HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

Defending against Tor Website Fingerprinting with Selective Cover Traffic

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

eingereicht von: Michael Kreikenbaum
geboren am: 13.09.1981
in: Northeim

Gutachter: Prof. Dr. Konrad Rieck (Universität Braunschweig)
Prof. Dr. Marius Kloft (Universität Kaiserslautern)

eingereicht am: . . . . . .

Summary

The anonymity network Tor is under attack. It provides access to an uncensored internet for people in oppressive regimes, and enables for example whistleblowers to report wrongdoing with no fear of retribution. If its anonymity were to be reduced, this would hamper its purpose, both in actual terms, and because this would dissuade users, as the number of users increase its anonymity.

Website fingerprinting can allow a local passive adversary, a very limited attacker, to fathom which websites a user is visiting, reducing Tor's anonymity. This has continued to improve as AI continues to improve, and researchers publish new website fingerprinting attacks against Tor.

This thesis presents a new defense against website fingerprinting. The defense creates camouflage traffic tailored to WWW distributions to hide which page the user visits. It is easily configurable for the level of required anonymity, and does not burden the user overly, neither in installation, nor in the time and size delay it imposes.

Zusammenfassung

Das Tor-Netzwerk wird angegriffen. Es gibt Menschen in Diktaturen die Möglichkeit, sich frei zu informieren, und erlaubt unter anderem Whistleblowern, ohne Furcht vor Vergeltung auf Verbrechen hinzuweisen. Eine Angriff auf Tors Anonymität reduziert die konkrete Nutzbarkeit, und bringt Nutzer von Tor ab. Dies würde die Anonymität zusätzlich vermindern.

Website Fingerprinting erlaubt einem minimalem, lokal passiven Angreifer zu erkennen, welche Websites besucht werden. Somit wird Tors Anonymität reduziert. Diese Angriffsart wird immer weiter verbessert, sowohl mit Fortschritten in KI, als auch mit neuen publizierten Angriffen gegen Tor.

Diese Arbeit präsentiert eine Verteidung gegen Website Fingerprinting. Die neue Verteidung erzeugt an WWW-Datenverkehr angepasste Camouflage-Daten, um die besuchte Seite zu verbergen. Die Benutzer können die Verteidungsstärke an die benötigte Schutzklasse anpassen. Die Verteidigung ist leicht zu installieren sein und verzögert Tor weniger als vergleichbare Verteidungen.

Contents

List of Figures

List of Tables

Glossary

AUC area under the ROC curve. 15, 38

FN false negatives. 14

ML machine learning. 8

OH overhead. 32, 51, 53

ROC receiver operating characteristic. 15, 38

SCT Selective Cover Traffic. 2, 7, 27, 28, 29, 30, 31, 32, 33, 32, 33, 34, 35, 38, 46, 47, 48, 56, 64

SVM support vector machine. 7, 12

WF website fingerprinting. 31

1. Introduction

Many parties like to observe users' online browsing: shops and advertisers want to tailor advertisement to focus groups; people who share your wireless LAN might just be curious; governments could aim for dragnet crime prevention, steering public discourse, and sometimes censorship; criminals want to create better scams. Against all these, Tor [1] helps individuals protect their online privacy. They can browse censored information [2], and for example publish leaks without fear of retaliation. Several constitutions implicitly [3] or explicitly [4, Art.10] acknowledge that communication privacy is fundamental to an open society [5, ch.10]. Tor protects many and diverse parts of society, such as journalists, businessmen, and military units deployed abroad.

Tor [1] works by creating an encrypted path through its network of servers. These servers are run by volunteers located all over the world. A local software on a client's computer negotiates this path step-by-step: It uses encryption to ensure that none of the forwarding servers can know the full path of the message. For example, a response from a web server is sent back to the local software along the same path. This is how Tor makes it possible for two parties to communicate anonymously over the internet.

People need to trust that Tor truly protects their online privacy [6]. The more they doubt this, the less they would use Tor, or the more they would self-censor their online activities. One as-of-yet-unfixed vulnerability in Tor is website fingerprinting. In website fingerprinting, the sizes and timing of each data packet for a web page retrieval are collected to form a *fingerprint*. This is matched to other observed retrievals to predict which page was visited. Originally, website fingerprinting is briefly mentioned by Wagner and Schneier [7]. Panchenko et al. [8] are the first to successfully attack Tor by website fingerprinting under laboratory conditions. The Tor project encourages research into website fingerprinting attacks and defenses against Tor [9].

Various defenses have been proposed so far ([10], [11], [12], [13], [14], [15], [16] [17], [18]). None have been deployed with the Tor Browser. The reasons are manifold: Defenses delay [19] web browsing, require tuning by hand or changes to the Tor code base, complicated setup, are only presented conceptually, etc. Still, website fingerprinting attacks have continued to improve ([20], [21], [10], [22], [8], [23], [14], [24], [25], [26]).

1.1. Contributions: a New Website Fingerprinting Defense and Tools

This thesis tries to find if a HTTP-specific defense increases protection for the same overhead.

A new defense should be easy to use, easy to deploy, and hinder users as little as possible. It should use resources responsibly, that is: it should try to get the best possible defense from a given resource level. Such a defense is attempted in this thesis. Ideally, it should also be easy to configure.

Selective Cover Traffic (SCT) tightly integrates into the Tor Browser; it tries to make each fetch of a webpage look like another random webpage; it also uses the Tor Browser's extension system for easy configurability, and deployability. SCT sets sensible defaults, allowing users to adjust its resource use to account for their personal security needs.

SCT needs to be compared to existing defenses: This thesis reimplements and validates the state-of-the art attack by Panchenko et al. [24]; it also provides a way to record web page browsing via Tor. The assorted data in trace format includes more than 150 capture sets. A capture set took more than a day on average. More than 50 captures are without defense. This data needs more than 20 GB of disk space, in a format similar to those of Wang et al. [14] and Panchenko et al. [24]. The included code can convert from and to their formats.

1.2. Structure: Foundations, Design, Evaluation

The organization of this thesis follows thesis best practices: a foundational chapter is followed by the problem statement and design, which is evaluated in the following chapter. The final chapter contains a conclusion with future work.

Chapter 2 introduces website fingerprinting, Tor, and defenses against website fingerprinting on a need-to-know basis. Section 2.1 gives a gentle introduction to Tor, with an example how Tor achieves privacy. Section 2.2 explains how website fingerprinting works in general and against Tor in particular. A big part of website fingerprinting is machine learning, which is explained in Section 2.3. How to defend against website fingerprinting is presented in Section 2.4.

This thesis' new website fingerprinting defense is introduced in Chapter 3.1. It also presents the aspects the defense is based upon: Section 3.2 justifies the chosen HTTP traffic model; whereas the stochastic size cache based on Bloom filters [27] is described in Section 3.3.

Chapter 4 puts the defense to the test, analyzing how it compares to the most recent website fingerprinting defense by Cherubin et al. [18]. The analysis follows website fingerprinting literature best practices: it first evaluates how a subset of sites are distinguished one from the other in Section 4.2; it then tries to distinguish these sites from a bigger set of background sites in Section 4.3. All results are summarized in Section 4.5.

Chapter 5 summarizes the findings and presents courses of future work in Section 5.2.

2. Tor Basics, Attacking and Defending with Website Fingerprinting

2.1. How Tor Works

In the wake of both the Snowden revelations in the western world, and increased internet censorship in countries such as Iran, Saudi-Arabia, and China [28], more and more Internet users search for ways to keep online communication and web browsing both private and free of censorship. The *Tor* project [1] provides this. It protects whistleblowers, journalists, the people in oppressive regimes [2], even the military, and regular internet users, against for example nation-states or businesses which want to follow user's online steps. It routes encrypted data traffic via intermediaries, obscuring who connects to whom.

Let us conceive of the internet as a series of tubes. Each internet message sent from Alice [29, sec.II] to Bob passes many of their connections. At each joint, there are many paths in and out. The message needs to find a way to Bob, so it contains Bob's address on the envelope. In case Bob wants to answer, the envelope also contains Alice's address.

This is why the internet is not anonymous by design. To provide partial anonymity, a group of tube intersections can join, wrap each data packet in layers (of encryption), and bounce it along the group randomly, unwrapping a layer at each bounce. After several bounces, for example to Carol, Dave, and Frank, the data packet is completely

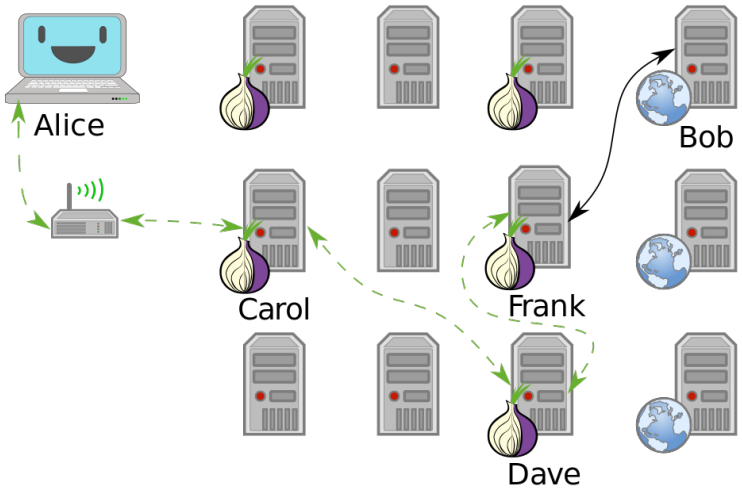

Figure 2.1.: Tor Network. Tor-protected links are dashed and Tor green [30]. Onion icon marks Tor servers, ©Tor project. Inspired by Wikipedia.

unwrapped again. Its destination is Bob, but Alice's name is blotted out. Frank sends the packet to Bob. To answer the packet, Bob sends the packet back to Frank, who sends it via Dave and Carol back to Alice. Because encryption, Alice knows the full path, but Carol only knows Alice and Dave, Dave only knows Carol and Frank, and Frank only knows Dave and Bob.

This closely models the Internet: Each Internet Protocol [31] packet lists the sender and destination. This makes it easy to identify communication partners. To achieve anonymity, the Tor software forms a path to the destination along multiple hops, establishing separate encryption with each hop. The hops are globally-distributed volunteer servers. Each intermediary hop only knows its predecessor and successor. Only Alice knows the full path.

The local Tor software selects three globally-distributed hops to initialize a connection. It makes a connection to the first, establishes encryption, asks the first hop to make a connection to the second, sets up encryption to this, and from there to the third. The third hop establishes a connection to its destination.

Each message is encrypted three times using same-length encryption and sent along this path. The first router decrypts the first layer, and so on, like layers of an onion. As a result of this setup, each hop can only see its direct neighbors along the path. Even if one hop of a three-hop setup is compromised, directly linking source and destination becomes pretty hard.

While the above allows Tor's users to anonymously use all TCP-related servers, which comprise most of the internet, Tor also provides server anonymity via its location-hidden services[32, sec.5]. Here, a host's IP address is hidden behind the Tor network. Clients need to know the host's `.onion` address. Using this, they can contact a rendezvous point, which holds a Tor connection to the server, and forwards all data along this connection. Thus, the server's IP address stays hidden from all actors. As of Cherubin et al. [18, sec.5.2], web services that run behind Tor hidden services tend to have smaller HTML-pages and contain less embedded objects.

2.2. Website Fingerprinting Attack

Some groups dislike other people's privacy. It's too resource-intensive to protect against all of them. Tor, as any privacy system, has elected to protect against certain threats [1]. For example, Tor does not protect against an adversary that can see all Tor network traffic; this level of observation would make correlation attacks ([33], [34]) easy [32, sec.9]. Tor is designed to defend for example against a local passive adversary: someone who can see the traffic from a client to the network.

Let us assume that Eve is a concerned mother who wants to find out whether Alice visits Bob's website, see Figure 2.2. If Alice is just using a vanilla web browser, Eve can see where Alice connects. Alice needs to use a middle-man. She downloads the Tor Browser Bundle. This program routes all traffic via three Tor servers, for example Carol, Dave, and Frank. These three are regularly replaced with other random nodes. Eve is frustrated: she can not see whom Alice connects to. Also, all traffic is encrypted, so she can only see that packets are sent, not their content.

Eve is not to be thwarted: website fingerprinting to the rescue! She uses Tor on her

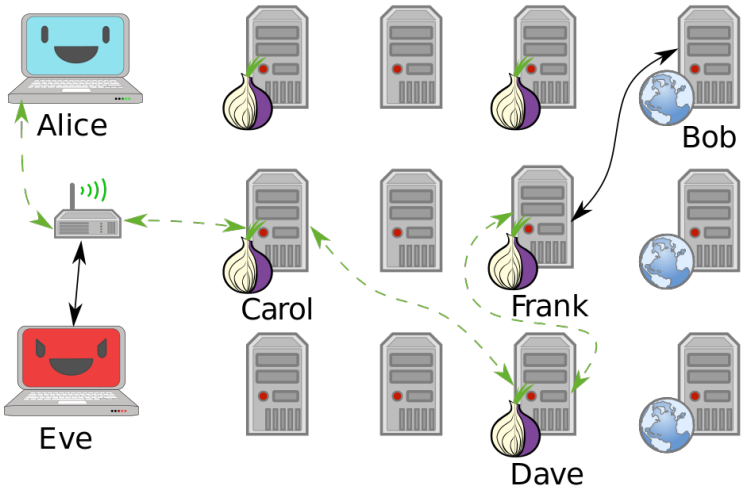

Figure 2.2.: Tor Network as in 2.1. Alice's mother Eve can see all of Alice's WLAN traffic, and tries to perform website fingerprinting.

own computer to both connect to Bob's website, recording the network traffic of Bob's site, as well as other random web browsing traffic. (She could of course just block Tor, but then Alice might sneak off in the middle of the night to see Bob.) This traffic trains a machine learning classifier. When Alice uses Tor, her traffic is input to this classifier, which can decide in almost real-time whether the site Alice visits is Bob's. If so, Eve can devise targeted ways to keep her daughter occupied with other things.

In website fingerprinting, the time and size of users' data packets (called *traces*) are recorded. Figure 2.3 shows a visualization of these traces: The similarities can be expected, as every retrieval retrieves similar content. Website fingerprinting distinguishes between 30 and many more sites with high accuracy ([24], [14] [25], [26]). The main attack used to test SCT is CUMUL [24]. Section 2.3.1 describes its features. CUMUL's features are analysed using a support vector machine (SVM) classifier [35] with 10-fold cross-validation, which is described in 2.3.2.

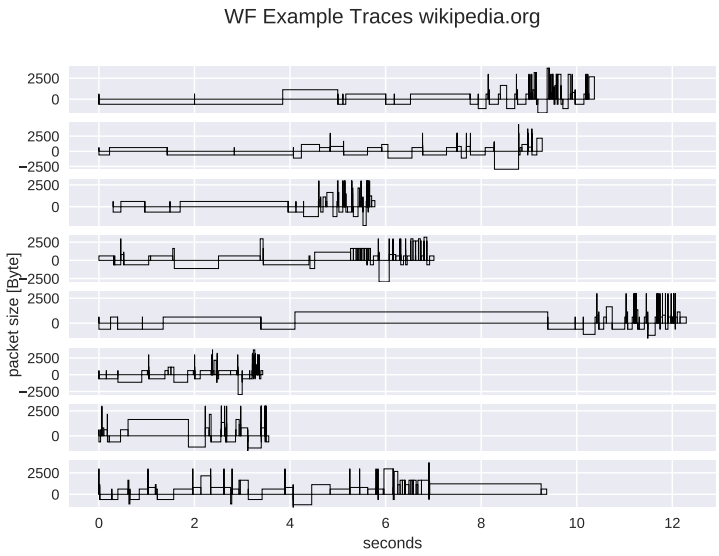

Figure 2.3.: Example of traces of wikipedia.org. Box width is the time to the next packet, box height the size of the packet (positive incoming, negative outgoing). The top traces seem similar to the naked eye (modulo time dilation). The bottom trace does not fit this pattern. The whole of Wikipedia's traces were recognized with 100% accuracy in a set of 30 sites. (overall accuracy 98.57%)

2.3. Machine Learning

In machine learning, a computer [36] algorithm extracts and generalizes patterns from learning data [37, ch.1.2]. This abstraction is used to classify further patterns (for example for handwriting recognition [38, sec.11.7]), or to act on the generalizations (for example for self-driving cars [39]). For the purpose of this thesis, machine learning is more of a black box: it transforms traces into a website prediction.

Few real world examples are just given as a vector of numbers [40, sec.1.3.1]. Thus, machine learning needs some prior work: *Feature extraction* (sec. 2.3.1), uses domain-specific knowledge to extract meaningful features. It also has to transform raw input data — in our case, website traces — into *features* — in our case, numbers, for example the number of outgoing packets. All these features are combined into classifier input. Classification (sec. 2.3.2) takes feature extraction's output as input. Its task is two-fold: it first generalizes from training data. Trained, it assigns input data into categories. In order to evaluate how well this works, Section 2.3.3 presents measures to evaluate machine learning performance.

2.3.1. Feature Extraction

The natural world has an abundance of information. Russell and Norvig [41, ch.24] mention that even a 1 megapixel camera, sampled at 60Hz, produces more than 10 GB of data per minute. The more features are used in machine learning, the higher the amount of data required to correctly train a classifier [42, sec.1.2.3]. Knowing the subject domain can help in condensing information: the aim is to keep class-specific information, and to discard individual attributes and noise [40, sec.1.3.1]. Some classifiers expect their input as elements of for example Hilbert spaces [43, sec.1.3.3]. In this case some types of input, for example email texts, *must* be transferred to some other representation.

Feature extraction is highly domain-dependent [40, sec.1.3.1]. Let us examine website fingerprinting features en detail: website fingerprinting input data needs to be wrangled [44] for the classification to work: extra information that might change from request to request — such as IP addresses [31], or the absolute time of the retrieval — needs to be removed or unified to a common format. The trick is as always: keeping the signals and discarding the noise; in other words: finding those features with the biggest difference of the class means relative to class standard deviations [40, sec.10.14.2].

The source data in website fingerprinting are traces, for example in `pcap` [45] format. From these, only the size, and timing of each packet is extracted. The packet direction is encoded in the size ([8, sec.3.1], [15, sec.3.1]), either positive or negative. These uniquely describe the web retrieval for website fingerprinting [15, Fact 1]. The size of files is hidden by the traffic's encryption; the closest approximation [46] is the size of each TLS [47] record. Alternatives to this are sizes of TCP [48] packets, Tor [1] cells, or IP [31] packets. As of Wang and Goldberg [46], these work similarly well.

The earliest website fingerprinting attacks ([20], [21], [10]) only use packet sizes as features to attack SSL [49]. [1, sec.7] conjecture that website fingerprinting against Tor would be hampered by Tor's fixed (data) cell size [50, sec.0.2,3]. Five years later, [22] confirm this resilience in comparison with other privacy-enhancing technologies, but still show slightly better-than-random classification.

Panchenko et al. [8] increased closed-world website fingerprinting accuracy on Her-

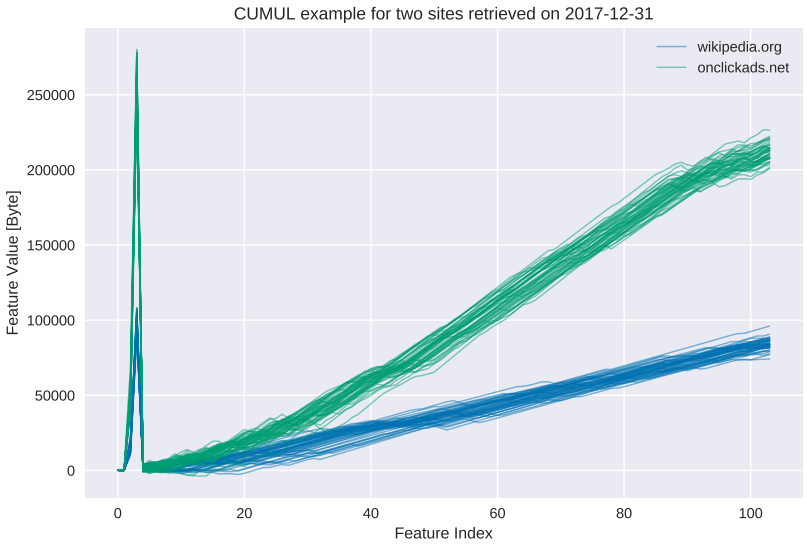

Figure 2.4.: CUMUL features example

rmann et al. [22]'s dataset from 2.96% to 54.61% (Section 2.3.3 defines accuracy etc.). They add website fingerprinting-specific features, such as the percentage of incoming packets, the estimated size of the HTML [51] page, etc. Wang et al. [14] use close to 4000 features. On a smaller dataset, they achieve a true-positive rate of 84%, with a false-positive rate of 0.6%. Panchenko et al. [24] use a cumulative size metric, CUMUL. As seen in Figure 2.4 these provide a graphical representation of traces, while still allowing for computer-based comparison after normalization. To create this, they sum the incoming and outgoing bytes, producing a list of steps of uploaded and downloaded bytes. To extract the same number of features from each trace, they interpolate 100 data points from these. After feature extraction, CUMUL removes outliers from the data [24, sec.V-B]. Hayes and Danezis [25], in addition to the website fingerprinting attack, also measure how much each feature adds to classification. Their approach uses approximately 150 features with a unique Random Forest [52] fingerprint classification that appears to use error-correcting output codes ([25, sec.3.1], [53]).

2.3.2. Classification

Website fingerprinting *classification* tries to assign a website to a trace's features extracted in the previous step. The previous step — feature extraction — transforms traces (raw input data) to features. In *classification*, these features are used for two purposes: some traces' features are used for training the classifier, others for testing.

For Mitchell [42, sec.1.1], in machine learning, a computer program needs to learn how to do a task *T*. It learns from experience *E*. The program's success is measured by a performance measure *P*. The program is said to *learn* from *E*, if it improves at *T* as measured by *P*. Muller et al. [54] provide a formal definition: The task of (binary) classification is to approximate a function $f : \mathbb{R}^n \to \{-1, 1\}$. The given *training data* points $(\mathbf{x}, y) \in X \times Y = \mathbb{R}^n \times \{-1, 1\}$ are drawn from an independent and identically distributed (i.i.d.) distribution. Using these, a hypothesis h [41, sec.18.2] can be estimated. The aim is to minimize the *hypothesis's error*

$$R[h] = \int l(h(\mathbf{x}), y) \mathrm{d}P(\mathbf{x}, y) \tag{2.1}$$

with an adequately defined *loss function* l, for example $l(h(x), y) = 0$ if $h(x) == y$ else 1. As the probability distribution P is not known, the hypothesis error is often estimated as the *empirical risk*

$$R_{emp}[h] = \frac{1}{n} \sum_{i=1}^{n} l(h(\mathbf{x_i}), y_i) \tag{2.2}$$

over all points of training data. As of Smola and Vishwanathan [43, sec.1.1], this can be extended to multi-class-classification, where Y contains more than two labels. More and other classes of input and output are possible. Russell and Norvig [41, sec.18.1] makes the distinction that class labels are not always provided. Providing the labels is called *supervised learning*. Finding patterns in unlabeled data is called *unsupervised learning*. In between these, there is also *semi-supervised learning*, where some data is labeled, and/or these labels are not necessarily accurate (for example user-reported ages). *Reinforcement learning* provides guidance only after the fact. This guidance is only in the form of "yes, you did well", or "no, do better next time".

While classification input can have many types, in website fingerprinting it is always a vector, most often of numbers. In classifier *training* [43, sec.1.3.1], a classifier

gets as input several feature vectors $\{x_1, \ldots, x_n\} \subset X$ and their respective classes $\{y_1, \ldots, y_n\} \subset Y$ as pairs (x_i, y_i) and tries to generalize a relationship. This combination of feature vectors and their classes is called *training data* [38, sec.2.2]. In actual *classification*, the classifier only receives input feature vectors, and needs to predict the class label. In website fingerprinting, this is: the web page. This data is called *test data* and is used to test classifier performance.

What happens in classification depends on the classifier. Most classifiers, such as SVMs [35] form an internal model from which further input data is classified. Others, notably k-Nearest-Neighbors, classify directly without an intermediary model. SVMs [35] are a linear classifier: they find a linear boundary between points. Given a set $X = \{x_1, ..., x_n\}$ with a dot product $\langle \cdot, \cdot \rangle : X \times X \to \mathbb{R}$ and tuples $(x_1, y_1), ..., (x_m, y_m)$, with $x_i \in X, y_i \in \{-1, 1\}$ as a *binary classification* task [43, ch.6f]. The SVM's job is to find a separating hyperplane / affine set

$$\{x \in X \mid \langle w, x \rangle + b = 0\} \tag{2.3}$$

such that the data lie on the correct sides: $\langle w, x_i \rangle + b \geq 0$ whenever $y_i = 1$, and $\langle w, x_i \rangle + b < 0$ whenever $y_i = -1$. With added normalization, this can be rewritten as

$$y_i \cdot (\langle w, x_i \rangle + b) \geq 1.$$

A SVM tries to find a hyperplane between two groups of points and maximize its distance to the closest points, called *margin*. If the data points lie such that a line cannot be found, a *soft-margin classifier* introduces slack variables $\xi \geq 0$, which it tries to reduce while maximizing the margin. This alters the equations to $y_i(\langle w, xi \rangle + b) \geq 1 - \xi_i$ for the optimization problem

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + \frac{C}{m} \sum_{i=1..m} \xi_i \tag{2.4}$$

The *error term* C weighs minimizing training errors against maximizing the margin [43, sec.7.2.1]. Yet, straight lines do not always distinguish classes correctly. This would seem a drawback to using SVMs, yet they can compute these not only on the original data, but also on a projected *feature space* $\mathcal{F}$. This allows for complex decision

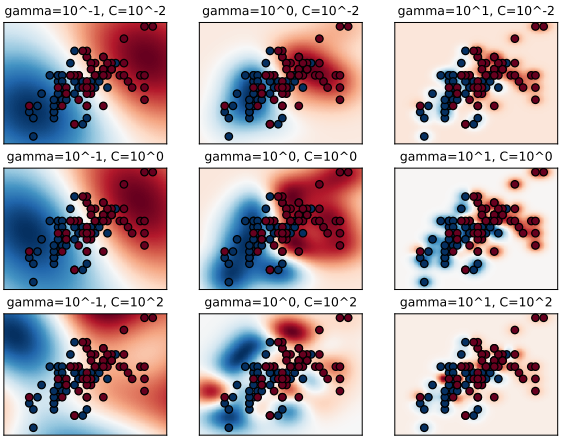

Figure 2.5.: Example svm-rbf classification with different parameters for C and γ. Source [58, Figure 42.328], recreated for higher resolution.

boundaries. By using the kernel trick [55, sec.2.2.2], a SVM can not only use the dot product $\langle \cdot, \cdot \rangle$, but another kernel $k(\cdot, \cdot)$ instead to efficiently find a solution, without explicitly computing the mapping to the feature space [54, sec.III]. The default kernel used by wei Hsu et al. [56] for SVMs is the (gaussian) *radial basis function* (RBF) kernel [57, sec.5]:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\gamma^2}\right). \tag{2.5}$$

The effect of regularization parameter C which weighs how much to penalize outliers and the RBF-kernel width γ can be seen in Figure 2.5. These parameters need to be adjusted to achieve optimal classification accuracy, while avoiding overfitting to the training data. One way to avoid this overfitting is cross-validation [59]: The data set is split into k disjoint subsets, called *folds*, of equal size ± 1. Of those, $k - 1$ are used combinedly for training the classifier, while the last is used for prediction evaluation. In other words, for each run, $k - 1$ folds are used as training data, while the last fold is used as testing data, as shown in Table 2.1. The accuracy over all folds is averaged. CUMUL[24] uses 10-fold cross-validation.

Table 2.1. Cross-validation splits. The training folds are combined, the classifier is trained on them, then accuracy is computed on the test fold. This is done k times, the results are averaged.

folds	1	2	3	4	5
experiment 1	**test**	train	train	train	train
experiment 2	train	**test**	train	train	train
experiment 3	train	train	**test**	train	train
experiment 4	train	train	train	**test**	train
experiment 5	train	train	train	train	**test**

2.3.3. Measuring Performance

To find out if website fingerprinting attacks work, and if defenses prevent them from working, their success can be measured. The simplest form of measurement is simply counting how many traces are classified correctly, and dividing by the total number of traces. This is called *accuracy* [60]. For other types of measurement, a *confusion matrix* helps to illustrate the different cases that can occur in website fingerprinting. See Table 2.2.

This matrix counts the number of classifications. For example, if a trace from *wikipedia.org* is classified as *onclickads.net*, this increases the false negatives (FN) count by 1. Each trace is categorized by whether it *is* a sensitive website (here: wikipedia.org), and whether it is *classified* as such. From these values, metrics can be derived [60]. Apart from *accuracy*, the main metrics used in website fingerprinting literature are *True-*, and *False-Positive-Rate*. True-Positive-Rate is also known as recall. These are defined as

$$\begin{aligned} \text{Accuracy} &:= (TP+TN)/(TP+FP+FN+TN) \\ \text{True Positive Rate} &:= TP/(TP+FN) \\ \text{False Positive Rate} &:= FP/(FP+TN) \end{aligned}$$

There are two scenarios of testing website fingerprinting in experiments, *closed-world*

Table 2.2. (binary) Confusion matrix. Correctly classified traces are in bold. The distinction positive/negative is chosen based on the classification objective.

	real wikipedia.org	real onclickads.net
predicted as wikipedia.org	**True Positives (TP)**	False Positives (FP)
predicted as onclickads.net	False Negatives (FN)	**True Negatives (TN)**

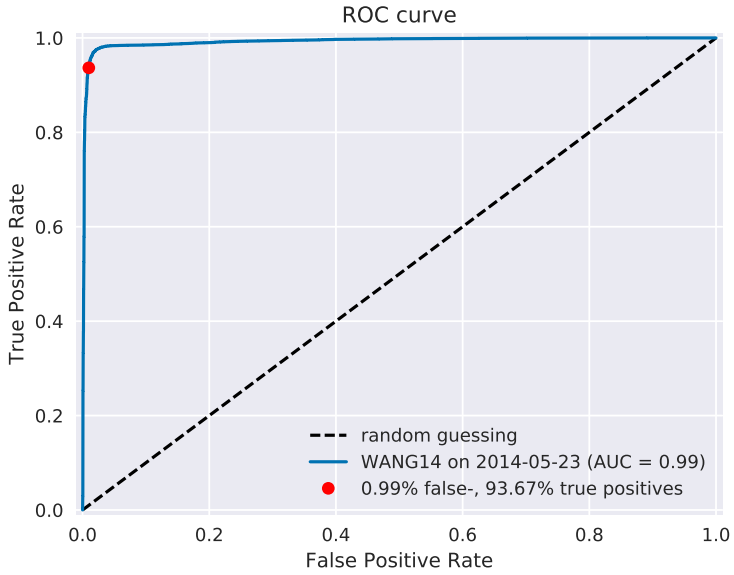

Figure 2.6.: (Example) ROC-curve for the WANG14 [24, sec.4.1] data set. The point at the top left (0, 1) is optimal. The ROC-curve shows possible true- to false-positive ratios. Each point on the curve corresponds to one confusion matrix [61, sec.4.0]: the farther to the right on the curve, the more classification tends towards positive classification.

and *open-world* [8, sec.3.2]. In a closed-world scenario, a fixed number of pages, most often 100, are compared one to the other, the classifier only has to distinguish between these. Accuracy measures this classifier's success. In an open-world scenario, a certain number, for example 5000, of background pages are additionally captured. The previously captured (100) sites are called *foreground* pages. The classifier's task is to distinguish between foreground and background pages. The scenario is that there are certain censored sites, for example wikipedia.org, which need to be distinguished from normal web browsing, for example onclickads.net.

Some classifiers not only yield the input's class, but also output a certainty or probability: This indicates how sure the classifier is of the classification. In this dual scenario of having background and foreground pages, it becomes possible to weight how important each classification is. For example, if a nation state were to raid people's houses if they access a certain website, it better be very sure that they did in fact access the website. If Eve wants to check in on Alice whenever her daughter visits Bob's website, she might prefer some other sites to classify as Bob's.

A receiver operating characteristic (ROC)-curve [61] — see Figure 2.6 — shows the classifier strictness trade-off. This diagram contrasts classifier true-, and false-positive-rate. The area under the ROC curve (AUC) can be measured. The closer this value is to 1, the better. If one is mainly interested in few false positives, the leftmost section of the ROC-curve is of particular interest.

2.4. Defending against Website Fingerprinting

This section describes defenses against website fingerprinting as described in Section 2.2: website fingerprinting attacks have become increasingly better using machine learning. Website fingerprinting thus effectively de-anonymizes the traffic that users thought private: it could for example expose a dissident to his nation state, nullifying this part of Tor's protection. The task of defenses is to confuse such a website fingerprinting attacker. As most machine learning, website fingerprinting uses statistical properties of the underlying data. It could possibly be defeated by shuffling these properties.

Traffic analysis [62] assumes that encryption is unbreakable, and tries to find information from metadata: observable streams of traffic, for example radio wave origin, or IP packet size and timing. From inception, Tor [1, sec.3.1] provides some defense against traffic analysis. For one, all Tor *data* cells have the same size, which protects against identifying them by size only. Tor also multiplexes all traffic into a single stream, making it hard to distinguish the multiple streams that most websites require, let alone parallel website retrieval. Unavoidably, Tor also increases traffic latency [63, sec.2.2], so that attacks have a harder time relying on packet timing [6]. This makes website fingerprinting harder, to the point that it was mentioned, but not hindered, in Dingledine et al. [1, sec.7].

To protect against website fingerprinting, several early attack authors also mention possible defenses: Wagner and Schneier [64, sec.5] propose padding SSL [49] so that HTTP GET [65, sec.4.3.1] URL's sizes would be concealed. Cheng and Avnur [21, sec.3] proposes three possible defenses: an additional padding layer between HTTP and SSL, modifying the web pages themselves, or using web proxies. Hintz [10, sec.8] suggests padding, switching off for example images, and transferring a whole

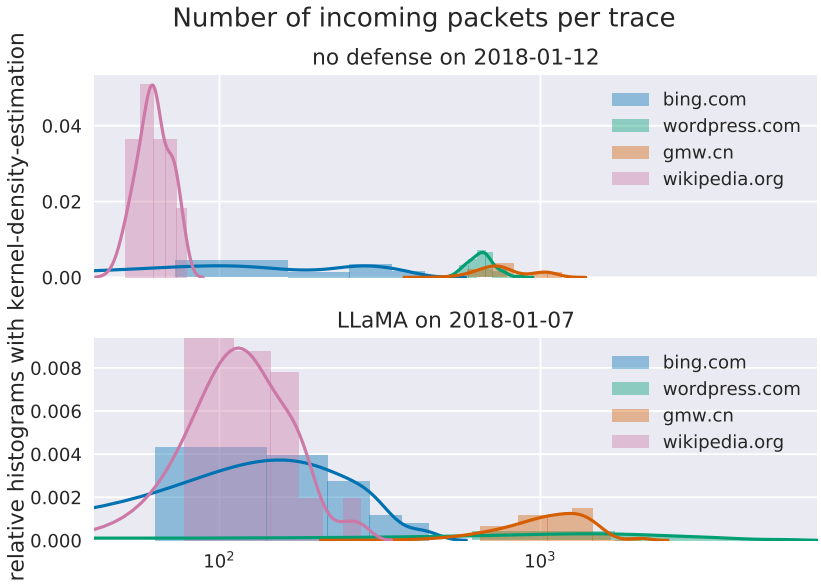

Figure 2.7.: Distribution of number of total incoming packets, once without defense, once using LLaMA [18].

page in one connection.

The first website fingerprinters considered only packet lengths. This made it seem sensible to defend by altering the lengths of packets by padding, as evaluated for example by Cheng and Avnur [21, sec.5.1]. The total number of incoming packets is a feature analysed by almost all modern attacks. Figure 2.7 illustrates the distribution of this main feature. As seen, Cherubin et al. [18] create additional packets, but preserve site separation and ordering.

While early defenses attempt to hinder website fingerprinting in general, not on Tor, several attack authors also propose Tor-specific defenses. Panchenko et al. [8, sec.6] add noise to traffic. They *camouflage* by loading another page simultaneously in the background. Cai et al. [23], Cai et al. [15], Hayes and Danezis [25], Wang et al. [14], Panchenko et al. [8], and Juárez et al. [16] found this simple defense to be surprisingly effective, albeit at a high overhead. As more and more features were used to classify the traces, different ways of altering the data were evaluated by several researchers: several ways of padding ([66], [13], [15], [23], [17]), or altering traffic sizes to fit another web page's ([11], [12]).

Prior to Dyer et al. [13], most website fingerprinting defenses altered specific machine learning features, for example single packet size. This created an arms race between attacks and defenses - the attacks finding new feature combinations to use, the defenses obfuscating these. To stop this, Dyer et al. [13] introduces the idea of a *general defense* into the context of website fingerprinting. The aim is to transform groups of web retrievals so that all members look the same. They propose a traffic-flow security [67, ch.10.3] solution called *BuFLO*: fixed-rate transmission of all data, with dummy traffic for gaps, for the estimated duration of web site retrieval. This idea was improved on by [15]. Both of these exhibit high bandwidth overhead, as they send data at a fixed rate. Furthermore, they have high time overhead, on average more than quadrupling the required time to download a page [15, sec.6.2]. As even sub-second delays cause users to use a service less [19], the effect of quadrupling the load time ought to be studied before adopting these. Wang et al. [14] propose the (offline) defense of morphing all traffic to supersequences of traffic patterns. This defense needs to know the traffic pattern beforehand, which for today's ever-changing webpages seems feasible only for a small fixed page set.

The stochastic defenses of Wang and Goldberg [17] and Juárez et al. [16] have less size overhead (33% and 64% respectively [26]) than the previous deterministic general defenses. Juárez et al. [16] derives its basic mechanism from Shmatikov and Wang [68], which aims at generally hiding *that traffic occurs*, not just which website is visited. It works on the network level. Wang and Goldberg [17] changes the traffic patterns to half-duplex: It either only sends or only receives, with added traffic. To do so, it modifies the Tor Browser Bundle source code. In 2017, LLaMA was introduced by Cherubin et al. [18]. They acknowledge the need for a client-side application-level defense. LLaMA delays traffic by a uniformly distributed delay, up to the median time observed to load a site[69, index.js:18], and uses the URLs of previously retrieved elements to provide cover traffic. It is implemented using the Mozilla Add-on SDK. The authors provide LLaMA as a secondary defense to the server-side ALPaCA defense, and emphasize its prototype status [69, README.md].

3. A New Defense

Website fingerprinting has deanonymized Tor [1] under laboratory conditions [8], and has continued to improve ([24], [25], [70], [26]). Several approaches have been taken to prevent website fingerprinting ([8], [18]). Yet none has made it into mainline Tor [1]. This thesis presents a deployable defense. It uses domain-specific properties of HTTP traffic [71]. This is akin to [72] who show good results in obfuscating faces using eigenfaces.

The question is whether this defense obfuscates effectively: Whether HTTP-specific cover traffic can yield better results than previous stochastic defenses. Most of these used non–subexponential [73] distributions. A bound on this effectiveness are previous defenses. The state-of-the-art CUMUL website fingerprinting attack by Panchenko et al. [24] measures effectiveness of obfuscation.

This chapter first presents the main thesis's defense design in Section 3.1. Section 3.2 goes into some detail about the traffic model. The optional Bloom filter based stochastic data structure that is used for size caches is described in Section 3.3.

The advantages of this approach are: Firstly, the application layer is used in website fingerprinting. The defense works at the same layer. Secondly, the defense works inside the browser, observing requests as they occur. It can thus send specific cover traffic fitted to each single request. Thirdly, the defense could possibly cache element sizes, in order to more closely tailor cover requests. Finally, the defense is provided as a Firefox Add-on. This eases installation and configurability.

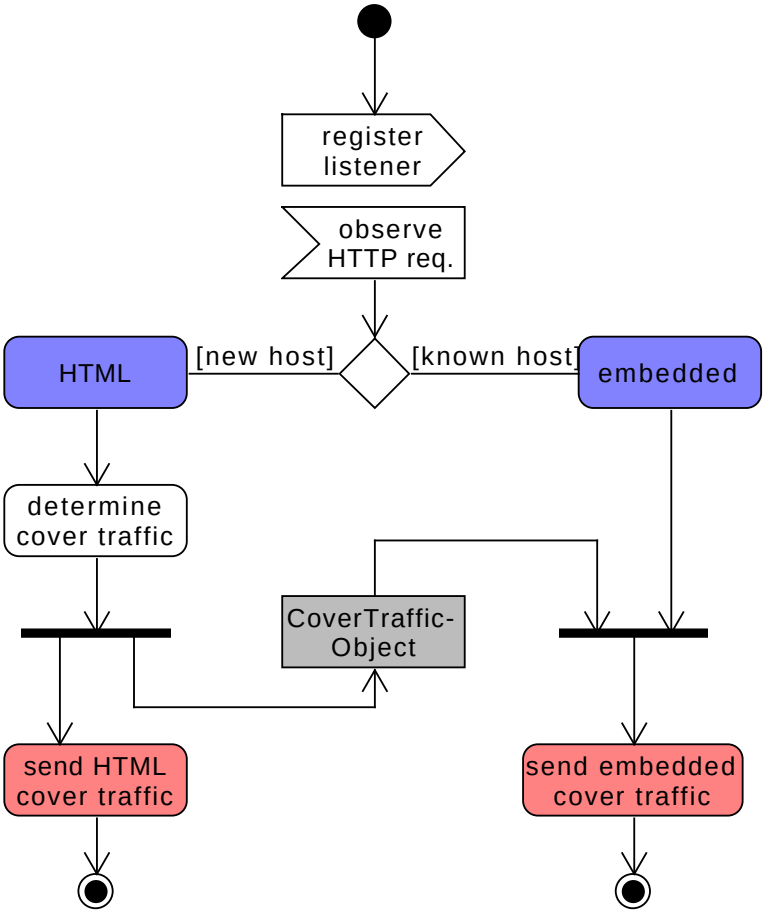

Figure 3.1.: Main functionality flowchart

3.1. Main Defense

The defense obfuscates web traffic: For each load of a website, additional cover traffic is created. To do this, the defense observes all web requests. The defense's main flow is modeled in Figure 3.1: If the website is not known, the defense determines parameters for HTML and embedded cover traffic, and directly sends a cover request for the HTML page. If the website is known, the pre-determined parameters determine the amount of embedded cover traffic sent. This is also described in Algorithm 1. Additionally, it details how the cover traffic parameters are set.

The retrieval of the (first) HTML page provides significant information to a website

Algorithm 1 Generate cover traffic

```
1: procedure OnHttpRequest(url)
2:     if !isRegistered(hostnameOf(url)) then        ▷ unknown hostname: HTML request
3:         targetHttpSize ← randomHttpSize()
4:         targetNumEmbedded ← randomNumEmbedded()
5:         urlHttpSize ← lookupOrGuessHttpSize(url)
6:         urlNumEmbedded ← lookupOrGuessNumEmbedded(url)
7:         coverHttpSize ← targetHttpSize − urlHttpSize
8:         coverNumEmbedded ← targetNumEmbedded − urlNumEmbedded
9:         requestCoverSized(coverHttpSize)
10:        registerHost(hostnameOf(url), coverNumEmbedded, urlNumEmbedded)
11:    else                                          ▷ known hostname: a resource request
12:        requestProbability ← getProbability(hostnameOf(url))
13:        while requestProbability > 1 do           ▷ if > 1, send multiple cover requests, else check probability
14:            requestCoverSized(randomEmbeddedSize())
15:            updateHosts(hostnameOf(url))
16:            requestProbability ← requestProbability − 1
17:        if withProbability(requestProbability) then        ▷ maybe send cover request
18:            requestCoverSized(randomEmbeddedSize())
19:            updateHosts(hostnameOf(url))
```

fingerprinting attacker [8, sec.4.1]. To counter this information gain, a first load is always covered by additional traffic. After this first request, the browser downloads the page resources: images, style sheets, JavaScript [74] files, etc. Each of these transfers might be covered by additional dummy traffic: The actual number of resources is subtracted from the target number. This difference is randomly spread out among the actual browser requests. The size of each cover response is adjusted to the page's actual values, if known. The addon augments data requests to match an imagined web page retrieval. The dummy traffic is currently provided by a server, see also Future Work 5.2. The *lookupOrGuess...*-functions need data structures to map URLs to both HTTP sizes and number of embedded objects. These data structures use Bloom-filter [27] based binning (see Section 3.3) to save values related to URLs in a fixed space, while not allowing an adversary to exactly determine which URLs are saved. After a timeout, the hostname and its data is removed from the data structure used by *registerHost*, *isRegistered*, *getProbability* and *updateHosts*.

Figure 3.2 shows a sequence diagram for a known host. The *watcher* class sees the request, notifies the *user* class, which has an already initialized *coverTraffic* object for

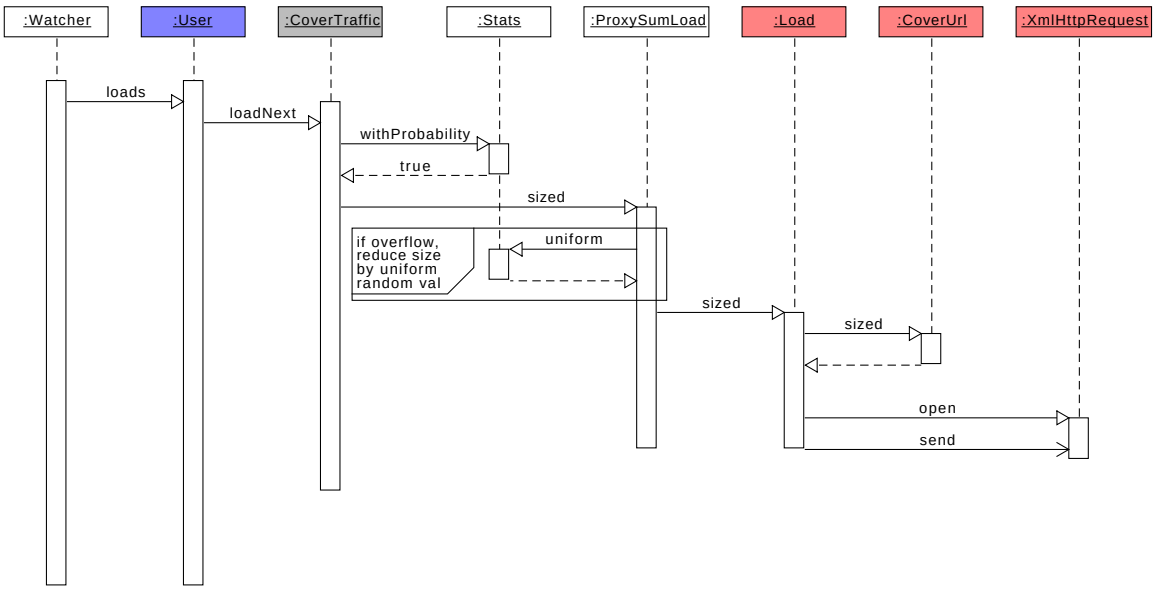

Figure 3.2.: Sequence chart of cover traffic for a known site with embedded probability < 1, with triggered cover traffic. The color coding corresponds to Figure 3.1.

this site. In our assumption, the probability of embedded cover traffic per element is less than 1, so the *coverTraffic* object checks if it should send cover traffic. It then calls the *proxySumLoad* class, which saves sizes that could not be fulfilled due to the HTTP protocol's minimal traffic size. After resolving the `overflow`, the *load* object is called to load cover traffic data. It receives the URL for the data from the *coverURL* class, and loads it using a `XMLHttpRequest`.

Once again, let us consider Alice. When she opens `http://bob.com` in her browser, it first loads `bob.com`'s main HTML [51] page. This page frequently contains other resources like CSS stylesheets, images, JavaScript [74] files, etc. It might also contain elements that request other elements again, like nested HTML, CSS `@import`, or JavaScript AJAX requests. For each of these, the browser sends a request to obtain it. Alice's defense sees the first load request to `bob.com`, and checks if it has seen this site before. Since it is a new request, it has not. It thus considers this to be a main HTML request, and determines the actual, target, and cover values. Using these, it then sends a request for HTML cover traffic concurrently to the first HTML request. Since all traffic is encrypted and secured over Tor, only the sizes of the requests and responses are seen. By design, on the wire this cover request is simultaneous to the

main HTML request. Bob's main website might be simple: it contains just a stylesheet, links to further posts, and a background image. The browser downloads the stylesheet, and the image after the first HTML page. The defense has determined to accompany each embedded resource with 1.37 additional cover requests. It randomly chooses to request two cover elements for the stylesheet, and one for the background image.

The created traffic is based on the HTTP model [71] described in Section 3.2. For each web page retrieval, the defense sets target retrieval parameters. From these, the web page's actual parameters are subtracted to set the probability of cover traffic for each element of this web page. If the page sizes are not known, they are guessed from the same distributions used to set the target values.

3.2. Modeling Web Retrieval

This thesis' defense uses the distribution of HTTP traffic: the sizes of HTML pages, the size of embedded resources, and the number of embedded resources per HTML page. There are several approaches on how to generate HTTP-shaped traffic. The naïve way, using HTTP dummy traffic [8], loads another page simultaneously in the background. Cai et al. [23], Cai et al. [15], Hayes and Danezis [25], Wang et al. [14], Panchenko et al. [8], and Juárez et al. [16] evaluate it to be surprisingly effective for all its simplicity, albeit at a high overhead.

This is why HTTP [77]-shaped cover traffic might prove more effective than chosing standard distributions: this would make it harder to separate cover and real traffic. In addition, it would work at the layer [78, ch.1.7] where the problem originates, as it mimics the HTTP/HTML [51]-specific request-response interaction. Yet, World Wide Web [79] traffic cannot be adequately modeled using standard distributions like normal or uniform [80]. This explains the many outliers reported by Cherubin et al. [18, sec.5.2]. Juárez et al. [16, sec.5.A] also distribute its website fingerprinting defense data partitions exponentially to better fit web traffic. [81] repeatedly mention a *long tail* of traffic. Several aspects of web traffic show subexponential distribution [73] behavior, where high sampling values are not as unlikely as in for example exponential or normal distributions. Lee and Gupta [71] show that log-normal distributions (truncated), as

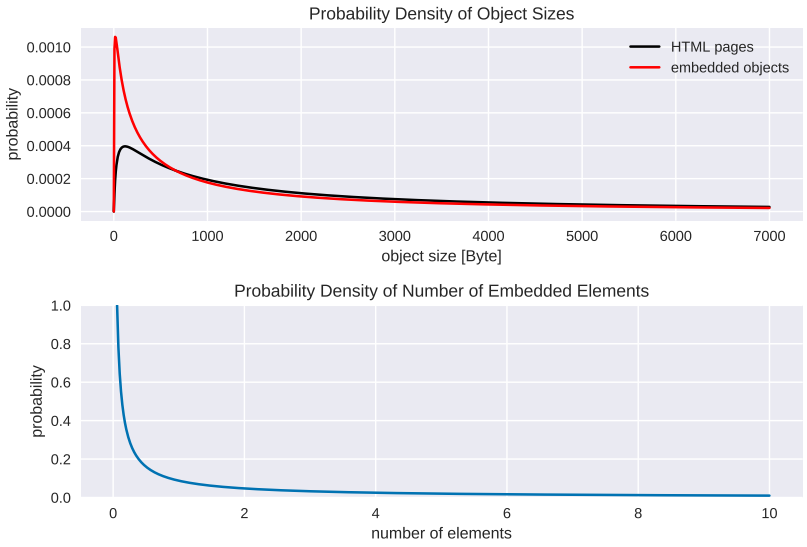

Figure 3.3.: Distribution of sizes for the HTTP traffic model. While the number of embedded elements seems very low, it has mean $\mu \approx 5.07$ [71] and standard deviation $\sigma \approx 15.16$ (computed as of [75, 5.1.11] and using [76, stats.gamma]).

seen in Figure 3.3 fit the sizes of both HTML and embedded resources best in their analysis. A truncated gamma function models the number of embedded objects. This model provides the cover traffic target sizes. It also sets the page's actual sizes, if they are not known.

3.3. Caching Sizes using Bloom Filters

As mentioned in the previous section, the model provides the page retrieval's actual numbers, if these are not known. The defense might further improve if the sizes of the webpage to be loaded are known beforehand: Cover traffic could be tailored more exactly, increasing obfuscation and/or reducing overhead. Knowing the exact retrieval pattern in advance even enables new defenses [14]. This section introduces Bloom filters, and their application to save data in a fixed-size probabilistic data structure.

The problem in caching is that

1. page properties change over time, making a fixed cache increasingly less accurate,

2. caching visited page sizes might yield an exact log of the visited web pages to an attacker who gains control over the defended computer. Perry et al. [82, sec.2.1] forbid writing sensitive data to disc, except on opt-in, and

3. this cache cannot store all page sizes. Even in a closed world, using a default mapping of string to size could take as much space as to preclude usage, depending on the subset of pages's sizes cached.

Storing the sizes of all pages as a mapping from their names to their sizes is impractical due to size constraints. A data structure that stochastically saves approximate sizes might solve problems *2* and (partially) *3*: Bloom Filters [27] have a small error rate in exchange for a fixed size. Their otherwise disadvantageous error probability is an advantage in this situation, as it further confounds possible attackers. The current implementation was initialized with fixed page sizes for the top pages. Dynamically updating the filter might solve problem *1* and further help with problem *3* (future work).

Bloom filters [27] are a stochastic fixed-width data structure to test membership in a set. In exchange for a small false-positive error rate, they require significantly less space than deterministic data structures: if an element is in the set, the filters accurately report this; if the element is not in the set, the Bloom filter might report that it is contained. The error rate depends on the number of included elements in relation to the Bloom filter's size. Bloom filters were developed for spell checking. They generalize on hash-coding, with a tunable false positive error rate. They have numerous uses in network applications, for example in distributed caches, and network analysis [83].

By ordering data into bins, it becomes possible to use Bloom filters for the estimation of sizes, using one Bloom filter for each bin: the histogram of observed values is split by quantiles into bins. For each bin, a Bloom filter is created. Each site/URL is then binned: An element is added to this filter if its size is inside the bin. To approximate the size of an element, all filters are checked. If one filter reports containment, its size is chosen. If zero report containment, the size is not known; if two or more report containment, it is saved wrongly. In both of these latter cases, the default distribution

is used. If the data structure is queried, the middle quantile of each bin is chosen to represent the bin. Each bin thus corresponds to a Bloom filter that saves whether the element's URL is modeled by the bin. This data structure has the additional advantage that, even if visited page sizes were saved, an adversary could not safely determine that pages were visited due to the Bloom Filter's false positive errors. See Appendices E and F for implementation details.

This data-structure, dubbed *BloomSort* is initialized with an array of *splits* between the bins, and an array of bin *sizes*. After initialization, it contains an array of Bloom *filters*. Given these, it becomes possible to add the elements to their bins, as shown in Algorithm 2.

Algorithm 2 Insert URL into BloomSort

```
1: procedure BLOOMSORT.ADD(url, size)
2:     idx ← splits.index(size) ▷ where size would be inserted into the sorted splits array
3:     filters[idx].add(url)
```

Algorithm 3 describes the retrieval of element sizes. It looks into each Bloom filter, checking whether it might contain the `url`. If one Bloom filter reports containment, its corresponding element- `size` is returned. If several or no Bloom filters report containment, an exception is thrown. The exception is used to allow all possible return values, not blocking one of them, for example `-1`, for the error condition.

Algorithm 3 Query URL size from BloomSort

```
1: procedure BLOOMSORT.QUERY(url)
2:     pos ← −1
3:     for i ← 0; i < length(filters); i ← i + 1 do
4:         if filters[i].contains(url) then
5:             if pos = −1 then
6:                 pos ← i
7:             else
8:                 throw(name: 'BloomError', message: 'Contains multiple entries')
9:     if pos = −1 then
10:        throw(name: 'BloomError', message: 'Contains no entries')
11:    return sizes[pos];
```

Consider a *BloomSort* named `htmlSizes` with parameters *sizes* $[400, 1000, 20000]$ and *splits* $[700, 10000]$. An element is added via `htmlSizes.add("http://google.com/", 613)`. Querying via `htmlSizes.query("http://google.com/")` would yield `400`.

3.4. Defense Configurability

These altogether enable the four configuration options: overhead factor, use of cached sizes, which target model to use, and whether to send bursts after retrieval has ended.

- SCT offers configurable overhead to account for personal security needs. The higher this *factor* is, the more data gets sent.
- The *cache* option decides whether to use the (fixed) cached page properties to adjust retrieval. The alternative is to guess the page's properties from the same HTTP model [71] that sets the target values.
- The *target* option determines how the target values are set: If the page is cached, its size is inside a bloom bin, and approximated by that bin's size. The alternative is to sample from the www traffic distributions presented in the previous section.
- What happens when the retrieval finishes but the defense has not yet sent all the traffic it should send? The `burst` option sends all of this traffic at the end, while the `noburst` discards this.

If the sizes for a URL are not known, the http model is a fallback both the cache and the target. While several scenarios were tested, the latter three options have little effect, except for very fresh caches (a week). For the sake of comparison, the defense is compared with two *factor* values:

- Selective Cover Traffic light (SCT light) sets the factor to 20, and
- Selective Cover Traffic heavy (SCT heavy) sets the factor to 100.

These values should to illustrate configurability, and show usable presets. Users can select other values, depending on their personal security needs.

3.5. Summary

While website fingerprinting attacks continue to evolve, its defenses have been found wanting. SCT's main advantages are

- works at the application layer, using a traffic model [71] to augment traffic
- easily deployable due to using Firefox's Add-on SDK
- configurable to personal security needs

Its efficiency and configurability are examined in the next chapter.

4. Evaluation

This work proposes Selective Cover Traffic (SCT), a new defense against website fingerprinting, which builds upon the HTTP traffic model [71] for efficient obfuscation of world-wide-web [79] traffic. The complexity of HTTP traffic renders the theoretical analysis of website fingerprinting difficult, hence the prevalent technique for assessing the performance of a website fingerprinting defense is an empirical evaluation.

In empirical evaluation, traffic data needs to first be captured and then analyzed. Wang et al. [14] and Panchenko et al. [24] provide their capture data, but it does not contain SCT's traces, so new traffic data needed to be collected. The collection process is described in Section 4.1. Section 4.2 first validates the attack re-implementation, then evaluates SCT, and compares it with the latest defense by Cherubin et al. [18], both in a closed-world scenario. Section 4.3 describes the open-world analysis. Complicated aspects of the capture process are discussed in Section 4.4, with concluding remarks in Section 4.5.

4.1. Capturing Web Traffic

The aim of this chapter is to test if the defense prevents website fingerprinting, and if so, it works more efficiently than other defenses. For this, data first needs to be captured.

To do this, real Tor web browsing should be closely reproduced. The current Tor Browser Bundle (up to version 7.5.6) downloads the web sites traces: it is based on the Firefox Browser, which provides the Marionette framework [84] for instrumentation.

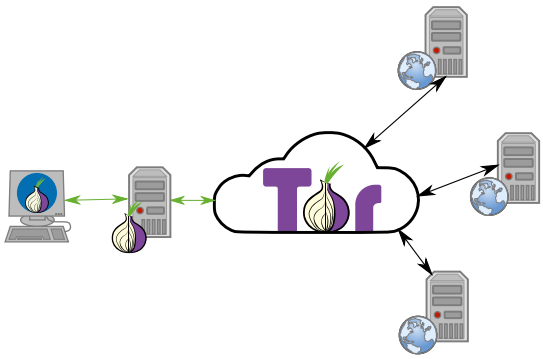

Figure 4.1.: Setup to capture web page traffic: Tor Browser on *client* machine, connects to Tor server on *bridge* machine, connects to Tor network, connects to web servers

To be able to evaluate the WTF-PAD [16] defense[1], all traffic is routed over a Tor Bridge [85, sec.client options], see Figure 4.1. This also eases defense comparability. The sites to be downloaded come from Alexa's top million sites list[2] with duplicate sites and sites with high load error rate removed. Section 4.4.2 describes this in further detail. Removing sites with problems increases the classification ratio, that is: improves website fingerprinting accuracy. Making website fingerprinting easier is admissible in the evaluation of a website fingerprinting defense. WTF-PAD [16]'s main author co-authored LLaMA [18]. This thesis compares its results to that latest defense.

Web traces are captured in batches in a round-robin fashion: each site is captured once until all sites are done. The process then starts again until all pages are captured the approximate number of times. Prior to this capture process, one or no defense is enabled. In the case of SCT, it is configured for low or high overhead, if necessary. The Tor Browser Bundle clears its cache and saves data on each restart. This takes care of resetting the circuit and deletes all cached data. This procedure also reproduces the time gaps described in Wang et al. [14, sec.5.1]: By fetching websites in a round-robin fashion, there is a delay between two consecutive fetches of the same page: the time for the other pages to be fetched in between.

The data is analyzed via a re-implemented version of CUMUL [24], which is used as state-of-the-art comparison in the latest attacks by Hayes and Danezis [25] and Rimmer et al. [70].

[1] original provided at `https://bitbucket.org/mjuarezm/obfsproxy_wfpadtools`, this thesis' version at `https://github.com/kreikenbaum/capture.git`

[2] available at `http://s3.amazonaws.com/alexa-static/top-1m.csv.zip`

Table 4.1. Validation on 100 sites in a closed-world setup. The traces captured at Technische Universität (TU) Braunschweig show similar accuracy to [24]. The traces captured using Google Cloud virtual machines show higher accuracy. A similar phenomenon was reported by Juarez et al. [86, sec.4.7].

Data source	Number of instances	Accuracy [%]
Panchenko et al. [24]	40	92.03
TU Braunschweig (*mlsec*), using bridge	50	91.61
Google Cloud (*gcloud*)	50	95.12
TU Braunschweig (*mlsec*), direct	50	86.99

4.2. Analyzing Closed World Scenarios

As Section 2.3.3 details, there are two settings in analyzing website fingerprinting: open-world and closed-world. Closed-world analysis distinguishes only between main sites, for example Eve knows that Alice is either visiting `bob.com` or her uncle `charlie.com`, and the aim is to distinguish these two sites, assuming both are being monitored by Eve. This section presents the results of closed-world analysis. It first compares the accuracy of Panchenko et al. [24]'s original to the re-implementation on defenseless traces.

Recent studies ([25] [18] [70]) consider CUMUL by Panchenko et al. [24] the state-of-the-art website fingerprinting attack. Table 4.1 ensures attack accuracy by comparing CUMUL [24] to its reimplementation on defenseless web site traces. In most cases, the original CUMUL attack and the reimplementation exhibit similar results. This validates that the CUMUL-reimplementation[3] is used as the main method of evaluating the extension. The traces captured at TU Braunschweig with a direct connection, without a Tor bridge, yield less accuracy. This is presented for the sake of completeness, and because the LLaMA-comparison does not use a bridge (see below), while all following comparisons use the bridge. The bridge was omitted for LLaMA to reduce a point of failure in the capturing process. Repeating the captures with a bridge proved too time-consuming before the submission deadline of the thesis: It would take approximately more than 35 days to capture both foreground and background traffic. Panchenko et al. [24]'s accuracy results could be reproduced on the traces captured with bridge with the re-implementation.

[3]available at `https://gitlab.com/kreikenbaum/classify.git`

Table 4.2. Results for capturing traces using LLaMA, as compared to no defense, and SCT. This compares all captured traces to those where text was successfully loaded, to those that completed loading. website fingerprinting (WF) defaults are keeping timeouts, and [24, sec.V.B]'s outlier removal, with removal of small (< 30) sites. The traces were captured at TU Braunschweig, without a bridge.

Scenario	captured	text loaded	fully loaded	WF default	duration
LLaMA	5055	2760 (54.6%)	1038 (20.5%)	1158 (22.9%)	22d 16h
defenseless	4930	4596 (93.2%)	4560 (92.5%)	4192 (85.0%)	2d 2h
SCT light	5000	4916 (98.3%)	4700 (94.0%)	4601 (92.0%)	2d 13h
SCT heavy	4996	4917 (98.4%)	4718 (94.4%)	4633 (92.7%)	2d 15h

Table 4.2 compares the captured trace statistics for defenseless traces to Cherubin et al. [18]'s LLaMA's traces, and this thesis' defense. While SCT only moderately slows the loading process, the table clearly illustrates the effects of LLaMA's delaying action. For one, the duration to capture 50 traces each for 100 web pages increased from 2 days 2 hours to 22 days 16 hours. This is also illustrated by the loading statistics. At the timeout of 4 1/2 minutes, approximately half of all pages had not loaded the HTML text. Only approximately one fifth finished loading the whole content. In the default machine learning setting with outlier removal [24, sec.V.B] and removal of sites with less than 30 instances (keeping the text-only loads) due to cross-validation limitations, only 22.9% of traces were usable. If no outlier removal was performed, there were 42 remaining sites with at least 30 instances for machine learning. These are evidently easier to classify than the whole set of 100 sites, as it reduces the classification problem from one on 100 classes to one on 42. The time overhead of all of LLaMA's traces which loaded at least the text is 881.83%. Reduced to the fully loaded traces, the time overhead compared to no defense is 228.98%. The slightly higher trace load rates of SCT as compared to no defense are conceivably due to circuit performance variations in Tor. Another explanation might be the percentiles in outlier removal. All of the LLaMA comparison data was captured without a Tor bridge to further reduce the possible causes of errors.

The classification results in Table 4.3 compare CUMUL's [24] results on these captures. Both variants of SCT decrease accuracy enough to make website fingerprinting seem impractical, for moderate and configurable overhead. While LLaMA decreases accuracy more than SCT, keep in mind that this also compares 1722 non-completely loaded pages, combined with 1038 completely loaded. Obviously, the text-loaded traces are different, as they show less object requests/response pairs. If the non-

complete-load-traces are removed as well, no sites remain for analysis after removal of small (≤ 30 instance) sites, with CUMUL's outlier removal [24, sec.5-B] disabled. A time overhead of 846% (out of the half of traces that were loaded at all) might encumber Tor's usability design goal [32, ch.3]. The variability in the time overhead could again be due to Tor circuit and web site changes.

This obvious problem with LLaMA could stem from the difference in www top site's structure as opposed to Tor sites. As mentioned in the previous section, this thesis evaluates defenses on Alexa's top sites, while Cherubin et al. [18] inspect Tor's `.onion` hidden service [1, sec.5] sites: Tor hidden service sites contain fewer embedded elements, especially JavaScript and CSS files [18, sec.5.2]. Each element triggers LLaMA's cover traffic behavior, and could thus explain higher overhead on Alexa's top sites.

As LLaMA hinders web browsing, where even minimal delays make users less likely to use a service [19], the source code[4] was edited (a constant was changed) to disable the delay. This version is called *LLaMA-nodelay*. Table 4.4 compares this modified version of LLaMA to this thesis' SCT, and to no defense. This thesis' defense offers both configurability and a lower size overhead: SCT light decreases accuracy by ~50%, for only 60% size overhead. SCT heavy decreases accuracy by 10% more than LLaMA-nodelay for 60% lower time overhead and comparable size overhead. Both LLaMA-nodelay and SCT have negligible time overhead. Even compared to non-delayed LLaMA, SCT offers similar accuracy/size overhead relationship for a much lower time overhead. All further data was captured using the bridge setup described in

[4] gratefully provided by Marc Juarez by email, also available at Cherubin et al. [69]

Table 4.3. Results for evaluating LLaMA, compared to no defense, and SCT. LLaMA only has 34 sites with at least 30 instances. The size and time overhead (OH) are for the website fingerprinting default setting, as described in the previous Table 4.2. Size and time OH are for ~34 sites for comparability to LLaMA, 100 site values are similar. These were also captured without a bridge at TU Braunschweig.

Defense	~100 site acc. [%]	~34 site acc. [%]	Size OH [%]	Time OH [%]
defenseless	86.99	93.10	0	0
SCT light	42.50	57.16	66.57	30.75
SCT heavy	30.21	46.25	130.04	50.37
LLaMA	n.a.	33.19	23.03	845.82

Table 4.4. Comparison of LLaMA-nodelay to this thesis's defense on 100 sites, captured at TU Braunschweig, using the bridge. The table is sorted by size increase. It also illustrates SCT's configurability.

Scenario	Accuracy	Size overhead [%]	Time overhead[%]
defenseless	0.93	0.00	0.00
SCT light	0.43	61.81	22.23
SCT heavy	0.26	142.22	28.16
LLaMA-nodelay	0.36	205.18	29.04

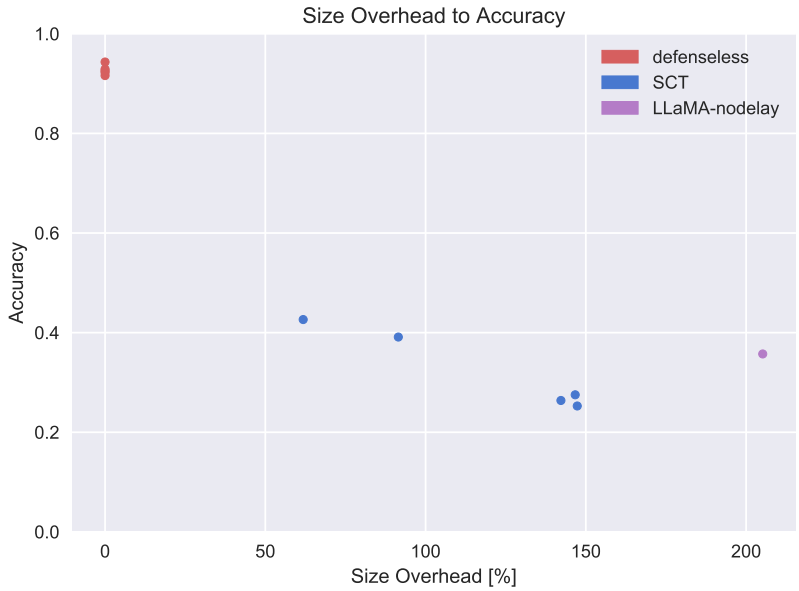

Figure 4.2.: Size overhead to accuracy trade-off for SCT, LLaMA-nodelay, and no defense on 100 sites, captured at TU Braunschweig.

Section 4.1.

Table 4.4 shows SCT's effects, and compares it to LLaMA-nodelay, on 100 sites. This shows that SCT reduces accuracy more, for the same overhead, or requires less overhead, for a similar accuracy reduction. The main difference between LLaMA-nodelay and SCT lies in the HTTP-specific distributions to generate cover traffic. Cherubin et al. [18] report less overhead on Tor hidden service [1, sec.5] sites. This is arguably due to the high number of embedded objects for standard websites as compared to Tor [18, sec.5.2], bas each element potentially triggers a request for an already-seen element with LLaMA-nodelay, as well as a delay with LLaMA.

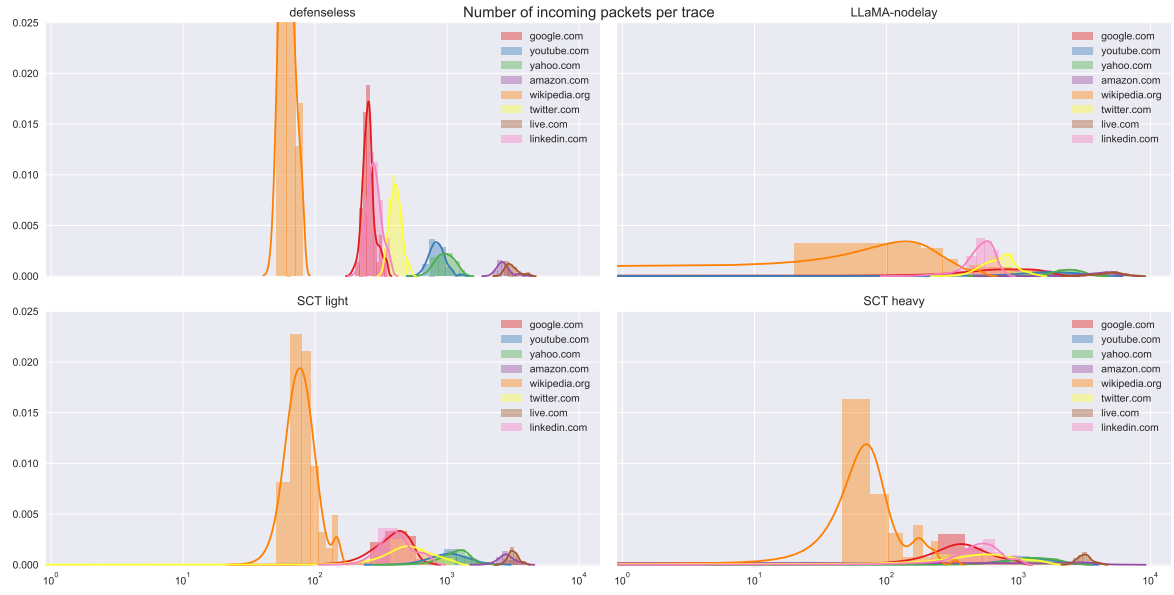

Figure 4.3.: Relative histograms of number of incoming packets on top sites. They were capped at y=0.025 to clearly show the defense's differences, even though wikipedia.org's bars on defenseless traces go above 0.06

Figure 4.2 shows SCT's configurability and contrasts its accuracy/overhead trade-off to LLaMA-nodelay. This scatter plot includes data for captures that were affected by retrieval inaccuracies. Still, it clearly shows that SCT surpasses LLaMA-nodelay in the accuracy-overhead trade-off. Furthermore, its configurability can be seen: low size overhead in the 20% yields accuracy decrease from ~90% to ~65%. Higher size overheads yields accuracies of ~35%. This allows users to configure the defense based on their individual security needs. Interested readers can find further evaluation results in Appendix A.

Figure 2.7 in Chapter 2 compares the relative histograms of the total number of incoming packets. It can be extended to provide some insight into the effects of both LLaMA-nodelay [18] and SCT in Figure 4.3. Again, all defenses reduce the clarity of separation that is visible for defenseless data. All defenses flatten and intermingle the peaks of this feature. The height of the relative histograms seems linked to defense overhead: defenseless data has the highest peaks, followed by SCT light on 2018-02-13, SCT heavy on 2018-03-28, and LLaMA-nodelay, which has the highest, non-configurable overhead.

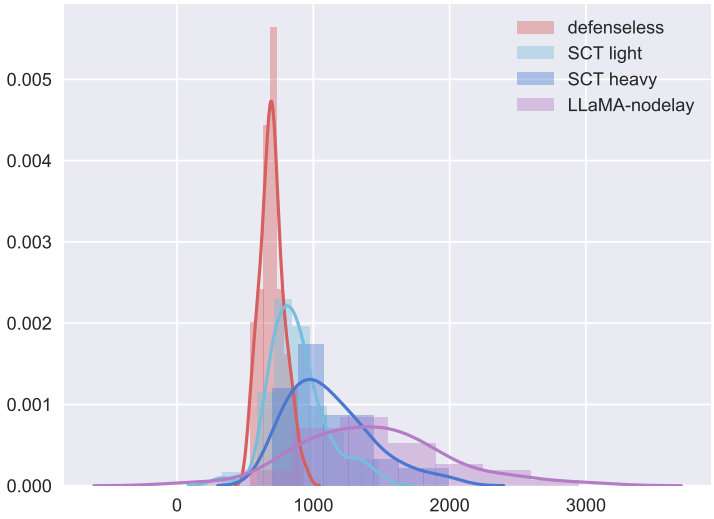

Figure 4.4.: Relative histograms of the number of incoming packets for quora.com for each defense — with kernel-density estimation (kde).

Figure 4.4 compares the number of incoming packets for a single site. The site is `https://quora.com`, which was randomly picked. In the figure, the curves for defended data are all considerably flatter than the defenseless data. SCT light is steeper than SCT heavy, which is again steeper than LLaMA [18]. The aim is a flat curve that still has its values as far as possible to the left. SCT offers configurability in the amount of cover traffic produced.

What is the effect of caching page statistics as described in described in Section 3.3? This cache saves the size of the HTML page, plus the number of embedded elements, possibly providing more closely tailored cover traffic. A first comparison on the top 10 pages yields approximately 7% higher accuracy reduction when using the cache. On 30 pages, this effect is reversed, albeit lower. All sets of traces were captured with a time-gap of one to three months after configuration was enabled in the source code. The decrease might be due to the top sites increasing their sizes less, and being generally lighter, as visible in Figure 4.5 and documented by Cremin [87].

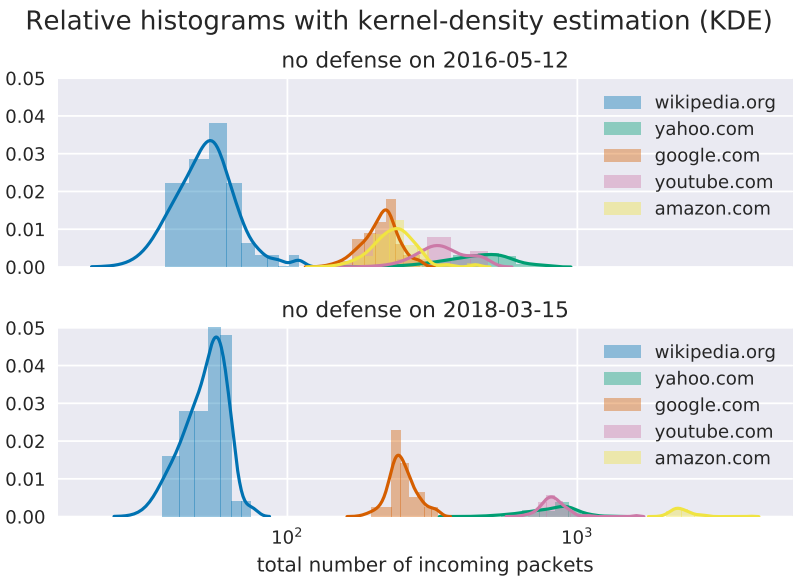

Figure 4.5.: Change of number of incoming packets per site

4.3. Analyzing Open World Scenarios

Closed world evaluation sets a strong assumption [86, sec.3.1] due to its limitation to known sites. As described in Section 2.3.3, the open world setting compares a certain number of monitored sites against a bigger number of background pages, and is considered more realistic [14, sec.5.1]. This setting adds additional background pages to the mix: An attacker's scenario is to find out if a web trace is to a sensitive site, for example Wikileaks, as opposed to normal web browsing. The background pages model the "normal" web browsing. Let us assume Eve knows that her daughter Alice might visit any number of pages, but is only interested in whether she visits `bob.com`. All visits that are wrongly classified as `bob.com`, say Alice really visited `bbc.com`, are called *false positives*, whereas if Alice visited `bob.com` and it was classified as `bbc.com`, is called a *false negative*. Ideally, none of these exist. In practice, Eve has to weigh between leniency and strictness: would she rather let slip a visit to `bob.com` as `bbc.com`, or would she rather check in on her daughter when she just visits `bbc.com`, because it was classified as `bob.com`? This trade-off is illustrated in Figure 2.6 in Section 2.3.3.

Table 4.5. Attack validation on 100 sites of the WANG14 [24] dataset in the open-world setting (100 foreground sites with 90 instances each, 9000 background instances). The traces were captured at TU Braunschweig using a bridge.

Data source	Classes	Num. inst.	TPR [%]	FPR [%]
Panchenko et al. [24, Table III]	two-	90	96.92	1.98
	multi-	90	96.64	9.61
own result	two-	90	96.9	1.87
	multi-	90	96.63	9.19

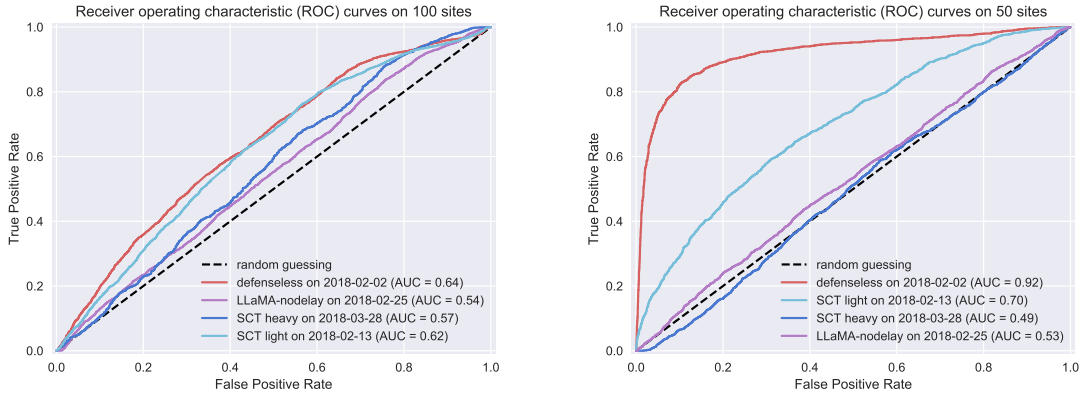

Figure 4.6.: ROC-curves that compare SCT with no defense and LLaMA [18]. The left distinguishes 100 monitored pages in the foreground-set, the right 50, the size of the world/background set is 5000 pages.

On the WANG14 [24, sec.4.1] data set[5], the results of the reimplementation closely match CUMUL's original, see Table 4.5. This validates the re-implementation. Interested readers can find additional results on this dataset in Table B.1 in Appendix B. As visible from that table, the WANG14 data set is extraordinary in its classification accuracy.

To evaluate SCT, and to compare and contrast to LLaMA [18], defenseless captures were compared and contrasted to those protected by LLaMA-nodelay and two configurations of SCT. The reader is kindly reminded of their closed-world accuracy and size overhead results in Table 4.4.

Figure 4.6 compares the defenses using ROC-curves with 100 and 50 pages. 100 pages on the left already show SCT's configurability, and the effect of its overhead con-

[5]available at `https://cs.uwaterloo.ca/~t55wang/wf.html`

figuration: the light-overhead version obfuscates traffic less than the heavy-overhead version. LLaMA [18] is close to SCT heavy in its obfuscation (for considerably higher size overhead). As all curves are close to the diagonal — random guessing — the world size of Juárez et al. [16, Fig. 11] was used in the right graph: The left ROC-curve for 100 sites shows the curves close to each other. To better distinguish between the defenses, the right ROC-curve illustrates the classification strictness trade-off on 50 sites with an open world size of 5000 background traces. This is the same size configuration that Juárez et al. [16, Fig. 11] use in their ROC-curve comparison. The curve of defenseless data has a similar AUC (0.92) to their reports (0.82). The slightly higher value could be due to CUMUL's [24] increased accuracy as compared to KNNs [14]. SCT light at 62% overhead visibly decreases classification efficacy. SCT heavy at 142% overhead reduces classification accuracy to a level similar to random guessing at AUC 0.49; as does LLaMA-nodelay at AUC 0.53, but with 205% size overhead. SCT's web-model-based [71] defense seems to surpass LLaMA-nodelay's load-repeating defense in this scenario, too, yielding close to total obfuscation for much less space overhead, and added configurability.

4.4. Capture Pitfalls

This section lists problems that arose during capturing of data in the hope that future researchers will not have to repeat these.

4.4.1. Accuracy Decay from 2016 to 2017

Accuracy for captures until 2017 was markedly lower than that reported in the literature, both on 30 and 100 sites. This is true regardless of if data was captured with defense or without. On 100 sites, the comparison to literature results shows this handicap more clearly. As both Panchenko's original implementation and the reimplementation yield the same results, the cause seemed due to capture methodology.

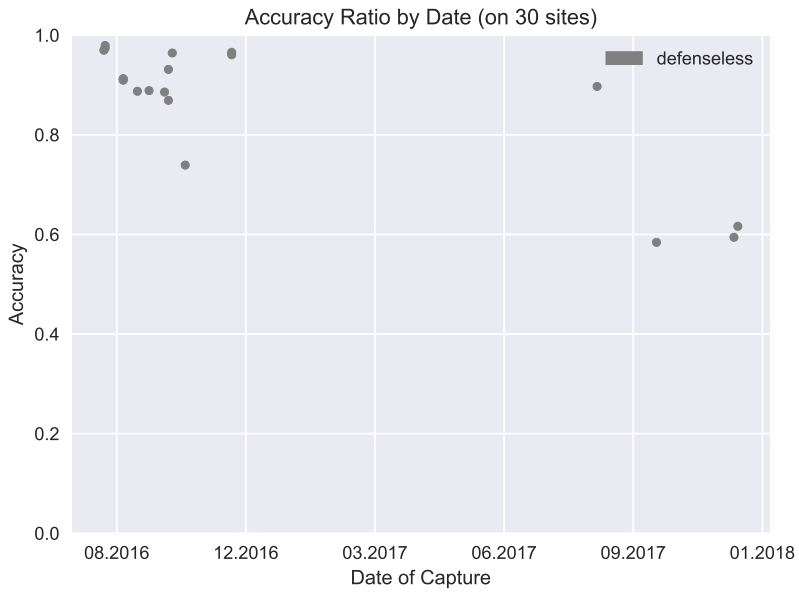

Figure 4.7.: Accuracy decay on 30 sites.

Surprisingly, the capture's accuracies decayed over time: The accuracy on 30 sites decayed from 97.93% on 2016-08-15 to 58.41% on 2017-10-16, as displayed in Figure 4.7. On 100 sites, the accuracy likewise fell from 90.57% on 2016-06-17 to 43.46% on 2017-10-22, as displayed in Figure 4.8.

A high-accuracy confusion matrix from 2016-08-15 is shown in Figure 4.9. This is contrast with a low-accuracy confusion matrix from 2017-10-16 in Figure 4.10. While there is much more class-bleed-off, there is no clear culprit: errors are mostly distributed.

A site that existed on both captures is *msn.com*. It was chosen to illustrate the problem. The CUMUL-features are shown in Figure 4.11.

For several low-accuracy retrievals, very little data is received back from the server. To re-increase accuracy, the text of the loaded page was checked. This revealed that the Marionette framework [84] sometimes returned from the `.navigate` method without the page being loaded. If this text indicated a load error, or a captcha, the capture was

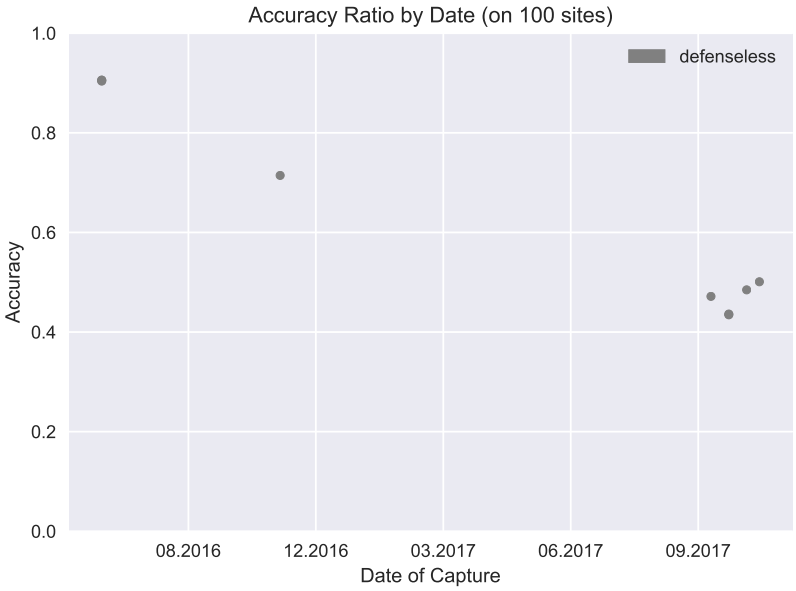

Figure 4.8.: Accuracy decay on 100 sites.

marked accordingly, and discarded in the analysis. Additionally, the code was modified to ensuring that the Marionette framework [84] reports successful connection to the Tor Browser. Also, websites with a high number of capture errors were removed from the list of sites.

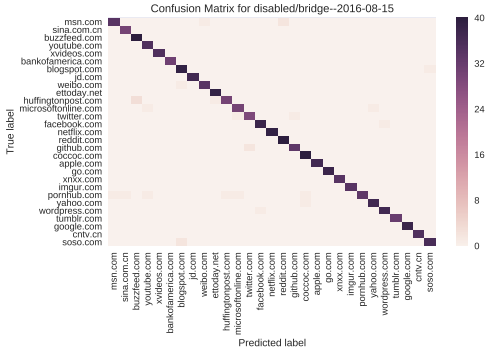

Figure 4.9.: Confusion matrix for 30 sites at 2016-08-15, overall accuracy at 98%.

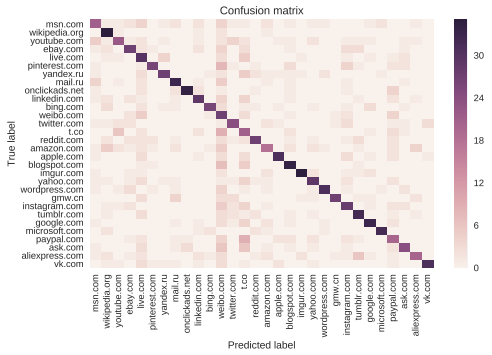

Figure 4.10.: Confusion matrix for 30 sites at 2017-10-16, overall accuracy at 58%.

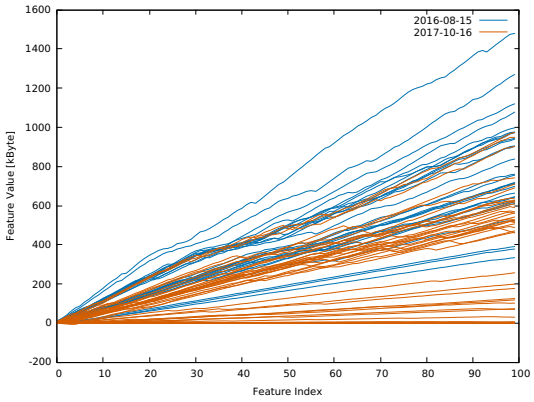

Figure 4.11.: CUMUL-traces for `msn.com` compared for both dates.

4.4.2. Open World Misclassification

While closed-world results are easier to compare and analyze, website fingerprinting is modeled more accurately by an open-world model. This brought additional challenges in classification. The confusion matrix in Figure 4.12 shows which sites were misclassified how often:

One site that was exclusively classified as background is t.co (on the chart between `microsoftonline.com` and `amazonaws.com`). For t.co, 48 of 48 (valid) traces were classified as background sites. The next-highest rates of false-negative misclassification is rakuten.co.jp, which has 37 of 49 sites incorrectly classified as background pages. This might be due to their highly varied traces, as can be seen in Figure 4.13:

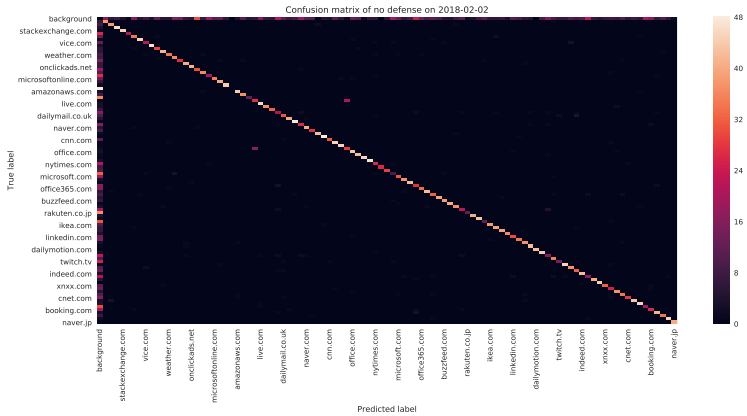

Figure 4.12.: Open world confusion matrix. Background to background classification (5928) was removed to normalize colors.

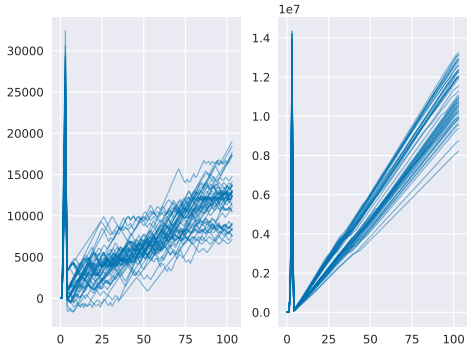

Figure 4.13.: CUMUL-Traces for the websites t.co and rakuten.co.jp. These show high (intra-class) variability, which could explain the high misclassification rates. Compare Figure 2.4 and 4.14 for lower-variability CUMUL traces.

This can be explained by the content: t.co shows only 3 lines of text, and thus any variation in the download of its little content leads to higher variability. rakuten.co.jp has shopping results, which are bound to show high variation.

As a contrast, the website dailymotion.com has 42 of 42 correctly classified captures. Its CUMUL-traces show low variability in Figure 4.14.

Table 4.6 shows the mis-classification rates of the sites that were mis-classified the most often ($\geq$ 30 false negatives or $\geq$ 20 false positives). Let us examine the top-5 false-positive sites in detail. As Figure 4.15 shows, four of these show high intra-class variability:

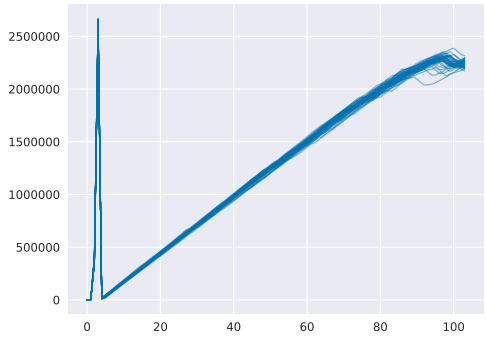

Figure 4.14.: CUMUL-Traces for well-classified website dailymotion.com.

Table 4.6. Misclassification count (open world)

site	site as background (FN)	background as site (FP)
ameblo.jp	19	30
aliexpress.com	17	24
godaddy.com	22	23
youporn.com	16	21
google.com	30	20
aol.com	32	19
dropbox.com	31	11
netflix.com	36	5
rakuten.co.jp	37	2
t.co	48	0

- ameblo.jp (and youporn.com) show a range of dynamic content
- aliexpress.com sells different items at different times. The images at least are bound to have different byte sizes
- godaddy.com has versions in various languages, serving different content, which explains the varied traces

The fifth site, google.com also serves international content. Yet, its traces show low variability, as seen in Figure 4.16. While google.com sister sites, say google.fi, were removed from the foreground set, the background set contained hundreds of google sites as well. This of course made classification infeasible.

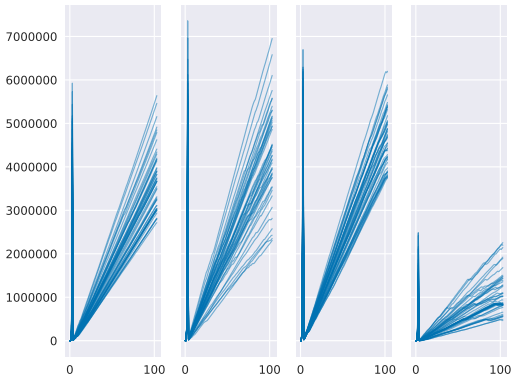

Figure 4.15.: CUMUL-Traces for the websites with high false-positive rates. The sites are (left-to-right) ameblo.jp, youporn.com, aliexpress.com, and godaddy.com. These also show high (intra-class) variability, which could explain the high misclassification rates.

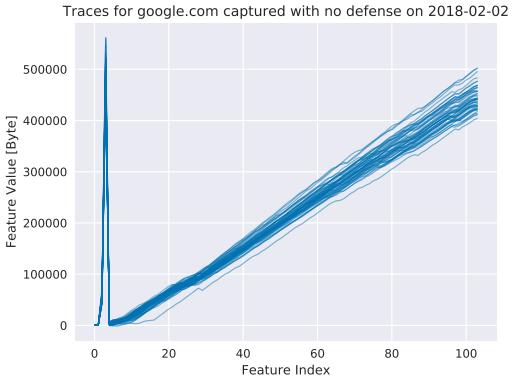

Figure 4.16.: CUMUL-Traces for the last website with high false-positive rates: google.com. It shows low variability. Its high misclassification rate is explained by other google sites being in the background set (for example google.fi).

4.5. Summary

More than 20 GB of traces were collected. These serve both to validate the re-iplemented attack, and to evaluate the defenses. The attack re-implementation matches both the provided software's values, as well as the theoretic values of Panchenko et al. [24], both in the closed-, and open-world scenario.

In all comparisons, SCT shows its effectiveness, and configurability. It compares favorably to LLaMA [18] using four methods in Panchenko et al. [24] and Juárez et al. [16]: where metrics exist, it yields the same accuracy reduction for lower overhead, or more accuracy reduction for comparable overhead. It is also configurable, offering a trade-off of security to size overhead. The defense has a slight time overhead that is not due to design, but to increased network traffic. Using cached sizes did not consistently increase the defense's effectiveness.

5. Conclusions

To counter the increasing threat of website fingerprinting, general defenses need to create traffic that is easily confused with other websites's. The current defenses have several drawbacks, mostly high overhead, hard to deploy, and static overhead. While the base rate fallacy [88], might hinder this detection due to the large number of background sites, Axelsson [88]'s original arguments have been called into question by Rieck [37]'s effectiveness. As of Koehler [89], the base rate fallacy should generally be considered with a grain of salt.

This thesis addresses the problem of creating cover traffic at the application layer of network communication. In particular, it presents a stochastic solution, which applies Lee and Gupta [71]'s web traffic model to create realistic cover traffic, augmenting existing traffic so as to effectively obfuscate web page patterns. The creation of cover traffic is realized using the Firefox Addon-SDK.

The selective cover traffic is rooted in the web traffic model of Lee and Gupta [71]. This model supports the generation of cover traffic that closely resembles real WWW [79] traffic. While SCT does not defend against any particular attack, it obfuscates traffic so as to defeat any conceivable attack. This is based on Panchenko et al. [8]'s camouflage traffic, but adjusts it so that the whole page load is accompanied by cover traffic. It also enables users to select their level of website fingerprinting defense as it behooves them.

5.1. Summary of Results

The main result of this thesis can by summarized in three contributions:

Web-model Based Defense

SCT, introduced in Section 3.1, is a web-model [71] based, easy-to-use website fingerprinting defense that allows for various levels of protection. This defense easily integrates into the Tor Browser Bundle, and works out of the box (given a cover traffic server). The default settings weigh protection with extra network traffic. For the security-conscious, increasing protection is as easy as changing a preference. For evaluation purposes, SCT offers other options, such as the exact way of setting cover traffic targets, whether to use the BloomSort size cache, and wether to send additional burst traffic at the end.

Bloom-filter Based Cache

Section 3.3 proposes BloomSort, an approximate size cache based on Bloom filters [27]. These use Bloom filters's configurable error rate to approximately save target page values. Bloom filters's size benefit allows for efficient disk storage of page properties, whereas their otherwise disadvantageous error rate could protect users if the filter were updated dynamically, obfuscating on-disk storage of properties.

Website Fingerprinting Traces

Chapter 4 evaluates this new method using real Tor traffic covering a period of two years, and comprising over 400,000 instances of web retrievals. In the open world setting, the prototype matches the performance of the latest LLaMA [18] client-side defense, for lower overhead. In the closed world setting, it can either match the

accuracy reduction of LLaMA, for significantly lower overhead, or provide the same overhead, for significantly higher accuracy reduction. It also evaluates the BloomSort-cache, and finds improvements only for very fresh caches. It thus confirms that the cache works, yet points to the need for fresh data, if it were to be used.

5.2. Future Work

As it seems to provide adequate protection, the main improvement for the defense would be an update to the web extension framework, which succeeds the Add-On SDK. As the extension was thought as a prototype, a server was used instead of more elaborate solutions, such as caching retrieved sizes, and using these. Minor possible improvements would be UI-based approximating overhead, and a web model that is updated to modern sites. Further research is if this defense could possibly defeat targeted attacks such as Arp et al. [90], and if it can defeat the modern deep-learning based attacks by Rimmer et al. [70] and Sirinam et al. [26]. In case the defense is found wanting, the methods of Guan et al. [91] might be used to strengthen obfuscation by selective rerouting, as Tor could already provide for this with `.exit` domain names [92, sec.2].

On the attack side, Cheng and Avnur [21] confirm that website fingerprinting profits from using hidden markov models in traffic analysis [62]. They could be combined with recent improvements in single-page detection, as written in the last paragraph. For single page detection, it could be explored if multiple kernel learning as presented by Kloft et al. [93] that incorporates timing-based features [94] can surpass current deep-learning based approaches.

Part I.

Appendix

A. Closed-World (Accuracy) Results

The classification results decreased over time, see Section 4.4.1, until the capture process was fixed: the loading process needed to check that a page was loaded, and retry several times, as the Marionette framework [84] sometimes returned from the `.navigate` method without the page being loaded. Sometimes, the loaded page was clearly not the real page. This needed to be noted as well.

A.1. Complete 100 Sites CUMUL Results, Sorted by Size Overhead

Table A.1.: CUMUL-results on 100 site data sets. S. and T. OH denote size and time overhead, respectively. Acc. denotes Accuracy. Size is the number of evaluated classes.

Scenario	Site	Date	Bridge	Size	Accuracy	S.OH [%]	T.OH [%]
defenseless	mlsec	2016-06-17	No	100	0.90	0.00	0.00
defenseless	mlsec	2016-06-17	No	98	0.91	0.00	0.00
defenseless	mlsec	2016-06-17	No	98	0.91	0.00	0.00
defenseless	mlsec	2016-11-04	No	120	0.71	0.00	0.00
defenseless	mlsec	2017-10-08	Yes	98	0.47	0.00	0.00
defenseless	mlsec	2017-10-22	Yes	100	0.43	0.00	0.00
defenseless	mlsec	2017-10-22	Yes	99	0.44	0.00	0.00
defenseless	mlsec	2017-11-05	No	100	0.48	0.00	0.00
defenseless	mlsec	2017-11-15	No	99	0.50	0.00	0.00
defenseless	mlsec	2018-01-17	Yes	99	0.93	0.00	0.00
defenseless	gcloud	2018-01-30	Yes	100	0.95	0.00	0.00

Continued on next page

Continued from previous page

Scenario	Site	Date	Bridge	Size	Accuracy	S.OH [%]	T.OH [%]
defenseless	mlsec	2018-02-02	Yes	100	0.92	0.00	0.00
defenseless	gcloud	2018-02-11	Yes	97	0.95	0.00	0.00
defenseless	mlsec	2018-03-15	Yes	97	0.93	0.00	0.00
defenseless	mlsec	2018-08-03	No	99	0.87	0.00	0.00
defenseless	mlsec	2018-08-03	No	99	0.87	0.00	0.00
defenseless	mlsec	2018-08-03	No	99	0.87	0.00	0.00
defenseless	mlsec	2018-08-03	No	42	0.93	0.00	0.00
defenseless	mlsec	2018-08-03	No	34	0.93	0.00	0.00
defenseless	mlsec	2018-08-03	No	99	0.87	0.00	0.00
defenseless	mlsec	2018-08-31	No	37	0.51	0.00	0.00
defenseless	mlsec	2018-08-31	No	90	0.34	0.00	0.00
defenseless	mlsec	2018-08-31	No	90	0.34	0.00	0.00
defenseless	mlsec	2018-08-31	No	37	0.50	0.00	0.00
defenseless	mlsec	2018-09-07	Yes	97	0.94	0.00	0.00
defenseless	mlsec	2018-10-17	Yes	99	0.92	0.00	0.00
SCT	mlsec	2016-06-04	No	100	0.78	14.94	26.32
SCT	mlsec	2016-06-04	No	100	0.79	18.23	41.70
SCT 5	mlsec	2016-09-23	No	78	0.87	7.03	2.83
SCT	mlsec	2017-11-21	No	100	0.37	397.37	69.82
SCT	mlsec	2017-11-24	No	100	0.30	654.68	80.88
SCT heavy	mlsec	2018-01-29	Yes	99	0.25	147.32	22.72
SCT light	gcloud	2018-02-02	Yes	100	0.79	22.89	17.49
SCT heavy	gcloud	2018-02-02	Yes	99	0.66	37.82	14.69
SCT light	mlsec	2018-02-13	Yes	98	0.43	61.81	22.23
SCT heavy	mlsec	2018-03-28	Yes	96	0.26	142.22	28.16
SCT 5aI	mlsec	2018-05-16	Yes	93	0.61	61.54	19.38
SCT medium	mlsec	2018-09-10	No	98	0.36	91.66	44.53
SCT medium	mlsec	2018-09-10	No	42	0.53	81.91	44.74
SCT medium	mlsec	2018-09-10	No	34	0.53	83.88	45.76
SCT light	mlsec	2018-09-13	No	42	0.58	60.43	64.73
SCT light	mlsec	2018-09-13	No	98	0.43	64.15	54.28
SCT light	mlsec	2018-09-13	No	34	0.57	63.37	65.59

Continued on next page

Continued from previous page

Scenario	Site	Date	Bridge	Size	Accuracy	S.OH [%]	T.OH [%]
SCT heavy	mlsec	2018-09-16	No	99	0.39	95.07	52.15
SCT heavy	mlsec	2018-09-16	No	98	0.30	118.73	47.33
SCT heavy	mlsec	2018-09-16	No	34	0.46	130.04	50.37
SCT heavy	mlsec	2018-10-01	Yes	98	0.28	146.68	6.74
SCT medium	mlsec	2018-10-05	Yes	98	0.39	91.50	3.89
SCT medium	mlsec	2018-10-05	Yes	98	0.39	91.50	3.89
SCT medium	mlsec	2018-10-05	Yes	98	0.39	91.50	3.89
LLaMA-nodelay	mlsec	2017-10-18	Yes	100	0.41	294.80	75.02
LLaMA-nodelay	mlsec	2017-10-18	Yes	100	0.43	294.80	75.02
LLaMA-nodelay	mlsec	2017-11-07	No	100	0.43	708.98	72.39
LLaMA-nodelay	mlsec	2017-11-28	No	30	0.61	254.32	318.52
LLaMA-nodelay	gcloud	2018-02-11	Yes	96	0.45	213.02	14.79
LLaMA-nodelay	mlsec	2018-02-25	Yes	97	0.36	205.18	29.04
LLaMA-nodelay	mlsec	2018-10-29	Yes	99	0.48	250.38	20.04
LLaMA	mlsec	2018-07-17	No	34	0.33	28.03	845.82
LLaMA	mlsec	2018-07-17	No	34	0.33	28.03	845.82
LLaMA	mlsec	2018-07-17	No	100	0.21	nan	nan
LLaMA	mlsec	2018-07-17	No	42	0.31	44.44	765.05

A.2. Complete 30 Sites CUMUL Results, Sorted by Date

Table A.2.: CUMUL-results on 30 site data sets. S. and T. OH denote size and time overhead, respectively. Acc. denotes Accuracy. OR-level refers to Panchenko et al. [24]'s outlier removal levels, as described in Panchenko et al. [8] and given in the code at `http://lorre.uni.lu/~andriy/zwiebelfreunde/`. small-removed describes if sites with less than 30 instances were removed from the data set.

Scenario	Date	Acc.	S. OH [%]	T. OH [%]	OR-lvl.	small removed
wtf-pad	2016-07-05	0.95	9.74	6.49	2	True
wtf-pad	2016-07-05	0.97	9.74	6.49	2	True

Continued on next page

Continued from previous page

Scenario	Date	Acc.	S. OH [%]	T. OH [%]	OR-lvl.	small re
simple1	2016-07-06	0.77	68.80	10.51	2	True
simple1	2016-07-07	0.94	7.82	7.57	2	True
new defense@30	2016-07-10	0.83	47.55	-4.33	2	True
new defense@30-burst	2016-07-11	0.83	49.35	3.86	2	True
tamaraw	2016-07-11	0.94	18.02	4.84	2	True
new defense@30al	2016-07-13	0.58	99.51	1.48	2	True
new defense@50al	2016-07-13	0.53	118.26	13.12	2	True
new defense	2016-07-15	0.62	119.57	-3.02	2	True
new defense@20	2016-07-17	0.85	17.49	-5.74	2	True
new defense@5	2016-07-17	0.91	0.39	-9.69	2	True
new defense@5	2016-07-17	0.91	3.43	-9.02	2	True
new defense@5all	2016-07-18	0.69	38.62	0.18	2	True
new defense@10-maybe-al	2016-07-23	0.61	51.05	13.48	2	True
new defense@2al	2016-07-23	0.73	34.51	2.23	2	True
new defense@30al	2016-07-25	0.56	81.13	27.80	2	True
new defense@5al	2016-07-25	0.69	46.36	4.73	2	True
new defense@50al	2016-07-26	0.52	96.25	15.53	2	True
defenseless	2016-08-14	0.97	0.00	0.00	2	True
defenseless	2016-08-15	0.97	0.00	0.00	2	True
defenseless	2016-08-15	0.98	0.00	0.00	2	True
new defense@5all	2016-08-25	0.63	40.59	14.12	2	True
new defense@5al	2016-08-26	0.67	30.35	16.82	2	True
new defense@5bll	2016-08-27	0.63	31.56	3.44	2	True
new defense@5bl	2016-08-27	0.67	37.07	9.08	2	True
defenseless	2016-08-29	0.91	0.00	0.00	2	True
defenseless	2016-08-29	0.91	0.00	0.00	2	True
defenseless	2016-08-29	0.91	0.00	0.00	2	True
defenseless	2016-09-09	0.89	0.00	0.00	2	True
new defense@20all	2016-09-10	0.61	65.77	9.41	2	True
new defense@20al	2016-09-10	0.63	64.78	9.11	2	True
new defense@20al	2016-09-10	0.64	64.78	9.11	2	True
new defense@20bll	2016-09-12	0.62	59.92	26.70	2	True

Continued on nex

Continued from previous page

Scenario	Date	Acc.	S. OH [%]	T. OH [%]	OR-lvl.	small removed
new defense@20bI	2016-09-13	0.63	52.80	13.17	2	True
defenseless	2016-09-18	0.89	0.00	0.00	2	True
defenseless	2016-09-30	0.89	0.00	0.00	2	True
defenseless	2016-10-03	0.93	0.16	0.15	2	True
defenseless	2016-10-03	0.87	0.00	0.00	2	True
defenseless	2016-10-06	0.96	0.00	0.00	2	True
new defense@20aII	2016-10-07	0.62	67.56	4.99	2	True
new defense@20aII	2016-10-07	0.62	67.38	5.02	2	True
new defense@20aI	2016-10-07	0.63	72.47	1.50	2	True
new defense@20bII	2016-10-08	0.59	74.02	2.75	2	True
new defense@20bI	2016-10-08	0.64	136.92	0.85	2	True
new defense@20bI	2016-10-08	0.64	136.92	0.85	2	True
new defense@5aII	2016-10-09	0.66	47.25	-1.46	2	True
new defense@5aII	2016-10-09	0.66	47.25	-1.46	2	True
new defense@5aI	2016-10-09	0.69	57.92	3.22	2	True
new defense@5aI	2016-10-09	0.69	57.92	3.22	2	True
new defense@5bII	2016-10-10	0.65	67.76	3.97	2	True
new defense@5bII	2016-10-10	0.65	67.76	3.97	2	True
new defense@5bI	2016-10-10	0.70	136.93	7.89	2	True
new defense@5bI	2016-10-10	0.70	136.93	7.89	2	True
defenseless	2016-10-16	0.74	-0.13	1.80	2	True
defenseless	2016-11-21	0.97	0.00	0.00	2	True
defenseless	2016-11-21	0.96	-0.41	1.59	2	True
new defense@10-with-errors	2017-01-19	0.64	128.28	4.99	2	True
new defense@10	2017-01-19	0.65	128.28	4.99	2	True
wtf-pad	2017-08-27	0.92	-1.05	6.26	2	True
wtf-pad	2017-08-29	0.93	0.87	40.60	2	True
defenseless	2017-08-31	0.90	0.00	0.00	2	True
wtf-pad	2017-09-06	0.94	-5.67	76.60	2	True
LLaMA	2017-09-07	0.35	17.04	3031.46	2	True
LLaMA	2017-09-07	0.26	17.04	3031.46	0	False
LLaMA-nodelay	2017-09-25	0.63	279.67	58.86	0	False

Continued on next page

Continued from previous page

Scenario	Date	Acc.	S. OH [%]	T. OH [%]	OR-lvl.	small re
LLaMA-nodelay	2017-09-25	0.63	279.67	58.86	2	True
new defense@175al	2017-10-15	0.34	390.85	103.21	2	True
defenseless	2017-10-16	0.58	0.00	0.00	2	True
new defense@150al	2017-10-17	0.36	416.45	95.96	2	True
new defense@100al	2017-12-14	0.46	284.35	65.92	2	True
defenseless	2017-12-15	0.59	0.00	0.00	2	True
LLaMA-nodelay	2017-12-17	0.72	348.13	56.64	2	True
defenseless	2017-12-18	0.62	0.00	0.00	2	True
defenseless	2017-12-25	0.98	0.00	0.00	2	True
defenseless	2017-12-26	0.96	0.00	0.00	2	True
new defense@100al	2017-12-27	0.42	162.20	-0.88	2	True
LLaMA-nodelay	2017-12-29	0.68	163.08	-4.02	2	True
defenseless	2017-12-31	0.97	0.00	0.00	2	True
LLaMA	2018-01-07	0.71	77.59	1126.55	2	True
LLaMA	2018-01-07	0.55	94.25	1085.25	0	False
defenseless	2018-01-12	0.99	0.00	0.00	2	True

SCT is denoted as `new defense@xxxyz`, which encodes configuration options described in 3.4:

- *factor* is `xxx`
- *cache* is denoted by the `y` position: if it is `a`, cached properties are used, if `b`, the properties are guessed from the HTTP model in the following section.
- In *target* configuration `I` the bin's border is chosen as the target value. Configuration `II` samples from the traffic distributions presented in the previous section.
- The `burst` option sends all of this traffic at the end, while the `noburst` discards this. The default is `noburst`.

If the sizes for a URL are not known, `b` is a fallback for `a` and `II` the fallback for `I`.

B. Open-World (False/True–Positive) Results

The WANG14 data set shows higher accuracy than many other captures. It also shows higher true- and lower false- positive rates. Its original data files were lost in a hard drive crash.

B.1. Table of all 100-Site Results

Table B.1.: Table of all results on 100 sites, sorted by scenario capture date. If the background-size is "auto" the same number as in the foreground set were chosen; if "max", the maximum number was chosen. For size overheads, refer to the closed-world accuracy results in section A.1 above.

Scenario	Date	Accuracy	Background-size	TPR	FPR
WANG14	2014-05-23	0.93	1000	0.985	0.051
WANG14	2014-05-23	0.94	2000	0.982	0.043
WANG14	2014-05-23	0.94	3000	0.977	0.028
WANG14	2014-05-23	0.95	5000	0.978	0.019
WANG14	2014-05-23	0.96	9000	0.973	0.014
WANG14	2014-05-23	0.95	8303	0.976	0.016
WANG14	2014-05-23	0.93	1000	0.985	0.066
WANG14	2014-05-23	0.93	2000	0.981	0.040
WANG14	2014-05-23	0.94	3000	0.982	0.029
WANG14	2014-05-23	0.95	5000	0.977	0.019

Continued on next page

Continued from previous page

Scenario	Date	Accuracy	Background-size	TPR	FPR
WANG14	2014-05-23	0.96	9000	0.973	0.014
WANG14	2014-05-23	0.95	8303	0.968	0.013
defenseless	2016-06-17	0.92	max	0.817	0.024
defenseless	2016-06-17	0.92	auto	0.923	0.036
defenseless	2016-08-30	0.84	auto	0.920	0.064
defenseless	2016-09-21	0.56	max	0.497	0.263
defenseless	2016-09-21	0.78	auto	0.795	0.142
defenseless	2016-09-26	0.84	auto	0.907	0.073
defenseless	2016-11-04	0.60	max	0.586	0.221
defenseless	2016-11-04	0.75	auto	0.821	0.123
defenseless	2016-11-27	0.56	max	0.396	0.188
defenseless	2016-11-27	0.80	auto	0.829	0.140
defenseless	2016-12-02	0.84	auto	0.849	0.069
defenseless	2016-12-23	0.74	max	0.741	0.181
defenseless	2016-12-23	0.83	auto	0.854	0.082
defenseless	2016-12-26	0.84	auto	0.926	0.046
defenseless	2016-12-26	0.87	auto	0.899	0.044
defenseless	2018-01-17	0.85	auto	0.875	0.126
defenseless	2018-02-02	0.55	auto	0.609	0.425
defenseless	2018-02-02	0.54	max	0.646	0.523
new defense (100al)	2018-02-02	0.61	auto	0.711	0.304
defenseless	2018-02-02	0.82	auto	0.846	0.171
defenseless	2018-02-02	0.89	100	0.992	0.900
defenseless	2018-02-02	0.87	200	0.988	0.855
defenseless	2018-02-02	0.84	500	0.973	0.710
defenseless	2018-02-02	0.81	1000	0.953	0.530
defenseless	2018-02-02	0.80	2000	0.897	0.305
defenseless	2018-02-02	0.80	3000	0.878	0.236
defenseless	2018-02-02	0.82	5000	0.827	0.149
defenseless	2018-02-02	0.84	max	0.814	0.123
new defense (20al)	2018-02-02	0.71	auto	0.725	0.209
defenseless	2018-02-02	0.82	auto	0.843	0.162

Continued on next page

Continued from previous page

Scenario	Date	Accuracy	Background-size	TPR	FPR
defenseless	2018-02-02	0.86	300	0.977	0.670
defenseless	2018-02-02	0.89	100	0.994	0.880
defenseless	2018-02-02	0.88	200	0.989	0.855
defenseless	2018-02-02	0.85	500	0.965	0.566
defenseless	2018-02-02	0.82	1000	0.943	0.446
defenseless	2018-02-02	0.80	2000	0.907	0.312
defenseless	2018-02-02	0.81	3000	0.875	0.219
defenseless	2018-02-02	0.82	5000	0.842	0.157
defenseless	2018-02-02	0.84	max	0.814	0.123
defenseless	2018-02-02	0.83	auto	0.829	0.137
defenseless	2018-02-02	0.85	4586	0.881	0.138
defenseless	2018-02-02	0.85	4586	0.869	0.123
defenseless	2018-02-02	0.60	5000	0.544	0.346
defenseless	2018-02-02	0.83	5000	0.839	0.140
defenseless	2018-02-02	0.85	4622	0.879	0.138
defenseless	2018-02-02	0.86	4538	0.879	0.131
defenseless	2018-02-02	0.61	4586	0.528	0.315
defenseless	2018-02-02	0.86	4586	0.877	0.131
defenseless	2018-02-02	0.85	4586	0.877	0.150
defenseless	2018-02-02	0.86	4586	0.880	0.133
defenseless	2018-02-02	0.61	4586	0.511	0.310
defenseless	2018-02-02	0.61	4586	0.527	0.316
defenseless	2018-02-02	0.61	4586	0.523	0.313
defenseless	2018-02-02	0.61	4586	0.521	0.311
defenseless	2018-02-02	0.61	4586	0.530	0.312
defenseless	2018-02-02	0.61	4538	0.527	0.318
defenseless	2018-02-02	0.61	4622	0.565	0.347
defenseless	2018-02-02	0.82	1000	0.999	0.998
defenseless	2018-02-02	0.70	2000	1.000	0.996
defenseless	2018-02-02	0.66	3000	0.874	0.658
defenseless	2018-02-02	0.60	5000	0.542	0.345
defenseless	2018-02-02	0.60	6754	0.000	0.000

Continued on next page

Continued from previous page

Scenario	Date	Accuracy	Background-size	TPR	FPR
defenseless	2018-02-02	0.62	4538	0.731	0.500
defenseless	2018-02-02	0.82	1000	0.945	0.439
defenseless	2018-02-02	0.82	1000	0.946	0.456
defenseless	2018-02-02	0.80	2000	0.914	0.313
defenseless	2018-02-02	0.82	3000	0.891	0.219
defenseless	2018-02-02	0.83	5000	0.847	0.158
defenseless	2018-02-02	0.84	6754	0.827	0.119
defenseless	2018-02-02	0.82	4538	0.846	0.153
new defense (100al)	2018-02-02	0.82	1000	1.000	0.991
new defense (100al)	2018-02-02	0.70	2000	1.000	0.993
new defense (100al)	2018-02-02	0.65	3000	0.859	0.683
new defense (100al)	2018-02-02	0.60	5000	0.653	0.456
new defense (100al)	2018-02-02	0.59	6783	0.000	0.000
new defense (100al)	2018-02-02	0.61	4622	0.723	0.511
new defense (100al)	2018-02-02	0.59	1000	0.928	0.656
new defense (100al)	2018-02-02	0.57	2000	0.835	0.474
new defense (100al)	2018-02-02	0.59	3000	0.733	0.327
new defense (100al)	2018-02-02	0.64	5000	0.593	0.203
new defense (100al)	2018-02-02	0.68	6783	0.443	0.110
new defense (100al)	2018-02-02	0.64	4622	0.636	0.227
new defense (20al)	2018-02-13	0.52	auto	0.347	0.147
new defense (20al)	2018-02-13	0.43	100	0.992	0.960
new defense (20al)	2018-02-13	0.42	200	0.986	0.885
new defense (20al)	2018-02-13	0.41	500	0.975	0.894
new defense (20al)	2018-02-13	0.39	1000	0.926	0.764
new defense (20al)	2018-02-13	0.40	2000	0.745	0.479
new defense (20al)	2018-02-13	0.44	3000	0.603	0.348
new defense (20al)	2018-02-13	0.61	max	0.175	0.053
new defense (20al)	2018-02-13	0.52	auto	0.345	0.144
new defense (20al)	2018-02-13	0.40	1000	0.935	0.734
new defense (20al)	2018-02-13	0.40	2000	0.771	0.512
new defense (20al)	2018-02-13	0.40	1000	0.935	0.734

Continued on next page

Continued from previous page

Scenario	Date	Accuracy	Background-size	TPR	FPR
new defense (20al)	2018-02-13	0.40	2000	0.771	0.512
new defense (20al)	2018-02-13	0.59	5000	0.615	0.426
new defense (20al)	2018-02-13	0.52	5000	0.000	0.000
new defense (20al)	2018-02-13	0.82	1000	1.000	1.000
new defense (20al)	2018-02-13	0.70	2000	0.998	0.989
new defense (20al)	2018-02-13	0.63	3000	0.834	0.669
new defense (20al)	2018-02-13	0.59	5000	0.618	0.430
new defense (20al)	2018-02-13	0.60	6780	0.000	0.000
new defense (20al)	2018-02-13	0.60	4554	0.700	0.498
new defense (20al)	2018-02-13	0.38	1000	0.915	0.757
new defense (20al)	2018-02-13	0.40	2000	0.722	0.456
new defense (20al)	2018-02-13	0.44	3000	0.556	0.297
new defense (20al)	2018-02-13	0.52	5000	0.000	0.000
new defense (20al)	2018-02-13	0.61	6780	0.175	0.053
new defense (20al)	2018-02-13	0.52	4554	0.410	0.183
LLaMA-nodelay	2018-02-25	0.35	1000	0.925	0.801
LLaMA-nodelay	2018-02-25	0.38	2000	0.685	0.430
LLaMA-nodelay	2018-02-25	0.43	3000	0.441	0.213
LLaMA-nodelay	2018-02-25	0.55	5000	0.111	0.033
LLaMA-nodelay	2018-02-25	0.59	6075	0.000	0.000
LLaMA-nodelay	2018-02-25	0.34	1000	0.930	0.822
LLaMA-nodelay	2018-02-25	0.51	4184	0.170	0.055
LLaMA-nodelay	2018-02-25	0.81	1000	0.996	0.971
LLaMA-nodelay	2018-02-25	0.68	2000	0.985	0.951
LLaMA-nodelay	2018-02-25	0.62	3000	0.815	0.647
new defense (100al)	2018-03-28	0.82	1000	0.998	0.960
new defense (100al)	2018-03-28	0.71	2000	0.990	0.941
new defense (100al)	2018-03-28	0.64	3000	0.895	0.748
new defense (100al)	2018-03-28	0.59	5000	0.589	0.402
new defense (100al)	2018-03-28	0.59	6557	0.000	0.000
new defense (100al)	2018-03-28	0.26	1000	0.923	0.787
new defense (100al)	2018-03-28	0.33	2000	0.645	0.391

Continued on next page

Continued from previous page

Scenario	Date	Accuracy	Background-size	TPR	FPR
new defense (100al)	2018-03-28	0.39	3000	0.444	0.245
new defense (100al)	2018-03-28	0.59	6557	0.000	0.000
new defense (100al)	2018-03-28	0.59	5000	0.589	0.402
new defense (100al)	2018-03-28	0.59	6557	0.000	0.000
new defense (100al)	2018-03-28	0.26	1000	0.923	0.787
new defense (100al)	2018-03-28	0.33	2000	0.645	0.391
new defense (100al)	2018-03-28	0.39	3000	0.444	0.245
new defense (100al)	2018-03-28	0.59	6557	0.000	0.000
new defense (100al)	2018-03-28	0.59	5000	0.589	0.402
new defense (100al)	2018-03-28	0.59	6557	0.000	0.000
new defense (100al)	2018-03-28	0.26	1000	0.923	0.787
new defense (100al)	2018-03-28	0.33	2000	0.645	0.391
new defense (100al)	2018-03-28	0.39	3000	0.444	0.245
new defense (100al)	2018-03-28	0.59	6557	0.000	0.000

B.2. CCDF-curves

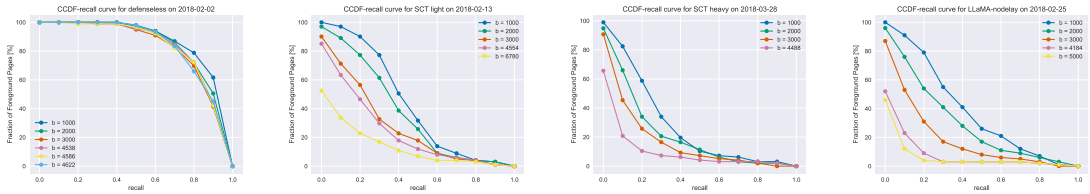

Figure B.1.: CCDF-curves for true-positive-rates for (left-to-right) (1) no defense, (2) new defense at 02-13, (3) new defense at 03-28, and (4) LLaMA-nodelay at 02-25. The number in the legend is the size of the background set/open world

.

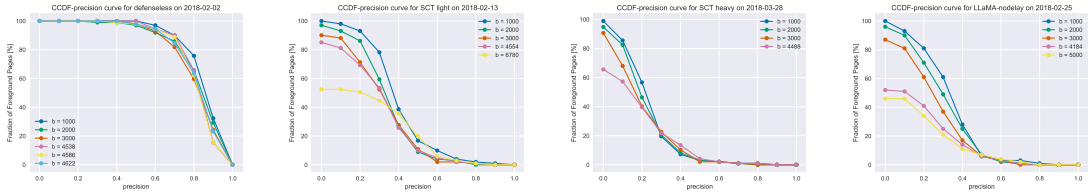

Figure B.2.: CCDF-curves for precision-rate for (left-to-right) (1) no defense, (2) new defense at 02-13, (3) new defense at 03-28, and (4) LLaMA-nodelay at 02-25. The number in the legend is the size of the background set/open world

.

Both sets of curves drop off steeply. Defenseless data has a very high true-positive rate and precision overall. SCT light already shows an effect in reducing true-positive rate and precision. Both LLaMA-nodelay and SCT reduce both true-positive rate and precision so much as to yield little to no positive tpr for bigger background sizes. *Precision* is defined as [60] (terms from 2.2).

$$TP/(TP + FP).$$

C. Determining Size of HTML-Documents for the Cache

The HTTP traffic model[71]'s statistical size generation is based on application-level[1] sizes on the network, as its authors analyzed log-files of the Squid[2] proxy.

These sizes could not be trivially obtained from the HTTP `Content-Length`-header[77, sec.3.3.2], as it does not represent additional headers and size-reduction via compression. Thus, the sizes were determined by retrieving the files with `wget` via Squid, and reading the sizes from the Squid log.

Retrieval is implemented in the `html_top_100.sh` script. It initially empties Squid's log file and cache by restarting it. Afterwards, the [[#top100][top-100] files are retrieved with `wget` via Squid.

From Squid's `access.log` log file, the sizes are extracted via the command

```
sudo cat /var/log/squid3/access.log | tr -s ' ' | cut -d ' ' -f 5,7 >
HTML-sizes
```

These sizes are then converted to a JSON[95]-array via the `htmlSizeToJSON.py`-file. It also checks for duplicate sizes for each file-URL, choosing the lower one. This could increase traffic, but the opposite might be too little traffic, thus easier website fingerprinting, which should be avoided.

[1] the data sizes transported by TCP
[2] `http://www.squid-cache.org`

D. Cached: Number of Embedded Objects

The second parameter for generating cover traffic is the number of embedded objects per HTML-page. These are extracted via the python script `htmlToNumEmbedded.py`.

To extract, Python's `lxml` module to parse the HTML's DOM [96] extracts the URLs of embedded files from the features of several tags, f.ex. the `src` element of `img` tags.

This approximation currently omits some possibly embedded elements, f.ex. those embedded in CSS files and `style` tags via the `@url` CSS-directive. It seems better for cover traffic to slightly underestimate the number of embedded elements. This might generate more traffic than strictly necessary, but here, safe seems better than sorry.

E. Bloom-sort

By ordering data into bins, it becomes possible to use Bloom filters for the estimation of sizes, using one Bloom filter for each bin.

To achieve this, sensible separation criteria (called *splits*) for the bins need to be found. Afterwards, each bin needs to be assigned a value (called *size*) for all contained elements. See appendix F on determining the sizes and splits.

This data-structure, called *Bloom-sort* is initialized with an array of splits, and an array of sizes. The sizes-array needs to have one more element than the splits-array, as the bins are bounded on the left by 0, and on the right by infinity.

```
/**
 * @param {sizes Array} array of values for each bin, must be sorted
 * @param {splits Array} array of bin borders, must be sorted
*/
function BloomSort(sizes, splits) {
    this.sizes = sizes;
    this.splits = splits;
    this.filters = [];
    for ( let i = 0; i < sizes.length; i++ ) {
        this.filters[i] = new Bloom.BloomFilter(NUM_BITS, NUM_HASH);
    }
}
```

Thus, you get

$$-\infty \leq \text{size}_0 \leq \text{split}_0 \leq \text{size}_1 \leq \text{split}_1 \leq ... \leq \text{split}_{n-1} \leq \text{size}_n < \infty$$

Given the splits, it becomes possible to add the elements to their bins:

```
BloomSort.prototype.add = function(id, size) {
    this.filters[_.sortedIndex(this.splits, size)].add(id);
};
```

where `_.sortedIndex()` gives the index at which `size` would be inserted into the sorted `this.splits` array.

The retrieval of element sizes looks into each Bloom filter, checking whether it might contain the element `id`. If one Bloom filter reports containment, its corresponding element- `size` is returned. If several or no Bloom filters report containment, an exception is thrown. The exception is used to allow all possible return values, not blocking one of them, say `-1`, for the error condition.

```
/** determines size of element, raises exception if unclear */
BloomSort.prototype.query = function(id) {
    let pos = -1;
    for ( let i = 0; i < this.filters.length; i++ ) {
        if ( this.filters[i].test(id) ) {
            if ( pos === -1 ) {
                pos = i;
            } else {
                throw {
                    name: 'BloomError',
                    message: 'Contains multiple entries'
                };
            }
        }
    }
```

```
    if ( pos === -1 ) {
        throw {
            name: 'BloomError',
            message: 'Contains no entries'
        };
    }
    return this.sizes[pos];
};
```

It is initialized with

```
let htmlSizes = new BloomSort.BloomSort([400, 1000, 20000], [700, 10000]);
```

then adding elements via `htmlSizes.add("http://google.com/", 613)` and querying via `htmlSizes.query("http://google.com/")`, which would yield `400`. See usage in `size-cache.js`.

F. Determining Bloom-Sort Parameters

It is impractical to store the sizes of many URLs, as this would greatly increase the size of the add-on. A possibility to save space is to use Bloom Filters to aggregate groups of URLs with similar values, as described in Bloom-sort.

Each group needs borders (given via the *splits* method parameter) and a size to represents its contained elements.

Determining the optimal number of groups, splits and sizes is a topic of further work. Here, initially the quantiles of the HTTP-model (see [[#HTTP traffic model][HTTP traffic model]) were used. When the data were to be inserted, it turned out that especially the numbers of embedded elements did not match the theoretically proposed groups:

For three groups, the splits would be given by the 33 1/3 and 66 2/3 quantiles, as 0.0107 and 1.481. This would create two single-valued groups, which only would contain the elements 0 and 1. The next group would contain all other elements: The (representative) sizes of the groups were given as 7.915E-05, 0.188, and 8.260 (quantiles 16 1/6, 50, and 83 5/6).

The actual data to be inserted (see Cached: Number of Embedded Objects) had the *splits* (quantiles) at 10 2/3 and 36 2/3 and the *sizes* (middle quantile) at 6, 20, and 59 2/3.

In addition to using the observed sizes for the Bloom filter, the number of groups was increased to 5.

Bibliography

[1] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, August 2004.

[2] Eric Jardine. Tor, what is it good for? political repression and the use of online anonymity-granting technologies. *new media & society*, 2016.

[3] Supreme Court of the United States. Katz v. United States, 389 U. S. 347, 1967. 88 S. Ct. 507; 19 L. Ed. 2d 576; 1967 U.S. LEXIS 2.

[4] Parlamentarischer Rat. Grundgesetz für die Bundesrepublik Deutschland. `http://www.bundestag.de/dokumente/rechtsgrundlagen/grundgesetz/gg_01.html`, Mai 1949. Zuletzt geändert durch Art. 1 G v. 13.7.2017 I 2347.

[5] Sir Karl Popper. *The open society and its enemies*. Routledge, 2012.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Challenges in deploying low-latency anonymity. Available at `https://www.onion-router.net/Publications/challenges.pdf`, 2005.

[7] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. Revised edition of [64], April 1997.

[8] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas 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.

[9] Mike Perry. Experimental defense for website traffic fingerprinting. `https://blog.torproject.org/blog/`

experimental-defense-website-traffic-fingerprinting, 2011. [Online; accessed 14-September-2015].

[10] Andrew Hintz. Fingerprinting websites using traffic analysis. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[11] Charles Wright, Scott Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium - NDSS '09*. IEEE, February 2009.

[12] 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 *Proceedings of the Network and Distributed Security Symposium - NDSS '11*. IEEE, 2011.

[13] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas 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*, May 2012.

[14] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23th USENIX Security Symposium*, 2014.

[15] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 21th ACM conference on Computer and Communications Security (CCS 2014)*, November 2014.

[16] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. WTF-PAD: toward an efficient website fingerprinting defense for tor. *CoRR*, abs/1512.00524, 2015. URL http://arxiv.org/abs/1512.00524.

[17] Tao Wang and Ian Goldberg. Walkie-talkie: An effective and efficient defense against website fingerprinting. Technical report, Technical Report 2015-09, CACR, 2015. http://cacr.uwaterloo.ca/techreports/2015/cacr2015-09.pdf, 2015.

[18] Giovanni Cherubin, Jamie Hayes, and Marc Juárez. Website fingerprinting defenses at the application layer. *PoPETs*, 2017(2):186–203, 2017. doi: 10.1515/popets-2017-0023. URL https://doi.org/10.1515/popets-2017-0023.

[19] Jake Brutlag. Speed matters for google web search, 2009.

[20] Shailen Mistry and Bhaskaran Raman. Quantifying traffic analysis of encrypted web-browsing. Project paper, University of Berkeley. Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5823&rep=rep1&type=pdf, 1998.

[21] Heyning Cheng and Ron Avnur. Traffic analysis of SSL encrypted web browsing. Project paper, University of Berkeley. Available at http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps as of [8], 1998.

[22] Dominik Herrmann, Rolf Wendolsky, and Hannes 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, October 2009. ACM. ISBN 978-1-60558-784-4. doi: http://doi.acm.org/10.1145/1655008.1655013.

[23] Xiang Cai, Xincheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*, October 2012.

[24] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at internet scale. In *Proceedings of the 23rd Internet Society (ISOC) Network and Distributed System Security Symposium (NDSS 2016)*., 2016. Available at https://www.comsys.rwth-aachen.de/fileadmin/papers/2016/2016-panchenko-ndss-fingerprinting.pdf.

[25] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, Austin, TX, 2016. USENIX Association. ISBN 978-1-931971-32-4. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes.

[26] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. *arXiv preprint arXiv:1801.02265*, 2018.

[27] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[28] Philipp Winter and Stefan Lindskog. How the great firewall of china is blocking Tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet, FOCI '12, Bellevue, WA, USA, August 6, 2012*, 2012. URL `https://www.usenix.org/conference/foci12/workshop-program/presentation/winter`.

[29] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21 (2):120–126, 1978.

[30] Tor Project. Tor style guide. `https://media.torproject.org/image/Tor%20Style%20Guide%20v1.3.pdf`, mar 2017.

[31] J. Postel. Internet Protocol. RFC 791, RFC Editor, September 1981. URL `http://www.ietf.org/rfc/rfc791.txt`.

[32] Roger Dingledine, Nick Mathewson, Steven Murdoch, and Paul Syverson. Tor: The second-generation onion router (2014 draft v1). Revised edition of [1]. Available at `http://sec.cs.ucl.ac.uk/users/smurdoch/papers/tor14design.pdf`, August 2014.

[33] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. On flow correlation attacks and countermeasures in mix networks. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*, pages 207–225, May 2004.

[34] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 20th ACM conference on Computer and Communications Security (CCS 2013)*, November 2013.

[35] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42): 230–265, 1936. URL `http://www.cs.helsinki.fi/u/gionis/cc05/OnComputableNumbers.pdf`.

[37] Konrad Rieck. *Machine Learning for Application-Layer Intrusion Detection*. PhD thesis, 2009.

[38] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2008.

[39] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9): 569–597, 2008.

[40] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[41] Stuart Russell and Peter Norvig. *Artificial Intelligence - A modern approach*. 1995.

[42] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, first edition, 1997. ISBN 0-07-042807-7.

[43] Alex Smola and S.V.N. Vishwanathan. *Introduction to Machine Learning*. Cambridge University Press, 2008.

[44] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.

[45] *PCAP(3PCAP) manual*, Apr 2014.

[46] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2013)*. ACM, November 2013.

[47] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246, RFC Editor, August 2008. URL http://www.ietf.org/rfc/rfc5246.txt.

[48] J. Postel. Transmission Control Protocol. RFC 793, RFC Editor, September 1981. URL http://www.ietf.org/rfc/rfc793.txt. Updated by RFCs 1122, 3168, 6093, 6528.

[49] A. Freier, P. Karlton, and P. Kocher. The ssl protocol version 3.0. Technical report, IETF Draft (Netscape Communications), 1996. Available at https://tools.ietf.org/id/draft-ietf-tls-ssl-version3-00.txt.

[50] Roger Dingledine and Nick Mathewson. Tor Protocol Specification. Technical report, Tor Project Inc., https://git.torproject.org/torspec.git, Jan 2016. most current edit as of commit f9e111ea.

[51] Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, and Silvia Pfeiffer. Html5. Technical report, WHATWG and W3C, 2014. Available at https://www.w3.org/TR/html5/single-page.html.

[52] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL http://dx.doi.org/10.1023/A:1010933404324.

[53] Thomas G Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2: 263–286, 1994.

[54] K. R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2): 181–201, Mar 2001. ISSN 1045-9227. doi: 10.1109/72.914517.

[55] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. The MIT Press, 2002.

[56] Chih wei Hsu, Chih chung Chang, and Chih jen Lin. A practical guide to support vector classification, 2003. Available at http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

[57] Alex J Smola, Bernhard Schölkopf, and Klaus-Robert Müller. The connection between regularization operators and support vector kernels. *Neural networks*, 11(4):637–649, 1998.

[58] scikit-learn developers. *scikit-learn user guide*, release 0.16.1 edition, apr 2015.

[59] Leo Breiman and Philip Spector. Submodel selection and evaluation in regression. the x-random case. *International statistical review/revue internationale de Statistique*, pages 291–319, 1992.

[60] David M.W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011. ISSN 2229-3981.

[61] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8): 861–874, 2006.

[62] George Danezis and Richard Clayton. Introducing traffic analysis, Feb 2007. Available at `https://www.cl.cam.ac.uk/~rnc1/TAIntro-book.pdf`.

[63] Ed. R. Callon. The twelve networking truths. RFC 1925, RFC Editor, April 1, 1996. URL `http://www.ietf.org/rfc/rfc1925.txt`.

[64] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *IN PROCEEDINGS OF THE SECOND UNIX WORKSHOP ON ELECTRONIC COMMERCE*, pages 29–40. USENIX Association, 1996.

[65] Ed. R. Fielding and Ed. J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, RFC Editor, June 2014. URL `http://www.rfc-editor.org/rfc/rfc7231.txt`.

[66] Marc Liberatore and Brian Neil Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS 2006)*, pages 255–263, November 2006.

[67] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, second edition, 1996.

[68] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of ESORICS 2006*, September 2006.

[69] Giovanni Cherubin, Jamie Hayes, and Marc Juárez. Llama, a tor browser add-on to hamper website fingerprinting. https://github.com/camelids/LLaMA, 2017. Commit-ID 73ed026f of 2017-06-16.

[70] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. 2018.

[71] Jeongeun Julie Lee and Maruti Gupta. A new traffic model for current user web browsing behavior. *Intel corporation*, 2007. White Paper, Intel. Available at http://blogs.intel.com/wp-content/mt-content/com/research/HTTP%20Traffic%20Model_v1%201%20white%20paper.pdf.

[72] Elaine M Newton, Latanya Sweeney, and Bradley Malin. Preserving privacy by de-identifying face images. *IEEE transactions on Knowledge and Data Engineering*, 17(2):232–243, 2005.

[73] Sergey Foss, Dmitry Korshunov, Stan Zachary, et al. *An introduction to heavy-tailed and subexponential distributions*, volume 6. Springer, 2011.

[74] International Ecma. *ECMAScript Language Specification*. Rue du Rhone 114; CH-1204 Geneva, 5.1 edition, June 2011. Available at http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf as of [97].

[75] R. Saucier. Computer Generation of Statistical Distributions. Technical Report ARL-TR-2168, Army Research Laboratory, March 2000.

[76] community SciPy. *SciPy Reference Guide*, release 0.17.0 edition, feb 2016. Available at http://scipy.org (or the Debian package *python-scipy-doc*.

[77] Ed. R. Fielding and Ed. J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, RFC Editor, June 2014. URL http://www.rfc-editor.org/rfc/rfc7230.txt.

[78] James F. Kurose and Keith W. Ross. *Computer networking - a top-down approach featuring the internet*. Addison-Wesley-Longman, 2001. ISBN 978-0-201-47711-5.

[79] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, RFC Editor, May 1996. URL http://www.ietf.org/rfc/rfc1945.txt.

[80] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[81] Sunghwan Ihm and Vivek S. Pai. Towards understanding modern web traffic. In *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11, Berlin, Germany, November 2-, 2011*, pages 295–312, 2011. doi: 10.1145/2068816.2068845. URL http://doi.acm.org/10.1145/2068816.2068845.

[82] Mike Perry, Erinn Clark, and Steven Murdoch. The design and implementation of the Tor browser. Draft, Tor Project, may 2015. Available at https://www.torproject.org/projects/torbrowser/design/.

[83] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[84] Mozilla Automation and Tools team. *Marionette Python Client Documentation*, July 2015.

[85] Tor Project. *tor (1) Linux User's Manual*, tor v0.2.6.10 edition, May 2015.

[86] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 21th ACM conference on Computer and Communications Security (CCS 2014)*, November 2014.

[87] Ronan Cremin. The web is doom. https://mobiforge.com/research-analysis/the-web-is-doom, 2016. [Online; accessed 02-May-2016].

[88] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[89] Jonathan J Koehler. The base rate fallacy reconsidered: Descriptive, normative, and methodological challenges. *Behavioral and brain sciences*, 19(01):1–17, 1996.

[90] Daniel Arp, Fabian Yamaguchi, and Konrad Rieck. Torben: Deanonymizing tor communication using web page markers. Technical report, University of Göttingen, Germany, 2014.

[91] Yong Guan, Chengzhi Li, Dong Xuan, Riccardo Bettati, and Wei Zhao. Preventing traffic analysis for real-time communication networks. In *MilCom*, volume 1, pages 744–750, 1999.

[92] Nick Mathewson. Special Hostnames in Tor. Technical report, Tor Project, `https://git.torproject.org/torspec.git`, Sep 2011. most current edit as of commit 06cf12f8.

[93] Marius Kloft, Ulf Brefeld, Sören Sonnenburg, and Alexander Zien. ℓ_p-norm multiple kernel learning. *Journal of Machine Learning Research*, 12(Mar):953–997, 2011.

[94] Saman Feghhi and Douglas J Leith. A web traffic analysis attack using only timing information. *IEEE Transactions on Information Forensics and Security*, 11(8): 1747–1759, 2016.

[95] Ed. T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, RFC Editor, March 2014. URL `http://www.rfc-editor.org/rfc/rfc7159.txt`.

[96] W3C. Document Object Model (DOM) Level 3 Core Specification. Technical report, W3C, April 2004.

[97] Douglas Crockford. *JavaScript: The Good Parts*. Yahoo! Inc., 2008. ISBN 978-0-596-51774-8.

Index

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Hannover, den 23.11.2018 .....

Michael Kreikenbaum