Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail

Kevin P. Dyer*, Scott E. Coull†, Thomas Ristenpart‡, and Thomas Shrimpton*

*Department of Computer Science, Portland State University, Portland, USA. Email: {kdyer, teshrim}@cs.pdx.edu
† RedJack, LLC., Silver Spring, MD, USA Email: scott.coull@redjack.com
‡Department of Computer Sciences, University of Wisconsin-Madison, USA. Email: rist@cs.wisc.edu

***Abstract*—**
We consider the setting of HTTP traffic over encrypted tunnels, as used to conceal the identity of websites visited by a user. It is well known that traffic analysis (TA) attacks can accurately identify the website a user visits despite the use of encryption, and previous work has looked at specific attack/countermeasure pairings. We provide the first comprehensive analysis of general-purpose TA countermeasures. We show that nine known countermeasures are vulnerable to simple attacks that exploit coarse features of traffic (e.g., total time and bandwidth). The considered countermeasures include ones like those standardized by TLS, SSH, and IPsec, and even more complex ones like the traffic morphing scheme of Wright et al. As just one of our results, we show that despite the use of traffic morphing, one can use only total upstream and downstream bandwidth to identify — with 98% accuracy— which of two websites was visited. One implication of what we find is that, in the context of website identification, it is unlikely that bandwidth-efficient, general-purpose TA countermeasures can ever provide the type of security targeted in prior work.

***Keywords*-traffic analysis countermeasures; privacy; machine learning; padding; encrypted traffic**

I. Introduction

Internet users increasingly rely on encrypted tunnels to keep their web browsing activities safe from eavesdroppers. A typical scenario involves a user establishing an encrypted tunnel to a proxy that then relays all subsequent HTTP traffic (in both directions) through the tunnel. Another is when one browses the web on a wireless network that uses WPA to encrypt all traffic. In both cases, the use of encryption should hide the contents of the traffic and, intuitively, the identity of the destination website(s). Yet modern encryption does not obfuscate the length of underlying plaintexts, nor the number of plaintexts that are encrypted. This information may seem harmless, but in fact it enables *traffic analysis* (TA) attacks. Among other things, TA attacks can reveal the identity of the websites viewed by the user [1, 9, 10, 15, 19].

One commonly suggested TA countermeasure is to hide the plaintext length by adding padding prior to encryption. Padding countermeasures are standardized in TLS, explicitly to "frustrate attacks on a protocol that are based on analysis of the lengths of exchanged messages" [5]. Similar allowances for padding appear in SSH and IPSec. More advanced countermeasures, such as traffic morphing [19], manipulate whole streams of packets in order to precisely mimic the distribution of another website's packet lengths.

The seemingly widespread intuition behind these countermeasures is that they patch up the most dangerous side channel (packet lengths) and so provide good protection against TA attacks, including website identification. Existing literature might appear to support this intuition. For example, Liberatore and Levine [10] show that padding packets to the network MTU (e.g., 1500 bytes) reduces the accuracy of one of their attacks from 98% to 7%.

Our results strongly challenge this intuition. We perform the first comprehensive analysis of low-level countermeasures (e.g., per-packet padding) for the kind of website identification attacks considered by prior work (c.f., [8, 10, 14, 22]): a closed-world setting for privacy sets, in which the *a priori* set of possible websites a user might visit is known to the attacker, coupled with the ability for the attacker to train and test on traffic traces that are free of real-world artifacts (e.g., caching effects, interleaved flows, and user-specific content). We consider nine distinct countermeasures, apply them to two large, independent datasets of website downloads, and pit the resulting obfuscated traffic against a total of seven different attacks. The results are summarized in Figure 1. What we uncover is surprisingly bleak:

None of the countermeasures are effective. We show that two classifiers —a new naïve Bayes classifier called VNG++ and a support vector machine classifier due to Panchenko et al. [14]— achieve better than 80% accuracy in identifying which of $k = 128$ websites was visited in a closed-world experiment. (Random guessing achieves 0.7% accuracy.) When $k = 2$ these classifiers achieve over 98% accuracy. This holds for all nine countermeasures considered, including ones inspired by the SSH, TLS and IPSec RFCs, and state-of-the-art ones such as traffic morphing [21].

Hiding packet lengths is not sufficient. We initiate a study of classifiers that do not directly use fine-grained features such as individual packet lengths. The VNG++ classifier just mentioned uses only "coarse" information, including overall time, total bandwidth, and size of bursts. In fact, we provide a naïve Bayes classifier that uses *only* the total bandwidth for training and testing, yet still achieves greater

Attack	Classifier	Features Considered	$k = 2$	$k = 128$	$k = 775$
Liberatore and Levine [10] (LL)	naïve Bayes (NB)	Packet lengths	85%	25%	8%
Herrmann et al. [8] (H)	multinomial naïve Bayes (MNB)	Packet lengths	71%	3%	0%
Panchenko et al. [14] (P)	support vector machine (SVM)	Packet lengths, Order, Total bytes	99%	82%	63%
Time (TIME)	naïve Bayes	Total trace time	82%	9%	3%
Bandwidth (BW)	naïve Bayes	Upstream/Downstream total bytes	98%	41%	18%
Variable n-gram (VNG)	naïve Bayes	Bytes in traffic bursts	99%	69%	54%
VNG++	naïve Bayes	Total trace time, Upstream/Downstream total bytes, Bytes in traffic bursts	99%	80%	61%

Figure 1. Summary of attacks evaluated in our work. The $k = 2$, $k = 128$ and $k = 775$ columns indicate the classifier accuracy for a privacy set of size k when using the most effective countermeasure for the Herrmann dataset (see Section II).

than 98% accuracy at $k = 2$ and 41% accuracy at $k = 128$. This implies that any effective countermeasure must produce outputs that consume indistinguishable amounts of bandwidth.

Coarse information is unlikely to be hidden efficiently. Our coarse-feature attacks, in particular the bandwidth-only attack, strongly suggest that resource-efficient countermeasures will not (on their own) effectively hide website identity within a small privacy set. So, we investigate an inefficient strawman countermeasure, Buffered Fixed-Length Obfuscation (BuFLO, pronounced "buffalo"), that combines and makes concrete several previous suggestions: it sends packets of a fixed size at fixed intervals, using dummy packets to both fill in and (potentially) extend the transmission. We subject it to the same analysis as the other countermeasures. This analysis shows that should BuFLO fail to obfuscate total time duration and total bandwidth, then attacks *still* achieve 27% accuracy at $k = 128$. With a bandwidth overhead of over 400%, we can, in theory, finally reduce $k = 128$ accuracy to 5%.

Relevance to other settings. While the adversarial model that we consider is consistent with previous work, we admit that there are several factors (e.g., caching, open-world identification) that are not captured. Indeed, these may reduce the effectiveness of the attacks, and improve countermeasure efficacy, in practice. There may also be some other settings, such as Voice over IP (VoIP) traffic [18–21], where the nature of the application-layer protocol enables some countermeasures to work very well. That said, the model considered in this paper (and its predecessors) is one that a general-purpose countermeasure ought to cover.

Finally, our analysis does not cover application-layer countermeasures such as Camouflage [8] and HTTPOS [12], which both make intimate use of spurious HTTP requests to help obfuscate traffic patterns. We suspect, however, that the lessons learned here might help direct future analysis of application-layer countermeasures, as well.

II. Experimental Methodology

Like previous works [8, 10, 14, 22], our experiments simulate a closed-world setting in which an adversary has access to the timing, lengths, and directionality of packets sent over an encrypted HTTP tunnel (e.g., to or from a proxy server). We assume secure encryption algorithms are used and no information can be learned from the encrypted contents itself.

We base our simulation on two datasets that have been widely used by previous works on web page identification. The Liberatore and Levine dataset [10] contains timestamped traces from 2,000 web pages. The Herrmann et al. [8] dataset contains timestamped traces from 775 web pages. A *trace* is defined as a record of the lengths and timings of ciphertexts generated by accessing a web page using an OpenSSH single-hop SOCKS proxy. Please refer to the previous works [8, 10] for further details about data collection methodology.

Each of our experiments is performed with respect to a particular classifier, a particular countermeasure, and a specified set of n web pages. An experiment consists of a number of trials; we will say in a moment how the particular number of trials is determined. At the start of each experimental trial, we uniformly select a subset of $k \leq n$ web pages to define the privacy set for that trial.[1] Next we establish k sets of 20 traces, one for each web page, as follows. For every web page in the data set, there are $m > 20$ chronologically sorted sample traces. We select a random trace index $i \in \{0, 1, \ldots, m - 19\}$, and take traces $i, i + 1, \ldots, i + 19$ for each of the k web pages. The first $t = 16$ of the traces from each of the k sets are used as the training data for the classifier, and the remaining $T = 4$ traces form the testing data set.[2] The countermeasure is applied to both the training and testing data, and the classifier is trained and then tested to determine its accuracy. Classifier accuracy is calculated as

[1] We do not believe the uniform distribution represents typical user web-browsing behavior. In practice, we expect that biased sampling from the privacy set would further aid an attacker.

[2] We considered values of $t \in \{4, 8, 12, 16\}$ and observed effects consistent with those reported by Liberatore and Levine [10]: as t increases there was a consistent, modest increase in classification accuracy.

(c/Tk), where c is the number of correctly classified test traces and k is our privacy set size.

In each experiment, we perform $2^{(15-\log_2(k))}$ trials, so that there are a total of $T \cdot 2^{15}$ test data points per experiment. We consider values of $k \in \{2, 4, 8, 16, 32, 64, 128, 256, 512, 775\}$ in order to capture countermeasure performance across a number of scenarios. Intuitively, smaller values of k present easier classification (attack) settings, and larger values of k present more difficult classifier settings.

We note that the engineering effort required to produce our results was substantial. To aid future research efforts, the Python framework used for our experiments is publicly available[3].

III. Traffic Classifiers

A sequence of works detail a variety of TA attacks, in the form of classifiers that attempt to identify the web page visited over an encrypted channel. These classifiers use *supervised* machine learning algorithms, meaning they are able to train on traces that are labeled with the destination website. Each algorithm has a *training* and a *testing* phase. During training, the algorithm is given a set $\{(X_1, \ell_1), (X_2, \ell_2), \ldots, (X_n, \ell_n)\}$, where each X_i is an *vector* of *features* and ℓ_i is a *label*. During testing the classification algorithm is given a vector Y and must return a label. In our case, a vector X_i contains information about the lengths, timings, and direction of packets in the encrypted connection containing a web page ℓ_i, and the format of a vector X_i is dependent upon the classifier. In the remainder of this section, we present a high-level overview of the operation of the three published classifiers that we use in our evaluation, and we refer interested readers to more detailed descriptions elsewhere [8, 10, 13, 14].

A. Liberatore and Levine Classifier

Liberatore and Levine [10] (LL) proposed the use of a naïve Bayes classifier (NB) to identify web pages using the direction and length of the packets. The naïve Bayes classifier determines the conditional probability $\Pr(\ell_i|Y)$ for a given vector of features Y using Bayes' rule: $\Pr(\ell_i|Y) = \frac{\Pr(Y|\ell_i)\Pr(\ell_i)}{\Pr(Y)}$. The probability is computed for all labels ℓ_i with $i = \{1, 2, \ldots, k\}$ and k representing the size of the privacy set (or number of labels being considered), and the label with the highest probability is selected as the classifier's guess. The probability $\Pr(Y|\ell_i)$ is estimated using kernel density estimation over the example feature vector provided during training, and $\Pr(\ell_i)$ is assumed to be $1/k$. The feature vectors used by the LL classifier are derived from the count of the lengths of the packets sent in each direction of the encrypted connection. Specifically, the feature vector contains $2 \cdot 1449 = 2898$ integers that represent the number of packets seen in the given vector with each of the potential direction and packet length combinations (i.e., $\{\uparrow, \downarrow\} \times \{52, \ldots, 1500\}$). For example, if we observe a packet of length 1500 in the $\downarrow$ direction (e.g., server to client) we would increment the counter for ($\downarrow$,1500).

B. Herrmann et al. Classifier

Herrmann, Wendolsky and Fedarrath [8] (H) take a similar approach to Liberatore and Levine, however they make use of a multinomial naïve Bayes (MNB) classifier. Like the naïve Bayes classifier with density estimation, the multinomial naïve Bayes classifier attempts to estimate the probability $\Pr(\ell_i|Y)$ for each of the $i = \{1, 2, \ldots, k\}$ potential labels and the given feature vector Y. The key difference is that the multinomial classifier does not apply density estimation techniques to determine the probability $\Pr(Y|\ell_i)$, but instead uses the aggregated frequency of the features (i.e., normalized distribution) across all training vectors. Thus, the H classifier uses normalized counts of $(\textit{direction}, \textit{length})$, whereas the LL classifier examined raw counts. Furthermore, Herrmann et al. suggest a number of approaches for normalizing these counts. For our evaluation, we combine *term frequency transformation* and *cosine normalization*, as these were identified by Herrmann et al. to be the most effective in the SSH setting.

C. Panchenko et al. Classifier

Panchenko et al. [14] (P) take a completely different approach by applying a support vector machine (SVM) classifier to the problem of identifying web pages. A support vector machine is a type of binary linear classifier that classifies points in a high-dimensional space by determining their relation to a separating hyperplane. In particular, the SVM is trained by providing labeled points and discovering a hyperplane that maximally separates the two classes of points. Classification occurs by determining where the point in question lies in relation to the splitting hyperplane. Due to the complexity of SVM classifiers, we forego a detailed discussion of their various parameters and options.

We configure our SVM as follows. We use the same radial basis function (RBF) kernel as Panchenko et al. with parameters of $C = 2^{17}$ and $\gamma = 2^{-19}$. The P classifier uses a wide variety of coarse and detailed features of the network data mostly derived from packet lengths and ordering. Some of these features include the total number of bytes transmitted, total number of packets transmitted, proportion of packets in each direction, and raw counts of packet lengths. There are also several features known as "markers" that delineate when information flow over the encrypted connection has changed direction. These markers aggregate bandwidth and number of packets into discrete chunks. Each of the features considered by the P classifier are rounded and all 52 byte TCP acknowledgement packets are removed to minimize noise and variance in the training and testing vectors.

[3] `http://www.kpdyer.com/`

IV. Countermeasures

For ease of exposition and analysis, we organize the considered countermeasures into three categories: those that are inspired by the padding allowed within the SSH, TLS and IPSec standards (Type-1); other padding-based countermeasures (Type-2); and countermeasures that make explicit use of source and target packet-length distributions (Type-3). In what follows, we describe the operation of the countermeasures we evaluate and discuss the overhead they generate. Lengths are always measured in bytes.

A. Type-1: SSH/TLS/IPSec-Motivated Countermeasures

A common suggestion, already used in some implementations, like GnuTLS[4], is to obfuscate plaintext lengths by choosing random amounts of extra padding to append to the plaintext prior to encryption. SSH, TLS and IPSec allow up to 255 bytes of padding in order to align the to-be-encrypted plaintext with the underlying block cipher boundary, and also to provide some obfuscation of the original plaintext length. We consider two ways in which this might be implemented within SSH/TLS/IPSec: (1) choose a single random amount of padding to be applied across all plaintexts in the session, or (2) choose a random amount of padding for each plaintext.

Session Random 255 padding: A uniform value $r \in \{0, 8, 16 \ldots, 248\}$ is sampled and stored for the session.[5] Each packet in the trace has its length field increased by r, up to a maximum of the MTU.

Packet Random 255 padding: Same as Session Random 255 padding, except that a new random padding length r is sampled for each input packet.

We note that our simulation of Session Random and Packet Random padding in this setting are not exactly what would be implemented in reality because we do not have access to the size of the plaintext data from the datasets available to us. Instead, our assumption is that the plaintext data is sufficiently short to fit into a single TCP packet and therefore is closely approximated by simply adding the padding to the length of the ciphertext. What we simulate, therefore, is likely to overstate the efficacy of the countermeasure since the (at most) 255 bytes of padding would be dominated by the true size of the plaintext (e.g., up to 2^{14} bytes for TLS), thereby providing relatively little noise. In contrast, our simulation allows for a much larger ratio of plaintext to padding, which in turn adds significantly more noise.

B. Type-2: Other Padding-based Countermeasures

The second class of countermeasure we consider are those padding mechanisms that are not easily supported in existing encrypted network protocol standards due to the amount of padding added. In this scenario, we assume the countermeasure will be capable of managing fragmentation and padding of the data before calling the encryption scheme. Most of the countermeasures considered by prior work fall into this category, though we also consider a randomized scheme that has not been previously explored.

Linear padding: All packet lengths are increased to the nearest multiple of 128, or the MTU, whichever is smaller.

Exponential padding: All packet lengths are increased to the nearest power of two, or the MTU, whichever is smaller.

Mice-Elephants padding: If the packet length is ≤ 128, then the packet is increased to 128 bytes; otherwise it is padded to the MTU.

Pad to MTU: All packet lengths are increased to the MTU.

Packet Random MTU padding: Let $\mathtt{M}$ be the MTU and ℓ be the input packet length. For each packet, a value $r \in \{0, 8, 16, \ldots, \mathtt{M} - \ell\}$ is sampled uniformly at random and the packet length is increased by r.

C. Type-3: Distribution-based Countermeasures

Wright et al. [22] presented two novel suggestions as improvements upon traditional per-packet padding countermeasures: *direct target sampling* (DTS) and *traffic morphing* (TM). On the surface, both techniques have the same objective. That is, they augment a protocol's packets by chopping and padding such that the augmented packets appear to come from a pre-defined target distribution (i.e., a different web page). Ideally, DTS and TM have security benefits over traditional per-packet padding strategies because they do not preserve the underlying protocol's number of packets transmitted nor packet lengths. Although the full implementations details of DTS and TM are beyond scope of this paper (see [22]), we give a high-level overview here.

Direct target sampling: Given a pair of web pages A and B, where A is the source and B is the target, we can derive a probability distribution over their respective packet lengths, D_A and D_B. When a packet of length i is produced for web page A, we sample from the packet length distribution D_B to get a new length i'. If $i' > i$, we pad the packet from A to length i' and send the padded packet. Otherwise, we send i' bytes of the original packet and continue sampling from D_B until all bytes of the original packet have been sent. Wright et al. left unspecified morphing with respect to packet timings. We assume a negligible overhead to perform morphing and specify a 10ms inter-packet delay for dummy packets.

In our experiments, we select the target distribution uniformly at random from our set of k potential identities. The selected web page remains unchanged (i.e., no

[4] `http://www.gnu.org/software/gnutls/`

[5] We assume that the underlying encryption block size is 8 bytes. For the Liberatore and Levine dataset, we know this assumption is true. We do not expect classification accuracies to be different if, in fact, the block size was 16 bytes.

	Overhead (%)	
Countermeasure	LL	H
Session Random 255	9.0	7.1
Packet Random 255	9.0	7.1
Linear	4.2	3.4
Exponential	8.7	10.3
Mice-Elephants	41.6	39.3
Pad to MTU	81.2	58.1
Packet Random MTU	40.1	28.8
Direct Target Sampling	86.4	66.5
Traffic Morphing	60.8	49.8

Figure 2. Bandwidth overhead of evaluated countermeasures calculated on Liberatore and Levine (LL) and Herrmann et al. (H) datasets.

countermeasures applied), while the remaining $k-1$ web pages are altered to look like it. After the source web page has stopped sending packets, the direct target sampling countermeasure continues to send packets sampled from D_B until the L1 distance between the distribution of sent packet lengths and D_B is less than 0.3.

Traffic morphing: Traffic morphing operates similarly to direct target sampling except that instead of sampling from the target distribution directly, we use convex optimization methods to produce a morphing matrix that ensures we make the source distribution look like the target while simultaneously minimizing overhead. Each column in the matrix is associated with one of the packet lengths in the source distribution, and that column defines the target distribution to sample from when that source packet length is encountered. As an example, if we receive a source packet of length i, we find the associated column in the matrix and sample from its distribution to find an output length i'. One matrix is made for all ordered pairs of source and target web pages (A, B). The process of padding and splitting packets occurs exactly as in the direct target sampling case. Like the direct target sampling method, once the source web page stops sending packets, dummy packets are sampled directly from D_B until the L1 distance between the distribution of sent packet lengths and D_B is less than 0.3. In our simulations we select a target distribution using the same strategy described for DTS.

D. Overhead

Although the focus of our evaluation lies in understanding the security provided by these countermeasures, we realize that their cost in terms of bandwidth overhead and latency is an important factor that determines whether they are applicable in practice or not. To this end, we present the bandwidth overhead induced by the countermeasures for both the Liberatore and Levine and Herrmann et al. datasets in Figure 2. Overhead is calculated as (*bytes sent with countermeasure*)/(*bytes sent without countermeasure*) times 100. We note that these overhead measurements differ from those of earlier work because we do not apply countermeasures to TCP acknowledgement (52-byte) packets. For example, Liberatore and Levine [10] report a Pad to MTU overhead of 145% and Wright et al. [22] report 156%. We argue that acknowledgement packets are present regardless of the content being downloaded and there is no standard mechanism for application-layer countermeasures to apply padding to TCP acknowledgement (52-byte) packets. Nevertheless, as we will see in the following section, there is almost no correlation between overhead and the level of confidentiality provided by the countermeasure

		LL	H	P
$k=2$	Type-1	85%	71%	99%
	Type-2	97%	80%	99%
	Type-3	98%	76%	99%
$k=128$	Type-1	41%	13%	91%
	Type-2	46%	5%	90%
	Type-3	25%	3%	82%

Figure 3. The lowest average accuracy for each countermeasure class against LL, H, and P classifiers using the Hermann dataset. Random guessing yields 50% ($k=2$) or 0.7% ($k=128$) accuracy.

V. Existing Countermeasures versus Existing Classifiers

We pit the LL, H, and P classifiers from Section III against traffic simulated as per the nine countermeasures of the previous section. The testing methodology used was described in Section II. We also look at classifiability of the raw traffic, meaning when no countermeasure (beyond the normal SSH encryption) is applied.

We note that despite the appearance of the LL, H, and P classifiers in the literature, all the results we report are new. In particular, the H and P classifiers were never tested against any of these countermeasures, while the LL classifier did look at efficacy against Linear , Exponential, Mice-Elephants, and Pad to MTU but only at $k=1000$. Figure 3 contains a high-level summary for $k=2$ and $k=128$. We refer the interested reader to Appendix A for comprehensive results.

In the rest of this section we analyze the results from various points of view, including the role of the dataset, the relative performance of the classifiers, and the relative performance of the different countermeasures.

A. Comparing the Datasets

Before beginning digging into the results in earnest, we first evaluate the consistency and quality of the two available datasets. We do so to determine the extent to which results gathered using them represent the identifiability of the web pages rather than artifacts of the collection process, such as connection timeouts and general collection failures. In Figure 4, we show the silhouette of the accuracy achieved by the three classifiers across a number of universe sizes and countermeasures using each of the

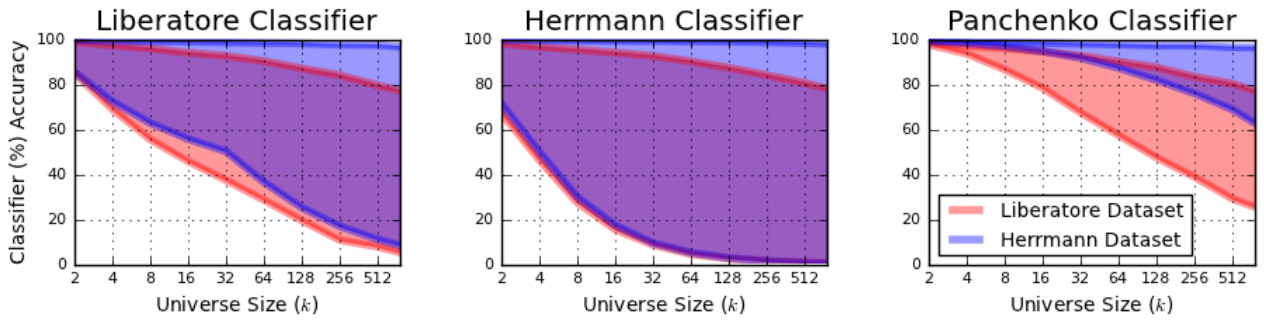

Figure 4. Comparison of accuracy silhouettes for the Liberatore and Levine and Herrmann datasets across all countermeasures for the LL, H, and P classifiers, respectively.

datasets. That is, the lower boundary of each silhouette is the best-performing countermeasure while the upper boundary represents the worst-performing (which turned out to always be no countermeasure, as one would expect).

Ideally, the classifier accuracies should be roughly similar, or at least show similar trends. Instead, what we notice is a trend toward strong drops in performance as the web page universe size increases in the Liberatore dataset, whereas in the Herrmann dataset we see a much smoother drop across multiple universe sizes and across all classifiers. This is most notable under the P classifier (far right of Figure 4).

To take a closer look at the differences between the datasets, we report some basic statistics in Figure 5. The fraction of traces that have short duration, particularly ones that are clearly degenerate (≤ 10 packets), is much higher in the Liberatore dataset. Such degenerate traces act as noise that leads to classification errors. We suspect that they arise in the dataset due to collection errors (e.g., incomplete website visits), and may imply that some previous works [10, 22] may underestimate the privacy threat posed by web page traffic analysis attacks. Despite the extra noise, the classifiers performed well, just consistently lower at high values of k as compared to the Herrmann dataset. In addition, the Herrmann dataset was collected in 2009, as opposed to the Liberatore dataset, which was collected in 2006. Despite all these differences we found the high-level trends and conclusions are the same across both datasets. For these reasons, we will focus our analysis only on the Herrmann dataset for the remainder of this paper. Appendix A contains details for classifier performance using the Liberatore dataset at $k = 128$.

B. Comparison of Classifiers

Figure 6 gives a three-by-three grid of graphs: one column per classifier and one row for countermeasure type. We start by quickly comparing the relative performance of the three classifiers, which is observable by comparing the performance across the three columns.

The first thing to notice is that at $k = 2$, essentially all of the classifiers do well against all of the countermeasures.

	LL	H
Traces with 0 packets in one direction	3.1%	0.1%
Traces with $\leq$ 5 bidirectional packets	5.2%	0.2%
Traces with $\leq$ 10 bidirectional packets	13.8%	0.4%
Traces with $\leq$ 1s duration	29.4%	6.4%
Median trace duration	2.4 sec.	3.6 sec.
Median bidirectional packet count	106	256
Median bandwidth utilization (bytes)	78,382	235,687

Figure 5. Statistics illustrating the presence of degenerate or erroneous traces in the Liberatore and Levine and Hermann datasets.

The LL and P classifiers are particularly strong, even against the DTS and TM countermeasures. The overall best classifier is clearly the P classifier. It is robust to all the countermeasures. The H classifier edges out both the P and LL classifiers for raw traffic, but is very fragile in the face of all but the simplest countermeasure (Linear padding). The LL classifier proves more robust than the H classifier, but has more severe accuracy degradation compared to P as k increases.

C. Comparison of Countermeasures

Consider the first row of Figure 6, where we see a comparison of the two Type-1 randomized padding schemes. Curiously, it is better to pick a single random padding amount to apply to each packet within a trace than to pick fresh random amounts per packet. Applying a single random amount across all packets shifts the distribution of packet lengths in a way that is unlikely to have been seen during training. On the other hand, randomizing per packet "averages out" during training and testing.

Common intuition about the Pad to MTU countermeasure is that it ought to work well against TA attacks since it ensures that no individual packet length information is leaked. However, as we seen in the second row of Figure 6, we see this intuition is wrong in large part because the *number* of packets is still leaked. The LL classifier, for example, exploits this fact, since it trains on the number of packets of each (*direction*, *length*). When the packets are padded to the MTU, there are only two numbers, namely for ($\uparrow$,1500) and ($\downarrow$,1500). The LL classifier does well

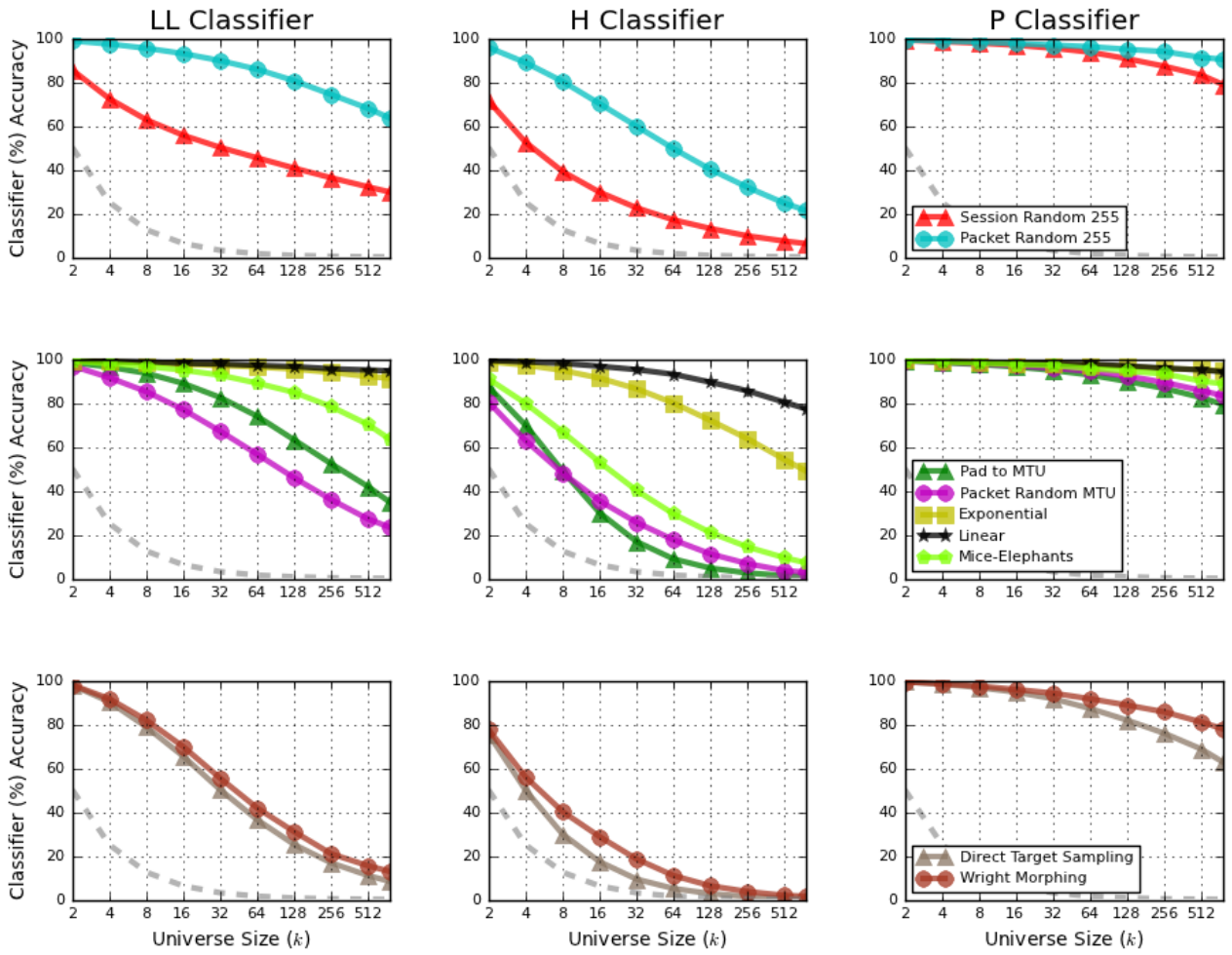

Figure 6. Average accuracy as k varies for the LL (left column), H (middle column), and P (right column) classifiers with respect to the Type-1 (top row), Type-2 (middle row), and Type-3 (bottom row) countermeasures. The dotted gray line in each graph represents a random-guess adversary.

because the number of packets transmitted is relatively consistent across traces for a particular web page. (We will investigate this more in the next section.) This also is our first evidence that exact packet-length information is *not* necessary for high-accuracy classification.

Next, we turn to the Type-3 countermeasures. Recall that these countermeasures focus on altering a specific feature of the web page traffic, namely the distribution of normalized counts, so that one web page looks like another with respect to that feature. In theory then, the distribution of packets produced by the DTS and TM countermeasures should match that of the target web page and, unlike Type-1 and Type-2 countermeasures, the number of packets from the source web page should be concealed, in part. This is not true in all cases, however, as Type-3 countermeasures do not substantially change the total bandwidth of data transmitted in each direction, nor the duration of the trace with regards to time. In fact, no countermeasure considered here substantially changes the total bandwidth. Moreover, these countermeasures do not hide "burstiness" of the data, which may be correlated to higher level structure of the underlying HTTP traffic (e.g., a downstream burst represents a web page object). Therefore, DTS and TM perform best against the H classifier, which examines the same normalized packet count distribution, while the P classifier performs particularly well with its use of packet burst information.

We compare the best countermeasure from each type in Figure 7: Session Random 255 (Type-1), Pad to MTU (Type-2), and DTS (Type-3). A few surprises arise in this comparison. First, Session Random 255 performs better or about the same as Pad to MTU. This is surprising, as Session Random 255 is a significantly lighter-weight countermeasure. It has only 7% overhead compared to Pad to MTU's 58%, and can potentially be dropped into existing deployments of SSH and TLS. That said, even at $k = 128$, it is unlikely to be satisfying to drop accuracy only down to 90%. DTS does better than the others across all values of k against the best classifier (P), but we note that simpler countermeasures actually can do a better job

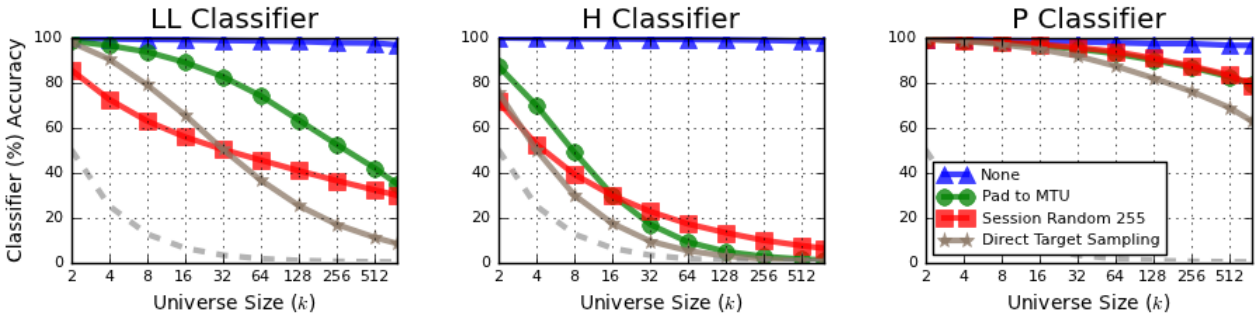

Figure 7. Comparison of the overall best performing countermeasure of each type against the LL, H, and P classifiers.

against the LL and H classifiers for lower k values.

VI. Exploring Coarse Features

Our study of existing classifiers reveals that some fine-grained features, such as individual packet lengths, are *not* required for high-accuracy classification. Indeed, the fact that the P classifier performs so well against the Pad to MTU countermeasure means that it is using features other than individual packet lengths to determine classification. This leads us to the following question: Are coarse traffic features sufficient for high-accuracy classification?

To answer this question, we explore three coarse features: total transmission time, total per-direction bandwidth, and traffic "burstiness".[6] From these features we build the time (TIME), bandwidth (BW), and the variable n-gram (VNG) classifier using naïve Bayes as our underlying machine learning algorithm. See Figure 9 for a visual summary of their performance. Later, we put these three coarse features together, and build the VNG++ naïve Bayes classifier. We will see that VNG++ is just as accurate as the (more complex) P classifier.

A. Total Time

We begin with the most coarse and intuitively least useful feature, the total timespan of a trace. How much do traces differ based on total time? The left-most plot in Figure 8 depicts the time of the first 50 traces from five websites in the Herrmann dataset. There is clear regularity within traces from each website, suggesting relatively low variance for this feature.

To test the usefulness of total time in classification, we implemented a naïve Bayes classifier that uses time as its only feature. This simple time-only classifier is quite successful for small k, as shown in Figure 9. At $k = 2$, it is able to achieve better than an 80% average accuracy against the three best countermeasures from each class as determined by performance on the P classifier. As the privacy set increases, the likelihood of multiple websites having similar timing increases, and so the accuracy of the time classifier goes down. At $k = 775$, it achieves only about 3% accuracy, although this is still substantially better than random guessing (0.1%) and may provide value as a supplementary feature in order to increase a classifier's accuracy.

Figure 9 also shows that the time classifier performs roughly the same against raw traffic (i.e., the "None" countermeasure) and with traffic countermeasures applied. As one might expect padding-based countermeasures (Type-1 and Type-2), do not directly modify the total time taken by traces. On the other hand, distribution-based countermeasures (Type-3) potentially inject dummy packets into a trace, but this is most often no more than 10-12 packets sent in quick succession. Thus, these also do not change the total time significantly.

B. Total Per-Direction Bandwidth

Next, we turn to total bandwidth consumed per direction. We see the consistency of total bandwidth in the center plot in Figure 8, which displays the upstream and downstream bandwidths of the first 50 traces of five websites from the Herrmann dataset. This plot shows a clear clustering of the websites with both very low variance within website clusters and high degrees of separability (i.e., spacing) between clusters.

Therefore, we expect bandwidth-based classification will work well as long as websites within the privacy set do not have too much overlap in terms of total per-direction bandwidth. Figure 9 shows that, indeed, the bandwidth classifier performs well. In fact, the real surprise is just *how* well the bandwidth-only classifier works for all privacy set sizes despite the coarse nature of the feature. At $k = 2$, the classifier provides close to perfect accuracy of over 99% against all countermeasures. Moreover, compare the behavior of the bandwidth-only classifier to that of the LL and H classifiers (c.f., Figure 7), which do not use bandwidth as a feature, as k increases. The bandwidth classifier is clearly more robust to changes in privacy set size. This might seem surprising, since countermeasures such as Pad to MTU and Session Random 255 should, intuitively, obfuscate bandwidth usage. They do, but these per-packet paddings only add noise to the low order bits of total

[6] We note that these features are more coarse than individual packet lengths, in the sense that knowing the latter likely implies knowing the former, but not the other way around.

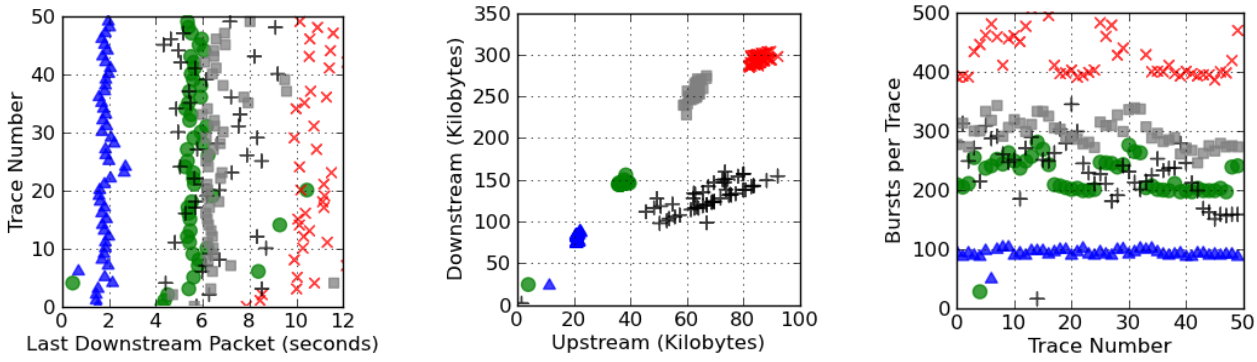

Figure 8. Each scatterplot is a visual representation of the first fifty traces, from the first five websites in the Herrmann dataset. Each symbol of the same shape and color represents the same web page. (left) Distribution of traces with respect to duration in seconds. (middle) Distribution of traces with respect to bandwidth utilization, where we distinguish the upstream and downstream directions. (right) Distribution of traces with respect to the number of bursts per trace.

bandwidth. Specifically, the change to bandwidth usage is too small relative to what would be needed to make two websites' bandwidths likely to overlap significantly. This is true for all of the padding-based countermeasures (Type-1 and Type-2). Distribution-based countermeasures DTS and TM, however, offer the best resistance to the bandwidth classifier for higher k values. Here, they outpace other countermeasures by several percentage points. This seems to be due to the insertion of dummy packets, which can add more noise than per-packet padding for total bandwidth use.

C. Variable n-gram

The time and bandwidth features already provide impressive classification ability despite their coarse nature, but do not yet give the accuracy that the Panchenko classifier achieves. We therefore look at a third feature, that of burst bandwidth. A burst is a sequence of non-acknowledgement packets sent in one direction that lie between two packets sent in the opposite direction. The bandwidth of a burst is the total size of all packets contained in the burst, in bytes. For instance, if we have a trace of the form

$$(\uparrow, 100), (\downarrow, 1500), (\downarrow, 100), (\uparrow, 200), (\uparrow, 300)$$

then there are three bursts with bandwidth 100, 1600, and 500. The intuition underlying this is that bursts correlate with higher-level properties of the traffic, such as individual web requests. This observation was first made by Panchenko et al. [14].

The right-most plot in Figure 8 shows the number of bursts for each of the first 50 traces for five websites in the Herrmann dataset. Even the number of bursts correlates strongly with the web page visited. Although this relatively limited information is capable of providing some classification ability, it turns out that burst bandwidths prove even more powerful.

Recalling that an n-gram model would coalesce n packets together into one feature, we can view bandwidth bursts as a variable n-gram model in which n varies across the trace. Then, our VNG (Variable n-Gram) classifier partitions a trace into bursts, coalesces packets into variable n-grams described by (*direction*, *size*) pairs, rounds the resulting sizes up to the nearest multiple of 600 bytes[7], and then applies a naïve Bayes classifier. Figure 9 shows how well the VNG classifier performs, already achieving better than 80% accuracy for all padding-based countermeasures, and achieving significantly higher accuracy levels for distribution-based approaches than any other classifier except the P classifier.

D. Combining Coarse Features: the VNG++ Classifier

To extract all potential identifying information from these coarse features, we combine the time, bandwidth, and variable n-gram classifiers to give a simple, yet impressively effective, classifier that dispenses with use of individual packet lengths for classification. Specifically, we use total time, bandwidth in each direction of the connection, and variable n-grams as features of a naïve Bayes classifier. A graph of the VNG++ classifier's accuracy as k varies is given in Figure 11.

In comparing VNG++ to the P classifier, we note that the latter uses a large assortment of features (as discussed in Section III), including fine-grained ones such as frequency of individual packet lengths. It also applies a more complicated machine learning algorithm in the form of an SVM. Figure 11 depicts the performance of the P and VNG++ classifiers against the best performing countermeasures of each type, as well as data with no countermeasure applied. Note that for clarity the y-axis starts at 50%, unlike other graphs. From this figure, two clear trends arise. First, VNG++'s performance against no countermeasure degrades slightly faster with k than the P classifier. This highlights that fine-grained features can provide some small benefit in classifying unprotected traces. Second,

[7]Panchenko et al. experimentally determine this rounding value as a way to maximize classification accuracy via dimensionality reduction.

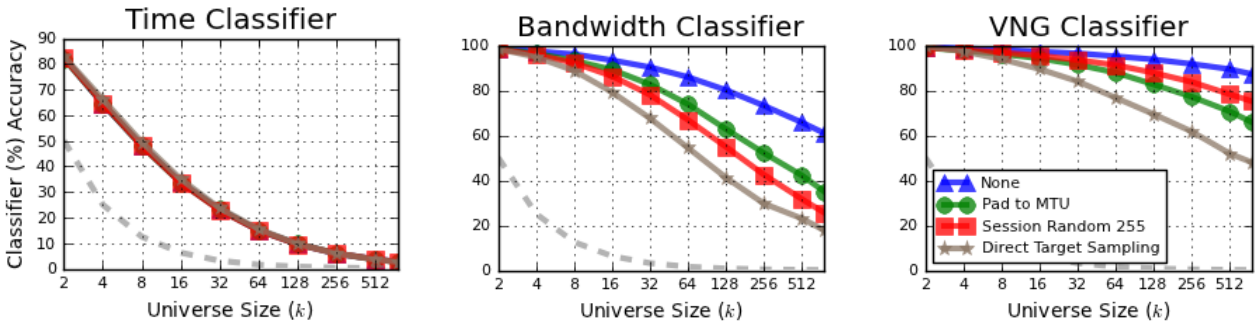

Figure 9. The average accuracy against the raw encrypted traffic (None), and the best countermeasures from each type, as established in Section V. (left) the time-only classifier. (middle) the bandwidth only classifier. (right) the VNG ("burstiness") classifier.

	Classifier		
Countermeasure	P	P-NB	VNG++
None	97.2 ± 0.2	98.2 ± 0.9	93.9 ± 0.3
Session Random 255	90.6 ± 0.3	59.1 ± 2.3	91.6 ± 0.3
Packet Random 255	94.9 ± 0.3	93.7 ± 1.6	93.5 ± 0.3
Linear	96.8 ± 0.2	96.9 ± 1.1	94.3 ± 0.3
Exponential	96.6 ± 0.3	97.4 ± 0.9	94.8 ± 0.3
Mice-Elephants	94.5 ± 0.6	95.1 ± 0.8	91.7 ± 0.4
Pad to MTU	89.8 ± 0.4	91.7 ± 1.5	88.2 ± 0.4
Packet Random MTU	92.1 ± 0.3	84.1 ± 1.7	87.6 ± 0.3
Direct Target Sampling	81.8 ± 0.5	76.8 ± 2.5	80.2 ± 0.5
Traffic Morphing	88.7 ± 0.4	82.6 ± 5.6	85.6 ± 0.7

Figure 10. Accuracies (%) of P, P-NB, and VNG++ classifiers at $k = 128$.

when we consider countermeasures, VNG++ matches P in performance. This holds despite the use of fewer features and the simpler machine learning algorithm used by the former. As it turns out, in the face of countermeasures, the coarse features are the damaging ones and fine-grained features are not particularly helpful.

A final question lingers: does using an SVM provide any advantage over a naïve Bayes classifier? We implemented a naïve Bayes version of the P classifier. This P-NB classifier uses a 1-1 mapping of the features used by P to analogues suitable for use with a naïve Bayes classifier. A comparison of performance at $k = 128$ for P, P-NB, and VNG++ are given in Figure 10. Overall, we see that the results are consistent across all three classifiers. A single exception is the accuracy of P-NB for Session Random 255, which results in a surprisingly low classifier accuracy.

E. Discussion

The nine countermeasures considered so far attempt to obfuscate leaked features of the traffic via padding and insertion of dummy packets. As we've seen, however, these fail to protect significant amounts of identifying information from being leaked from coarse features of the encrypted traffic, rather than the fine-grained, per-packet features typically targeted by TA countermeasures. Unfortunately, these kinds of features are precisely the ones that are most difficult to efficiently hide.

Obfuscating total bandwidth is an obvious case in point. To prevent this feature from leaking information, a countermeasure must ensure a similar amount of bandwidth use across all websites in any given privacy set. Since we do not want to forego functionality (e.g., shutting down connections prematurely), this translates into a countermeasure that inserts dummy traffic until we achieve a total bandwidth close to that of the maximum bandwidth usage of any website in the privacy set.

Hiding burst bandwidth is also problematic. As seen in Figure 8, different websites can have quite different patterns of bursts. A countermeasure must smooth out these patterns. In theory, a traffic morphing-like countermeasure can attempt to imitate a target trace's burst patterns, however this will require buffering packets for potentially long periods of time. Thus, countermeasures for preventing website traffic analysis must incur both bandwidth and latency overheads.

In all, our analyses leaves little wiggle room for countermeasures to operate within. Providing robust protection against fingerprinting attacks for arbitrary websites in a closed-world setting, such as the one presented here, is going to have to be inefficient.

VII. BuFLO: Buffered Fixed-Length Obfuscator

Our analysis thus far leaves us with the conclusion that, despite the long line of work on TA attacks and countermeasures, we have no packet-oriented countermeasure that prevents website fingerprinting attacks. We therefore want to know whether any measure can work, even prohibitively inefficient ones.

Following the analysis of the last section, we know that any effective countermeasure must hide the total time, bandwidth use, and burst patterns. To that end, we consider a new countermeasure **Bu***ffered* **F***ixed-***L***ength* **O***bfuscator*, or BuFLO. It is a realization of the "fool-proof" folklore countermeasure that, intuitively, should defeat any TA classifier by removing all side-channel information. BuFLO operate by sending fixed-length packets at a fixed

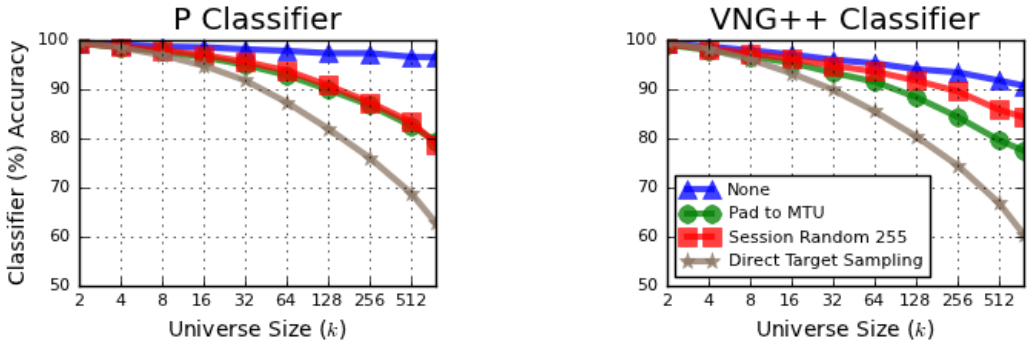

Figure 11. Accuracy of P (left) and VNG++ (right) classifiers against the best-performing countermeasures from Section III.

interval for at least a fixed amount of time. If a flow goes longer than the fixed time out, BuFLO lets it conclude while still using fixed-length packets at a fixed interval. In an ideal implementation, BuFLO will not leak packet lengths or packet timings, and so BuFLO should do a good job at closing side-channels that enable TA classifiers. This type of countermeasure has been investigated in the context of other TA attacks, such as those on anonymity networks [17, 23]

Our simulation-based analysis of BuFLO provides some positive evidence for packet-level countermeasures, but in fact our results here are mostly negative, thereby reinforcing the lessons learned in prior sections. BuFLO is, as one might expect, incredibly inefficient. Moreover, we will see that even mild attempts to claw back some efficiency can fail: setting the minimum session too aggressively short opens up vulnerability to our coarse-feature classifiers.

A. BuFLO *Description*

A BuFLO implementation is governed by three integer parameters d, ρ and τ:

- Parameter d determines the size of our fixed-length packets.
- Parameter ρ determines the rate or frequency (in milliseconds) at which we send packets.
- Parameter τ determines the minimum amount of time (in milliseconds) for which we must send packets.

A BuFLO implementation at the start of communications will send a packet of length d every ρ milliseconds until communications cease and at least τ milliseconds of time have elapsed. Specifically, data is buffered into discrete chunks, and these chunks are sent as quickly as possible via the steady flow of the fixed-length packets. When no data is in the buffer, dummy data is sent instead. This assumes that the application-layer signals the start and end of communication. Alternatively, we could have chosen τ as an *upper bound* on the duration of our communications session and forcibly close the connection even if communications are still in progress. This would disable any websites that take longer to load, making it unlikely to be a pragmatic choice.

B. *Experiments*

In this section, we examine BuFLO for various parameters using the Hermann dataset and provide detailed results in Figure 12. Since we are using a simulation-based experiment, these results reflect an ideal implementation that assumes the feasibility of implementing fixed packet timing intervals. This is at the very least difficult in practice [7] and clearly impossible for some values of ρ. Simulation also ignores the complexities of cross-layer communication in the network stack, and the ability for the BuFLO implementation to recognize the beginning and end of a data flow. If BuFLO cannot work in this setting, then it is unlikely to work elsewhere, aiding us in our exploration of the goal of understanding the limits of packet-level countermeasures.

We evaluated BuFLO empirically with parameters in the ranges of $\tau \in \{0, 10000\}$, $\rho \in \{20, 40\}$ and $d \in \{1000, 1500\}$. The least bandwidth-intensive configuration, at $\tau = 0$, $\rho = 40$ and $d = 1000$ would require at least 0.2 Mbps of continuous synchronous client-server bandwidth to operate[8]. Surprisingly, with this BuFLO configuration and a privacy set size of $k = 128$, the P classifier still identifies sites with an average accuracy of 27.3%. This is compared to 97.5% average accuracy with no countermeasure applied. At the other extreme of our experiments with $\tau = 10000$, $\rho = 20$ and $d = 1500$ it would require at least 0.6 Mbps of synchronous client-server bandwidth to operate. Here, the P classifier can *still* identify sites with a privacy set size of $k = 128$ with an average 5.1% accuracy.

C. *Observations about* BuFLO

BuFLO cannot leak packet lengths, nor can it leak packet timings. Yet, our experiments indicate that an aggressively configured BuFLO implementation can still leak information about transmitted contents. This is possible because BuFLO can leak *total bytes transmitted* and the *time required to transmit a trace* in two circumstances:

[8]Calculated by $\left(\frac{1000}{\rho}\right) \cdot \left(\frac{8d}{10^6}\right)$.

	Overhead		Classifier Accuracy (%)				
Parameters	Bandwidth (%)	Latency (s)	LL	H	P	VNG++	P-NB
BuFLO (τ=0, ρ=40, d=1000)	93.5	6.0	18.4 ± 2.9	0.8 ± 0.0	27.3 ± 1.8	22.0 ± 2.1	21.4 ± 1.0
BuFLO (τ=0, ρ=40, d=1500)	120.0	3.6	16.2 ± 1.6	0.8 ± 0.0	23.3 ± 3.3	18.3 ± 1.0	18.8 ± 1.4
BuFLO (τ=0, ρ=20, d=1000)	140.5	2.4	16.3 ± 1.2	0.8 ± 0.0	20.9 ± 1.6	15.6 ± 1.2	17.9 ± 1.7
BuFLO (τ=0, ρ=20, d=1500)	201.3	1.2	13.0 ± 0.8	0.8 ± 0.0	24.1 ± 1.8	18.4 ± 0.9	18.7 ± 1.0
BuFLO (τ=10000, ρ=40, d=1000)	129.2	6.0	12.7 ± 0.9	0.8 ± 0.0	14.1 ± 0.9	12.5 ± 0.8	13.2 ± 0.7
BuFLO (τ=10000, ρ=40, d=1500)	197.5	3.6	8.9 ± 1.0	0.8 ± 0.0	9.4 ± 1.3	8.2 ± 0.8	9.3 ± 1.3
BuFLO (τ=10000, ρ=20, d=1000)	364.5	2.4	5.4 ± 0.8	0.8 ± 0.0	7.3 ± 1.0	5.9 ± 1.0	6.8 ± 0.9
BuFLO (τ=10000, ρ=20, d=1500)	418.8	1.2	4.4 ± 0.2	0.8 ± 0.0	5.1 ± 0.7	4.1 ± 0.8	5.3 ± 0.5

Figure 12. Overhead and accuracy results for the BuFLO countermeasure at $k = 128$.

- The data source continued to produce data beyond the threshold τ.
- The data source ceases to produce data by the threshold τ, but there is still data in the buffer at time τ.

The first situation can occur if our our threshold τ is not sufficiently large to accommodate for all web pages that we may visit. The latter situation occurs when values ρ and d are not sufficiently configured to handle our application's data throughput, such that we transmit all data by time τ.

What is more, in some circumstances an inappropriately configured BuFLO implementation can can actually benefit an adversary. At $k = 128$ with $\tau = 0$, $\rho = 40$ and $d = 1000$ (see Figure 12) the BuFLO countermeasure can increase the accuracy of the Time classifier from 9.9% to 27.3%! In retrospect this is not surprising. If we throttle the bandwidth of the web page transfer, we will amplify its timing fingerprint.

These results reinforce the observations of prior sections. Namely, that TA countermeasures must, in the context of website identification, prevent coarse features from being leaked. As soon as these features leak, adversaries will gain some advantage in picking out web pages.

VIII. Related Work

Traffic analysis of encrypted data has been studied extensively. Our focus is on identification (or fingerprinting) of web pages within encrypted tunnels, and we do not discuss other contexts, such as analysis of encrypted VoIP traffic [18–21] or revelation of web page contents [2, 3]. Even so, there is a significant amount of literature focused on website identification, including a wide diversity of evaluation methodologies, attacks, and countermeasures.

To the best of our knowledge, the first academic discussion of TA attacks in this context was by Wagner and Schneier [16]. They relayed an observation of Yee that SSL might leak the URL of an HTTP get request because ciphertexts leak plaintext length. Wagner and Schneier suggested that per-ciphertext random padding should be included for all cipher modes of SSL.

Cheng and Avnur [4] provided some of the first experimental evidence of web page fingerprinting attacks by analyzing pages hosted within one of three websites. Their attack assumes perfect knowledge of HTML and web page object sizes, which is not always precisely inferred from ciphertexts. They also suggested countermeasures including padding of HTML documents, Pad to MTU, and introduction of spurious HTTP requests. They evaluated the first two in the context of their attack, and claim some efficacy for the considered websites.

Sun et al. [15] investigated a similar setting, in which the adversary can precisely uncover the size of individual HTTP objects in a non-pipelined, encrypted HTTP connection. They provided a thorough evaluation utilizing a corpus of 100,000 websites. They described a classifier based on the Jaccard coefficient similarity metric and a simple thresholding scheme. It was successful against raw traffic, and while we did not implement their attack, several of the classifiers we consider are likely to outperform it. They also explored numerous countermeasures, including per-packet padding, byte-range requests, client-based prefetching, server-based pushing of content, content negotiation, web ad blockers, pipelining, and using multiple browsers in parallel. Their evaluation of the countermeasures only considered their attack, and the results indicate that the countermeasures provide improved TA resistance to it.

Hintz [9] discussed a simple attack for identifying which of five popular web pages was visited over a single-hop proxy service called SafeWeb. The proposed attack does not require exact knowledge of web request sizes, but there is little evaluation and it remains unclear how the attack would fair with larger privacy sets.

Bissias et al. [1] demonstrated a weaker adversary than that of Sun et al. [15], which could observe an SSH tunnel and view only the length, direction, and timing of each ciphertext transmitted, rather than web page objects. They used cross-correlation to determine webpage similarity, which is a metric commonly used for evaluating the similarity of two time series. They achieved worse performance than the classifiers we consider, and they did not explore any countermeasures.

Liberatore and Levine [10] showed that it is possible to infer the contents of an HTTP transaction encapsulated in an SSH connection by observing only encrypted packet lengths and the directions of unordered packets. We provided a detailed description of their classifier in section III, and we use their publicly-available dataset in our analy-

ses. They quantify the ability of several countermeasures, including Linear, Exponential, Mice-Elephants, and Pad to MTU padding schemes, to protect against their attack, but only report on a privacy set size of $k = 1000$. These results cast a positive light on some padding approaches, like Pad to MTU, which reduces the accuracy of their proposed classifier from 68% to around 7%. We did not consider $k = 1000$ in order to ensure consistency with other datasets in our evaluation, but projecting out from the observed trends we expect that, for example, the VNG++ classifier will do significantly better than 7% at $k = 1000$ (c.f., Figure 9).

Herrmann et al. [8] collected encrypted traces from four different types of single-hop encryption technologies, and two multi-hop anonymity networks. We use a portion of their dataset for our analyses. They were the first to suggest the use of a multinomial naïve Bayes classifier for traffic classification that examines normalized packet counts. A discussion of their classifier was given in Section III. Their evaluation of countermeasures was restricted to application-layer countermeasures.

Panchenko et al. [14] presented a support vector machine classifier as an improvement upon the work of Herrmann et al. [8]. We discussed details of the Panchenko classifier in Section III. They apply it to Tor [6] traffic they generated in both a closed-word and open-world setting, showing good accuracy, though worse than those that the classifiers we consider achieve. Tor's encryption mechanisms already obfuscate some information about plaintext lengths, making it harder, in general, to classify. They did not report on their classifier's efficacy against the countermeasures we consider.

In an effort to minimize overhead incurred by previously suggested padding schemes, Wright et al. proposed the notion of traffic morphing [22]. Their countermeasures can minimize overhead while still making one web page "look" like another with respect to specific features. As Wright et al. suggested [22, Section 4.1], and Lu et al. later confirmed with their experimental evaluation [11], traffic morphing is only effective when the attacker restricts attention to the same feature(s) targeted by the morphing routine. Our results likewise indicate that attackers can still succeed even when traffic morphing is used to ensure the normalized distribution of packet sizes is similar to some target web page.

Both Panchenko et al. [14] and Luo et al. [12] suggest concrete application-layer countermeasures. Panchenko et al. propose the Camouflage countermeasure, which makes spurious HTTP requests in parallel with legitimate ones, and show that it renders their classifier significantly less effective. The Luo et al. system is called HTTPOS and uses a number of client-side mechanisms that take advantage of existing HTTP functionality to add noise to encrypted web traffic. For example, HTTPOS randomizes HTTP GET requests by adding superfluous data to headers and utilizing HTTP byte range functionality to request subsets of data non-sequentially. They evaluate their countermeasure in the presence of four existing classifiers [1, 3, 10, 15] and show that HTTPOS is effective against all of them. We do not consider these kinds of application-layer mechanisms, and indeed our results suggest that such countermeasures may be better positioned to defend against web page identification attacks.

IX. Concluding Discussion

Although a significant amount of previous work has investigated the topic of TA countermeasures, and specifically the case of preventing website identification attacks, the results were largely incomparable due to differing experimental methodology and datasets. Our work synthesizes and expands upon previous ones, and it provides sharper answers to some of the area's central questions:

Do TA countermeasures prevent website fingerprinting? None of the nine countermeasures considered here prevents the kind of website fingerprinting attack addressed by prior works [8, 10, 14, 22]. From a security perspective this setting is conservative, and makes several simplifying assumptions. (The attacker knows the privacy set; it trains and tests on traffic generated in the same way; the collected traffic does not account for (potentially) confounding effects, such as browser caching, interleaved web requests, etc.) Nevertheless, our negative results suggest that one should not rely *solely* upon these countermeasures to prevent website fingerprinting attacks.

Do TA attacks require individual packet lengths? No. We implemented three coarse-feature classifiers: one using only total time as a feature, one using only total per-direction bandwidth, and one tracking only data bursts (the VNG classifier). These did not make direct use of individual packet lengths or packet counts as features, yet attained high accuracy against the countermeasures. This highlights the point that masking fine-grained information is insufficient, unless such masking also hides telling large-scale features (e.g., individual object requests, size of web objects, etc.).

Does classification engine matter? Our experiments suggest it is the *features*, and not the underlying classification engine, that matters. We implemented a naïve Bayes-based classifier that used the same features as those exploited by the SVM-based Panchenko et al. classifier, and our experiments show that these two perform almost identically.

Does the privacy-set size (k) matter? For the considered setting, it seems not to matter much. When no countermeasure is used, attacks can achieve roughly the same accuracy for $k = 2$ through $k = 775$. When countermeasures are applied, the best classifier's accuracy does drop slowly as k increases. This suggests that the countermeasures do obfuscate some features that can improve accuracy. That said, at the largest k, the best classifiers offer better than 60% accuracy against all of the countermeasures.

Our work paints a pretty negative picture of the usefulness of efficient, low-level TA countermeasures against website-fingerprinting attacks. But pessimism need not prevail. Future work could investigate more detailed modelings of real-world traffic, and investigate applications of TA countermeasures beyond website fingerprinting. This may uncover settings in which some countermeasures are more successful than they were in our experiments. In addition, the coarse features (e.g. bandwidth) that appear near impossible to obfuscate efficiently at the level of individual packets might be better handled at the application layer. Previous works [8, 12] suggest application-layer countermeasures with promising initial evaluations. Future work could provide more extensive investigation of such countermeasures.

REFERENCES

[1] George Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Proceedings of the Privacy Enhancing Technologies Workshop*, pages 1–11, May 2005.

[2] Peter Chapman and David Evans. Automated Black-Box Detection of Side-Channel Vulnerabilities in Web Applications. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 263–274, November 2011.

[3] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 191–206, May 2010.

[4] Heyning Cheng and Ron Avnur. Traffic Analysis of SSL Encrypted Web Browsing, December 1998. Available at: http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps.

[5] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. Updated by RFCs 5746, 5878, 6176. Available at: http://www.ietf.org/rfc/rfc5246.txt.

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

[7] Xinwen Fu, Bryan Graham, Riccardo Bettati, Wei Zhao, and Dong Xuan. Analytical and Empirical Analysis of Countermeasures to Traffic Analysis Attacks. In *Proceedings of the International Conference on Parallel Processing*, pages 483–492, October 2003.

[8] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-Bayes Classifier. In *Proceedings of the ACM Workshop on Cloud Computing Security*, pages 31–42, November 2009.

[9] Andrew Hintz. Fingerprinting Websites Using Traffic Analysis. In *Proceedings of the Privacy Enhancing Technologies Workshop*, pages 171–178, April 2002.

[10] Marc Liberatore and Brian Neil Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 255–263, November 2006.

[11] Liming Lu, Ee-Chien Chang, and Mun Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Proceedings of the European Symposium on Research in Computer Security*, volume 6345 of *Lecture Notes in Computer Science*, pages 199–214, September 2010.

[12] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Proceedings of the Network and Distributed Security Symposium*, February 2011.

[13] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[14] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website Fingerprinting in Onion Routing-based Anonymization Networks. In *Proceedings of the Workshop on Privacy in the Electronic Society*, pages 103–114, October 2011.

[15] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 19–30, May 2002.

[16] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 Protocol. In *Proceedings of the USENIX Workshop on Electronic Commerce*, pages 29–40, November 1996.

[17] Wei Wang, Mehul Motani, and Vikram Srinivasan. Dependent Link Padding Algorithms for Low Latency Anonymity Systems. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 323–332, November 2008.

[18] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on fon-iks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–18, May 2011.

[19] Charles V Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M Masson. Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 35–49, May 2008.

[20] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Uncovering Spoken Phrases in Encrypted Voice over IP Conversations. *ACM Transactions on Information and Systems Security*, 13:1–30, December 2010.

[21] Charles V. Wright, Lucas Ballard, Fabian Monrose, and Gerald M. Masson. Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In *Proceedings of the USENIX Security Symposium*, pages 1–12, August 2007.

[22] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Proceedings of the Network and Distributed Security Symposium*, pages 237–250, February 2009.

[23] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. On Flow Correlation Attacks and Countermeasures in Mix Networks. In *Proceedings of the Privacy Enhancing Technologies Workshop*, volume 3424 of *Lecture Notes in Computer Science*, pages 207–225, May 2004.

APPENDIX

	Classifier						
Countermeasure	LL	H	P	BW	TIME	VNG	VNG++
None	98.1 ± 0.1	98.9 ± 0.1	97.2 ± 0.2	80.1 ± 0.6	9.7 ± 0.1	93.7 ± 0.2	93.9 ± 0.3
Session Random 255	40.7 ± 0.3	13.1 ± 0.2	90.6 ± 0.3	54.9 ± 0.4	9.5 ± 0.1	87.8 ± 0.3	91.6 ± 0.3
Packet Random 255	80.6 ± 0.4	40.1 ± 0.3	94.9 ± 0.3	77.4 ± 0.6	9.4 ± 0.1	91.6 ± 0.2	93.5 ± 0.3
Pad to MTU	63.1 ± 0.5	4.7 ± 0.1	89.8 ± 0.4	62.7 ± 0.6	9.6 ± 0.2	82.6 ± 0.4	88.2 ± 0.4
Packet Random MTU	45.8 ± 0.4	11.2 ± 0.2	92.1 ± 0.3	64.6 ± 0.5	9.5 ± 0.1	77.8 ± 0.3	87.6 ± 0.3
Exponential	95.4 ± 0.2	72.0 ± 0.4	96.6 ± 0.3	77.1 ± 0.6	9.6 ± 0.1	95.1 ± 0.2	94.8 ± 0.3
Linear	96.6 ± 0.2	89.4 ± 0.2	96.8 ± 0.2	79.5 ± 0.6	9.6 ± 0.2	93.5 ± 0.2	94.3 ± 0.3
Mice-Elephants	84.8 ± 0.4	20.9 ± 0.3	94.5 ± 0.3	72.3 ± 0.6	9.6 ± 0.1	89.4 ± 0.3	91.7 ± 0.4
Direct Target Sampling	25.1 ± 0.6	2.7 ± 0.1	81.8 ± 0.5	41.2 ± 0.9	9.7 ± 0.2	69.4 ± 0.6	80.2 ± 0.5
Traffic Morphing	31.0 ± 0.7	6.3 ± 0.3	88.7 ± 0.4	43.0 ± 0.9	9.8 ± 0.2	81.0 ± 0.5	86.0 ± 0.4

Figure 13. Classifier performance for $k = 128$, using the Herrmann dataset.

	Classifier						
Countermeasure	LL	H	P	BW	TIME	VNG	VNG++
None	87.1 ± 0.6	87.4 ± 0.3	87.5 ± 0.6	55.7 ± 0.7	11.6 ± 0.7	72.0 ± 1.1	76.3 ± 1.0
Session Random 255	25.3 ± 0.4	9.5 ± 0.1	66.1 ± 0.6	38.6 ± 0.5	12.1 ± 0.6	60.5 ± 1.1	68.7 ± 1.2
Packet Random 255	43.6 ± 0.7	13.1 ± 0.3	74.0 ± 0.7	51.5 ± 0.7	11.8 ± 0.6	65.6 ± 1.3	71.8 ± 1.0
Pad to MTU	41.3 ± 0.6	5.0 ± 0.1	69.2 ± 0.7	41.8 ± 0.6	11.6 ± 0.7	56.8 ± 1.2	65.7 ± 1.1
Packet Random MTU	21.8 ± 0.5	7.5 ± 0.1	69.1 ± 0.7	40.2 ± 0.6	11.4 ± 0.8	47.1 ± 1.0	60.3 ± 1.0
Exponential	72.9 ± 0.6	61.2 ± 0.4	82.1 ± 0.8	54.8 ± 0.8	10.5 ± 0.7	74.1 ± 0.8	78.0 ± 0.9
Linear	79.2 ± 0.7	73.9 ± 0.3	84.2 ± 0.6	54.4 ± 0.9	12.0 ± 0.7	70.3 ± 0.9	74.3 ± 1.4
Mice-Elephants	55.9 ± 0.9	25.6 ± 0.3	75.6 ± 0.7	49.3 ± 0.6	11.7 ± 0.5	65.9 ± 1.1	71.2 ± 1.0
Direct Target Sampling	19.4 ± 1.0	2.5 ± 0.3	47.4 ± 1.4	26.8 ± 1.1	11.1 ± 0.7	35.7 ± 3.0	49.7 ± 1.9
Traffic Morphing	20.1 ± 1.2	4.1 ± 0.5	55.3 ± 1.3	25.6 ± 1.1	12.3 ± 0.7	45.4 ± 2.1	56.7 ± 2.0

Figure 14. Classifier performance for $k = 128$, using the Liberatore dataset.

	Privacy Set Size						
Classifier	$k = 16$	$k = 32$	$k = 64$	$k = 128$	$k = 256$	$k = 512$	$k = 775$
None	96.9 ± 0.1	95.9 ± 0.2	95.1 ± 0.2	93.9 ± 0.3	93.3 ± 0.4	91.6 ± 0.6	90.6 ± 0.9
Session Random 255	96.0 ± 0.1	94.7 ± 0.2	93.4 ± 0.2	91.6 ± 0.3	89.4 ± 0.4	85.6 ± 0.6	84.1 ± 0.5
Packet Random 255	96.6 ± 0.1	95.7 ± 0.2	94.5 ± 0.2	93.5 ± 0.3	92.2 ± 0.5	89.8 ± 0.7	88.5 ± 0.8
Pad to MTU	95.2 ± 0.1	93.2 ± 0.2	91.4 ± 0.2	88.2 ± 0.4	84.2 ± 0.5	79.5 ± 0.8	77.3 ± 0.7
Packet Random MTU	95.0 ± 0.1	93.1 ± 0.2	90.8 ± 0.2	87.6 ± 0.3	83.3 ± 0.5	78.6 ± 0.6	74.8 ± 0.8
Exponential	97.1 ± 0.1	96.5 ± 0.1	95.6 ± 0.2	94.8 ± 0.3	93.7 ± 0.4	92.5 ± 0.6	90.8 ± 1.2
Linear	96.9 ± 0.1	96.0 ± 0.2	95.1 ± 0.2	94.3 ± 0.3	92.8 ± 0.5	91.8 ± 0.6	89.5 ± 1.1
Mice-Elephants	96.2 ± 0.1	95.1 ± 0.2	93.6 ± 0.2	91.7 ± 0.4	90.0 ± 0.5	86.5 ± 0.8	84.0 ± 0.8
Direct Target Sampling	93.1 ± 0.2	89.8 ± 0.2	85.3 ± 0.3	80.2 ± 0.5	74.3 ± 0.8	65.8 ± 1.8	61.0 ± 4.1
Traffic Morphing	94.8 ± 0.2	92.8 ± 0.2	90.2 ± 0.3	85.6 ± 0.7	83.3 ± 0.7	77.7 ± 2.3	75.1 ± 3.1

Figure 15. Performance of the VNG++ classifier, for varying values of k, using the Herrmann dataset.

	Privacy Set Size						
Classifier	$k = 16$	$k = 32$	$k = 64$	$k = 128$	$k = 256$	$k = 512$	$k = 775$
None	98.4 ± 0.1	98.0 ± 0.1	97.7 ± 0.2	97.2 ± 0.2	97.2 ± 0.3	96.4 ± 0.5	96.4 ± 0.4
Session Random 255	96.8 ± 0.1	95.5 ± 0.2	93.6 ± 0.2	90.6 ± 0.3	87.2 ± 0.5	83.2 ± 0.5	78.7 ± 0.9
Packet Random 255	97.6 ± 0.1	96.9 ± 0.2	96.2 ± 0.2	94.9 ± 0.3	93.9 ± 0.3	91.2 ± 0.8	90.3 ± 0.7
Pad to MTU	96.4 ± 0.1	94.8 ± 0.2	92.7 ± 0.3	89.8 ± 0.4	86.6 ± 0.5	82.4 ± 0.8	79.2 ± 0.9
Packet Random MTU	97.0 ± 0.1	95.8 ± 0.2	94.4 ± 0.2	92.1 ± 0.3	89.3 ± 0.5	85.6 ± 0.7	83.2 ± 0.6
Exponential	98.1 ± 0.1	97.9 ± 0.1	97.2 ± 0.2	96.6 ± 0.3	95.6 ± 0.4	95.2 ± 0.3	94.6 ± 0.4
Linear	98.1 ± 0.1	97.7 ± 0.1	97.6 ± 0.2	96.8 ± 0.2	95.8 ± 0.5	95.0 ± 0.6	94.2 ± 0.7
Mice-Elephants	97.5 ± 0.1	97.0 ± 0.1	95.6 ± 0.2	94.5 ± 0.3	93.2 ± 0.4	89.9 ± 0.9	88.7 ± 1.0
Direct Target Sampling	94.7 ± 0.2	91.7 ± 0.2	87.2 ± 0.3	81.8 ± 0.5	75.9 ± 0.7	68.7 ± 0.9	62.5 ± 1.3
Traffic Morphing	95.9 ± 0.1	94.2 ± 0.2	91.6 ± 0.3	88.7 ± 0.4	85.6 ± 0.6	81.0 ± 0.9	77.8 ± 1.3

Figure 16. Performance of the Panchenko classifier, for varying values of k, using the Herrmann dataset.