CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense

Xiang Cai
Stony Brook University
xcai@cs.stonybrook.edu

Rishab Nithyanand
Stony Brook University
rnithyanand@cs.stonybrook.edu

Rob Johnson
Stony Brook University
rob@cs.stonybrook.edu

ABSTRACT

Website fingerprinting attacks[10] enable an adversary to infer which website a victim is visiting, even if the victim uses an encrypting proxy, such as Tor[19]. Previous work has shown that all proposed defenses against website fingerprinting attacks are ineffective[6, 4]. This paper advances the study of website fingerprinting defenses by first laying out the complete specifications of the Congestion-Sensitive BuFLO scheme outlined by Cai, et al. [4]. CS-BuFLO, which is based on the BuFLO defense proposed by Dyer, et al.[6], was not fully-specified by Cai, et al, but has nonetheless attracted the attention of the Tor developers [16, 17]. Next, a full working implementation of CS-BuFLO is provided. Finally, a thorough evaluation of CS-BuFLO is performed using empirical data (rather than data from simulations). Our experiments find that Congestion-Sensitive BuFLO has high overhead (around 2.3-2.8x) but can get $6\times$ closer to the bandwidth/security trade-off lower bound than Tor or SSH.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General–Security and protection
; K.4.1 [**Computing Milieux**]: Computers and Society–Privacy

Keywords

Anonymity; Website fingerprinting attacks and defenses

1. INTRODUCTION

Website fingerprinting attacks have emerged as a serious threat against web browsing privacy mechanisms, such as SSL, Tor, and encrypting tunnels. These privacy mechanisms encrypt the content transferred between the web server and client, but they do not effectively hide the size, timing, and direction of packets. A website fingerprinting attack uses these features to infer the web page being loaded by a client.

Researchers have engaged in a war of escalation in developing website fingerprinting attacks and defenses, with two recent papers demonstrating that all previously-proposed defenses provide little security[6, 4]. At the 2012 Oakland conference, Dyer, et al. showed that an attacker could infer, with a success rate over 80%, which of 128 pages a victim was visiting, even if the victim used network-level countermeasures. They also performed a simulation-based evaluation of a hypothetical defense, which they call BuFLO, and found that it required over 400% bandwidth overhead in order to reduce the success rate of the best attack to 5%, which is still well-above the ideal 0.7% success rate from random guessing. At CCS 2012, Cai et al. proposed the DLSVM fingerprinting attack and demonstrated that it could achieve a greater than 75% success rate against numerous defenses[4], including application-level defenses, such as HTTPOS[13] and randomized pipelining[15]. As a result, it is not currently known whether there exists any efficient and secure defense against website fingerprinting attacks.

Cai, et al. also proposed Congestion-Sensitive BuFLO, which extended Dyer's BuFLO scheme to include congestion sensitivity and some rate adaptation, but they left many details unspecified and did not implement or evaluate their scheme. Despite the lack of data on CS-BuFLO, the Tor project has indicated interest in incorporating CS-BuFLO into the Tor browser [17, 16].

In order to get a better understanding of the performance and security of the CS-BuFLO protocol, this paper presents a complete specification of CS-BuFLO, describes an SSH-based implementation, and evaluates its bandwidth overhead, latency overhead, and security against current attacks.

Cai's description of the CS-BuFLO protocol outlines solutions to several performance and practicality problems in the original BuFLO protocol – CS-BuFLO is TCP-friendly, it pads streams in a uniform way, and it uses information collected offline to tune BuFLO's parameters to the website being loaded. We propose two further improvements: we modify CS-BuFLO to adapt its transmission rate dynamically, and we improve its stream padding to use less bandwidth while hiding more information about the website being loaded. Dynamic rate adaptation makes CS-BuFLO much more practical to deploy, since it does not require an infrastructure for performing offline collection of statistics about websites, but poses a challenge: adapting too quickly to the website's transmission rate can reveal information about which website the victim is visiting. CS-BuFLO balances these performance and security constraints by limiting the rate and precision of adaptation.

Defense	n	Method	Source	Panchenko	VNG++	DLSVM	BW	Latency
CS-BuFLO (CTSP)	200	Empirical	this paper	18.0	13.0	20.6	2.796	3.271
CS-BuFLO (CPSP)	200	Empirical	this paper	24.2	16.5	34.3	2.289	2.708
CS-BuFLO (CTSP)	120	Empirical	this paper	23.4	20.9	28.9	2.799	3.444
CS-BuFLO (CPSP)	120	Empirical	this paper	30.6	22.5	40.5	2.300	2.733
BuFLO ($\tau = 0, \rho = 40, d = 1000$)	128	Simulation	[6]	27.3	22.0	N/A	1.935	N/A
BuFLO ($\tau = 0, \rho = 40, d = 1500$)	128	Simulation	[6]	23.3	18.3	N/A	2.200	N/A
BuFLO ($\tau = 0, \rho = 20, d = 1000$)	128	Simulation	[6]	20.9	15.6	N/A	2.405	N/A
BuFLO ($\tau = 0, \rho = 20, d = 1500$)	128	Simulation	[6]	24.1	18.4	N/A	3.013	N/A
BuFLO ($\tau = 10^5, \rho = 40, d = 1000$)	128	Simulation	[6]	14.1	12.5	N/A	2.292	N/A
BuFLO ($\tau = 10^5, \rho = 40, d = 1500$)	128	Simulation	[6]	9.4	8.2	N/A	2.975	N/A
BuFLO ($\tau = 10^5, \rho = 20, d = 1000$)	128	Simulation	[6]	7.3	5.9	N/A	4.645	N/A
BuFLO ($\tau = 10^5, \rho = 20, d = 1500$)	128	Simulation	[6]	5.1	4.1	N/A	5.188	N/A
HTTPOS	100	Empirical	[4]	57.4	N/A	75.8	1.361	N/A
Tor+rand. pipe.	100	Empirical	[4]	62.8	N/A	87.3	1.745	N/A
Tor	100	Empirical	[4]	65.4	N/A	83.7	N/A	N/A
Tor	120	Empirical	this paper	56.3	36.8	77.4	1.247	4.583 [a]
Tor	200	Empirical	this paper	50.1	31.8	75.1	1.244	4.919
Tor	775	Empirical	[14]	54.6	N/A	N/A	N/A	N/A
Tor	800	Empirical	[4]	40.1	N/A	50.6	N/A	N/A
SSH	120	Empirical	this paper	86.5	75.0	80.7	1.128	1
SSH	200	Empirical	this paper	84.4	72.9	79.4	1.111	1

Table 1: **Main evaluation results for CS-BuFLO, and comparison to results on other schemes reported in other papers. Bandwidth and Latency are reported as overhead ratios.**

We have implemented CS-BuFLO in a custom version of OpenSSH. Our implementation also includes a Firefox browser plugin that informs the SSH client when the browser has finished loading a web page. The CS-BuFLO implementation uses this information to reduce the amount of padding performed after the page load has completed.

We evaluate CS-BuFLO, and compare it to Tor, on the Alexa top 200 websites in the closed-world setting. The Alexa top 200 websites represent approximately 91% of page loads on the internet [1], so these results reflect the security users will obtain when using these schemes in the real world. Furthermore, prior work on website fingerprinting attacks has found that an attackers success rate only goes down as the number of websites increases, so our results give a high-confidence upper bounds on the success rate these attacks may achieve in larger settings.

In our experiments, CS-BuFLO uses 2.8 times as much bandwidth as SSH (i.e. no defense) and the best known attack had only a 20% success rate at inferring which of 200 websites a victim was visiting. This is a substantial improvement over previously-proposed schemes – the same attack had a success rate over 75% against Tor and SSH under the same conditions. Table 1 compares our results with results reported in other papers. These comparisons must be done carefully, since the experiments used different numbers of websites and methodologies. Nonetheless, the following conclusions are clear from the data:

- CS-BuFLO hides more information than Tor, SSH, HTTPOS, and Tor with randomized pipelining, albeit with higher cost. For example, the DLSVM attack has a lower success rate against CS-BuFLO in a closed-world experiment with 100 websites than it has against Tor with 800 websites.

- Overall, CS-BuFLO achieves approximately the same bandwidth/security trade-off in our empirical analy-sis as BuFLO achieved in Dyer's simulated evaluation. For example, CS-BuFLO in CTSP mode had a bandwidth ratio of 2.8 and Panchenko's attack had a success rate of 23.4% on 120 websites. BuFLO with $\tau = 0$, $\rho = 40$, and $d = 1500$ had almost identical security, but a bandwidth ratio of 2.2. Although CS-BuFLO optimizes many aspects of the BuFLO protocol, an empirical evaluation presents issues that do not arise in a simulation, such as dropped packets, retransmissions, and application-level timing dependencies.

We use results from previous work [3] to compare defenses that offer different security/bandwidth trade-offs by comparing how close they are to the lower bound. We find that Congestion-Sensitive BuFLO gets over 6× closer to the bandwidth/security trade-off lower bound than Tor or plain SSH. Dyer's reported experiments with BuFLO showed somewhat better trade-off performance, but those results were based on simulations and are not directly comparable. Despite the improvement of CS-BuFLO over Tor and SSH, there is still a large gap between the lower bounds and the best defenses.

In summary, this paper makes the following contributions: Section 3 gives a complete specification of the CS-BuFLO protocol, describing optimizations to make the protocol congestion sensitive, rate adaptive, and efficient at hiding macroscopic website features, such as total size and the size of the last object. Section 4 describes our prototype implementation in SSH, which also includes a Firefox plugin to notify the proxy when the browser finishes loading a web page. Section 5 presents empirical evaluation results for CS-BuFLO, Tor, and SSH, and shows that CS-BuFLO provides better security, albeit at higher bandwidth costs. We also show that CS-BuFLO is closer to the lower bound on the security/bandwidth trade-off than Tor and SSH.

1.1 Website Fingerprinting Attacks

In a website fingerprinting attack, an adversary is able to monitor the communications between a victim's computer and a private web browsing proxy. The private browsing proxy may be an SSH proxy, VPN server, Tor, or other privacy service. The traffic between the user and proxy is encrypted, so the attacker can only see the timing, direction, and size of packets exchanged between the user and the proxy. Based on this information, the attacker attempts to infer the website(s) that the user is visiting via the proxy. The attacker can prepare for the attack by collecting information about websites in advance. For example, he can visit websites using the same privacy service as the victim, collecting a set of website "fingerprints", which he later uses to recognize the victim's site.

Fingerprinting attacks are an important class of attacks on private browsing systems. For example, Tor states that it "prevents anyone from learning your location or browsing habits."[19] Successful fingerprinting attacks undermine this security goal. Fingerprinting attacks are also a natural fit for governments that monitor their citizens' web browsing habits. The government may choose not to (or be unable to) block the privacy service, but nonetheless wish to infer citizens' activities when using the service. Since it can monitor international network connections, the government is in a good position to mount website fingerprinting attacks.

Researchers have proposed two scenarios for evaluating website fingerprinting attacks and defenses: closed-world models and open-world models. A closed-world model consists of a finite number, n, of web pages. Typical values of n used in past work range from 100 to 800 [6, 4, 14]. The attacker can collect traces and train his attack on the websites in the world. The victim then selects one website uniformly at random, loads it using some defense mechanism, such as Tor or SSH, and the attacker attempts to guess which website the victim loaded. The key performance metric is the attacker's average success rate. In an open-world model, there is a population of victims, each of which may visit any website in the real world, and may select the website using a probability distribution of their choice. The attacker does not know any individual victim's distribution over websites, but has aggregate statistics about website popularity. The attacker's goal is to infer which of the victims are visiting a particular "website of interest", i.e. an illegal or censored site. In this case, the primary evaluation criteria are false positives and false negatives. Perry has critiqued the closed-world model for its artificiality [16]. However, the two models are connected: Cai, et al., showed how to bootstrap a closed-world attack into an open-world attack, such that better closed-world performance yields better open-world performance [4]. Thus, although experiments in the closed-world cannot tell us whether an attack or defense will be successful in the real world, we can use closed-world experiments to compare different attacks and defenses.

2. RELATED WORK

Network-level website fingerprinting defenses pad packets, split packets into multiple packets, or insert dummy packets. Dyer, et al., list numerous approaches to padding individual packets, including pad-to-MTU, pad-to-power-of-two, random padding, etc.[6]. They showed that none of the padding schemes was effective against the attacks they evaluated. Wright, et al., proposed traffic morphing, in which packets are padded and/or fragmented so that they conform to a specified target distribution[21]. Dyer, et al., defeated this defense, as well[6]. Lu, et al., extended traffic morphing to operate on n-grams of packet sizes, i.e. their scheme pads and fragments packets so that n-grams of packet sizes match a target distribution[12]. Dyer, et al. also proposed BuFLO, which pads or fragments all packets to a fixed size, sends packets at fixed intervals, injecting dummy packets when necessary, and always transmits for at least a fixed amount of time[6]. They found that they could reduce their best attack's success rate to 5% (when guessing from 128 websites), at a bandwidth overhead of 400%. Fu, et al., found in early work that changes in CPU load can cause slight variations in the time between packets in schemes that attempt to send packets at fixed intervals, and recommended randomized inter-packet intervals instead[7].

Application-level defenses alter the sequence of HTTP requests and responses to further obfuscate the user's activity. For example, HTTPOS uses HTTP pipelining, HTTP Range requests, dummy requests, extraneous HTTP headers, multiple TCP connections, and munges TCP window sizes and maximum segment size (MSS) fields[13]. Tor has also released an experimental version of Firefox that randomizes the order in which embedded objects are requested, and the level of pipelining used by the browser during the requests[15]. Both schemes were defeated by Cai, et al[4].

Researchers have proposed numerous attacks on basic encrypting tunnels, such as HTTPS, VPNs, and IPSec[2, 5, 8, 9, 10, 11, 12, 18, 22, 23, 6]. These attacks focus primarily on packet sizes, which carry a lot of information when no padding scheme is in use. Herrmann, et al., developed an attack based on packet sizes that worked well on simple encrypting tunnels[9], but performed quite poorly against Tor, which transmits data in 512-byte cells. Panchenko, et al., designed an attack that used packet sizes, along with some ad hoc features designed to capture higher-level information about the HTTP protocol, and achieved good success against Tor[14]. Dyer, et al. performed a comprehensive evaluation of attacks and defenses, and developed their own attack, called VNG++, that achieved good success against many network-level defenses[6]. Cai, et al., proposed an attack, based on string edit distance, that performs well against a wide variety of defenses, included application-level defenses, such as HTTPOS and Tor's randomized pipelining[4]. Wang, et al. improved this attack's performance against Tor by incorporating information about the structure of the Tor protocol [20]. Danezis, Yu, et al., and Cai, et al., all proposed to use HMMs to extend web page fingerprinting attacks to web site fingerprinting attacks[5, 22, 4].

3. Congestion-Sensitive BuFLO

Dyer, et al., described BuFLO, a hypothetical defense scheme that hides all information about a website, except possibly its size, and performed a simulation-based evaluation that found that, although BuFLO is able to offer good security, it incurs a high cost to do so.

In this section, we describe Congestion-Sensitive BuFLO (CS-BuFLO), an extension to BuFLO that includes numerous security and efficiency improvements. CS-BuFLO represents a new approach to the design of fingerprinting defenses. Most previously-proposed defenses were designed in

response to known attacks, and therefore took a black-listing approach to information leaks, i.e. they tried to hide specific features, such as packet sizes. In designing CS-BuFLO, we take a white-listing approach – we start with a design that hides all traffic features, and iteratively refine the design to reveal certain traffic features that enable us to achieve significant performance improvements without significantly harming security.

3.1 Review of BuFLO

The Buffered Fixed-Length Obfuscator (BuFLO) of Dyer, et al., transmits a packet of size d bytes every ρ milliseconds, and continues doing so for at least τ milliseconds. If $b < d$ bytes of application data are available when a packet is to be sent, then the packet is padded with $d - b$ extra bytes of junk. The protocol assumes that the junk bytes are marked so that the receiver can discard them. If the website does not finish loading within τ milliseconds, then BuFLO continues transmitting until the website finishes loading and then stops immediately. Dyer, et al., did not specify how BuFLO detects when the website has finished loading. They also did not specify how BuFLO handles bidirectional communication – presumably independent BuFLO instances are run at each end-point. BuFLO effectively hides everything about the website, except possibly its size, but has several shortcomings:

- It either completely hides the size of the website or completely reveals it ($\pm d$ bytes). Thus it does not provide the same level of security to all websites.

- BuFLO has large overheads for small websites. Thus its overhead is also unevenly distributed.

- BuFLO is not TCP-friendly. In fact, it is the epitome of a bad network citizen. Further, BuFLO does not adapt when the user is visiting fast or slow websites. It wastes bandwidth when loading slow sites, and causes large latency when loading fast websites.

- BuFLO must be tuned to each user's network connection. If the BuFLO bandwidth, $\frac{1000d}{\rho}$ B/s, exceeds the user's connection speed, then BuFLO will incur additional delay without improving security.

- Past research by Fu, et al., showed that transmitting at fixed intervals can reveal load information at the sender, which an attacker can use to infer partial information about the data being transmitted[7].

Dyer, et al., proposed BuFLO as a straw-man defense system, so it is understandable that they did not bother addressing these problems. However, we show below that several of these problems have common solutions, e.g. we can simultaneously improve overhead and TCP-friendliness, simultaneously make security and overhead more uniform, etc. Thus, as our evaluation will show, CS-BuFLO may be a practical and efficient defense for users requiring a high level of security.

Further, as noted by its authors, BuFLO's simulation based results "reflect an ideal implementation that assumes the feasibility of implementing fixed packet timing intervals. This is at the very least difficult and clearly impossible for certain values of ρ. Simulation also ignores the complexities of cross-layer communication in the network stack" [6]. As a

result, it remains unclear how well the defense performs in the real world.

3.2 Overview of Congestion-Sensitive BuFLO

Algorithm 1 shows the main loop of the CS-BuFLO server. The client loop is similar, except for the few differences discussed throughout this section. Similar to BuFLO, CS-BuFLO delivers fixed-size chunks of data at semi-regular intervals. CS-BuFLO randomizes the timing of network writes in order to counter the attack of Fu, et al.[7], but it maintains a target average inter-packet time, ρ^*. CS-BuFLO periodically updates ρ^* to match its bandwidth to the rate of the sender (Section 3.3). Since updating ρ^* based on the sender's rate reveals information about the sender, CS-BuFLO performs these updates infrequently. CS-BuFLO uses TCP to be congestion friendly, and uses feedback from the TCP stack in order to reduce the amount of junk data it needs to send (Section 3.4). Also like BuFLO, CS-BuFLO transmits extra junk data after the website has finished loading in order to hide the total size of the website. However, CS-BuFLO uses a scale-independent padding scheme (Section 3.5) and monitors the state of the page loading process to avoid some unnecessary overheads (Section 3.6).
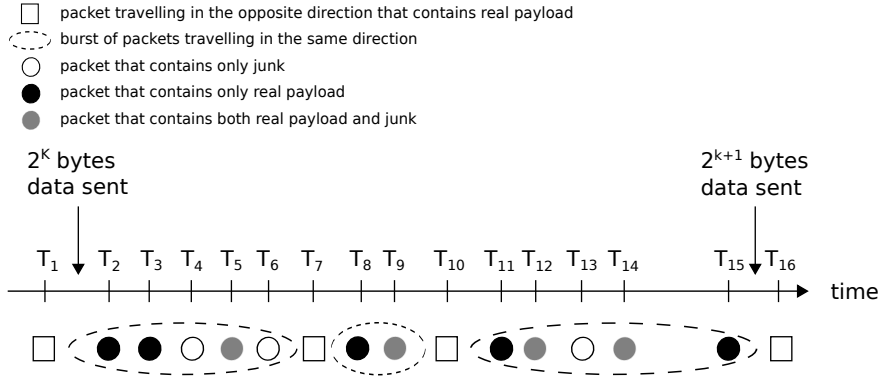
3.3 Rate Adaptation

CS-BuFLO adapts its transmission rate to match the rate of the sender. This reduces wasted bandwidth when proxying slow senders, and it reduces latency when proxying fast senders. However, adapting CS-BuFLO's transmission rate to match the sender's reveals information about the sender, and therefore may harm security.

As shown in Figure 1, CS-BuFLO takes several steps to limit the information that is leaked through rate adaptation. First, it only adapts after transmitting 2^k bytes, for some integer k. Thus, during a session in which CS-BuFLO transmits n bytes, CS-BuFLO will perform $\log_2 n$ rate adjustments, limiting the information leaked from these adjustments. This choice also allows CS-BuFLO to adapt more quickly during the beginning of a session, when the sender is likely to be performing a TCP slow start. During this phase, CS-BuFLO is able to ramp up its transmission rate just as quickly as the sender can.

CS-BuFLO further limits information leakage by using a robust statistic to update ρ^*. Between adjustments, it collects estimates of the sender's instantaneous bandwidth. It then sets ρ^* so as to match the sender's median instantaneous bandwidth. Median is a robust statistic, meaning that the new ρ^* value will not be strongly influenced by bandwidth bursts and lulls, and hence ρ^* will not reveal much about the sender's transmission pattern.

Note that the estimator only collects measurements during uninterrupted bursts from the sender. This ensures that the bandwidth measurements do not include delays caused by dependencies between requests and responses. For example, if the estimator sees a packet p_1 from the website, then a packet p_2 from the client, and then another packet p_3 from the website, it may be the case that p_3 is a response to p_2. In this case, the time between p_1 and p_3 is constrained by the round trip time, not the website's bandwidth.

Finally, CS-BuFLO rounds all ρ^* values up to a power of two. This further hides information about the sender's true rate, and gives the sender room to increase it's transmission rate, e.g. during slow start.

packet travelling in the opposite direction that contains real payload

burst of packets travelling in the same direction

packet that contains only junk

packet that contains only real payload

packet that contains both real payload and junk

2^K bytes data sent

2^{k+1} bytes data sent

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10} T_{11} T_{12} T_{13} T_{14} T_{15} T_{16}

time

Figure 1: Rate adaptation in CS-BuFLO. ρ^* **is updated based on the packets transmitted to the other end between** T_2 **and** T_{15}**. Time intervals between two consecutive packets are stored in an array** $Intervals[]$**. The two packets under consideration both contain some real payload data and they belong to the same burst. i.e.** $Intervals = [T_3 - T_2, T_5 - T_3, T_9 - T_8, T_{12} - T_{11}, T_{14} - T_{12}, T_{15} - T_{14}]$ **and** $\rho^* = 2^{\lfloor \log_2 Median(Intervals[]) \rfloor}$.

Algorithm 1 The main loop of the Congestion-Sensitive BuFLO server.

function CSBUFLO-SERVER(s)
   while true **do**
      (m, ρ) = READ-MESSAGE(ρ)
      if m is application data from website **then**
         $output\text{-}buff \leftarrow output\text{-}buff \parallel data$
         $real\text{-}bytes \leftarrow real\text{-}bytes +$ LENGTH(m)
         $last\text{-}site\text{-}response\text{-}time \leftarrow$ CURRENT-TIME
      else if m is application data from client **then**
         send m to the website
         $\rho\text{-}stats \leftarrow \rho\text{-}stats \parallel \bot$
         $onLoadEvent \leftarrow 0,\ padding\text{-}done \leftarrow 0$
      else if m is onLoad message **then**
         $onLoadEvent \leftarrow 1$
      else if m is padding-done message **then**
         $padding\text{-}done \leftarrow 1$
      else if m is a time-out **then**
         if $output\text{-}buff$ is not empty **then**
            $\rho\text{-}stats \leftarrow \rho\text{-}stats \parallel$ CURRENT-TIME
         end if
         $(output\text{-}buff, j) \leftarrow$ CS-SEND(s, $output\text{-}buff$)
         $junk\text{-}bytes \leftarrow junk\text{-}bytes + j$
      end if
      if DONE-XMITTING **then**
         reset all variables
      else   ▷ ρ^* : Average time between sends to client
         if $\rho^* = \infty$ **then**
            $\rho^* \leftarrow$ INITIAL-RHO
         else if CROSSED-THRESHOLD($real\text{-}bytes$, $junk\text{-}bytes$) **then**
            $\rho^* \leftarrow$ RHO-ESTIMATOR($\rho\text{-}stats$, ρ^*)
            $\rho\text{-}stats \leftarrow \emptyset$
         end if

         if m is a time-out **then**
            $\rho \leftarrow$ random number in $[0, 2\rho^*]$
         end if
      end if
   end while
end function

Algorithm 2 Algorithm for estimating new value of ρ^* based on past network performance.

function RHO-ESTIMATOR($\rho\text{-}stats$, ρ^*)
   $I \leftarrow [\rho\text{-}stats_{i+1} - \rho\text{-}stats_i \mid \rho\text{-}stats_i \wedge \rho\text{-}stats_{i+1} \neq \bot]$
   if I is empty list **then**
      return ρ^*
   else
      return $2^{\lfloor \log_2 \text{MEDIAN}(I) \rfloor}$
   end if
end function

3.4 Congestion-Sensitivity

There's a trivial way to make BuFLO congestion sensitive and TCP friendly: run the protocol over TCP. With this approach, we grab an additional opportunity for increasing efficiency: when the network is congested, CS-BuFLO does not need to insert junk data to fill the output buffer.

Algorithm 3 shows our method for taking advantage of congestion to reduce the amount of junk data sent by CS-BuFLO. Note first that CS-SEND always writes exactly d bytes to the TCP socket. Since the amount of data presented to the TCP socket is always the same, this algorithm reveals no information about the timing or size of application-data packets from the website that have arrived at the CS-BuFLO proxy.

This algorithm takes advantage of congestion to reduce the amount of junk data it sends. To see why, imagine the TCP connection to the client stalls for an extended period of time. Eventually, the kernel's TCP send queue for socket s will fill up, and the call to `write` will return 0. From then until the TCP congestion clears up, CS-BuFLO calls to CS-SEND will not append any further junk data to B.

3.5 Stream Padding

CS-BuFLO hides the total size of real data transmitted by continuing to transmit extra junk data after the browser and web server have stopped transmitting.

Table 2 shows two related padding schemes we experimented with in CS-BuFLO. Both schemes introduce at most a constant factor of additional cost, but reveal at most a logarithmic amount of information about the size of the web-

Algorithm 3 Algorithm for sending data and using feedback from TCP. Socket s should be configured with `O_NONBLOCK`.

> **function** CS-SEND(s, *output-buff*)
> $n \leftarrow$ LENGTH(*output-buff*)
> $j \leftarrow 0$
>   **if** $n <$ PACKET-SIZE **then**
> $j \leftarrow$ PACKET-SIZE $- n$
>   *output-buff* $\leftarrow$ *output-buff* $\| j$
>   **end if**
> $r \leftarrow$ `write`(s, *output-buff*, PACKET-SIZE)
>   **if** $r \geq n$ **then**   ▷ Optional: reclaim unsent junk
>   *output-buff* $\leftarrow$ empty buffer
> $j \leftarrow r - n$
>   **else**
>   remove last j bytes from *output-buff*
>   remove first r bytes from *output-buff*
> $j \leftarrow 0$
>   **end if**
>   **return** (*output-buff*, j)
> **end function**

Padding Schemes	Payload Before	Junk Before	Total Bytes After
payload padding	R	J	$c2^{\lceil \log_2 R \rceil}$
total padding	R	J	$2^{\lceil \log_2(R+J) \rceil}$

Table 2: Two different padding schemes for CS-BuFLO.

site. The first scheme, which we call *payload* padding, continues transmitting until the total amount of transmitted data $(R + J)$ is a multiple of $2^{\lceil \log_2 R \rceil}$. This padding scheme will transmit at most $2^{\lceil \log_2 R \rceil}$ additional bytes, so it increases the cost by at most a factor of 2, but it reveals only $\log_2 R$.

The second scheme, which we call *total* padding, continues transmitting until $R + J$ is a power of 2. This also increases the cost by at most a factor of 2 and reveals, in the worst case, $\log_2 R$, but it will in practice hide more information about R than payload padding.

Note that the CS-BuFLO server and the CS-BuFLO client do not have to use the same stream padding scheme. Thus, there are four possible padding configurations, which we denote CPSP (client payload, server payload), CPST (client payload, server total), CTSP (client total, server payload) and CTST (client total, server total).

In order to determine when to stop padding, the CS-BuFLO server must know when the website has finished transmitting. Congestion-Sensitive BuFLO uses two mechanisms to recognize that the page has finished loading. First, the CS-BuFLO client proxy monitors for the browser's onLoad event. The CS-BuFLO client notifies the CS-BuFLO server when it receives the onLoad event from the browser. Once the CS-BuFLO server receives the onLoad message from the client, it considers the web server to be idle (see Algorithm 4) and will stop transmitting as soon as it adds sufficient stream padding and empties its transmit buffer. As a backup mechanism, the CS-BuFLO server considers

Algorithm 4 Definition of the DONE-XMITTING function.

> **function** DONE-XMITTING
>   **return**   LENGTH(*output-buff*) $\leftarrow$ 0 $\wedge$CHANNEL-IDLE(*onLoadEvent,last-site-response-time*)$\wedge$ (*padding-done*
> $\vee$ CROSSED-THRESHOLD(*real-bytes* + *junk-bytes*))
> **end function**
>
> **function**   CHANNEL-IDLE(*onLoadEvent, last-site-response-time*)
>   **return** *onLoadEvent* $\vee$ (*last-site-response-time* + QUIET-TIME $<$ CURRENT-TIME)
> **end function**
>
> **function** CROSSED-THRESHOLD(x)
>   **return** $\lfloor \log_2(x -$ PACKET-SIZE$) \rfloor < \lfloor \log_2 x \rfloor$
> **end function**

the website idle if QUIET-TIME seconds pass without receiving new data from the website. We used a QUIET-TIME of 2 seconds in our prototype implementation.
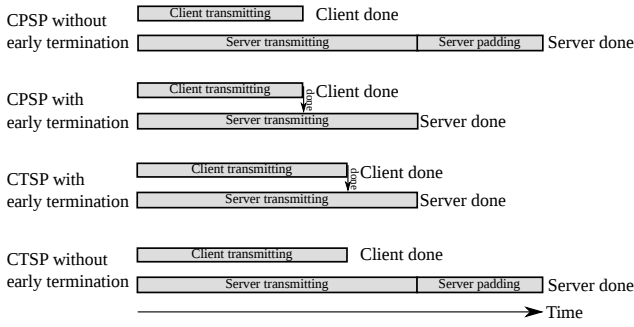
3.6 Early Termination

As described above, the CS-BuFLO server is likely to finish each page load by sending a relatively long tail of pure junk packets. This tail can be a significant source of overhead and, somewhat surprisingly, may not provide much additional security.

Our initial investigations revealed that the long tail served two purposes which could also be served through other, more efficient means. As mentioned above, the long tail helps hide the total size of the website. However, the interior padding performed by CS-SEND also obscures the total size of the website. Our evaluation in Section 5 investigates the security impact of additional stream padding.

In the specific context of web browsing, the long tail also hides the size of the last object sent from the web server to the client. The attacker can infer some information about the size of this object by measuring the amount of data the CS-BuFLO server sends to the CS-BuFLO client after the CS-BuFLO client stops transmitting to the CS-BuFLO server. However, this information can also be hidden by having the CS-BuFLO client continue to send junk packets to the CS-BuFLO server, i.e. more aggressive stream padding from the CS-BuFLO client may obviate the need for aggressive padding at the CS-BuFLO server.

Based on these ideas, we implemented an *early termination* feature in our CS-BuFLO prototype. The CS-BuFLO client notifies the CS-BuFLO server that it is done padding. After receiving this message, the CS-BuFLO server will stop transmitting as soon as the web server becomes idle and its buffers are empty.

Figure 2 illustrates how the padding scheme used by the client and server can interact, including the impact of early termination. Additional client padding can hide the size of the last HTTP object, and early termination can avoid unnecessary padding. Our evaluation investigates the security/efficiency trade-offs between different padding regimes at the client and server, and how they interact with early termination.

Figure 2: The interaction between client and server padding schemes and early termination. More padding at the client can help hide the size of the last object sent from the server to the client. Early termination can avoid unnecessary padding at the end of a page load.

3.7 Packet Sizes

Sending fixed-length packets hides packet size information from the attacker. Although any fixed length should work, it is important to choose a packet length that maximizes performance. Since we may transmit pure junk packets during the transmission, larger packets tend to cause higher bandwidth overhead, and on the other hand, smaller packets may not make full use of the link between the client and server, thus increase the loading time.

Preliminary investigations revealed that over 95.7% of all upstream packet transmissions are under 600 bytes, therefore, this was used as the standard packet size in our experiments.

4. PROTOTYPE IMPLEMENTATION

We modified OpenSSH5.9p1 to implement Algorithm 1. However, the optional junk recovery algorithm described in Algorithm 3 was not implemented.

The SSH client was also modified to accept a new SOCKS proxy command code, *onLoadCmd*. This command was used to communicate to the server when to stop padding (as described in Section 3.5). A Firefox plugin, *Onload-Notify*, that, upon detecting the page *onLoad* event, connects to the SSH client's SOCKS port and issues the *onLoadCmd*, was also developed. In addition, the following OpenSSH message types were used: (1) The OpenSSH message type `SSH_MSG_IGNORE`, which means all payload in a packet of this type can be ignored, was used to insert junk data whenever needed. (2) The `SSH_MSG_NOTIFY_ONLOAD` message was created to be used by the client to communicate reception of *onLoadCmd* from the browser, to the server. Upon receiving this message from the client, the CS-BuFLO server stops transmitting as soon as it empties its buffer and adds sufficient stream padding. (3) The `SSH_MSG_NOTIFY_PADDINGDONE` message was created to implement the early termination feature of CS-BuFLO. Upon receiving this message from the client, the CS-BuFLO server stops transmitting as soon as the web server becomes idle and its buffers are empty.

All the above messages were buffered and transmitted just like other messages in Algorithm 1, i.e. using CS-SEND,

therefore an attacker is unable distinguish these messages from other traffic.

5. EVALUATION

We investigated several questions during our evaluation: (1) How do the different stream padding schemes affect performance and security of CS-BuFLO? What is the effect of adding early termination to the protocol? (2) How does CS-BuFLO's security and overhead compare to Tor's, and how do they both compare to the theoretical minimums derived in [3]? (3) Can we use the theoretical lower bounds to enable us to compare defenses that have different security/overhead trade-offs?

5.1 Experimental Setup

For our main experiments, we collected traffic from the Alexa top 200 functioning, non-redirecting web pages using four different defenses: plain SSH, Tor, CS-BuFLO with the CTSP padding and early termination, and CS-BuFLO with CPSP padding and early termination. We also collected several smaller data sets using other configurations of CS-BuFLO, but these are only used in the padding scheme evaluations (Table 3).

We constructed a list of the Alexa top 200 functioning, non-redirecting, unique pages, as follows. We removed web pages that failed to load in Firefox (without Tor or any other proxy). We replaced URLs that redirected the browser to another URL with their redirect target. Some websites display different languages and contents depending on where the page is loaded, e.g. *www.google.com* and *www.google.de*. We kept only one URL for this type of website, i.e. we only had *www.google.com* in our set. Our data set consisted of Alexa's 200 highest-ranked pages that met these criteria.

We collected 20 traces of each URL, clearing the browser cache between each page load. We collected traces from each web page in a round-robin fashion. As a result, each load of the same URL occurred about 5 hours apart.

Measuring the precise latency of a fingerprinting defense scheme poses a challenge: we can easily measure the time it takes to load a page using the defense, but we cannot infer the exact time it *would have taken* to load the page without the defense. Therefore, every time we loaded a page using a defense, we immediately loaded it again using SSH to get an estimate of the time it would have taken to load the page without the defense in place. We then compute latency ratios the same way we compute bandwidth ratios, i.e. if $L(t)$ is the total duration of a packet trace, the latency ratio of a defense scheme is $\frac{E\left[L(T_W^D)\right]}{E[L(T_W)]}$.

We collected network traffic using several different computers with slightly different versions of Ubuntu Linux – ranging from 9.10 to 11.10. We used Firefox 3.6.23-3.6.24 and Tor 0.2.1.30 with polipo HTTP Proxy. All Firefox plugins were disabled during data collection, except when collecting CS-BuFLO traffic, where we enabled the *OnloadNotify* plugin. Three of the computers had 2.8GHz Intel Pentium CPUs and 2GB of RAM, one computer had a 2.4GHz Intel Core 2 Duo CPU with 2GB of RAM. We scripted Firefox using Ruby and captured packets using tshark, the command-line version of wireshark. For the SSH experiments, we used OpenSSH5.3p1. Our Tor clients used the default configuration. SSH tunnels passed between two machines on the same local network.

Padding	Early Termination	Bandwidth Ratio	Latency Ratio	VNG++ Accuracy
CTSP	Yes	3.59	3.91	29.0%
CTSP	No	3.73	3.51	29.6%
CPSP	Yes	2.60	2.87	34.2%
CPSP	No	3.42	3.52	36.0%

Table 3: Security and performance of Congestion-Sensitive BuFLO variants. VNG++ success rate is the probability that the attack was able to correctly guess which of 50 web pages the user was visiting.

Figure 4: Bandwidth ratios of various defense schemes as a function of the number of possible web pages.

We measured the security of each defense by using the three best traffic analysis attacks in the literature: VNG++ [6], the Panchenko SVM [14], and DLSVM [4]. We ran each of the above classifiers against the traces generated by each defense using stratified 10-fold cross validation.

5.2 Results

Padding Schemes: Table 3 shows the bandwidth ratio, latency ratio, and security (estimated using the VNG++ attack) of four different versions of CS-BuFLO on a data set of 50 websites. Note that early termination does not appear to affect security, although it can significantly reduce overhead in some configurations. All other experiments in this paper use early termination. The client padding scheme, on the other hand, appears to control a trade-off between security and overhead. Therefore we report the rest of our results for both CPSP and CTSP padding.

Security Comparison: Figure 3 shows the level of security various defense schemes provide against three different attacks, as the number of web pages the attacker needs to distinguish increases. Note that the CS-BuFLO schemes have significantly better security than Tor and SSH. For each defense scheme, we compute its average bandwidth ratio, BO, and plot the lower bound on security that can be achieved within that ratio, using the algorithm from [3].

Bandwidth Cost: Figure 4 plots the bandwidth ratios of SSH, Tor, and CS-BuFLO with CTSP and CPSP padding. SSH has almost no overhead, and Tor's overhead is about 25% on average. CS-BuFLO with CPSP has an average overhead of 129%, CTSP has average overhead 180%. Thus CS-BuFLO's improved security does come at a price.

Comparisons with Theoretical Bounds: Figure 5 evaluates CS-BuFLO, Tor, SSH, and BuFLO against the theoretical lower bounds [3]. Figure 5(a) shows the results of our empirical evaluation of CS-BuFLO, Tor, and SSH on $n = 120$ sites and using the DLSVM attack to estimate security. We limit to 120 sites to make it easier to compare with the BuFLO results reported by Dyer, et al., and which use 128 sites. There is a significant gap between the bandwidth of CS-BuFLO and the lower bound. However, as can be seen in Figure 5(c), CS-BuFLO in CTSP mode is over $6\times$ closer to the trade-off lower bound than Tor for 200 sites, and is the most efficient scheme across all sizes we measured.

Figure 5(b) presents the results of our empirical evaluation of CS-BuFLO, Tor, and SSH on $n = 120$ websites, using the Panchenko attack to estimate security. We also present Dyer's reported results from their experiments with BuFLO on 128 sites, also using the Panchenko attack. Note
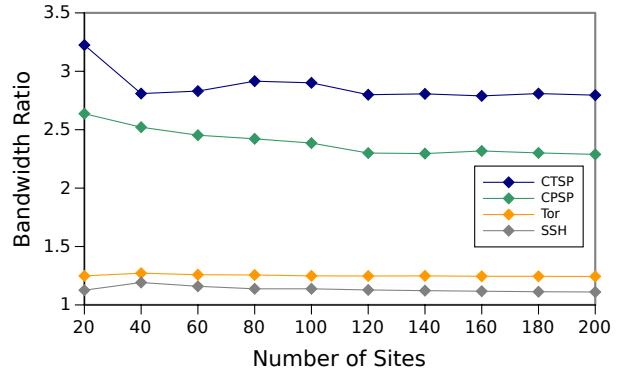
that, since Dyer used 128 sites to evaluate BuFLO, this slightly over-estimates BuFLO's security compared to the other schemes plotted in the figure. Also, recall that Dyer's experiments with BuFLO were all based on simulation. Despite the differences in experimental methodology, we can see that CS-BuFLO offers performance in the same general range as the BuFLO configurations from Dyer's paper, but has slightly worse security in our experiments.

Figure 5(d) shows that, based on our experiments and the simulation results of Dyer, et al., all but one BuFLO configuration get closer to the trade-off lower bound curve than CS-BuFLO, Tor, and SSH (SSH is omitted from the graph because its ratio to the lower bound was never less than 400). This figure also highlights a difference between the DLSVM and Panchenko attacks. In the DLSVM results shown in Figure 5(c), Tor and SSH diverge from CS-BuFLO. In the Panchenko results in Figure 5(d), Tor and CS-BuFLO appear to be equally close to the lower bound.

6. DISCUSSION AND CONCLUSIONS

Since early termination does not appear to affect security, the padding results suggest that the padding performed while transmitting a website sufficiently hides the size of the website, so that additional stream padding at the end of the transmission has little security benefit. Additional client padding does improve security, though – probably by obscuring the size of the final object requested by the client.

Overall, CS-BuFLO has better security than any other defense in our experiments, albeit at greater expense. It has the best security/overhead trade-off, as well. CS-BuFLO's security/overhead trade-off is in the same range as the estimates Dyer obtained for BuFLO in their simulations. For example, Dyer, et al., reported that, in one configuration of BuFLO, bandwidth overhead was 200% and the Panchenko SVM had an 24.1% success rate on 128 websites. We found that CS-BuFLO with CTSP padding had an overhead of 180% on 120 websites, and that the Panchenko SVM had a success rate of 23.4%.

CS-BuFLO's congestion-sensitivity likely had little impact in these experiments, which were carried out on a fast local network, so that congestion was rare. However, CS-BuFLO's congestion-sensitivity means that, in a real deployment, it would have even better bandwidth overhead. CS-BuFLO's latency overhead is approximately 3 in all our
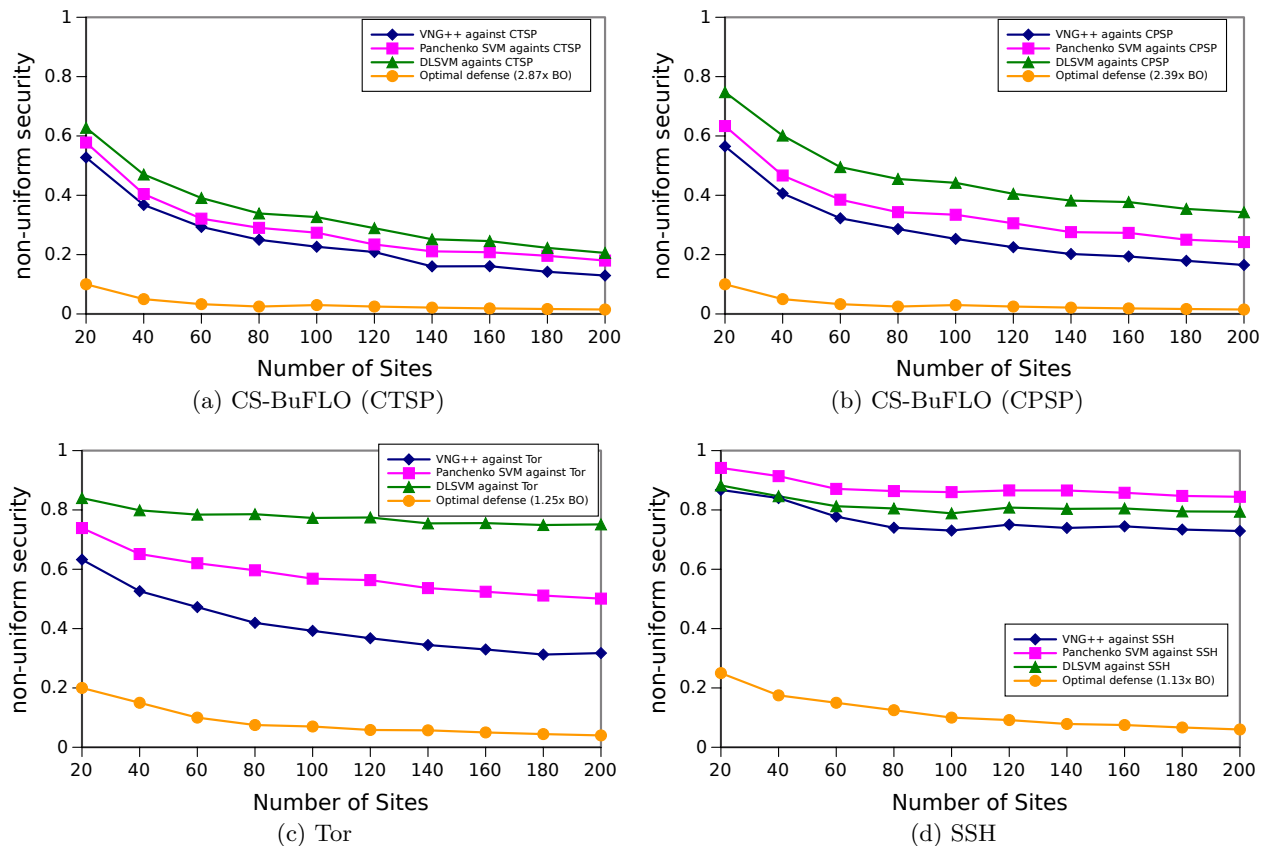
Figure 3: Security of CS-BuFLO, Tor, and SSH compared to the lower bounds defined in [3], as a function of the number of possible web pages.

experiments. This is better than Tor's latency, although Tor has the additional overhead of onion routing, so no fair comparison is possible. We cannot compare with the latency estimates reported by Dyer, et al., because they gave only absolute latency values.

CS-BuFLO offers a real world implementation of a high-security, moderate-overhead solution to website fingerprinting attacks. Compared to SSH and Tor, it achieves a better security/bandwidth trade-off, i.e. it uses its bandwidth efficiently to provide extra security. Our experiments also show that it has acceptable latencies. The padding schemes developed in this paper, along with browser-coordination and early-termination algorithm, can improve security with less overhead than previous stream padding schemes. Interestingly, we also found that padding from one end of a connection can sometimes be an efficient way to hide information about the data sent from the other side of the connection.
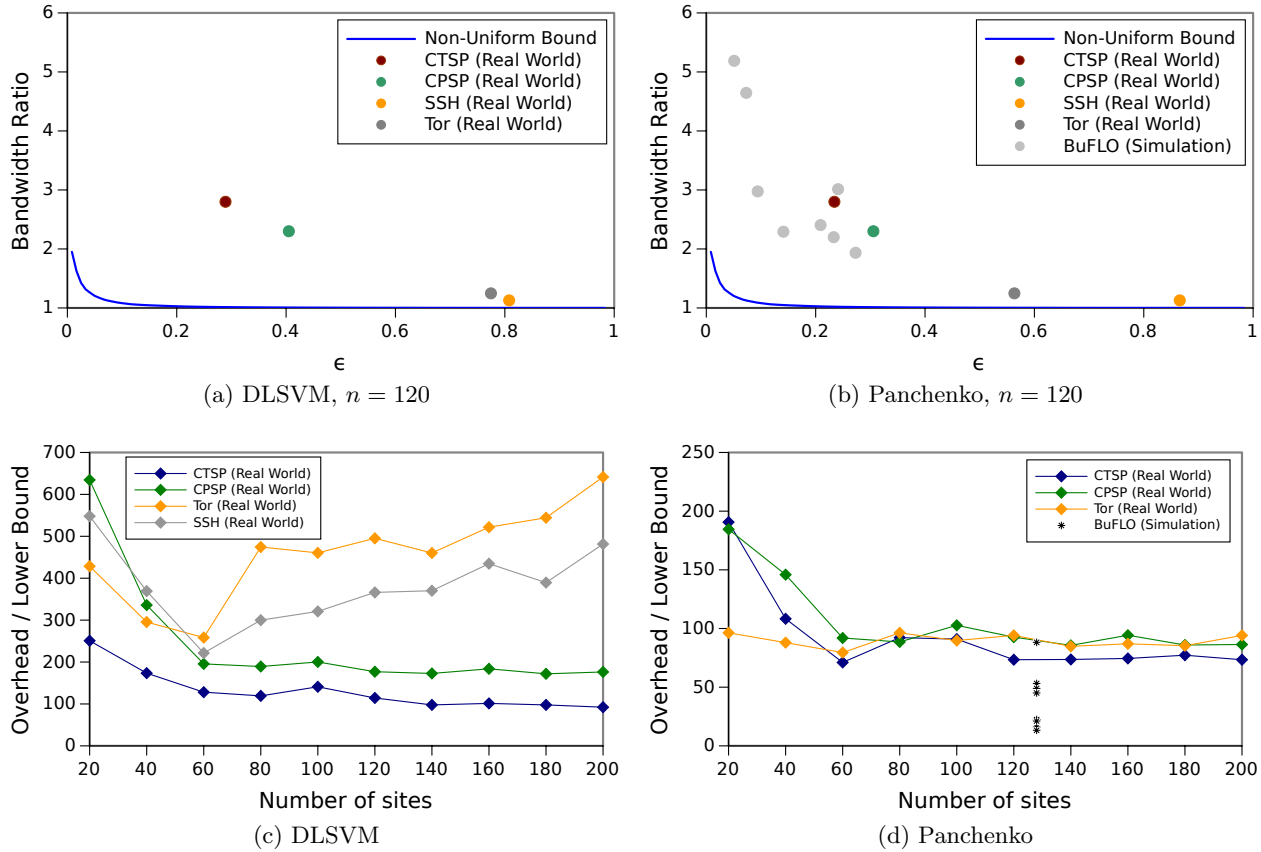
To ease further comparative evaluation of fingerprinting defenses, a fully working implementation of CS-BuFLO along with traces used in our experiments are available[1].

7. REFERENCES

[1] Amazon web services - alexa top sites, October 2013.
[2] George Bissias, Marc Liberatore, David Jensen, and Brian Levine. Privacy vulnerabilities in encrypted http streams. In *PETS*, 2006.
[3] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *ACM CCS*, 2014.
[4] Xiang Cai, Xincheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: website fingerprinting attacks and defenses. In *ACM CCS*, 2012.
[5] George Danezis. Traffic analysis of the HTTP protocol over TLS.
[6] 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 *IEEE Security and Privacy*, 2012.
[7] X. Fu, B. Graham, R. Bettati, and W. Zhao. On countermeasures to traffic analysis attacks. In *Information Assurance Workshop*, 2003.
[8] Xun Gong, Negar Kiyavash, and Nikita Borisov. Fingerprinting websites using remote traffic analysis. In *ACM CCS*, 2010.
[9] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In *ACM Workshop on Cloud Computing Security*, 2009.
[10] Andrew Hintz. Fingerprinting websites using traffic analysis. In *PETS*, 2002.

[1] at https://github.com/xiang-cai/CSBuFLO

(a) DLSVM, $n = 120$

(b) Panchenko, $n = 120$

(c) DLSVM

(d) Panchenko

Figure 5: **Non-uniform lower bounds on bandwidth ratio, as a function of the security parameter, ϵ, and specific trade-off points of the systems evaluated. The BuFLO results are taken from Dyer, et al. [6], and therefore use $n = 128$. SSH is omitted from Figure 5(d) because its ratio to the lower bound was always greater than 400.**

[11] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *ACM CCS*, 2006.

[12] Liming Lu, Ee-Chien Chang, and Mun Chan. Website fingerprinting and identification using ordered feature sequences. In *ESORICS*, 2010.

[13] 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 *NDSS*, 2011.

[14] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *WPES*, 2011.

[15] Mike Perry. Experimental defense for website traffic fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, September 2011.

[16] Mike Perry. A critique of website traffic fingerprinting attacks, November 2013.

[17] Mike Perry, Erinn Clark, and Steven Murdoch. The design and implementation of the tor browser [draft], March 2013.

[18] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Security and Privacy*, 2002.

[19] Tor project: Anonymity online, August 2011.

[20] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Workshop on Privacy in the Electronic Society*, 2013.

[21] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.

[22] Shui Yu, Wanlei Zhou, Weijia Jia, and Jiankun Hu. Attacking anonymous web browsing at local area networks through browsing dynamics. *The Computer Journal*, 2011.

[23] Fan Zhang, Wenbo He, Xue Liu, and Patrick G. Bridges. Inferring users' online activities through traffic analysis. In *ACM Conference on Wireless Network Security*, 2011.