ELE 298 Written Report

Website Traffic Analysis:

Bursting into Burst Length Countermeasures

Michael Alan Chang and Michael D. Freyberger

Advisor: Professor Prateek Mittal

Submitted: May 13, 2014

I hereby declare that this independent work report represents my own work in accordance with University regulations.

/s Michael Freyberger

Michael Freyberger

I hereby declare that this independent work report represents my own work in accordance with University regulations.

/s Michael Alan Chang

Michael Alan Chang

Website Traffic Analysis:

Bursting into Burst Length Countermeasures

Michael Alan Chang and Michael D. Freyberger

ABSTRACT

We consider the setting where clients communicate with servers in the Tor environment for the purpose of concealing the identity of the website they are accessing. While Tor is more effective at preventing traffic analysis attacks, the websites accessed on Tor are still susceptible to attacks by top performing website classifiers such as the ones developed by Panchenko et al., and Dyer et al. These classifiers exploit coarse features in the communication trace, most importantly the ordering of the packets. While countermeasures at the packet level cannot sufficiently defend against these attacks, we demonstrate the effectiveness of applying countermeasures that mask the bursts in the data by sending strategically placed dummy packets. In this paper, we propose and experiment with eleven novel burst-level countermeasures that provided varying tradeoffs between accuracy and byte level overhead. The countermeasures that most significantly reduce classifier accuracy require slightly more than 100% byte level overhead. Our findings imply that, with strategic placement of dummy packets, traffic analysis countermeasures can provide more desirable web security and anonymity.

Acknowledgments:

We acknowledge Professor Prateek Mittal for being a great advisor.

We acknowledge Kevin Dyer for responding promptly responding to our emails, and providing well documented code on Github.

We acknowledge Xiang Cai for his code base and helping us understand Levenshtein-distance, as well as SVM’s.

We acknowledge Amy Cutchin for helping proofread this report.

Table of Contents:

I. Introduction

Maintaining anonymity between parties who communicate over the Internet is crucial for users living under oppressive regimes. Without anonymizing the sites one visits, users will never be able to communicate freely or evade oppressive censorship. Indeed, the mere usage of anonymization networks increases scrutiny on the user, thus secure and reliable anonymization networks must be in place. Modern day cryptographic protocols like SSL successfully encrypt the contents of the individual packets, but this alone is insufficient for protecting the identity of the site being visited. Instead, users must rely on secure networks such as Tor, an anonymity network which utilizes a multi-hop proxy channel to facilitate Internet traffic. While Tor has arguably been the most effective defense against web classifiers, multiple studies have proven that Tor offers ineffectual protection against attacks that leverage the “coarse” features of HTTP traffic [1, 2]. It is our goal to offer defenses against the top performing classification methods – primarily the classifiers proposed by Dyer et al., Panchenko et al., and Cai et al. - that are capable of penetrating the Tor anonymity network [1, 2, 3].

Existing countermeasures rely on packet level padding, and are designed to mask the packet lengths, which are extremely informative about which website is being visited. Tor itself employs a form of this strategy, padding the data cell within a packet to 512 bytes [4]. However, more recent classifiers no longer rely entirely on packet sizes for classification, but on the burst sizes, or the total number of uninterrupted packets in one direction. Panchenko et al., combines the concepts of burst sizes with other coarse features such as total number of bytes and the overall time of transmission [2]. In our tests, Panchenko’s classifier was able to classify websites, collected within the Tor environment, with classification rates as high as 34% and with a universe size of 512 sites (see Table 1). Padding all bursts to the same size (in the same manner as packets), effectively reduces the accuracy of the classifier; however, the overhead required is substantial; our experiments reveal that the accuracy is reduced to 2% while the overhead exceeds 100% (see Table 1). See Section VII: Burst Level Countermeasures for more details.

Due to the nature of bursts, we design countermeasures that effectively split and mask bursts both deterministically and probabilistically. Our results strongly indicate that careful application of burst-level padding can greatly reduce the accuracy of the most accurate classifiers in the literature to 6.4% while achieving overheads as small as 11.1% (see Table 1). We implement novel features such as strategically placed dummy packets and random burst level padding. Furthermore, we look at a novel practical approach towards handling the issue of website fingerprinting that leverages user habit in the camouflaging of websites. We hope that this paper will be able to facilitate more effective burst-level padding in Tor defenses and help direct future analysis of effective countermeasures.

Countermeasure	**Characteristics of countermeasure**	**Overhead**	**Panchenko**	**VNG++**
Pad To MTU	Packet Padding	41.3%	23.5%	13.4%
Session Random 255	Packet Padding, Packet Length Randomness	6.5%	10.6%	16.6%
Burst Fixed Max Length	Dummy Packets Client to Server	2.9%	30.0%	18.3%
Burst Fixed Max Length with Packet Padding	Dummy Packets Client to Server, Packet Padding, Packet Length Randomness	9.5%	12.1%	14.7%
Burst Session Random Fixed Max Length	Dummy Packets Client to Server, Packet Padding, Packet Length Randomness, Burst Length Randomness	11.1%	9.1%	6.4%
Fixed Burst Length Packet Level	Dummy Packets Both Directions, Packet Padding, Packet Length Randomness, Burst Length Randomness	112.4%	3.7%	2.0%
Fixed Burst Length Byte Level	Dummy Packets Both Directions, Packet Padding, Packet Length Randomness, Burst Length Randomness	137.5%	7.1%	1.3%

Table 1: Summary of the important countermeasures explored in our work against the most effective classifiers. The two classifier columns on the right represent the accuracy of the classifier with a universe size of 512 sites. The countermeasures are applied to traces from a data set generated in the Tor environment.

II. Terminology

The terms that are critical to the comprehension of this paper are introduced in this section.

Packet- Individual units of data sent from the client to the server, or vice versa. Packets are of variable size, but will never exceed the Maximum Transmission Unit (MTU), which depends on settings established by the communication interface or communication standards. In our tests the MTU is 1500 bytes. In the context of this paper, classifiers typically leverage the packet size, direction, and timing of transmitted packets in order to fingerprint websites. Figure 1 is a trace of 204 packets.

Trace- When a client accesses a certain web page, there is a continual exchange of packets between the client and the server. Each packet is described by its size and direction, and a record of the packets in the exchange forms the trace. In the Tor setting, many packets are of the same size due to padding of the data cells. According to Herrmann et al., the packet lengths 1500, -52, 52, -638, and 1150 bytes comprise 87.6% of all Tor packets [5].

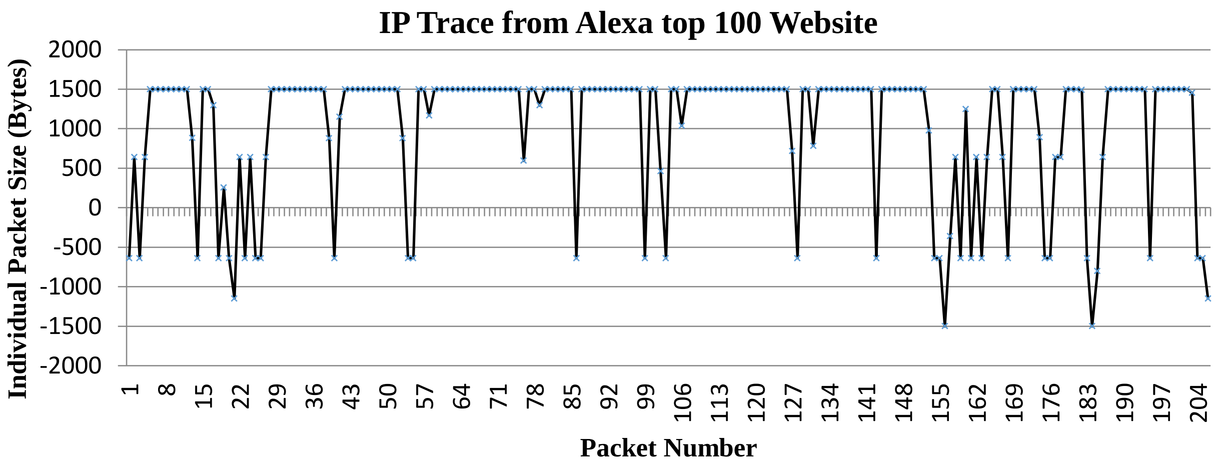

Figure 1: Example trace from an Alexa top 100 website [3]

Bursts- A burst consists of packets that are sent in one direction before they are interrupted by a packet of a different direction. In Figure 1, there is a large burst between the 57th packet and the 85th packet, where the packet sizes are all positive (indicating a continuous transmission of packets from the server to the client). The size of the burst can be defined by the number of packets or the cumulative number of bytes.

Setting- Setting simply describes the configuration of one's computer and the network over which a trace is acquired. This would include anything that would affect the collection of the trace, such as the browser setting, whether caching is enabled, etc.

III. Scenario

With the advent of SSL, the Internet has established an effective method of encrypting the data sent between a client and the server. Therefore, potential attackers are not able to inspect the contents of the packet, even if they were able to directly intercept it. This situation is modeled by Figure 2, thus it is clear that in this setting the attacker has access to the destination server.

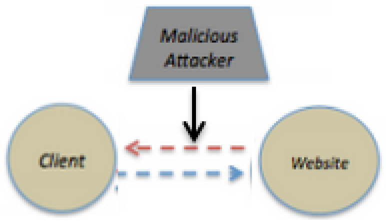

Figure 2: Setting that models a basic encrypted connection, with an attacker observing the link between the client and the server.

In order for the attacker to not have direct access to the destination server, the connection can be made by going through other nodes or proxies. Therefore, the attacker can only view the link between the client and the first node. In this setting, which is the very setting used by Tor [4], the attacker does not have information from the packets and cannot see the destination server as directly as he could in Figure 2. For Tor specifically, the network randomly reconfigures a

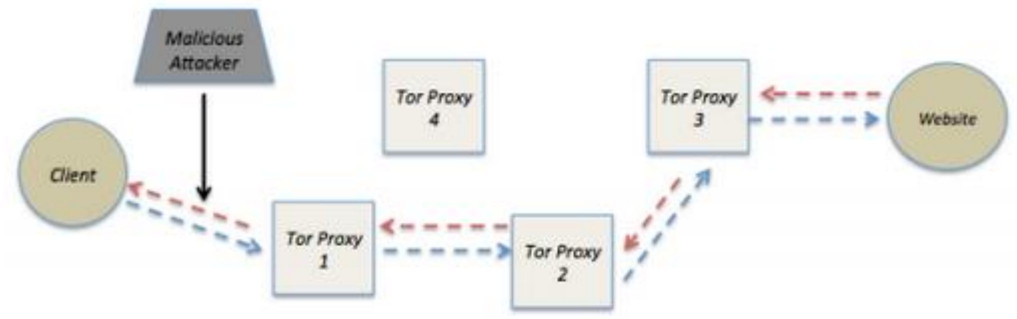

Figure 3: This setting models an encrypted connection with three intermediate nodes. The attacker is viewing the link between the client and Node 1.

different path through three random Tor proxies between the client and the server [4]. This setting is demonstrated by Figure 3. Note that Tor Proxy 4 in Figure 3 is not actively involved in the illustrated communication; however, a later request could randomly use Tor Proxy 4 instead of one of the others.

IV. Related Works:

Previous research in the field of website traffic analysis typically utilizes the same methodology. The classification typically assumes that the attacker possesses a list of possible websites and records traces of these websites at a desired setting, forming what are known as "training traces". The attacker can subsequently observe a victim's Internet traffic between a guard node and the client, recording the website trace and forming a "testing trace". The features of the testing trace (e.g. packet size, time of transmission, etc.) are embodied in a vector *G*. Multiple training traces are then used to train machine learning algorithms such as Support Vector Machines (SVM) or Naive Bayes Classifiers, before the testing trace *G* is identified by the classifier [1,2,3,5,6]. Typically, multiple training traces are used for each website to account for the inherent variability in website traces (e.g. packet reordering, advertisements, etc.); for example, in Dyer's experiments, he trains the machine learning algorithms with 16 training traces per website [1]. In the machine learning algorithms, the testing trace G is matched to one particular website. While this paper primarily explores the various countermeasures against the classifiers in detail, we discuss the classifiers to identify the inspiration for our defenses.

The concepts of website fingerprinting and traffic analysis were initially developed by Andrew Hintz who characterized website traces using the total number of bytes of HTTPS data that are sent to the user on any of the user's ports [7]. In particular, the premise of the research was that web objects were identifiable through examination of the TCP connection between the client and the server. The earlier work eventually evolved to characterizing websites by packet size, particularly in the works of Hermann et al. and Liberatore et al.. Hermann used the aggregated normalized frequency of the encrypted packet size across all training sets, while Liberatore composed a vector of the actual packet sizes across the training sets. Both tests were done in OpenSSH tunnels [5,6].

With the introduction of the onion-routing services Tor and JonDonym (formally known as JAP), the accuracy of the aforementioned classifiers decreased greatly. With the advent of pipelining and persistent connections, classifiers that depended on accurately identifying the start of HTML objects were no longer effective. Additionally, these anonymity networks send the packets in data cells of a fixed size, which cause many packet lengths to be undistinctive; for example, Tor sends data in cells of size 512 bytes. However, this does not mean that all *packets* are 512 bytes—as discussed in Section II: Terminology. In the aforementioned Hermann et al., which explicitly uses packet sizes for its classification, the classifier only successfully identified 2.96% of Tor traces, while achieving a rate of 96.65% on OpenSSH. For the purpose of simplicity, this paper focuses on the reinforcement of Tor, the onion-routing network that is consistently used by over 2 million users [4].

Despite Tor's success in hiding website information from packet-level classifiers, more recent classifiers have successfully penetrated Tor's defenses. The first of these classifiers was proposed and tested by Cai et al., who captured the packet size and direction in a vector [3]. While the features Cai extracts from the packets are no different than the features extracted by Hermann and Liberatore, Cai distinguishes his work by calculating the Damerau-Levenshtein edit distance (typically used for string comparisons). The edit distance is calculated between the testing trace and all the training traces, which is then used to define the SVM Kernel function. Cai's classifier is unique in that it factors in the order of the packets, accounting for its success against Tor Nodes. On randomly selected websites through Tor nodes, Cai's classification successfully classifies websites at a rate of 83.7% on a set of 100 websites [3].

The second such classifier was proposed by Panchenko et al., who utilizes a wide variety of features to train the Support Vector Machine. Such features include the total number of transmitted bytes, the number of packets, packet size occurrences, and most importantly, the number of bytes in a burst. The inclusion of trace burst sizes as features lead to a large increase in website classification accuracy, from 35.07% to 43.7%, in a set of 775 sites. On top of burst sizes, Panchenko exploits several other features of the trace; using all features, Panchenko achieves a successful classification rate of 54.61% in a closed-world dataset created by Hermann et al. [2]. We recreated Panchenko's classifier and ran tests with traces generated in the Tor setting. Results are provided in Figure 4.

The third and final such classifier was proposed by Dyer et al., whose proposed classifier, VNG++, trains a naïve Bayes classifier using "coarse features". Using only the total transmission time, total size of transmission, and byte-level burst size, Dyer demonstrates a 61% success rate in an OpenSSH setting on a relatively small set of 128 possible websites. Dyer's paper is the first to propose a classifier that does not feature individual packet sizes. While Dyer does not discuss the effectiveness of his classifier on Tor Nodes in his paper, our own experiments with his code base demonstrate 20.3% classification accuracy in a Tor setting, with a universe size of 512 websites. These results are presented in Figure 4.

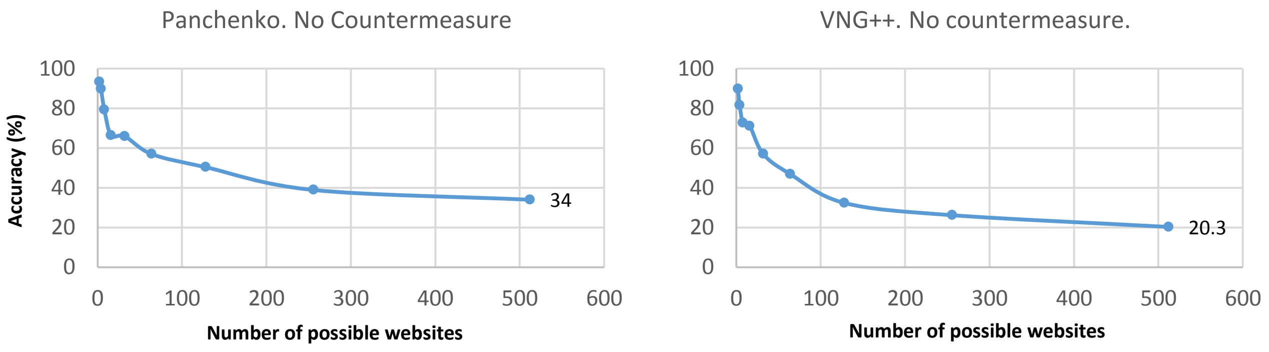

Figure 4: Results of applying Panchenko and VNG++ in the Tor environment, with no additional countermeasure, and with increasing Universe size.

Our goal is to find a defense that circumvents the website fingerprinting attacks of Cai, Panchenko, and Dyer, while finding a balance between accuracy and overhead. All three classifiers leverage different aspects of a trace – packet size, burst size, and order – each of which must be addressed.

V. Recreation of Experiments

In order to create various defenses, we attempt to recreate and eventually modify the experiments of Panchenko, Dyer, and Cai. We attempted to reach out to Panchenko to access his code base but were unsuccessful. As a result, at the time of this report, we have attempted to recreate the experiments of Cai et. al. and Dyer et al..

A. Recreating Cai's Classifier

The distinguishing characteristics of Cai's algorithm revolved around the application of the Damereau-Levenshtein Edit Distance calculation. While Cai's online Github contained the necessary code for the classification process, there was little documentation and much of the code was hard-coded to specific tests and settings, making it extremely difficult to recreate the experiment from the existing code. We decided, after some discussion with Cai, to rewrite parts of the code, in particular the Tor Node trace parser and the Damereau-Levenshtein Edit Distance algorithm. In this process, writing the edit distance algorithm revealed potential improvements in the algorithm, in particular the assignment of penalties to each transformation and the type of transformation. In other words, if there are two vector traces *t* and *t'* containing packet sizes and directions, how many insertions, deletions, transpositions, and substitutions are necessary to turn *t* into *t'*? Each transformation in the context of Cai's experiment represented a reordering of the packets, omission of packet request (due to caching), or other slight variations that are a cause of different settings. However, the attacker must determine how much to weight each transformation in calculating the edit distance. Cai tests several penalty assignments and simply weighs transpositions as 20 times the penalty of the other transformations. We believe that the accuracy of Cai's code could be improved by taking into account burst sizes.

Through the works of Dyer and Panchenko, we have seen how burst sizes are highly indicative of a website's identity. The Damerau-Levenshtein distance can be used to further include the order of the bursts in the classification activity. Because of this, we suggest that it may be effective to introduce additional transformations into the trace that take into account direction; in particular, a higher penalty would be assigned to transformations that disrupt a burst. The following are possibilities we considered for assigning higher penalties:

1.) Insertion with Direction Change: Insertion of packet that is in the opposite direction of its surrounding packets, thus separating a burst.

2.) Deletion with Direction Change: deletion of packet that is in the opposite direction of its surrounding packets, thus combining two disparate bursts

3.) Inverse substitution: Replacing an packet with a packet in the opposite direction

Although we successfully developed new code in Java to test the newly assigned burst-specific penalties, we were unable to fully comprehend the existing code base's stratified 10-fold cross validation that occurred before the Support Vector Machine training. As a result, we were unable to precisely recreate Cai's experiments, or test our own ideas beyond the calculation of the edit distance. We continue to work with Cai to improve the classifier and improve our understanding of how our proposed countermeasures perform against the Damerau-Levenshtein Edit Distance.

B. Recreating Dyer's Classifier: VNG++

Dyer provided a well-documented code base on Github (https://github.com/kpdyer/website-fingerprinting), and well detailed instructions for recreating his tests in the Tor environment. We run tests on all countermeasures explored by Dyer primarily against Panchenko's classifier and Dyer's VNG++ classifier. We run each test with varying universe size, from 2 to 512 possible websites, with data acquired from Herrmann's Tor dataset. For each possible website we train the classifier with 16 traces, that also have the countermeasure applied, and test the classifier with 2 traces. Also, for each test we run 512 trials. For instance, in the case of a universe size of 2 websites, we repeat the tests 256 times in order for there to be 512 tests. Lastly, it is important to note that we apply the countermeasure to the training traces, as well as the testing traces.

C. Setting under which Traces were collected

We decided to use the same trace data set as Dyer (which he had taken from Hermann), which consisted of 8510 websites in 2009 collected at a rate of 85 websites per second. The websites were collected from a Linux client machine and the browser used was Firefox 2.0. Dyer focused on the OpenSSH traces of Herrmann's data set, but we focus on the Tor traces of Herrmann's data set.

D. Metric for Evaluating Defenses

The primary means with which we evaluated each novel countermeasure was using accuracy and overhead.

$$Accuracy = \frac{c}{Tk}$$

where c is the number of correctly classified test traces, k is the size of the privacy set size, and T is the number of test traces per set (we set this value to 2 for all traces). The accuracy is eventually averaged with other trials.

$$Overhead = \frac{Number\ bytes\ in\ trace\ without\ countermeasure}{Number\ bytes\ in\ trace\ with\ countermeasure}$$

E. Anatomy of the Trace

Our strategy in identifying effective defenses against classifiers revolves around close inspection of website traces derived from Tor Nodes. Figure 5 shows the size of each burst in a trace (this is the same trace that was included in the introduction).

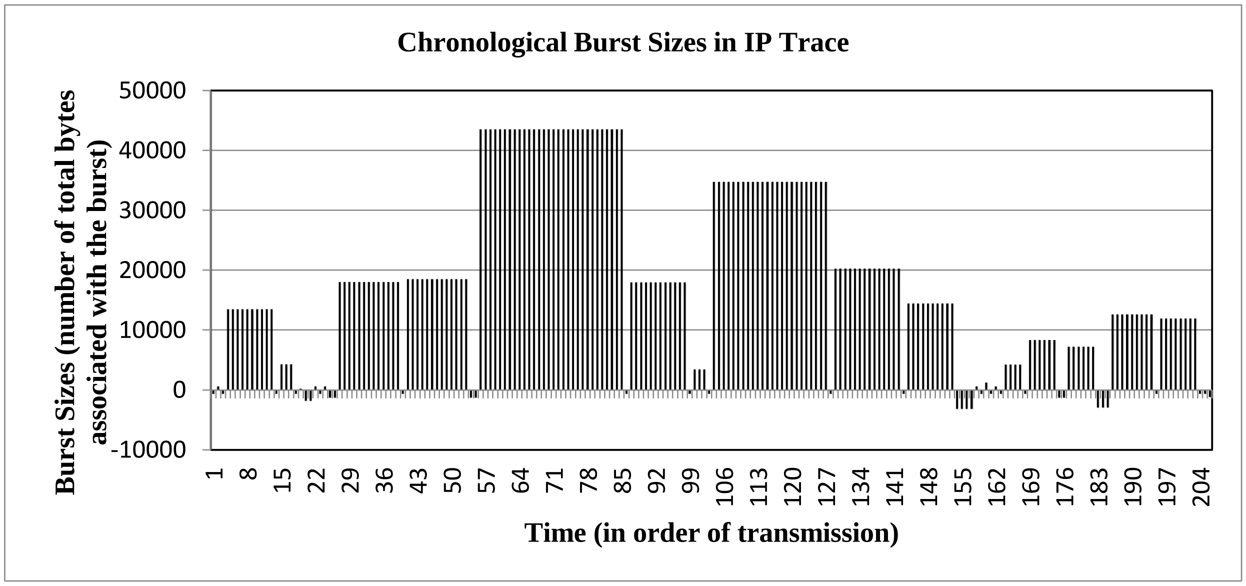

Figure 5: A representation of the bursts in a trace.

While the Tor packet-level padding reduces the amount of unique packet sizes in the trace, the number of bytes in each burst varies dramatically. Furthermore, as expected, the burst sizes going from server to client (the positive byte sizes) are significantly larger than the burst sizes going in the opposite direction. Note that in this particular trace and plot, 52 byte acknowledgement packets are ignored; many of the effective classifiers, including Cai and Panchenko, ignore 52 byte acknowledgement packets as they disrupt bursts, making it harder to classify website traces. A successful defense against the top performing classifiers would need to follow this pattern to disrupt the bursts.

VI. Packet Level Countermeasures:

Dyer's work presents the effects of packet level countermeasures very clearly. However, his work demonstrates the effects of packet level countermeasures in an OpenSSH environment rather than a Tor environment. Therefore, Dyer's tests have been re-created to see the effectiveness of packet level countermeasures in a Tor environment [1]. Herrmann's dataset provides traces generated in a Tor environment, and Dyer's code allows for analysis of that dataset; therefore, re-creating these tests was significantly enhanced by the previous work done by Herrmann and Dyer. It is important to emphasize that while the Tor program pads all data cells 512 bytes, packets can still be varying in size throughout the trace. Thus, packet level countermeasures can still have an impact in the Tor environment.

A. Existing Countermeasures [1]

Pad To MTU: All packet lengths are set to the MTU size. For our tests that length is 1500 bytes.

Exponential Padding: All packet lengths are increased to the nearest power of 2, or the MTU, whichever is smaller.

Session Random 255: For every application of this countermeasure, a random multiple of 8 between 8 and 248 is selected. Then every packet is increased by the random value, or to the MTU, whichever is smaller.

Mice and Elephants Padding: If the packet length is less than 128 bytes, the packet is padded to 128 bytes. Otherwise, it is padded to the MTU.

The effectiveness of these existing countermeasures are reflected in Figure 6.These countermeasures have been tested against the two most effective classifiers, VNG++ and Panchenko. The results highlight the effectiveness of these classifiers and the weaknesses of packet level countermeasures. Even with a universe size of 512 websites the lowest recorded accuracy is 10.6%, which is the Session Random 255 countermeasure applied against the Panchenko classifier.

B. Setting Discussion

The results produced here, in comparison with Dyer's results demonstrate the effectiveness of the current implementation of Tor. The effectiveness of these classifiers is significantly less in the Tor environment than in an OpenSSH environment. With a universe size of 512 the accuracy of Panchenko decreased from 96.4% to 34%, and the accuracy of VNG++ decreased from 91.6% to 20.3%. However, it is clear that Tor is still not highly secure, as demonstrated by Panchenko's 34% accuracy when no countermeasure is applied. The effectiveness of these classifiers in the current implementation of Tor make it clear that research in this field is necessary to enhance the security and anonymity of the Tor environment.

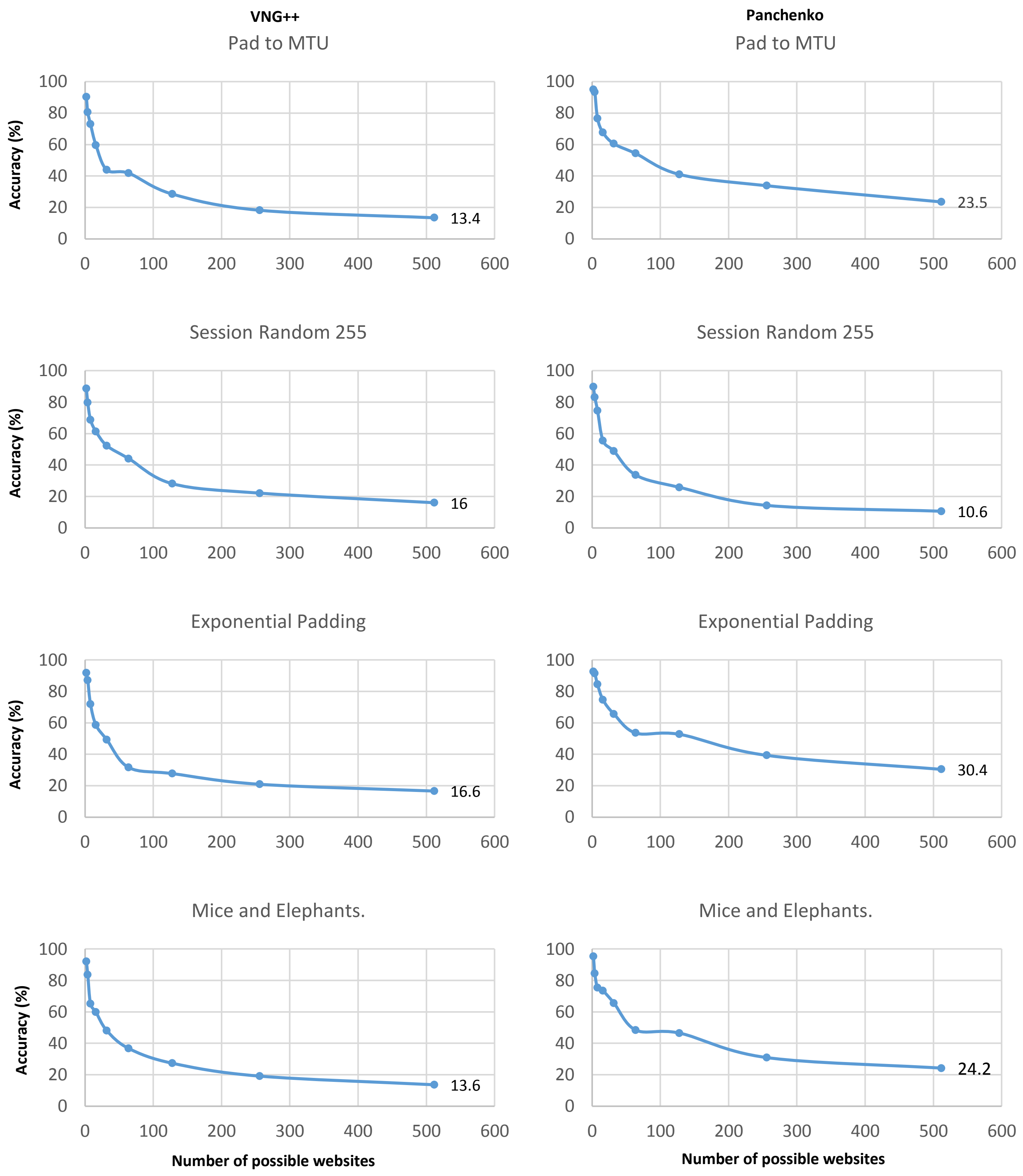

Figure 6: Accuracy vs. universe size plots demonstrating the ineffectiveness of packet level countermeasures.

C. Classifier Discussion

It is clear that the effectiveness of these countermeasures is not consistent across classifiers. First of all, it is important to note that in the setting without a countermeasure Panchenko achieves a higher accuracy than VNG++ (34% in comparison to 20.3%). This result is consistent with the classifier implementations because VNG++ only looks at coarse information, whereas Panchenko looks at the packet lengths, total bytes, and the order of packets. Therefore, prior to the application of a countermeasure, Panchenko is more effective because it utilizes more features. However, Panchenko appears to be highly dependent on packet lengths based on the fact that the countermeasure Session Random 255 decreased the accuracy from 34% to 10.6%. On the other hand, Session Random 255 only decreased the accuracy of VNG++ from 20.3% to 16.6%. In summary, the packet level countermeasures applied here had little impact on the effectiveness of VNG++. While both classifiers are certainly unique, they both look at the size of the bursts and that is why they are still very effective against packet level countermeasures.

D. Countermeasure Discussion

The packet level countermeasures applied yield very similar results against VNG++; however, the results vary significantly against Panchenko. It appears that Session Random 255 is the most effective, while the exponential padding is the least effective. Therefore, adding randomness is effective for packet level padding. Applying randomness to the countermeasures will be further explored at the burst level in our novel countermeasures.

D. Overhead Discussion

As seen in Table 2, the overhead of these packet level countermeasures varies significantly. Interestingly, the most effective countermeasure, Session Random 255, has the smallest byte level overhead, 6.5%. Furthermore, even though Pad to MTU has the largest overhead of 41.3%, it is not effective. In fact, Pad to MTU is the highest overhead possible with packet level countermeasures; therefore, if we are willing to increase byte level overhead, dummy packets must be sent.

Table 2: The average byte level overhead for the packet level countermeasure.

Countermeasure	**Overhead**
Pad To MTU	41.3%
Session Random 255	6.5%
Exponential Padding	16.3%
Mice and Elephants	39.8%

Sending dummy packets will increase the byte level overhead as well as the latency. In this paper we only investigate the byte level overhead. Dyer originally proposed the concept of sending dummy packets at fixed times. We further explore the effectiveness of sending dummy packets strategically to mask the bursts.

E. Packet Level Countermeasure Summary

In summary, packet level countermeasures are not effective. It is clear that effective classifiers can still identify a website with high accuracy. Therefore, even in a Tor environment packet level, countermeasures are not sufficient, thus more robust countermeasures must be developed to increase security and anonymity. With the consideration of the previous methods, and the emphasis on the information that is leaked due to burst sizes, we investigate methods to mask the bursts.

VII. Burst Level Countermeasures:

The burst level countermeasures that we developed seek to build upon the concepts discovered through testing the packet level countermeasures. Specially, we aim to determine the most effective way to include randomness in our countermeasures. Furthermore, we recognize Session Random 255 as an effective, low overhead, packet level countermeasure, so we include that on top of many of our Burst Level countermeasures. While low overhead countermeasures are desirable, we explore countermeasures with high overhead in order to discover if these top performing classifiers can be overpowered by robust countermeasures.

A. Novel Burst Level Countermeasures

i. Fixed Max Length Countermeasures

Results for all countermeasures proposed in this section against Panchenko and VNG++ are shown in Figure 7.

Burst Fixed Max Length 6 Packets: This countermeasure fixes the maximum length of a burst from the server to client to 6 packets. Therefore, after 6 packets are sent from the server to the client, the client sends a packet of 638 bytes (standard 512 byte Tor cell with packet overhead) to the server. Since packets of 638 bytes are commonly sent from the client to the server in the Tor environment, this dummy packet is difficult to identify as a dummy packet. The burst length of size 6 was selected to be a low value that would break up large bursts multiple times, but would not require significant byte level overhead. This is the only burst level padding countermeasure that does not include the Session Random 255 packet level countermeasure.

Burst Fixed Max Length 6 Packets with *Packet Level Padding*: This countermeasure is very similar to the Burst Fixed Max Length 6 Packets; however, this also includes the Session Random 255 packet level countermeasure. Thus, all packets are padded by a random value according to the Session Random 255 scheme, and dummy packets are sent to limit any given burst size to 6 packets. Therefore, this countermeasure is a combination of a low overhead packet level countermeasure and a low overhead burst level countermeasure.

Burst Session Random Fixed Max Length: This countermeasure selects a random value, *β,* between 2 and 10 before manipulating each trace. Then, while applying this countermeasure and a burst of size *β* is witnessed, the countermeasure sends a dummy packet, packet of 638 bytes from the client to the server, to break up the burst. Consequently, during each application of this countermeasure to a trace, every burst will have a maximum length between 2 and 10. It is important to note that when applying this countermeasure there will be significantly more overhead if *β* is 2 rather than 10. Therefore, we make sure to run multiple trials in order to determine the average overhead. Additionally, each packet is padded according to the Session Random 255 scheme.

Burst Session Random Fixed Max Length with Random Dummy Packets: This countermeasure is the same as above, but rather than sending a dummy packet of 638 bytes, a packet is sent from the client to the server of some random length. This random length is at least the length of the previous packet in the trace and at most 248 more bytes than the previous packet in the trace. This countermeasure is less effective and will not be explored further in this report.

Burst Random Maximum Length: This countermeasure is similar to Burst Session Random Fixed Max Length; however, every time the algorithm has to send a dummy packet, the algorithm selects a new value for *β,* thus the maximum burst size continuously changes throughout the algorithm. This algorithm intends to randomize the trace bursts further. This algorithm yields similar, yet slightly worse, results as Burst Session Random Fixed Max Length, so it will not be analyzed throughout this paper. The extra layer of randomness does not make the countermeasure more effective.

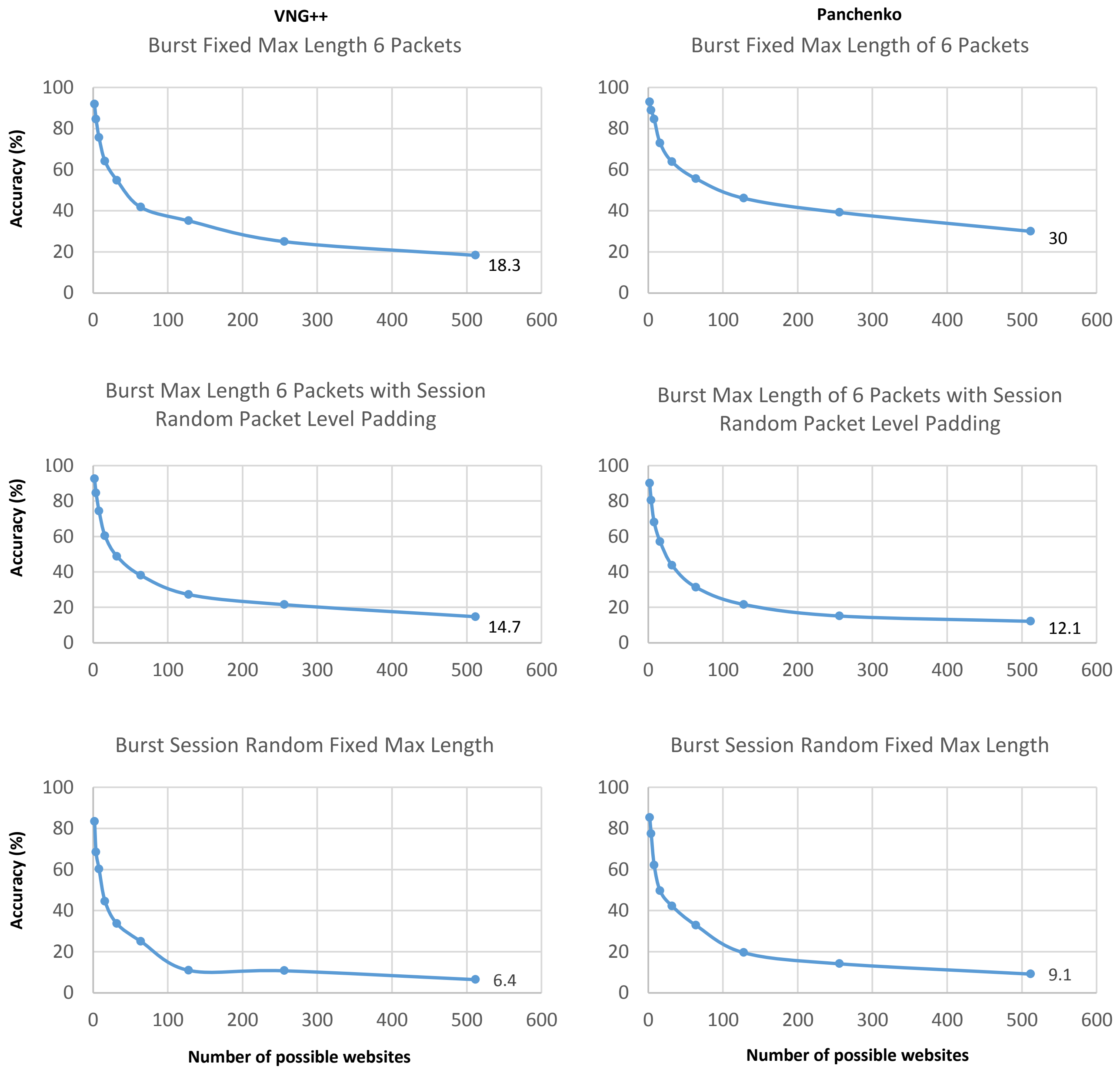

Figure 7: Summary of Burst Level countermeasures that focus on setting a maximum length for the bursts from the server to the client.

ii. Fixed Burst Length Countermeasures

The accuracy of VNG++ and Panchenko when classifying against these countermeasures is shown in Figure 8.

Fixed Burst Lengths at Byte Level: This countermeasure makes all bursts from the server to the client the same number of bytes. In the beginning of the countermeasure a random number, *G*, is selected within the range of 3000 and 15000 bytes, and all bursts are padded to this length. If a burst size is less than *G*, then packets of random size are added until the burst is of size *G*. If the burst is longer than *G*, then after *G* bytes are seen a dummy packet of 638 bytes is sent from the client to the server, and then the server continues to send data. An important aspect of this countermeasure is that some packets are split (also known as IP fragmentation), and delivered to the client via distinct packets despite the server sending it as one packet prior to the application of the countermeasure. The countermeasure appropriately pads the split packet to include the protocols. Again, each packet is padded according to the Session Random 255 scheme.

Random Burst Lengths at Byte Level: This countermeasure is similar to Fixed Burst Lengths at Byte Level; however, this countermeasure selects a new *G* value every time a Burst is padded or a Burst is split with a dummy packet. This countermeasure yields similar, yet slightly worse, results as Fixed Burst Lengths at Byte Level, so it will not be analyzed throughout this paper. Again, the extra layer of randomness does not increase the effectiveness of the countermeasure.

Fixed Burst Length at Packet Level: This countermeasure selects a random value β to be a value between 2 and 10, and pads all bursts from the server to the client to be β packets in length. Therefore, if a burst is less than β, dummy packets of the MTU size are sent from the server to the client. If the burst length is greater than β, then one dummy packet of 638 bytes is sent from the client to the server to break up the packet.

Random Burst Lengths at Packet Level: This countermeasure is similar to Fixed Burst Lengths; however, every time a burst is padded or a dummy packet from the client is sent to the server, a new β value is selected. Therefore, this algorithm is very similar to the Fixed Burst Length algorithm, except that there is an increase in randomness. This increase in randomness does not increase the effectiveness of the countermeasure and will not be analyzed further in this report.

Exponential Burst Lengths: This countermeasure pads all bursts to have a number of packets that is a power of 2, with a maximum burst length of 64 packets. For instance, a burst that has 6 packets will be padded with two dummy packets so that the burst is 8 packets long. This countermeasure is not effective and will not be explored further in this report.

Client to Server: This algorithm is similar to the Burst Session Random Fixed Maximum Length; however, this algorithm sends either 3 or 4 random sized packets from the client to the server rather than just one packet of 638 bytes to break up bursts into multiple bursts of a fixed maximum length. The random dummy packets are multiples of 8 between 638 and 904 bytes. This algorithm intends to take advantage of the asymmetric download and upload usage on a broadband bandwidth. In a typical residential Internet package, the download speed will be significantly faster than the upload speed, as the average Internet user retrieve objects from the Internet far more frequently than they put things on it. Even with this asymmetry in information transfer rates, much of the upload capacity is not fully utilized. While the calculated byte level overhead of this algorithm may be high in our experiments, it is possible that in a reality, the byte level overhead would not actually affect the speed of the web browser, since the dummy packets will be sent via the upload channel, which still has ample capacity. This certainly would not hold true for every user, but the average person spends vastly more time downloading than uploading on the Internet.

B. Classifier Discussion

When the burst level countermeasures that we explored are applied against VNG++ classifier and Panchenko classifier, Panchenko achieves a higher accuracy against all countermeasures except Client to Server Padding and Burst Max Length 6 Packets with Packet Level Padding. Panchenko's classifier has a large decrease in accuracy between for Burst Max Length 6 Packets and Burst Max Length 6 Packets with Packet Level Padding, emphasizing the classifiers dependence on packet lengths. VNG++, on the other hand, was not as affected by adding the packet level countermeasure on top of the burst level countermeasure, confirming the classifiers sole dependence on coarse features. While both classifiers are very comparable when the burst level countermeasures are applied, Panchenko's algorithm accounts for more meta-data available in the masked trace; therefore, it achieves a higher accuracy than VNG++.

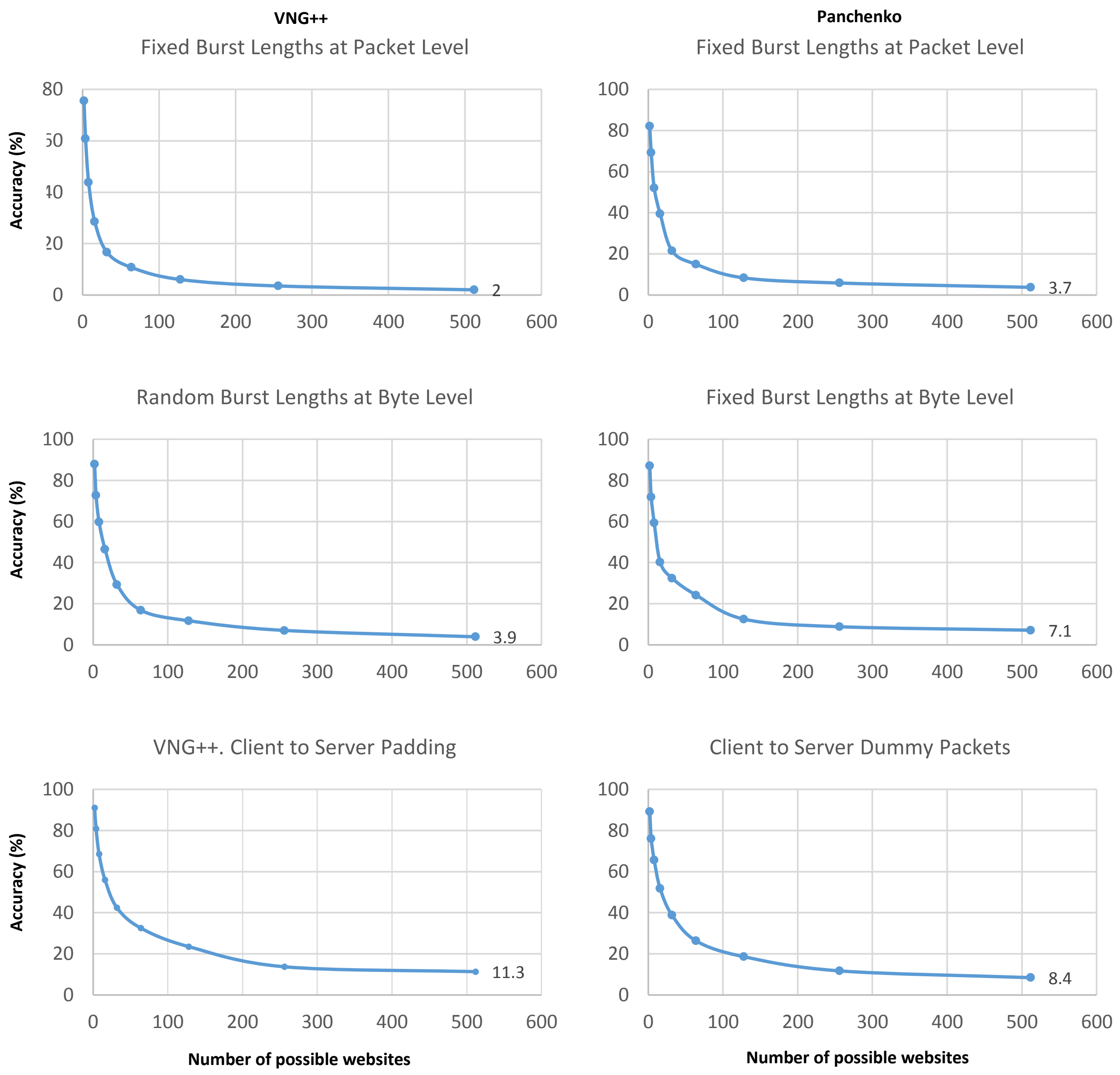

Figure 8: Summary of high overhead fixed burst lengths countermeasures, and the greedy client to server countermeasure.

C. Countermeasure Discussion

The first three burst level countermeasures of interest seek to mask the bursts by setting a maximum length—results are provided in Table 3. Burst Fixed Max Length of 6 Packets has very minimal effect on the accuracy of the classifiers, decreasing the accuracy of the classifier by less than 20%. When the Session Random 255 packet level countermeasure is applied in addition to Burst Fixed Max Length of 6 Packets, the accuracy of Panchenko drops significantly from 30% to12.1%, while VNG++ only drops from 18.3% to 14.7%. The drastic decrease in accuracy for Panchenko's classifier makes it clear that burst level countermeasures should be applied in addition to light weight packet level countermeasures. While VNG++ is barely affected by the additional packet level countermeasure, VNG++ exists to highlight how much information is contained in the coarse features of the trace; therefore, packet level padding will always be an important part of any effective countermeasure.

	VNG++	Panchenko
None	20.3	34
Burst Fixed Max Length	18.3	30
First Fixed Max Length with Packet Padding	14.7	12.1
Burst Session Random Fixed Max Length	6.4	9.1
Fixed Burst Lengths Packet Level	2	3.7
Fixed Burst Lengths Byte Level	1.3	7.1

Table 3: Accuracy results of burst level countermeasures with a universe size of 512.

Burst Fixed Max Length of 6 Packets with Packet Level Padding is altered to have a layer of randomness similar to the randomness in the packet level countermeasure Session Random 255. Randomness at this layer proves to be very effective, as the accuracy of VNG++ decreased to 6.4% and Panchenko decreased to 9.1%. When the amount of randomness involved in the countermeasure is applied more than once per trace, as in Burst Random Maximum Length, the effectiveness of the countermeasure decreases. This phenomenon was also found by Dyer as he shows that Session Random 255 is more effective than Packet Random 255. Therefore, while randomness increases the effectiveness of a countermeasure, excessive randomness decreases the effectiveness of the countermeasure.

The next set of countermeasures build directly off of the previous set of countermeasures. Again, randomness is applied once per trace and packet level padding is applied. However, these countermeasures are very distinct, as they pad all bursts from the server to the client (even if the burst is only one packet) to a certain number of packets. These countermeasures, Fixed Burst Lengths at Packet Level and Fixed Burst Lengths at Byte Level, naturally have a high overhead, but are very effective. This countermeasure is more effective when applied at the packet level against Panchenko's algorithm and more effective at the byte level against VNG++. The byte level implementation against VNG++ achieved an accuracy of 1.3%, the lowest recorded of all of our tests. When comparison to random guessing, which would yield an average accuracy of .2%, this countermeasure clearly still allows some information to be leaked; however, a significant amount of the coarse features exploited by Dyer's VNG++ are masked by the Fixed Burst Lengths countermeasures.

The greedy client to server countermeasure proved to be ineffective. With accuracy results of 11.3% against VNG++, it is clear that sending more packets along the open channel from the client to server did not mask a significant amount of information that is being exploited by these classifiers. In fact, the Burst Session Random Fixed Max Length algorithm similarly only sends packets along the client to server channel, and that algorithm achieves a lower accuracy with a lower byte overhead. Therefore, sending many packets along the open channel was ineffective, while sending packets on the full channel, server to client, has proven to be effective by the results produced by the Burst Fixed Lengths countermeasures.

D. Overhead Discussion

Of the three countermeasures that produced low classification accuracy, Burst Session Random Fixed Max length has a drastically lower byte level overhead of 11.1% rather than over 100%, data presented in Table 4. In comparing the two countermeasures that pad all bursts from the server to client, applying the countermeasure at the packet level has a lower byte level overhead than the byte level implementation. These overhead values only account for byte level overhead. We are currently researching

Countermeasure	**Overhead**
Burst Fixed Max Length	2.9%
Burst Fixed Max Length with Packet Padding	9.5%
Burst Session Random Fixed Max Length	11.1%
Fixed Burst Length Packet Level	112.4%
Fixed Burst Length Byte Level	137.5%

Table 4: Overhead results of the burst level countermeasures.

the latency overhead that these countermeasures would require if implemented in a real world setting.

VII. Conclusion:

How do the top performing classifiers perform in the Tor environment? The results produced in Dyer's paper suggest that traffic analysis countermeasures are not strong enough to protect against powerful classifiers, such as VNG++ and Panchenko, in the OpenSSH environment. We reproduce these tests in the Tor environment and prove that Dyer's results are consistent even in the Tor environment. While the Tor environment decreases the classification accuracy, the existing TA countermeasures are not sufficiently secure. The main reason for the lack of security in these countermeasures is the information that is available in the burst sizes; packet level padding is insufficient to mask the metadata.

Were our countermeasures effective? While these countermeasures can lead to significant amount of byte level overhead, strategically placed dummy packets can mask a significant amount of burst level information that is available in the communication traces. We suggest two particular countermeasures that are particularly effective, Burst Session Random Fixed Max Length and Fixed Burst Length at Packet Level. Both countermeasures include the Session Random 255 packet level countermeasure which serves as a first layer of defense that hides the information available in the packet lengths. Furthermore, both countermeasures have a level of randomness per session at the burst level. The only difference is that Burst Session Random Fixed Max Length only breaks up large bursts with dummy packets from the client to the server. On the other hand, Fixed Burst Length at Packet Level pads all bursts to a certain length, and breaks up long bursts; therefore, this countermeasure also sends dummy packets from the server to the client. In a universe size of 512 websites, Burst Session Random Fixed Max length leads to an accuracy of 6.4% and 9.1% against VNG++ and Panchenko respectively, with an overhead of 11.1%. With the same universe size Fixed Burst Lengths at Packet Level leads to an accuracy of 2% and 3.7% against VNG++ and Panchenko respectively, with an overhead of 112.4%.

Can burst information be concealed without excessive overhead? These results suggest that we can partially conceal the burst information that is available in the communication traces without a significant amount of overhead by simply breaking up the bursts, or can conceal the information more thoroughly by padding all bursts to a certain length. Rather than sending dummy packets at random or fixed times, we send dummy packets to strategically mask the bursts, and our results suggest that burst level padding is effective.

IX: Further Explorations:

Even with the proposed defenses in this paper, there is still much work to be done in order to optimize the trade-off between overhead and accuracy. After thoroughly exploring various probabilistic padding techniques, we decided that perhaps a different approach to the problem altogether might result in more optimal results. While we did not have the time to acquire the necessary data to verify these experiments, we provide a novel framework and implementation for this novel approach.

A. Proposing a Novel Framework in the Website Fingerprinting Approach

Existing papers have experimented with classifiers and defenses that navigate through a large *closed world* setting or an *open world* setting. A closed world setting assumes that the classifier is both trained and tested against the same subset of existing websites. An open world setting assumes that the classifier is trained using some subset of existing websites, but the testing trace can be any website in existence. All countermeasures in the existing art have attempted to make all websites in the open or closed world indistinguishable through various padding techniques, which are often cumbersome and leads to inefficiencies in padding overhead.

We propose that it is not necessary to pad away the unique features in the world of all possible websites, but only ones that are *similar* in terms of the various traffic features, particularly the burst size. Classifiers typically only identify the single most similar website, so this methodology would significantly boost the number of false positives. Research so far has focused on removing the uniqueness of a website trace and making all websites look alike; we suggest padding to make similar websites look even more similar to one another. We briefly propose a possible method for one implementation of this framework that leverages user habits. Due to constraints on time and a necessity for collecting an entirely novel data set, we were not able to test these frameworks.

B. Leveraging User Habits

This strategy applies to a very specific population of those who use network anonymity networks, and is particularly effective for a user who habitually returns to one or a few particular monitored websites (like a blog or a news site). In advance of accessing the site, the user could request a trace *t* of that particular website under a variety of settings. The user could then run tests locally to identify one or multiple sites *t'* that have "similar traces" but are not of interest to an attacker. We suggest that one effective measure for determining similarity would be the burst-level Damerau-Levenshtein Edit Distance, similar to the idea proposed by Cai. Once a similar trace is identified, *t* could be padded at the burst level to be identical to *t'*, such that any classifier would mistake *t* for *t'*. This has the potential to be an extremely low overhead defense (because of the inherent similarity in websites). However, there is much work to be done to design a padding algorithm that accounts for the fact that all burst sizes *t* are not bigger than all of the bursts in the *t'*.

This is a novel approach, and we encourage others in the field to explore this path, particularly with respect to new algorithms for determining similarity between traces, as well as a more flexible way of implementing this process. Furthermore, we hope that our probabilistic countermeasures will be able to benefit the Tor Community be reinforcing the security of the system without creating significant wait time for a page load.

References

[1] 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 33rd Annual IEEE Symposium on Security and Privacy, 2012.

[2] 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.

[3] Xiang Cai, Xincheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: website fingerprinting attacks and defenses. In ACM CCS, 2012

[4] Tor project: Anonymity online. https://www.torproject.org/, April 2014.

[5] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website Fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In Proceedings of the 2009 ACM workshop on Cloud computing security.

[6] Marc Liberatore and Brian Neil Levine. Inferring the sourceof encrypted http connections. In ACM CCS, 2006.

[7] Andrew Hintz. Fingerprinting websites using traffic analysis. In Privacy Enhancing Technologies. 2003.

X. Appendix

A. Accuracy results.

	Privacy Set Size								
Countermeasure	k=2	k=4	k=8	k=16	k=32	k=64	k=128	k=256	k=512
None	89.8	81.6	72.7	71.1	57	46.9	32.4	26.2	20.3
Pad To MTU	90.2	80.5	73	59.6	43.8	41.8	28.5	18.2	13.4
Session Random 255	88.7	79.7	68.8	61.3	52.3	44.1	28.1	22.1	16
Exponential Padding	91.8	87.1	71.9	58.6	49.2	31.6	27.7	20.9	16.6
Mice and Elephants	92	83.6	65.2	59.8	48	36.7	27.3	19.1	13.6

Table 5: VNG++ accuracy results for different universe sizes, tested against packet level countermeasures.

	Privacy Set Size								
Countermeasure	k=2	k=4	k=8	k=16	k=32	k=64	k=128	k=256	k=512
None	89.8	81.6	72.7	71.1	57	46.9	32.4	26.2	20.3
Burst Fixed Max Length	91.8	84.6	75.6	64.1	54.8	41.7	35.1	25	18.3
First Fixed Max Length with Packet Padding	92.6	84.6	74.4	60.3	48.8	38	27.2	21.5	14.7
Burst Session Random Fixed Max Length	83.3	68.4	60.2	44.5	33.6	25	10.9	10.7	6.4
Fixed Burst Lengths Packet Level	75.5	60.8	43.8	28.5	16.6	10.7	6	3.5	2
Fixed Burst Lengths Byte Level	85.6	51.2	40.6	27.3	15.6	10.2	4.7	4.1	1.3

Table 6: Panchenko accuracy results for different universe sizes, tested against packet level countermeasures.

	Privacy Set Size								
Countermeasure	k=2	k=4	k=8	k=16	k=32	k=64	k=128	k=256	k=512
None	93.4	89.8	79.3	66.4	66	57	50.4	38.9	34
Pad To MTU	94.9	93.4	76.6	67.6	60.5	54.3	41	33.8	23.5
Session Random 255	89.8	83.2	74.6	55.5	48.8	33.6	25.8	14.3	10.6
Exponential Padding	92.6	91.4	84.4	74.6	65.6	53.5	52.7	39.3	30.4
Mice and Elephants	95.3	84.4	75.4	73.4	65.6	48.4	46.5	30.9	24.2

Table 7: VNG++ accuracy results for different universe sizes, tested against burst level countermeasures.

	Privacy Set Size								
Countermeasure	k=2	k=4	k=8	k=16	k=32	k=64	k=128	k=256	k=512
None	93.4	89.8	79.3	66.4	66	57	50.4	38.9	34
Burst Fixed Max Length	93	88.9	84.6	72.9	63.8	55.6	46.1	39.2	30
First Fixed Max Length with Packet Padding	90.1	80.4	68	57	43.7	31.3	21.6	15.1	12.1
Burst Session Random Fixed Max Length	85.2	77.3	62.1	49.6	42.2	32.8	19.5	14.1	9.1
Fixed Burst Lengths Packet Level	82.1	69.2	52.1	39.4	21.4	14.9	8.3	5.8	3.7
Fixed Burst Lengths Byte Level	87.1	71.9	59.4	40.2	32.4	24.2	12.5	8.8	7.1

Table 8: Panchenko accuracy results for different universe sizes, tested against burst level countermeasures.

B. Python Code

The code used to generate these tests is available on Github at https://github.com/MikeFreyberger/burstingintobursts. In order to test this code, Kevin Dyer's code must also be accessed which is available at https://github.com/kpdyer/website-fingerprinting.