client is using (this is a basic assumption necessary for the privacy guarantees of Tor [4]). We must naturally ensure that we never use the same circuit for both training and testing in our experiments, as that would give an unrealistic advantage to the attacker.

By default, Tor can only use a circuit for up to ten minutes, after which Tor will not launch new connections on the circuit. If this setting is maintained, and if sites are visited in sequence, then instances of the same site are more likely to be accessed with the same circuit, while instances of different sites are more likely to be accessed with different circuits, which is another unrealistic advantage to the attacker. The inherent difference of circuits imposes a difference upon the traffic instances of different sites, which could help the machine learning algorithm separate them. We should not allow Tor to automatically close its circuits every ten minutes.

4.1.2 Timing

A site's content may change over time. A news site, for example, would not have the same content every day; it may have different images and different text, and perhaps even a different number of resources. We find that, in our metrics, processing the data to train the classifier could take more than two hundred CPU hours, whereas news sites could be updated every few minutes. Therefore, we cannot expect the attacker to keep up with ever-changing content. The attacker and client should train on sites that are loaded at least several hours apart.

4.1.3 Localization

Depending on the location of the exit relay, a site could present entirely different data. This is known as website localization and it is especially prevalent among popular sites, such as `www.google.com` and `www.yahoo.com`. Roughly speaking, there are two types of website localization that are of concern to us. The first type is redirection: a Canadian client attempting to access `www.google.com` will be redirected to `www.google.ca` by Google—the site that the client is actually accessing is different depending on locality.

The second type involves content changes: a German advertisement may be shown to a person connecting from Germany or a Tor client exiting from Germany, but it is unlikely for other clients to see the German advertisement. This does not involve redirection. When Tor is used, the locality of the client is determined by the location of the exit relay, and is thus dependent on what circuit the client is using. Tor does not attempt to choose the exit relay based on the client's location (to protect the client's privacy).

4.1.4 Our Methodology

We use a Tor controller [9] to gain full control over circuit construction. Our circuits are built according to Tor's algorithm, which chooses relays with a probability depending on their observed bandwidths. While clients can change this algorithm, it is not easy or encouraged to do so and thus we assume the client uses Tor with no modifications.

Our main data set consists of 40 traffic instances for each of 100 sites (other data sets we use for specific experiments are detailed in Section 6). To deal with circuit and timing issues, we collect our traffic instances in *batches*. Each batch consists of 4 traffic instances; they are each collected a few hours apart using different circuits. We later perform 10-fold cross validation, so that training instances are never collected from the same circuit used to collect the testing instances. We maximize Tor's permissiveness for circuit dirtiness (so Tor can keep using the same circuit), and we use each circuit as long as possible. As different circuits are used for training and testing, it is more difficult for the attacker to use timing information to attack the client.

The data set we use is based on Alexa's top sites,[2] modified to to avoid localization redirection, in order to ensure that traffic instances from the same site were consistent. We modified the top sites list in two steps. First, if a single site occurred many times under different localization instances in the list, we removed all repeated occurrences of it. In particular, different localizations of `www.google.com` occurred many times in the top 100 sites list, and all had similar traffic patterns. A classifier would not be expected to distinguish them, and a censor who may wish to block Google's search engine as a whole would not need to separate two different localizations of the site in any event. Second, whenever possible, we attempted to specify the localization version of a site we wanted to visit. That is to say, we did not attempt to visit `www.yahoo.com`; instead, we attempted to visit `www.yahoo.de`. This avoided localization redirection from various sites. Modification of the top sites list was done manually.[3] Our data was collected batch-by-batch, with each batch corresponding to one circuit (one client); each batch took around 4 hours and was separated from each other by around 4–12 hours.

We checked the size of all traffic instances. If the size of the traffic instance was less than 20% of the median size for that site, it was removed. These traffic instances are regarded as failed instances, which may be a failed connection to the server or a server-originated message that denied access to the client.

We did not attempt to control dynamic content changes based on client locality, such as advertisements. Furthermore, server-side caching of client settings may affect the content for clients using exit relays that have accessed those sites. We could limit the locality of the exit relays, but this would limit our selection of circuits. Our choices mean our results should more accurately reflect live conditions, where an attacker cannot account for such differences in content.

4.2 Processing Data

Once the attacker collects the traffic data (either the training or testing data), some processing may be done before using it to classify the site. After processing, each traffic instance is represented as a sequence of positive and negative integers. We describe three ways to process the data before using it as input to the SVM.

4.2.1 TCP/IP packet instances

A TCP/IP packet header contains the length of the given packet, which was used by previous authors [2, 11] to represent the packet. This length often ranges from 0 (an ACK) to 1448 for ethernet. Previous authors discarded ACK packets as they provide no useful information, and removing it increased the accuracy of their classifier. TCP/IP does not attempt to pad these packets when there is not enough data, but it attempts to send packets of the maximum segment size (MSS), which is 1448 for Ethernet. Packet lengths are viewed as a sequence of integers in order. Outgoing packets are represented as positive integers while incoming packets are represented as negative integers. Cai et al. also rounded the packet sizes up by increments of 600. For example, one GET request would be classified as 600, a SENDME and a GET request sent together would be 1200, and MSS packets on Ethernet would be 1800 (incoming packets are often MSS packets). This strategy offers a distinction between these types of packets. This is similar to a strategy used

[2] http://www.alexa.com/topsites

[3] We also removed `tumblr.com` from the list as it was unusually large (around 10 MB). This site impacted our processing time and does not impact our classifier comparison, as it was very easy to identify from its size alone.

by Panchenko et al. They round the total size of all packets in a traffic instance by increments of 600 and use it as a feature. We compare this data processing method to our new ones, which are below.

4.2.2 Tor cell instances

We experiment with extracting Tor cells directly. The Tor cell is a more consistent and basic unit than TCP/IP packets. For instance, TCP/IP packet retransmissions would produce duplicate entries in traffic instances processed as above, but not when we reconstruct the TCP streams, parse the TLS layer, and extract the cell counts. As the choice of circuits affects timing, we should remove timing-dependent factors from our data. Tor cells cannot be broken down or combined together by relays or routers in the middle, unlike TCP/IP packets. Tor cells are always padded to a constant size, while the sizes of TCP/IP packets can change based on the connection.

Tor encrypts its data in Transport Layer Security (TLS) records, which always contain a number of complete Tor cells. TLS is designed to encrypt information on the socket layer to prevent eavesdropping, message forgery, or tampering on the internet. As such, it operates independently of TCP/IP, and thus a TLS record could be contained in parts of one or several TCP/IP packets. There are three types of TLS records: data records, handshake records and alerts. TLS data records begin with one byte for type (17 for application data), followed by two bytes of the version (0301, TLS version 1, is used by Tor), followed by two bytes representing the length. We ignore TLS handshakes and alerts.

Parsing TCP/IP data allows the attacker to reconstruct full TLS records with their lengths. We are able to reconstruct these records while discarding retransmitted TCP/IP packets. We denote each size as positive if it is outgoing and negative if it is incoming. We round these record lengths down, to the closest multiple of 512, and then divide by 512. Each TLS record may contain a number of complete Tor cells (or none), and this method will produce the number of Tor cells in the record. The rounding accounts for the extra data in the record, such as encryption and MAC overhead, and empty TLS records used to defend against the BEAST attack [8].

We demonstrate with the following example. Consider a sequence with three TLS records of size 544, -1088, 1088. This sequence will be represented as 1, -1, -1, 1, 1 in the Tor cell instances. Each cell is recorded separately, so Tor cell instances only contain the integers 1 and -1.

4.2.3 Removing SENDMEs

As described in Section 3, Tor issues a circuit-level SENDME cell every 100 cells on each circuit and a stream-level SENDME cell every 50 cells on each stream. We attempt to automatically remove these cells as they provide us with no information.

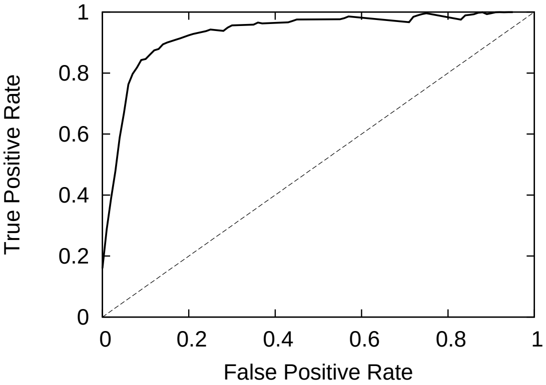

Figure 1: ROC to evaluate True Positive Rate and False Positive Rate with our experimental SENDME removal methodology

We remove SENDMEs with the following method. We scan through each traffic instance with a running counter of incoming cells. When the counter reaches some amount p_1, the next outgoing cell is guessed to be a SENDME. Then, the counter is decreased by some amount p_2 (the counter can become negative). We collect data to evaluate this scheme by instrumenting Tor to mark SENDME cells on 100 traffic instances. The results are presented as a receiver operating characteristic curve (ROC) in Figure 1. The false positive rate is taken over all outgoing packets, which is much greater than the total number of SENDMEs. The area under the curve, as a sum of trapezoids, is 0.88.

Specifically, for our experiments, we choose the parameters $p_1 = 45$ and $p_2 = 40$, which offers a true positive rate of 62% and a false positive rate of 5.7%. In our data set, around 19% of all outgoing packets (2.1% of all packets) were removed this way.

5 Classification

Our website fingerprinting problem can be viewed as a machine classification problem. Each class can be a group of sites, such as “Censored” or “Permitted” (in the open-world scenario) or a specific site (in the closed-world scenario). Given a sequence of packets collected from one access of a site, the classifier attempts to categorize the testing instance into one of the available classes. To do so, the classifier first learns about each class by training on a number of training instances. We next discuss the SVM classifier we use to tackle the classification problem, and then discuss how we construct the kernel used in the SVM.

5.1 Support Vector Machine

The Support Vector Machine (SVM), proposed by Vapnik and Chervonenkis [15], is a classifier that was used by previous authors for website fingerprinting [2, 11] and was shown to be fairly successful. We therefore use SVMs for our experiments as well. During training, SVMs take in a number of features, $\vec{f_s} = \{f_1, f_2, \ldots, f_F\}$, for each training instance s. Each training instance can then be considered a point in F-dimensional space belonging to a certain class. Consider the simplest case, with only two classes. If the points can be separated by a hyperplane, then the SVM finds the hyperplane that maximizes the gap between the two classes. The points that are closest to the gap determine (support) the hyperplane and are called support vectors. Each side of the hyperplane consists of only points from one class. The SVM would then classify each testing instance based on the side of the hyperplane on which it is located. If the training points cannot be separated by a hyperplane, we can use a cost parameter c to penalize training points that are on the wrong side of the hyperplane. The higher the value of c, the more importance the machine would place on avoiding wrongly classified points (relative to maximizing the gap between different classes). To avoid overfitting to the cost parameter c, we choose it exponentially. The machine can also assign a hyperplane in a higher dimension than F by extending all points to a higher dimension where a more complex hyperplane can be found.

As SVMs are based on finding a maximum gap between different classes, distance is central to the SVM. Given two instances s_1 and s_2 with feature vectors $\vec{f_{s_1}}$ and $\vec{f_{s_2}}$, where each feature is an F-dimensional vector in real space, a possible way to compute the distance between s_1 and s_2 is:

$$e^{-\gamma||\vec{f_{s_1}} - \vec{f_{s_2}}||^2}$$

This is the radial basis function, with parameter γ. A square kernel matrix of size $K \times K$ is built for the K training instances this way, where the element in row i, column j is computed as above for instances s_i and s_j. A value of 1 indicates perfect similarity, whereas a value approaching 0 indicates strong dissimilarity. In this work, we do not use the radial basis function; we use distance-based metrics to compute the square kernel matrix without using features, as described in Section 5.2.

We use the SVM provided by LIBSVM version 3.14, which deals with multiple classes using a technique known as "One-to-One" classification. Suppose the set of classes is $\mathcal{C}$, which is a list of integers. A different SVM is trained for each pair of classes, resulting in $|\mathcal{C}|(|\mathcal{C}| - 1)/2$ SVMs, as above; let the SVM responsible for distinguishing between classes $i, j \in \mathcal{C}$, be denoted as $SVM_{i,j}$. The classifier $SVM_{i,j}$ is responsible for deciding if each testing instance s is more likely to belong in class i or class j; this output is denoted as $SVM_{i,j}(s) \in \{i, j\}$. Then we find the class $a \in \mathcal{C}$ which maximizes $|\{j \in \mathcal{C}|SVM_{a,j}(s) = a\}|$. This technique resembles a tournament in which all players (classes) play against each other, and the player with the greatest number of wins is declared the victor. There are other ways to handle multiple classes in SVMs—for instance, we can train binary classifiers to pit each class against all other classes simultaneously in a "One-against-All" classification. Each $SVM_{i,j}$ can also output a "degree of certainty" in their decision. Other researchers have investigated more sophisticated techniques for multi-class classification [17]. We use the approach provided by LIBSVM in this paper.

5.2 Distance-based Metrics

A traffic instance is represented as a sequence of positive (outgoing) and negative (incoming) integers indicating the size of TCP packets or Tor cells. We train an SVM by directly computing the kernel matrix from these sequences using distance-based metrics. In this paper, we propose a number of new distance-based metrics. These metrics are designed to allow greater success in website fingerprinting by taking certain realistic observations in mind. These observations are described in detail in each corresponding section.

5.2.1 Optimal String Alignment Distance

Cai et al. use the optimal string alignment distance (OSAD)[4] to measure the distance between two traffic instances. OSAD is used in word matching; it describes the number of insertions, deletions, substitutions and transpositions required to transform a sequence of characters to another, with the specific requirement that transpositions can only affect elements that are adjacent in each string. This means that the distance between xAy and yx will be 3 operations (delete A, delete y, insert y before x) instead of 2 operations (delete A, transpose y and x). The latter operation is not allowed as x and y are not adjacent in the first string.

These operations can be assigned different costs without affecting the above algorithm's validity; the costs of insertions and deletions must be the same, however, in order to preserve distance symmetry. Cai et al. assign a lower cost to transpositions than insertions and deletions. The details of the OSAD algorithm can be found in Appendix A.

[4] Cai et al. called the metric they used the Damerau-Levenshtein distance, but a review of their code and their results showed that it is the OSAD, which is a restricted version of the DLD.

5.2.2 Damerau-Levenshtein Distance

The Damerau-Levenshtein distance (DLD) is closely related to the OSAD, with the chief difference being the removal of the restriction on transpositions. The Damerau-Levenshtein distance between two strings is therefore always less than or equal to the optimal string alignment distance when the operation costs are the same.

We can use Algorithm 2 in Appendix A for the Damerau-Levenshtein distance by making one additional change [16]. When computing the value of $M_{transpose}$ for the element at (i, j), instead of comparing with the element at $(i-2, j-2)$, we compare instead with the element at (i_1, j_1), where i_1 is the last occurrence in the first string of the jth element of the second string, and j_1 is the last occurrence in the second string of the ith element of the first string. The cost is then the cost of the number of deletions necessary to make those two elements adjacent, plus the cost of a transposition, plus the cost of re-inserting the required elements. This allows us to transpose elements that are not adjacent to each other in the original strings.

5.2.3 Removing Substitutions

The permissible operations on one traffic instance to transform it into another roughly correspond to possible events when accessing the same site several times. Insertions and deletions would be necessary if, for instance, a site loaded a different advertisement (which may be slightly larger or smaller), or if the text changed. Too many connections may be opened and some connections may be rejected, which would cause additional packets to be sent in a particular traffic instance. Transpositions are allowed at a lower cost because there is almost always some degree of reordering between two accesses of the same site. The timing of requests is highly dependent on how quickly data is received. As Tor relays are frequently congested [1], packet timings of the same site even across the same circuit will not be consistent (much less across different circuits).

Substitutions, however, do not seem to correspond to any realistic scenario. Accesses of the same site would not change the packet sequence in a way that can be represented by substitutions, unlike insertions, deletions or transpositions. We want to limit our permissible operations to those corresponding to possible situations that could cause two instances of the same site to be different. This is because the cost between two traffic instances should be low only if they correspond to the same site. We therefore experiment with removing substitutions from the list of possible operations. This can be done by removing the M_{sub} term when computing the minimum of different operation costs at $M_{i,j}$.

5.2.4 Different costs for Incoming/Outgoing Packets

The number of incoming packets depends on the size of web objects, which may change unpredictably for the same site for various reasons described above. In contrast, the number of outgoing packets depends on the number of resources (GET requests), the number of connections Firefox would open to download these resources (stream open requests), and the number of SENDMEs. The number of SENDMEs can vary based on the number of incoming packets, but the rest do not change easily. Since the number of outgoing packets is less likely to change when accessing the same site several times, outgoing packets should be more costly to insert or delete. On average, around one out of nine Tor cells when accessing a site are outgoing. We experiment with changing the cost of operations such that inserting and deleting outgoing packets are more costly than inserting and deleting incoming packets.

5.2.5 Varying transposition cost

Accesses to the same site can result in different packet sequences because of timing. Two resources that were downloaded simultaneously in one access may be downloaded sequentially in another, as time delays may cause the second to be requested late. Such delays, however, are less likely to affect the initial packets. This is because the order is more important initially: we must first connect to the site, request the main page, get the main page, and read the reference to resource locations before we can begin downloading other resources. We experiment with varying the cost of transpositions depending on the position of the element, with the cost being higher at first and lower down the line. For the element at (i, j), we define $P = min(\frac{i}{m}, \frac{j}{n})$. Then the transposition cost is $cost_{trans} = (1 - 0.9P)^2$. While other choices are possible, this satisfies our criteria and we will experiment with it.

5.2.6 Fast Levenshtein-like distance

The amount of time taken to compute OSAD and DLD is significant. We propose a new Levenshtein-like distance algorithm that reduces the algorithm time from $O(mn)$ to $O(m+n)$ on two strings of size m and n; it is around 2,000 to 3,000 times faster in our experiments. The algorithm is presented in Algorithm 3 in Appendix B.

5.3 Post-processing

The above metrics produce distances that are of the order of magnitude of the number of packets in each string, with smaller values indicating greater similarity. As discussed in Section 5.1, the elements of the kernel matrix should have values between 0 and 1, with 1 indicating perfect similarity. We use the strategy described by Cai et al. to transform these distances into values suitable for the kernel matrix. For a distance $D(s_1, s_2)$ between two strings (traffic instances) s_1 and s_2, we first compute $D'(s_1, s_2) = \frac{D(s_1, s_2)}{\min(|s_1|, |s_2|)}$ and then we take $K(s_1, s_2) = e^{-D'(s_1, s_2)^2}$. This gives us a value suitable for the kernel matrix. Cai et al. showed that this method was effective.

6 Experimental Results

6.1 Setting

Due to high computation costs, we used a parallel computing cluster to perform our experiments. As each entry of the SVM kernel is independent of the others, our problem is embarrassingly parallel. We used up to 200 cores to compute the SVM kernel matrix simultaneously.

We performed our experiments on the Tor Browser Bundle version 2.4.7-alpha-1. This bundle includes Tor Firefox version 10.0.12esr and Tor version 0.2.4.7-alpha. Tor Firefox includes HTTPS Everywhere and Noscript, as well as a number of patches, including one that enables pipelining and order randomization. We made one change to Firefox: we disabled caching by setting the value of network.http.use-cache to False in about:config.

We edited `torrc`, the configuration file for Tor, to change two options. We set `MaxCircuitDirtiness` to 600000 (seconds) so that our circuits would not be automatically closed after 10 minutes. We closed our circuits manually after each batch was completed (i.e. after each site was loaded 4 times). We set `UseEntryGuards` to 0 to disable the set of limited entry guards. Otherwise, we would have used only three entry guards for our site accesses, which limits the validity of our results across Tor. The client is not expected to make these changes; they are made for our experiments to gather realistic data, as described in Section 4.1.4.

We installed iMacros 7.6 on Firefox. We wrote iMacros code which loaded each site successively, and programmed iMacros to close the tab five seconds after each site finished loading. This ensures that no future streams will be opened for that site, and it roughly corresponds to a client who spends a small amount of time on each site after it loads. The iMacros code worked with our Tor Controller code in the experiment; the code is given in Algorithm 1, with the iMacros code in bold.

We used a Tor Controller [9] to control Tor's behaviour during the experiments. After each site access, we closed all open Tor streams and then reported the time. (We closed the open streams so that they would not affect the next traffic instance; a more natural option was to wait for them to close, but this significantly increased data collection time and did not affect the order of packets.) Next, after each site is visited 4 times in a batch, we closed all open circuits. This ensured that each circuit could not be used for more than 4 accesses of the same site. Only the packets captured between the marked start time and end time of each site were included in the traffic instance.

Algorithm 1 iMacros and Tor Controller code

```
for $0 \leq batchcounter < 10$ do
    for $0 \leq i < 100$ do
        for $0 \leq counter < 4$ do
            Mark the start time
            Load site $i$
            PAUSE 5 seconds
            Mark the end time
            Close the tab and start a new tab
            Close all streams
            PAUSE 4 seconds
        end for
    end for
    Close all active circuits
end for
```

6.2 Closed-World Results

We tested the various metrics described in Table 1 (used to create the SVM kernel matrix, as described in Section 5.3) against the data sets described in Table 2. Our metrics specified the cost of different operations. To prevent overfitting, we chose the operation costs once and did not attempt to vary the cost in order to find the one that returned the best results; it is possible that other operation costs may produce better results. We varied the cost parameter c of the SVM exponentially from 4^0 to 4^{10}, and chose the best result, which was different for each metric. We collected traffic instances using the methodology described in Section 4.1.4.

We experimented on 100 sites with 40 instances each, and performed 10-fold cross validation, so that there were 36 training cases and 4 testing cases for each site (400 tests for each fold). We computed the kernel matrix once and used different parts of the matrix to train and test for each fold. Our results are shown in Table 7 in Appendix C, and some interesting excerpts are shown in Table 3. We acquire one accuracy value for each of the folds, and we show the mean and standard deviation of these 10 accuracy values for each metric and data set.

The highest accuracy achieved was 90%, using set 4 (Tor cells with SENDMEs removed) and metric 5 (OSAD metric with a combination of strategies; hereafter referred to as the *combined OSAD*). This can be compared to 87% using set 2 and metric 1 (the data pro-

1	**OSAD, with $cost_{id} = cost_{sub} = 2$, $cost_{trans} = 0.1$**
2	OSAD, disabling substitutions
3	OSAD, with $cost_{id} = 6$ if the element is positive (outgoing)
4	OSAD, with varying transposition cost squared to its position in the sequence
5	**OSAD, with all changes in metrics 2, 3 and 4**
6	Damerau-Levenshtein Distance, with all costs = 2
7	Fast Levenshtein-like distance

Table 1: Distance metrics used to create the SVM kernel matrix, as described in Section 5.3. Metric 1 is that used by Cai et al. [2]; metric 5 is our *combined OSAD* metric.

1	TCP/IP packet sequence
2	**TCP/IP packet sequence, rounded up to 600**
3	Tor cell sequence
4	**Tor cell sequence, without SENDMEs**

Table 2: Data sets used as inputs to the SVM. Set 2 (Section 4.2.1) is that used by Cai et al. [2]; set 4 (Section 4.2.3) is our proposal.

Set	Metric	Accuracy (our data)	Accuracy (Cai's data)
1	1	0.74 (± 0.10)	0.82 (± 0.02)
2	1	0.87 (± 0.03)	0.86 (± 0.02)
2	5	0.65 (± 0.09)	0.76 (± 0.04)
3	4	0.86 (± 0.06)	0.84 (± 0.02)
4	2	0.87 (± 0.06)	0.89 (± 0.02)
4	5	0.90 (± 0.06)	0.91 (± 0.02)
4	7	0.70 (± 0.07)	0.71 (± 0.02)

Table 3: Closed-world results

Set	Metric	Site	TP	FP
2	1	google.de	29/40	1/860
		facebook.com	34/40	0/860
		wikipedia.org	39/40	2/860
		twitter.com	37/40	0/860
4	5	google.de	38/40	0/860
		facebook.com	39/40	2/860
		wikipedia.org	39/40	0/860
		twitter.com	39/40	0/860

Table 4: Open-world results

cessing and metric used by Cai et al.; hereafter referred to as *old OSAD*). **This is an over 20% decrease in the mean error rate.** Our fast Levenshtein-like algorithm achieved a somewhat lower accuracy of 70% on set 4, but with a substantially reduced computation cost: on the same cluster of cores, using the fast algorithm on set 4 (Tor cells with SENDMEs removed) in our data, kernel matrix computation took 283 CPU seconds, compared to 608,000 CPU seconds for combined OSAD. For each fold, training the SVM took around 6 CPU seconds for both metrics and testing was done in 0.7 CPU seconds.

6.2.1 Cai's Data

Cai et al. graciously shared with us the data they collected for their paper [2]. We performed the same analysis as above on their data. We show the results on Cai et al.'s data in Table 8 in Appendix C, and some excerpts are shown in Table 3. Although these sets were independently collected with different methodologies and selection of sites, the accuracy rates were largely comparable, suggesting that our analysis is robust. We note that with this data, the decrease in the mean error rate from the *old OSAD* metric of $1 - 0.86 = 0.14$ to our *combined OSAD* metric of $1 - 0.91 = 0.09$ is **over 35%**.

6.3 Open-World Results

We conduct an open-world experiment to evaluate the potential of censorship on Tor. A censor may choose to allow Tor, but monitor or block specific sites. Our own data set is used for these experiments. To emulate censorship, we choose four sites that have a history of being censored: `google.de`, `facebook.com`, `wikipedia.org` and `twitter.com` (hereafter referred to as the "censored sites" in this section).

We used Alexa's top 1,000 sites for this experiment. Sites ranked 101 to 1,000 were chosen as non-censored sites and one traffic instance from each site was loaded, out of which 860 were successfully loaded and used. For each of the censored sites, we trained an SVM on 40 censored instances and 860 non-censored instances for 10-fold testing. On the same fold, the attacker did not train on the same non-censored sites that the client visits. True Positives (TP) is the number of censored traffic instances that were classified as censored (population of 40); False Positives (FP) is the number of non-censored traffic instances that were classified as censored (population of 860). Alexa's top 1,000 sites were not manually processed in order to fix localization. The results are shown in Table 4. We see that most sites were correctly identified with over 90% success rate. With the combined OSAD (set 4, metric 5), in particular, we see that these particular sites reach a mean of 96.9% recall, compared to 86.9% on the old OSAD (set 2, metric 1). There are very few false positives in all cases. The cost c of the SVM was lowered to 0.25 to improve accuracy.

Set	Metric	With Def.	Without Def.
2	1	0.87 (± 0.05)	0.82 (± 0.03)
	5	0.80 (± 0.09)	0.79 (± 0.04)
4	1	0.79 (± 0.06)	0.77 (± 0.07)
	5	0.90 (± 0.03)	0.88 (± 0.06)

Table 5: Comparison of accuracy with and without Tor's Pipelining and Order Randomization Defense

6.4 Evaluation of Tor's Pipelining and Order Randomization Defense

The above experiments were conducted with Tor's pipelining and order randomization defense [12] enabled. Cai et al. demonstrated that this defense was ineffective against their technique [2]. We repeat this experiment on our metrics as well. We collected 10 instances each of 100 sites with pipelining and order randomization disabled, and trained the SVM using the same parameters on several set-metric combinations. Each circuit was closed after 1 access to each site, thus ensuring that the attacker does not train on the circuits that the client uses. We compare these results to those with pipelining and order randomization (our main data set), also on 10 instances each.

Our results are presented in Table 5; results collected in the data set with this defense enabled are listed in the column "With Def."; those with this defense disabled are listed in the column "Without Def." The results demonstrate no significant disparity with or without the defense; in fact, as Cai et al. observed, the metrics seemed to be marginally more accurate with the defense enabled.

7 Discussion

7.1 Damerau-Levenshtein Distance

The Damerau-Levenshtein Distance (metric 6) has a much lower success rate than other metrics, even though it allows unrestricted transpositions. We seek to investigate why its success rate was so low.

The combined OSAD (metric 5) is much more successful than DLD (metric 6). We compare the D' value (normalized distance) of those two metrics. We looked at the list of all pairs of traffic instances belonging to the same site, and took their D' value. For the combined OSAD, the median D' value was 0.972. For each site, we also looked for another site which had the minimum mean D' value to this site (the most similar site); the lower this value is, the more likely those two sites would be confused. The median D' value to the most similar site was 1.06 across 100 sites. We repeated these for DLD, and the values were 0.386 and 0.265 respectively. For combined OSAD, a site was persistently most similar only to itself, whereas for DLD, a site was often more similar to some other site than itself. This indicates that DLD is not successful in distinguishing different sites because it is too lenient. We showcase a comparison between the two metrics in Figure 2 on set 4, site 1 (google.de). The graph shows the number of sites within some distance of site 1, taking the mean distance over all pairs of instances belonging to the compared sites. We can see that the line for DLD (metric 6) is higher, which indicates that more sites are possible contenders for classification (thus resulting in misclassification more often). This is likely due to the fact that the effect of DLD's unrestricted transpositions is to make it much easier to transform any string into any other string. This increases the number of sites that are likely to be confused with a given site.

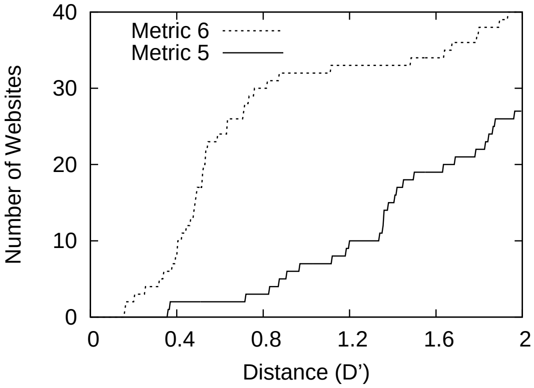

Figure 2: Number of sites within some distance of site 1 on set 4 for combined OSAD (metric 5) and DLD (metric 6)

7.2 Training set selection and parameters in SVM

We specified that our batches of 4 instances of each of 100 sites should use different circuits, and the batches should not be collected right after each other. We demonstrate the importance of our methodology with a small experiment. On 100 sites in the closed-world scenario, we take the first batch (4 sites), and we train our SVM on it using combined OSAD and 4-fold testing. This returned an accuracy value of 86%. We then trained on trials 1, 11, 21, and 31, which were from 4 different batches. The training size was therefore the same. This returned an accuracy of 74%. We therefore emphasize that the collection methodology cannot be taken lightly; different results must be compared on the same data set.

The size of the training set has a significant impact on training accuracy. We compare several metrics and how the size of the training set affects their accuracy. The results are presented in Figure 3. To gather more data points, for this particular section, we used 3-fold cross-validation rather than 10-fold cross validation. The maximum training size on our data set is 26. An increased training size generally improved the accuracy. We recall that the data was collected in batches of 4 immediately after one another using the same circuit for each batch, so that trial numbers 1 to 4 are similar to each other; 5 to 8 are similar to each other; and so on. To ensure that our training and testing data are from separate batches, we do the following. Suppose the size of the training plus testing sets is $K + L$, we computed the separation $s = \lfloor 40/(K + L) \rfloor$, and only the trials numbered $1, 1 + s, 1 + 2s, \ldots, 1 + (K + L - 1)s$ are used for each site. We observe that combined OSAD consistently outperforms other metrics when we vary the training size. It should be noted that the results presented in this section were different from those presented in Section 6.4 because the trials were chosen differently.

We used Alexa's top 100 sites for our closed-world experiments. We chose this number to ensure comparability with previous results such as those by Panchenko et al. and Cai et al. [2,11] We seek to understand if attempting to train on a greater number of sites would cause a significant change in accuracy. To do so, we varied the number of sites and observed the effects on our best metric, the combined OSAD (metric 5, set 4). We chose the first 10 sites, the first 20 sites, and so on. We performed 10-fold cross-validation on 40 trials for each site. The effect of varying our total number of sites on the accuracy is shown in Figure 4. We see that the accuracy for 10 sites was much higher; this may be because the top 10 sites are distinct from each other, reliable, and contain little dynamic content in their home pages. There is no noticeable difference in accuracy with more than 40 sites for training. This suggests to us that a more ambitious attacker seeking to identify a greater number of sites can use our techniques with no significant penalty.

For our experiments, we varied the cost parameter of the SVM, c, over a range of values to determine the best cost for the metric. In most cases this did not produce any significant change in the accuracy. However, varying the cost parameter was important for us to achieve our best results in the open-world model. We present the number of False Negatives (out of 40 actual positives) and False Positives (out of 860 actual negatives) for combined OSAD (metric 5, set 4) on `twitter.com` in Figure 5. We see that a high cost parameter leads to more false positives (uncensored sites that were classified as censored) and false negatives (censored sites that were classified as uncensored), which can be avoided with a lower cost parameter such as 0.25.

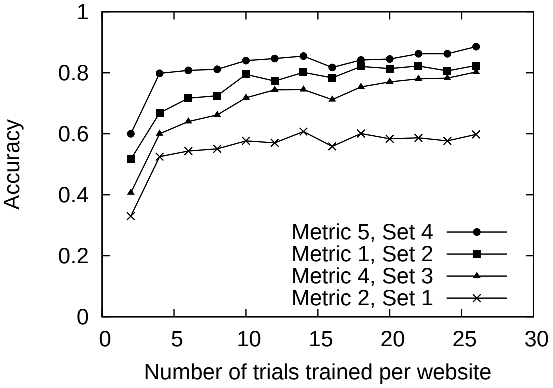

Figure 3: Effect of varying number of training trials on accuracy in closed-world results

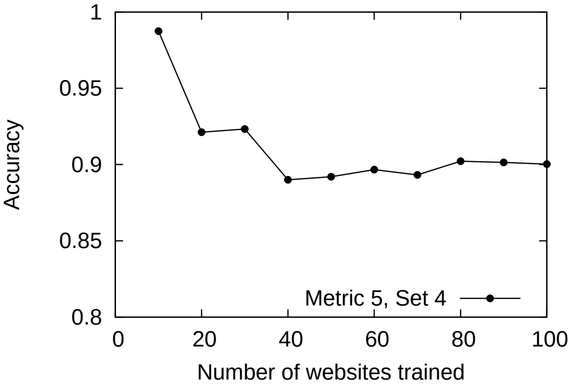

Figure 4: Effect of varying number of training sites on accuracy in closed-world results. Note that the y-axis is not 0-based.

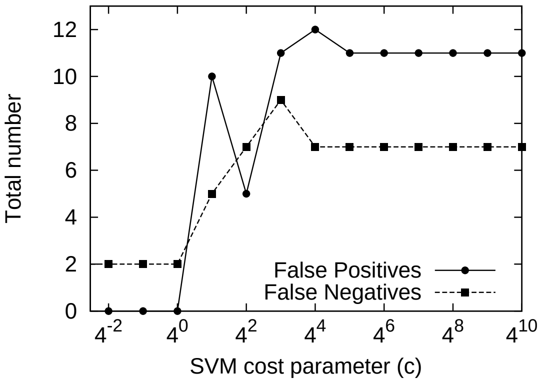

Figure 5: Effect of varying SVM cost on false negatives and false positives in open-world results

#	Site	Acc.	Mistaken as
1	blogspot.com	0.484	blogger.com (81%)
2	blogger.com	0.547	blogspot.com (87%)
3	msn.com	0.564	yahoo.de (11%)
4	tudou.com	0.584	mediafire.com (12%)
5	bbc.co.uk	0.603	godaddy.com (13%)

Table 6: The most difficult sites to classify with corresponding Accuracy (Acc.). The last column is the site which the incorrectly classified site was most often confused as, with the corresponding percentage (out of misclassified instances).

7.3 Difficult sites and Defenses

To understand how a site may defend itself against website fingerprinting, we looked at the most difficult sites to classify. The five most difficult sites to classify, with their corresponding classification rates, are shown in Table 6. The classification rate is a mean over all set-metric combinations that yielded an accuracy of 0.7 or above. We see that `blogspot.com` and `blogger.com` are extremely commonly mistaken for each other. It turns out the reason for this is simply that *these are the same site, with two different names*, so these "misses" were actually correct classifications. Apart from those, we seek to understand why the remaining commonly mistaken sites were significantly more difficult to classify. After some analysis, we observed a number of contributing factors, as follows:

Localization. We loaded `msn.com` without localization specification, and so it often redirected clients to completely different pages with different layouts. For instance, `ca.msn.com` and `plasa.msn.com` would always load different data. In the case of `bbc.co.uk`, specifying the localization of the site we wished to access did not stop the site from loading localized data. This particular site shows different news stories to UK residents than to international clients.

Updating content. Sites 3 and 5 were both active news sites that changed their content more than daily. Site 4 was a video site that constantly updated itself based on the latest popular videos, which are recommended to clients on the front page. Our data collection process could not keep up with their content updates.

Gradual content. Sites 3 to 5 all involved automatic slide shows. The images in these slide shows are not always loaded before the page is considered completely loaded. Rather, they are loaded when the slide itself is loaded. Therefore, the number of slide images that are loaded depended on the total load time, which varies across different circuits in Tor.

Randomized content. Site 4 involved randomized recommendations based on client preferences. As these recommendations would change across different accesses to the site, they are random to the Tor client.

The above strongly suggests that sites are harder to identify if they involve significant amounts of dynamic content, especially if such dynamic content affected the page size. Web designers themselves may help to protect the client's privacy this way. In addition, Tor may be configured to load random localizations of popular sites as a defense mechanism.

8 Conclusion

In this paper, we demonstrated improved website fingerprinting techniques on Tor. These attacks can be performed by a single observer, who may be any one of the hundreds of volunteer entry guard relays in the Tor network, or an attacker tapping the link between a client and her entry guard. We observed that measuring Tor cells, rather than TCP/IP packet sizes, yields more accurate information, and removing the Tor SENDME cells reduces the noise further. We described a new distance-based metric (*combined OSAD*) for comparing packet traces, and also showed a much faster (albeit less accurate) metric that an attacker with limited computation resources can easily perform.

Using our improved techniques, we demonstrated a marked improvement in accuracy with open-world experiments on Alexa's top 1000 sites to emulate a potential censor. With our techniques, the recall was above 95% on four commonly blocked sites and the false positive rate was less than 0.06% for those sites. It is possible that these results can be improved further with more sophisticated multi-class training approaches or some fine-tuning to the parameters used in the experiments. To compare our results with those of previous authors, we also performed closed-world experiments on Alexa's top 100 sites with 40 instances each, and we showed that our new metrics and data processing techniques yielded up to 35% fewer mistakes than previous work. Our results serve to warn us that even with TLS encryption, padding, and packet relaying, Tor may not be able to protect web-browsing clients from deanonymization by a passive observer with limited resources.

Acknowledgements. We are very grateful to Cai et al. for sharing their code and data with us, which greatly expanded the scope of our work. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:`www.sharcnet.ca`) and Compute/Calcul Canada. We gratefully acknowledge NSERC, the Ontario Research Fund, and The Tor Project for funding this work.

References

[1] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. Voelker. DefenestraTor: Throwing out windows in Tor. In *Proceedings of the 11th Privacy Enhancing Technologies Symposium*, pages 134–154. Springer, 2011.

[2] 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, 2012.

[3] J. R Crandall, D. Zinn, M. Byrd, E. Barr, and R. East. Conceptdoppler: a weather tracker for internet censorship. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 1–4, 2007.

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

[5] 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 *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[6] 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, 2009.

[7] M. Liberatore and B.N. Levine. Inferring the source of encrypted HTTP connections. In *Proceedings of the 13th ACM conference on Computer and Communications Security*, pages 255–263, 2006.

[8] N. Matthewson. Tor and the BEAST SSL attack. `https://blog.torproject.org/blog/tor-and-beast-ssl-attack`, September 2011. Accessed Feb. 2013.

[9] N. Matthewson and M. Perry. Tor Controller. `https://svn.torproject.org/svn/blossom/trunk/TorCtl.py`, September 2007. Accessed Feb. 2013.

[10] B. J. Oommen and R. K. S. Loke. Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions. *Pattern Recognition*, 30(5):789–800, 1997.

[11] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th ACM Workshop on Privacy in the Electronic Society*, pages 103–114, 2011.

[12] M. Perry. Experimental Defense for Website Traffic Fingerprinting. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`, September 2011. Accessed Feb. 2013.

[13] R. Pries, W. Yu, S. Graham, and X. Fu. On performance bottleneck of anonymous communication networks. In *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–11, 2008.

[14] Tor. Tor Metrics Portal. `https://metrics.torproject.org/`, July 2012. Accessed Feb. 2013.

[15] V.N. Vapnik and A.J. Chervonenkis. *Theory of pattern recognition*. Nauka, 1974.

[16] R. Wagner and R. Lowrance. An extension of the string-to-string correction problem. *Journal of the ACM (JCM)*, 22(2):177–183, 1975.

[17] J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the seventh European symposium on artificial neural networks*, pages 219–224, 1999.

[18] C.V. Wright, S.E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the 16th Network and Distributed Security Symposium*, pages 237–250, 2009.

A Details of the Optimal String Alignment Distance

The optimal string alignment distance can be calculated with the dynamic programming algorithm roughly described as follows. Given two strings s_1 and s_2 with m and n characters respectively, we construct a matrix M of size m by n. The element of matrix M at row i and column j is the distance between s_1 up to element i and s_2 up to element j. Elements are calculated one by one starting from $i = 1$ and $j = 1$, where each element is calculated based on the previously-calculated elements at positions $(i-1, j)$ (insertion), $(i, j-1)$ (deletion), $(i-1, j-1)$ (substitution), and $(i-2, j-2)$ (transposition) to minimize cost. Transpositions are only allowed when the last two elements of the two strings at i and j are indeed a transposition of each other. The algorithm is given in Algorithm 2. [10]

Algorithm 2 Optimal string alignment distance

```
Input: Strings $s_1, s_2$ with $|s_1| = m$ and $|s_2| = n$; insertion/deletion cost $cost_{id}$, substitution cost $cost_{sub}$, transposition cost $cost_{trans}$
Output: OSAD of $s_1$ and $s_2$
Initialize matrix $M$ of dimensions $m$ by $n$, with:
$M(i, 0) = i \cdot cost_{id} \quad \forall\, 0 \leq i \leq m$
$M(0, j) = j \cdot cost_{id} \quad \forall\, 0 < j \leq n$
for $0 < i \leq m, 0 < j \leq n$ do
    if $s_1(i) = s_2(j)$ then $cost_{idt} = 0$
    else $cost_{idt} = cost_{id}$
    end if
    $M_{ins} = M(i-1, j) + cost_{idt}$
    $M_{del} = M(i, j-1) + cost_{idt}$
    $M_{sub} = M(i-1, j-1) + cost_{sub}$
    if $s_1(i) = s_2(j-1)$ & $s_1(i-1) = s_2(j)$ then
        $M_{transpose} = M(i-2, j-2) + cost_{trans}$
    else
        $M_{transpose} = +\infty$
    end if
    $M(i, j) = \min\{M_{ins}, M_{del}, M_{sub}, M_{transpose}\}$
end for
Return $M(m, n)$
```

B Details of the Fast Levenshtein-like Distance

In this algorithm, the transposition and deletion costs are input as parameters. All elements in each string are initially marked as unused. We proceed as follows for each element k in s_1. We find the first identical element, k, in s_2 that is unused. The difference in position between these elements is multiplied by the transposition cost and then added to the total transposition distance. Then, we mark that element in s_2 as used. If for some element in s_1 we cannot find any identical element in s_2 that is unused, the deletion distance is incremented by the deletion cost. After all elements in s_1 have been processed, the deletion distance is further increased by the number of elements in s_2 that remain unused, multiplied with the deletion cost. The total distance is the sum of the transposition distance and the deletion distance.

The above algorithm can be efficiently implemented by first building a dictionary D of all elements in s_1, such that for each element k, $D(n) = \{D_1(k), D_2(k), \ldots, D_j(k)\}$ is a list of all positive integers such that the element in s_1 at the position $D_i(k)$ for $1 \leq i \leq j$ is k. The dictionary is computed in linear time and is used in the algorithm. The algorithm using the dictionary is given in Algorithm 3; it is equivalent to the above description. In our experiments, we take the transposition cost to be 0.01, the outgoing packet deletion cost to be 4, and the incoming packet deletion cost to be 1.

Algorithm 3 Fast Levenshtein-like distance

```
Input: Strings $s_1$, $s_2$ with $|s_1| = m$ and $|s_2| = n$, dictionary $D$ for element positions in $s_1$; insertion/deletion cost $cost_{id}$, transposition cost $cost_{trans}$
Output: Levenshtein-like Distance of $s_1$ and $s_2$
Initialize $cost = 0$
for $0 < i \leq n$ do
    if $D(s_2(i)) \neq \emptyset$ then
        $cost = cost + |D_1(s_2(i)) - i| \cdot cost_{trans}$
        $D(s_2(i)) = D(s_2(i)) \backslash \{D_1(s_2(i))\}$
    else
        $cost = cost + cost_{id}$
    end if
end for
$cost = cost + |D| * cost_{id}$
Return $cost$
```

C Full results for all metrics and data sets

Here we show the mean and standard deviations of 10 accuracy measurements for each combination of metric (from Table 1) and data set (from Table 2).

Table 7 shows the results for the data we collected, and Table 8 shows the results for the data collected by Cai et al. [2]

Table 7: Results on traffic traces we collected for each metric (M.) and set (S.)

	S.1	S.2	S.3	S.4
M.1	.74 ± .10	.87 ± .03	.86 ± .06	.83 ± .06
M.2	.65 ± .11	.82 ± .05	.88 ± .06	.87 ± .06
M.3	.55 ± .13	.72 ± .08	.89 ± .06	.90 ± .06
M.4	.74 ± .10	.87 ± .03	.86 ± .06	.83 ± .07
M.5	.46 ± .13	.65 ± .09	.89 ± .06	.90 ± .06
M.6	.20 ± .04	.23 ± .06	.21 ± .05	.22 ± .05
M.7	.46 ± .11	.53 ± .08	.69 ± .06	.70 ± .07

Table 8: Results on traffic traces collected by Cai et al. for each metric (M.) and set (S.)

	S.1	S.2	S.3	S.4
M.1	.82 ± .02	.86 ± .02	.84 ± .02	.87 ± .02
M.2	.71 ± .04	.84 ± .02	.88 ± .02	.89 ± .02
M.3	.65 ± .07	.79 ± .04	.90 ± .02	.87 ± .02
M.4	.82 ± .02	.86 ± .02	.84 ± .02	.87 ± .02
M.5	.59 ± .06	.76 ± .04	.91 ± .02	.91 ± .04
M.6	.12 ± .03	.14 ± .06	.09 ± .04	.10 ± .04
M.7	.54 ± .03	.52 ± .03	.70 ± .02	.71 ± .02