An Efficient Web Traffic Defence Against Timing-Analysis Attacks

Saman Feghhi and Douglas J. Leith
Trinity College Dublin, Ireland

Abstract—We introduce a new class of lower overhead tunnel that is resistant to traffic analysis. The tunnel opportunistically reduces the number of dummy packets transmitted during busy times when many flows are simultaneously active while maintaining well-defined privacy properties. We find that the dummy packet overhead is typically less than 20% on lightly loaded links and falls to zero as the traffic load increases i.e. the tunnel is capacity-achieving. The additional latency incurred is less than 100ms. We build an experimental prototype of the tunnel and carry out an extensive performance evaluation that demonstrates its effectiveness under a range of network conditions and real web page fetches.

Index Terms—Timing-analysis attacks, Traffic analysis defence, Website fingerprinting, Network privacy, VPN.

I. INTRODUCTION

Encrypting traffic to protect it from eavesdroppers is one of the basic building blocks of secure networks. However, while encryption conceals the contents of transmitted packets other features of the transmitted packet stream, such as the packet sizes and timings, often still contain revealing information. Taking advantage of this, over the last 10 years a number of increasingly powerful attacks have been demonstrated against encrypted packet streams, and encrypted web traffic in particular. While attacks against HTTPS were initially based on packet sizes and counts, attacks using only packet timing information have recently been demonstrated, e.g. [7]. The importance of the latter is that they are immune to defences currently being rolled out, such as padding packets to make them all the same size, plus they do not require a priori knowledge of the start and end of each web fetch. Such developments motivate revisiting how we transmit encrypted traffic.

Three main approaches to defence have been considered to date. One is to obfuscate the timing of information packets by transmitting packets at a constant rate, using buffering and insertion of dummy packets as needed [5], [4], [3]. While effective at disrupting attacks, this is recognised to involve such high overheads as to be unsuitable for general use. A second, application-layer, approach is to randomise the pipelining of the requests forming the fetch a web page and also to inject dummy requests [15]. However, while incurring a low overhead (few dummy packets and low latency) this is largely ineffective against modern attacks [2], [21], [22]. A third line of work is based on shaping traffic so that the transmitted trace is similar, in some appropriate sense,

to that for a different web page or application [23], [17], [21], [22]. However, either these approaches are of limited effectiveness [2], [21], [7] or again entail high overheads [17], [21], [22]. In summary, existing defences are either of limited ineffectiveness or carry an excessive overhead.

In this paper we introduce a new class of lower overhead tunnel that is resistant to traffic analysis. The basic idea used is to ensure that, given an observed sequence packet trace, many different sequences of web fetches could reasonably generate this trace. Users are therefore provided strong deniability that a specific web page was fetched. This indistinguishability idea is not new, but the approach used to realise it is novel. Important features are that (i) our approach is lightweight and does not, for example, require pre-clustering of web pages (and so maintenance of a packet trace database etc) and (ii) packets from multiple web fetches/flows are aggregated thereby allowing us to opportunistically reduce the number of dummy packets transmitted during busy times when many flows are simultaneously active while maintaining well-defined privacy properties. We find that the dummy packet overhead is typically less than 20% on lightly loaded links. Further, the fraction of dummy packets falls to zero as the traffic load increases i.e. the tunnel is capacity-achieving. The additional latency introduced by the tunnel is less than 100ms. We build an experimental prototype of the tunnel and carry out an extensive performance evaluation that demonstrates its effectiveness under a range of network conditions and real web page fetches.

II. PRELIMINARIES

A. Threat Model

The privacy disclosure scenario we consider arises naturally from the prevalence of relatively easy access to the network conditions and real web page fetches.
path connecting a client device to the internet. We imagine
two main types of attack/threat:

1) The first hop is a WiFi link and a nearby device equipped
with a wireless interface sniffs packets transmitted across
the wireless hop.
2) ISPs and hot spot providers sniff the packets carried on
the DSL/cable/ethernet link connecting the edge network
to the internet.

Notice that the sensitive asset here is the packet trace, consist-
ing of the time when a packet is transmitted together with the
packet header and contents. We assume that packet contents
are encrypted and packets are padded to be of the same size,
that is the user already has baseline defences in place by
routing traffic via a suitable tunnel as illustrated schematically
in Figure 1. The relevant part of the packet trace for an attacker
therefore consists of the timing of packet transmissions.

We focus on web traffic, since this a major source of
traffic in the internet and has been the subject of much of the
literature on traffic analysis attacks. We do not seek to conceal
the fact that the client device is browsing the web, but rather
to prevent an attacker from inferring which pages are being
browsed. Packet timing data is a rich source of information,
as discussed in more detail in Section II-B below, and in par-
icular it is known to be sufficient to allow an attacker
to infer with high probability the web page being browsed
by a user [7] while in [5] it is shown that packet padding is
not sufficient to hide coarse grained features such as bursts in
traffic or the total size and load time of a page.

Although a client device may transmit packets from multiple
flows, e.g. web and email, across its internet link, we want
to avoid making strong assumptions about the number and
characteristics of flows since this can change over time in
ways that are difficult to predict i.e. we require a defence that
provides a reasonable level of protection whether a single web
fetch is in progress or many parallel flows are active. That said,
we expect that flow re-identification attacks may sometimes be
harder to carry out when many flows are active simultaneously
and of course a defence can and should opportunistically take
advantage of this e.g. to reduce the overheads induced by the
defence.

Note that we do not seek to address attacks that target the
end host itself (viruses etc), nor attacks based on active packet
injection by the attacker (e.g. to force packet loss so as to
manipulate user or web server behaviour).

B. Anatomy of a Web Page Fetch

We briefly review the reasons why packet timing infor-
mation can be highly informative. When traffic is encrypted
the packet source and destination addresses and ports and the
packet payload are hidden and the packets may be padded to be
of equal size, so that packet size information is also concealed.
An attacker sniffing such encrypted traffic is therefore able
only to observe the direction and timing of packets. Figure
2 plots the timestamps of the uplink packets sent during
the course of fetching five different health-related web pages
(see below for details of the measurement setup). The $x$-axis
indicates the packet number $k$ within the stream and the $y$-
axis the corresponding timestamp. It can be seen that these
timestamp traces are distinctly different for each web site.

To gain insight into the differences between the packet
timestamp sequences in Figure 2 it is helpful to consider the
process of fetching a web page in more detail. To fetch a web
page the client browser starts by opening a TCP connection
with the server indicated by the URL and issues an HTTP GET
or POST request to which the server then replies. As the client
parses the server response it issues additional GET/POST
requests to fetch embedded objects (images, css, scripts etc.).
These additional requests may be to different servers from the
original request (e.g. when the object to be fetched is an
advert or is hosted in a separate content-delivery network),
in which case the client opens a TCP connection to each
new server in order to issue the requests. Fetching of these
objects may in turn trigger the fetching of further objects.
Note that asynchronous fetching of dynamic content using,
e.g. AJAX, can lead to a complex sequence of server requests
and responses even after the page has been rendered by the
browser. Also, typically the TCP connections to the various
servers are held open until the page is fully loaded so that
they can be reused for later requests (request pipelining in
this way is almost universally used by modern browsers). It
is this sequence of events while fetching the objects in a page
that creates a packet timing “signature” which can be used to
deanonimise the encrypted traffic and estimate the web page
fetched with high probability. For example, in [7] web pages
are correctly identified with $>90\%$ success rate.

C. Related Work

1) Attacks: With regard to attacks, a fairly large body of
literature now exists. Some of the earliest work specifi-
cally focussed on attacks against encrypted web traffic appears to
be that of Hintz [11]. In this setup (i) web page fetches occur
sequentially with the start and end of each web page fetch
known, and for each packet (ii) the client-side port number,
(iii) the direction (incoming/outgoing) and (iv) the size is observed. A web page signature is constructed consisting of the aggregate bytes received on each port (calculated by summing packet sizes), effectively corresponding to the number and size of each object within the web page.

Subsequently, Bissias et al. [1] considered an encrypted tunnel setup where (i) web page fetches occur sequentially with the start and end of each web page fetch known, and for each packet (ii) the size, (iii) the direction (incoming/outgoing) and (iv) the time (and so also the packet ordering) is observed. The sequence of packet inter-arrival times and packet sizes from a web page fetch is used to create a profile for each web page in a target set and the cross correlation between an observed traffic sequence and the stored profiles is then used as a measure of similarity.

Most later work has adopted essentially the same model as [1], making use of packet direction and size information and assuming that the packet stream has already been partitioned into individual web page fetches. For example in [20] the timing information is not considered in the feature set, hence the attack can be countered with defences such as BuFLO in [5] leading to a success rate of only 10%. In [13, 10] Bayes classifiers based on the direction and size of packets are considered while in [18] an SVM classifier is proposed. In [14] classification based on direction and size of packets is studied using Levenshtein distance as the similarity metric, in [16] using a Gaussian Bag-of-Words approach and in [20] using K-NN classification. In [2] using a SVM approach a classification accuracy of over 80% is reported for both SSH and Tor traffic and the defences considered were generally found to be ineffective. Similarly, [5] considers Bayes and SVM classifiers and finds that a range of proposed defences are ineffective. In [9] remote inference of packet sizes from queuing delay is studied.

Recently, powerful attacks based on using packet timing information alone have been demonstrated [7]. Such attacks are immune to defences based on packet padding etc, do not require a priori knowledge of the start/end of a web fetch and have low false positive rates in open-world settings where the set of web sites being browsed is not restricted to a defined set.

2) Defences: With regard to defences, one fairly obvious approach to obfuscating the timing of information packets is to transmit packets at a constant rate, inserting dummy packets when an information packet is not available to send (when encrypted dummy packers are indistinguishable from information packets to an attacker), and buffering information packets when a transmit slot is not available. This type of strategy is considered in detail by [5], although to reduce the bandwidth overhead transmission only continues until either a page has loaded or a minimum time has elapsed. It is concluded that this approach, referred to as BuFLO (Buffered Fixed-Length Obfuscator), involves excessive bandwidth overhead (>100%) and high latency (a ×2-3 increase) in return for uncertain security (the duration of pages longer than the minimum time is still revealed). Tamaraw [4] and CS_BuFLO [3] both attempt to refine BuFLO. Tamaraw ends transmission only at fixed multiples of a design parameter L which allows for stronger security guarantees at the cost of increased overhead. CS_BuFLO adjusts the transmit rate based on traffic load.

In view of the high overheads associated with constant rate transmission, a number of alternative approaches have been considered. In HTTPOS [15] and the work of Wright et al. [23] packet timing is modified by injecting dummy HTTP requests and randomising the pipelining of the requests forming the fetch a web page. Randomised pipelining is also used in Tor. However, this defence has been shown to be of limited effectiveness at protecting against attacks in several subsequent evaluations [2, 21, 12]. Another line of work is based on shaping traffic so that the transmitted trace is similar, in some appropriate sense, to that for a different web page or application. In early work [23] traffic morphing is considered whereby packet padding/splitting is used so that the distribution of packet sizes over a web fetch resembles that of a target web page. However this was found to be ineffective against later attacks [2, 21] and packet padding also provides little defence against attacks based only on timing information such as that of [7]. In [17] anonymity sets are created by clustering the packet traces for web page fetches and morphing them to look like the centroid of their cluster. A similar approach is considered by [21, 22]. However, in addition to still entailing a relatively high bandwidth overhead (>60%) these approaches also require maintaining a database of packet traces that may be both large and strongly dependent on the location of a client device (e.g. when moved from work to home, or between hot spots with different qualities of network connection).

On links carrying multiple web fetches interleaving of packets from these fetches can be expected to make it harder for an attacker to infer the web pages being browsed. Intuitively, the inference task of the attacker becomes that of estimating the components of a mixture, which can be expected to increase in difficulty as the number of components (web fetches) increases. The great advantage of this approach is that it avoids the costs associated with buffering and injecting dummy packets. However, it relies on the availability of sufficiently many active fetches at any given time. Further, while only a limited number of studies have been carried out that evaluate the protection provided by combining multiple web fetches, generally the results point towards this approach being of limited effectiveness against modern attacks, see for example [2]. In summary, it is difficult to quantify the degree of privacy provided at any given time when relying on interleaving of multiple fetches, and due to its opportunistic nature it is probably inherently impossible to provide up front guarantees of privacy to users using this type of approach.

III. Achieving Privacy

A. Indistinguishability

As already noted, we are not concerned with concealing the fact that the user is browsing the web but rather with concealing the particular pages visited. Privacy is therefore embodied in the indistinguishability of sequences of web fetches given with regard to the packet sequences which they generate.
It is important to note that we do not insist that fetches of single web pages individually be indistinguishable. Rather, we require that the observed packet sequence corresponding to the fetch of each web page can be reasonably explained by other combinations of web fetches. So, for example, the packet sequence generated by the fetch of a large web page could equally have been generated by sufficiently many sequences of fetches of small web pages. Fetches of single web pages must still be indistinguishable from one another when they cannot plausibly be generated by combinations of other page fetches. That is, we require fetches of sufficiently small web pages to be atomic in the sense that they are indistinguishable from one another.

B. Defence By Use of “Traces”

The basic problem with the packet sequence generated by a web fetch is that it contains many packets (often several hundred, frequently more) and so amounts to an observation of a point in high dimensional space. It is increasingly recognised that high dimensional data carries a major risk of de-anonymisation since data points tend to be sparsely distributed in high dimensions (each point can be individually distinguished). Our approach is therefore to reduce the dimension of the observed packet sequence data. We do this by gathering packets into larger groups and transmitting these groups in a uniform way.

This approach is illustrated in Figure 3a. Here, a web fetch starts at time $T_0$. Dummy packets (indicated in grey) are inserted so that the observed sequence of packets, which is a combination of dummy plus user packets, has a uniform profile and duration. We refer to this uniform profile as a “trace”. The rate and duration of the trace are design parameters, that we will return to later. In this example the web fetch completes by time $T_0 + T$ and so fits within a single trace. However, if the duration exceeded $T$ then a second trace would be started so as to mask the additional user packets, see upper schematic in Figure 3b. Similarly, if the rate at which user packets arrive much exceeds the rate of a trace then two traces can be started in parallel, see lower schematic in Figure 3b. With this approach the observed packet sequence now consists of a sequence of traces.

We note that [17], [21], [22] also consider indistinguishability, doing this through the creation of so-called anonymity sets by clustering the packet traces for web page fetches. However the approach considered here differs in a number of important and fundamental respects. One is that the sequence of traces is constructed on the fly as packets are transmitted. There is no need to pre-cluster web fetches and construct a trace in advance for each cluster, therefore also no need to maintain a history of web fetches (needed to perform pre-clustering) nor to explicitly adapt to location and time (which may change the link characteristics and so the packet traces and their clusters). A second key difference is that packets from multiple web fetches/flows can be packed together into the same trace, thereby allowing us to opportunistically reduce the number

1We might also use a trace where the rate is not constant over time and also allow the trace used to be drawn from a pre-defined set of traces.

Fig. 3: Illustrating use of dummy packets to force the number and timing of transmitted packets to conform to an uninformative pattern or “trace”. When insufficient information packets are available, dummy packets are inserted as needed. When too many information packets are available they are buffered until a transmission opportunity becomes available.

Fig. 4: Example illustrating an observed sequence of traces.
The cardinality of $W$ is needed to cover all 100 web sites and a trace of duration sites that fit inside the trace. A trace of duration 10890ms in a trace which are dummy packets vs the number of web by over-provisioning of network capacity and as a result modern networks quality of service is frequently achieved increased delay, perhaps much increased delay. However, in sending dummy packets. However, this comes at the cost of are available to fill a trace we can avoid the overhead of C. Impact of Rate and Duration of a Trace on Overhead it further here.

We note also that the option exists to inject dummy traces into the tunnel link to add a further level of deniability. Using such an approach the potential exists to formulate indistinguishability as a form of differential privacy. Namely, such that the addition of the fetch of any individual web page has limited impact on the sequence of traces observed on the link. However, we leave this as future work and do not pursue it further here.

### C. Impact of Rate and Duration of a Trace on Overhead

By buffering user packets at the tunnel ingress until enough are available to fill a trace we can avoid the overhead of sending dummy packets. However, this comes at the cost of increased delay, perhaps much increased delay. However, in modern networks quality of service is frequently achieved by over-provisioning of network capacity and as a result bandwidth is often relatively plentiful. Conversely, users are known to be sensitive to delay when using online services. For example, Amazon estimates that a 100ms increase in delay reduces its revenue by 1% 8, Google measured a 0.74% drop in web searches when delay was artificially increased by 400ms 24 while Bing saw a 1.2% reduction in per-user revenue when the service delay was increased by 500ms 19. Hence, in order to enhance resistance to traffic analysis attacks it is preferable to sacrifice some bandwidth by sending dummy packets rather than incurring excessive delay.

To gain some insight into the overhead of dummy packets associated with different durations of trace we fetched the home pages from the Alexa top 100 finance and health web sites in Ireland. Figure 5a plots the average fraction of packets in a trace which are dummy packets vs the number of web sites that fit inside the trace. A trace of duration 10890ms is needed to cover all 100 web sites and a trace of duration 4947ms to cover 50 web sites. As might be expected, the fraction of dummy packets increases as the duration of the trace is increased to include more web pages. When all 100 web pages fit inside a single trace the overhead is around 70% but this falls to around 30% for a trace that covers 50 web pages. Figure 5b plots the fraction of dummy packets vs the maximum number of consecutive traces used to cover all 100 web pages (so the value for one trace corresponds to the data in Figure 5a). It can be seen that as the number of traces used increases the overhead falls. However, use of smaller traces reduces the level of indistinguishability provided and so a trade-off exists between privacy and dummy packet overhead.

Importantly, the trade-off between privacy and dummy packet overhead also depends on the number of flows simultaneously active, since packets from multiple flows can be packed together into the same trace thereby reducing the number of dummy packets needed without compromising privacy. This is illustrated in Figure 6 which plots the fraction of dummy packets against the number of simultaneous flows (here a flow is a sequence of fetched pages randomly selected from Alexa’s top 100 web pages). It can be seen that as the number of flows increases, the level of dummy traffic falls towards zero. We will show analytically in the next section that the trace-based method is, in fact, capacity achieving as the traffic load increases, in contrast to previous traffic shaping approaches that come with privacy guarantees.

Since it is hard to analytically quantify the trade-off between privacy, dummy packet overhead and delay (the delay aspect is especially difficult to analyse mathematically), we will revisit this trade-off shortly using experimental data.

### IV. THROUGHPUT AND DUMMY PACKET OVERHEAD

A privacy-enhanced tunnel using the trace approach starts up trace dynamically so as to manage user delay and dummy packet overhead while ensuring privacy. In this section we

2There is no need to manage the termination of traces, each trace simply stops once its allotted time has expired.
consider policies for start-up of traces and analyse how the dummy packet overhead changes with the traffic load (analysis of delay is left to the experimental section below). We develop a scheduling policy that has the following important properties:

1) **Adaptive.** The number of dummy packets transmitted adapts with the traffic load. When the traffic load is light the scheduling policy inserts dummy packets as needed to fill out the active packet trace(s) and thus maintain privacy.

2) **Capacity Achieving.** The number of dummy packets falls to zero as the traffic load increases i.e. user packets are able to make full use of the available network throughput capacity as traffic load increases.

Our analysis is made complicated by (i) the non-preemptive nature of traces, namely once a trace has started it will continue until its allotted time has expired, and (ii) a new trace can start at any time and so when multiple traces are active their start (and so end) times need not be not aligned i.e. the traces may only partially overlap. We therefore begin by relaxing (ii) and requiring the synchronised start/stop of traces, and later relax this requirement.

### A. Network Setup and Notation

The setup considered is illustrated in Fig. 7. Let \( \mathcal{U} = \{1, 2, \ldots, n_u\} \) denote the set of users, \( a_k^{(u)} \in \mathbb{N} \) denote the number of packet arrivals for user \( u \) at time \( k \) and \( \bar{a}^{(u)} := \lim_{K \to \infty} \frac{1}{K} \sum_{k=1}^{K} a_k^{(u)} \) the average number of packet arrivals. Packets for each user \( u \) are held in separate queues, with the queue occupancy for user \( u \) at time \( k \) being denoted by \( q_k^{(u)} \). Letting \( w_k^{(u)} \in \mathbb{N} \) be the number of packets dequeued at time \( k \) then \( \Delta_k^{(u)} = [q_k^{(u)} + a_k^{(u)} - w_k^{(u)}]^+ \) where the notation \([\cdot]^+\) has the usual meaning that \([x]^+ = x \) when \( x \geq 0 \) and \([x]^+ = 0 \) otherwise.

User packets are dequeued according to a predefined trace. Multiple traces may be active simultaneously in order to increase the rate at which user packets are dequeued. A trace \( f \) is a sequence \( p_j \in \mathbb{N}, j = 1, 2, \ldots, n \) where \( p_j \) is the number of packets to be transmitted at time \( j \) and \( n \) is the duration of the trace. Importantly, if no user packets are available to send at time \( j \), then dummy packets will be transmitted to ensure that \( p_j \) packets are always transmitted. Let \( P = \sum_{j=1}^{n} p_j \) denote the total number of packets in a trace.

Packets from a trace are queued at the output link before transmission over the tunnel, see Fig. 7. As is usual in Internet links we leave rate control to the end hosts. If the aggregate send rate persistently exceeds the output link capacity then the queue at this link will eventually overflow and cause packet loss which end hosts can then use as a congestion indicator, e.g. by using TCP congestion control.

New traces are started as needed to service user packet arrivals, and multiple traces may be active at the same time so as to increase the sending rate if needed. Indexing these active traces by \( 1, 2, \ldots \) let \( T \) denote the set of trace indices, \( \tau_t \) the start time of trace \( t \in T \). At time \( k \) the set of active traces is \( T_k := \{t \in T : k \in \{\tau_t, \ldots, \tau_t + n\}\} \) and the number of packets that are transmitted at time \( k \) is \( \sum_{t \in T_k} p_{k-\tau_t} \), therefore \( \sum_{u \in \mathcal{U}} w_k^{(u)} \leq \sum_{t \in T_k} p_{k-\tau_t} \). The average rate of packet transmissions can be no more than the capacity \( c \) in packets/slot of the outgoing link i.e. \( \lim_{K \to \infty} \frac{1}{K} \sum_{k=1}^{K} \sum_{t \in T_k} p_{k-\tau_t} < c \).

Since \( \sum_{t \in T_k} p_{k-\tau_t} \) packets are sent then if there are insufficient user packets available to send at time \( k \) we need to send \( \sum_{t \in T_k} p_{k-\tau_t} - \sum_{u \in \mathcal{U}} w_k^{(u)} \) dummy packets. Our task is to activate traces and thereby adjust the number of user packets transmitted \( w_k^{(u)} \) so as to minimise the number of dummy packets sent while still servicing all of the user packets in a timely manner.

### B. Synchronised Scheduler

To proceed we first assume that active traces start/stop in a synchronised fashion. Since traces started in slot 1 finish at slot \( n \), traces started at slot \( n + 1 \) finish at slot \( 2n \) and so on then by restricting ourselves to starting traces at times \( 1, n + 1, 2n + 1, \ldots \) we can ensure synchronised start/stop of traces and avoid partial overlapping of traces. We will relax this restriction later, but the absence of overlapping traces simplifies the initial analysis and assists with gaining insight into the design issues to be considered.

Formally, partition time slots \( k = 1, 2, \ldots \) into groups of \( n \) slots, see Fig. 8. Denote the first group of slots by \( G_1 := \{1, \ldots, n\} \), the second by \( G_2 := \{n + 1, \ldots, 2n\} \) and so on. A new trace can only be activated at the start of a group of slots. Letting \( y_g \) denote the number of traces active in group \( g \), consider the following trace scheduler:

\[
y_g \in \arg \min_{y_g} \frac{\gamma - Q_g P}{x} \quad (2)
\]

\[
Q_{g+1} = [Q_g + b_g - y_g P]^+ \quad (3)
\]

where \( \gamma > 0 \) is a design parameter, \( b_g := \sum_{u \in \mathcal{U}} \sum_{k \in G_g} a_k^{(u)} \) is the number of user packet arrivals in group of slots \( g \) and recall \( P \) is the total number of packets in a trace. Scheduler (6)-(8) is intuitive.
The following lemma establishes that the scheduler maintains a bounded queue for all packet arrival rates up to $y_{\text{max}}$.

**Lemma 1 (Queue Boundedness).** Consider update (2)-(3). Suppose that $b_y \leq b_{\text{max}}$, $b = E[b_y]$, and $y_{\text{max}} P > b + \epsilon$, $\epsilon > 0$. Then as $g \to \infty$ we have $|\gamma - Q_g P| \leq 2\delta$ where $\delta := b_{\text{max}} + y_{\text{max}} P$.

**Proof.** Case (i) $\gamma - Q_g P > \delta$. Then $y_g = 0$ and $Q_{g+1} = [Q_g + b_y]^+ = Q_g + b_y$ since $Q_g, b_y > 0$. Hence, $\gamma - Q_{g+1} P = \gamma - Q_g P - b_y P$. Taking expectations conditioned on $Q_g$, $\gamma - E[Q_{g+1}|Q_g] P = \gamma - Q_g P - b P$.

Case (ii) $\gamma - Q_g P < -\delta$. Then $y_g = y_{\text{max}}$ and $Q_{g+1} = [Q_g + b_y - y_{\text{max}} P]^+$. Since $\gamma - Q_g P < -\delta$ then $Q_g > (\gamma + \delta)/P > y_{\text{max}}$. Hence, $Q_{g+1} = Q_g + b_y - y_{\text{max}} P$ and $\gamma - Q_{g+1} P = \gamma - Q_g P - b_y P + y_{\text{max}} P^2$. Taking expectations, $\gamma - E[Q_{g+1}|Q_g] P = \gamma - Q_g P - b P + y_{\text{max}} P^2 > \gamma - Q_g P + \epsilon P$.

Case (iii) $|\gamma - Q_g P| < \delta$. Since $Q_{g+1} \geq 0$ then $\gamma - Q_{g+1} P \leq \gamma$. For the left-hand bound, we use the fact that $Q_{g+1} - Q_g \leq b_{\text{max}} + y_{\text{max}} P$ and so $\gamma - Q_{g+1} P \geq -\delta - (b_{\text{max}} + y_{\text{max}} P)$.

Hence, when $|\gamma - Q_g P| > \delta$ then $\gamma - E[Q_{g+1}|Q_g] P$ is strictly decreasing and when $|\gamma - Q_g P| \leq \delta$ then it remains in the ball $|\gamma - E[Q_{g+1}|Q_g] P| \leq 2\delta$. \qed

We have the following immediate consequence.

**Corollary 1 (Capacity Achieving).** Suppose $y_{\text{max}} > \bar{b}/P$. Then the mean service rate provided by the traces is sufficient to serve the mean rate of user packet arrivals. That is, the trace-based tunnel with scheduler (2)-(3) is capacity achieving.

**Proof.** Observe from (3) that, $Q_{g+1} \geq Q_g + b_y - y_g P$ and so when $Q_1 = 0$ it follows that

$$\frac{1}{g} \sum_{i=1}^{g-1} (b_i - y_i P) = \bar{b} - y_0 P \leq Q_g / g$$

(5)

where $\bar{b}_y := \frac{1}{g} \sum_{i=1}^{g-1} b_y$ and $y_0 := \frac{1}{g} \sum_{i=1}^{g-1} y_i$. Since, by Lemma 1, $Q_g$ is bounded then $\frac{Q_g}{g} \to 0$ as $g \to \infty$ and therefore $\bar{b}_y \leq y_0 P$ as $g \to \infty$. That is, the mean service rate provided by the traces is sufficient to serve the mean rate of user packet arrivals. This holds for every $\epsilon > 0$ such that $y_{\text{max}} P > \bar{b} + \epsilon$. \qed

C. Discussion

The foregoing analysis establishes that the trace-based tunnel approach is capacity achieving. Intuitively, as the traffic load increases the probability increases that a user packet is available in the queue when a transmission slot arises and so the number of dummy packets decreases, eventually falling to zero.

**Scheduler (2)-(3)** is easy to analyse but rather harsh in that it selects either $y_{\text{max}}$ or 0 active traces in each group of slots. This can be alleviated by modifying the scheduler as follows:

$$x_g \in \arg \min_{x \in [\hat{t}^{-1}, 1]} (\gamma - Q_g P)x$$

(6)

$$y_{g+1} = [y_g + x_g]^0 \cdot y_{\text{max}}$$

(7)

$$Q_{g+1} = [Q_g + b_g + y_{g+1} - y_{g+1} P]^+$$

(8)

where $[x]^{0 \cdot y_{\text{max}}} = x$ when $x \in [0, y_{\text{max}}]$ and equals 0 when $x < 0$ and $y_{\text{max}}$ when $x > y_{\text{max}}$. This scheduler is intuitive, although the use of incremental updates to $y_g$ is somewhat unusual. Namely, queue $Q_g$ measures the accumulated mismatch between the arrival of user packets and service of packets. When this mismatch grows too large and exceeds $\gamma/P$ then we increase the number of active traces, when the mismatch falls to less than $\gamma/P$ we decrease the number of active traces.

We can also relax the synchronisation assumption made in the previous section by allowing new traces to start in any slot. This means that traces can, for example, now partially overlap. The payoff is that the potential exists to exploit this extra freedom to achieve improved performance, especially improved delay performance as a result of being able to start new traces in a more timely way.

As our baseline unsynchronised scheduler we use the following update,

$$x_k \in \arg \min_{x \in [\hat{t}_k^{-1}, 1]} (\gamma - Q_k P)x$$

(9)

$$z_{k+1} = [z_k + x_k]^0 \cdot y_{\text{max}}$$

(10)

$$\hat{Q}_{k+1} = [\hat{Q}_k + \alpha(\sum_{u \in U} a_k^{(u)} - \sum_{t \in T_k} \hat{p}_{k-t})]^+$$

(11)

where a new trace is started in slot $k$ (added to $T_k$) when $z_k$ increases or when a trace completes and the number of active traces $|T_k|$ falls below $z_k$.

V. Experimental Results

We built a prototype VPN implementation of the scheduler in (6) and also of its unsynchronised variant. The entire project including codes, scripts and datasets for all measurements in this paper is available at [6]. Using this VPN prototype, in this section we present experimental measurements of the scheduler performance under a wide range of conditions. In particular, while the analysis in the previous section focuses on throughput efficiency, our measurements allow us to also measure the delay performance of the scheduler and the impact of design choices (such as the use of unsynchronised trace activation) on this. We present results for both UDP and TCP traffic, a potential concern with TCP being possible overlap. The payoff is that the potential exists to exploit this extra freedom to achieve improved performance, especially improved delay performance as a result of being able to start new traces in a more timely way.

As our baseline unsynchronised scheduler we use the following update,
A. Hardware Setup

We begin by describing the hardware setup used. The network topology is shown schematically in Figure 9. Clients are connected to the VPN gateway A using an IPSec protected link. The link between nodes A and B (labelled Seculink) is the encrypted privacy-enhanced VPN tunnel. This is a public link, i.e., it is exposed to sniffing from adversaries. Seculink gateway A is a commodity server with an Intel(R) Core 2 Quad @2.66GHz CPU and 4GB RAM running Ubuntu 14.04.4 LTS. The other end of the tunnel, node B, is another commodity server with an Intel(R) Core 2 Duo @ 3.00GHz and 2GB RAM running Ubuntu 12.04.5 LTS. Nodes A and B use a mix of TP-LINK TG-3468 1000Mbps external ethernet cards and on-board gigabit ethernet cards (RTL8111 PCI Express and Intel 82567LM-3 respectively) for network connectivity.

Traffic from node B is routed to the Internet via a campus gateway using a 100Mbps link (it is this link which limits the network rate since the tunnel between A and B is a gigabit connection). A NETGEAR JGS516 Gigabit switch is used to connect client machines to Seculink gateway A. The Seculink tunnel uses IPSec and traffic shaping to protect against DPI and traffic analysis attacks. Outgoing traffic from clients is sent to node B via node A and then forwarded by B to the campus gateway. Incoming traffic arriving at node B is forwarded to node A and then sent to the corresponding client. The proposed tunnel is designed to perform the scheduling and traffic shaping on aggregated traffic from multiple clients. However in order to compare traffic patterns before and after the experiments of this section unless stated otherwise, which is found to provide a good compromise between responsiveness and sensitivity to small fluctuations.

B. Software Setup

Both Seculink and the links between clients and node A are protected with IPSec protocols AES-256 and SHA-256. Traffic shaping on Seculink is implemented using the netfilter and netfilter_queue libraries in C in conjunction with the iptables module in Linux. Gateway A queues packets from clients and transmits them over Seculink using our traffic shaping scheduler with slots of 1ms duration. When no client packets are queued for transmission yet the active traces require a packet to be sent then a dummy packet is generated and sent. Traffic received at node B is first filtered to remove dummy packets and remaining packets are then forwarded to the campus gateway over an unencrypted link. Responses received from the Internet are treated similarly, transmitted by node B to node A using our traffic shaping scheduler. In the UDP experiments traffic is generated using the PERIODIC method of the MGREN traffic generator. TCP traffic is generated by using wget to fetch dummy files of various sizes from a server located in the campus network.

C. Synchronised vs Unsynchronised Schedulers

With a synchronised scheduler traces only start and finish at the beginning of each cycle, a cycle being 9s duration in all of our tests unless otherwise stated. This means that when the arrival rate is changed in the middle of a cycle we have to wait until the start of the next cycle to update the number of active traces and so when new flows start they may experience significant delay. This is particularly an issue when no traces are active in the tunnel and a new client flow starts since in this case the scheduler cannot transmit any packets until the next cycle starts (when some traces are already active the scheduler has the option to transmit some packets from the new flow e.g. by substituting for packets from other flows or for dummy packets). Similarly, when the rate of incoming traffic is less than the rate of a single trace then sufficient packets need to be queued before a trace is activated, causing delay.

To mitigate such delays on lightly loaded links, two unsynchronised modifications are implemented to “wake” the channel from silence upon sensing new incoming traffic:

1) DNS Triggering. Upon receiving a DNS packet when no traces are active, a new trace is immediately activated.

2) Use of Running Average When Lightly-Loaded. We maintain a running average of packet arrivals, \( \bar{a} = (1 - \zeta)\bar{a} + \zeta \sum_{k \in U} Q^{(u)}_k \), where \( \zeta > 0 \) is a design parameter. When \( \bar{a} \) exceeds threshold \( a^* \) and no traces are active then a new trace is activated. In our experiments we use \( \zeta = 0.001 \), \( a^* = 0.005 \).

In addition, to avoid adding new traces in response to small spikes in arrivals, when \( \gamma - Q_k P < 0 \) we only activate a new trace when also \( Q_k - Q_{k-m} > 0 \), where \( m \) is a parameter. In this way we only respond to a sustained increase in \( Q \). Similarly, when \( \gamma - Q_k P > 0 \) we only decrease \( x_k \) when also \( Q_k - Q_{k-m} < 0 \). In our experiments we use \( m = 100 \) unless otherwise stated, which is found to provide a good compromise between responsiveness and sensitivity to small fluctuations (since slots are 1ms duration, this corresponds to a window of 100ms duration).
TCP and shows measurements for both the downlink and duration. This data is for an example web fetch using the delay and fraction of dummy packets vs the trace rate for a defined duration. Figure 11 shows measurements of D. Choice of Trace Parameters

a synchronised scheduler vs an unsynchronised scheduler. It time histories of the arrival and services rates when using use of this unsynchronised scheduler unless otherwise stated. has better performance, in the rest of this section we will make reducing the number of dummy packets transmitted). Since it arrival rate reduced (so reducing the delay experienced by user packets) but also the delay in reducing the service rate in response to a fall in the arrival rate is also reduced (so reducing the number of dummy packets transmitted). Since it has better performance, in the rest of this section we will make use of this unsynchronised scheduler unless otherwise stated.

D. Choice of Trace Parameters

We make use of traces which transmit at a constant rate for a defined duration. Figure 11 shows measurements of the delay and fraction of dummy packets vs the trace rate and duration. This data is for an example web fetch using TCP and shows measurements for both the downlink and uplink (the uplink carries the TCP ACKs). Figure 12 shows the corresponding impact of the trace rate and duration on the web fetch completion time. It can be seen from these plots that increasing the trace rate reduces completion time but increases the dummy packet overhead, but the duration of trace has limited effect on completion time. The dummy packet overhead for different trace durations is also shown in Figure 13. From now on we use a trace of duration 9s and rate 0.2 pkts/ms, with aim of achieving a reasonable balance between delay and dummy packet overhead.

To reduce sensitivity to small fluctuations in arrivals we also modify the \( Q \) update to use \( \max\{\bar{a}, \sum_{u \in U} a^{(u)}_{k}\} \), namely:

\[
\hat{Q}_{k+1} = \left[ \hat{Q}_k + \alpha (\max\{\bar{a}, \sum_{u \in U} a^{(u)}_{k}\} - \sum_{t \in T_k} \rho_{k-t}) \right]^{+}
\]

In this way when the number of packet arrivals falls temporarily \( \bar{a} \) is used instead of \( a_k \).

Figure 10 illustrates the impact of these changes, comparing time histories of the arrival and services rates when using a synchronised scheduler vs an unsynchronised scheduler. It can be seen that the unsynchronised scheduler is much more responsive to changes to traffic load. Not only is the delay in increasing the service rate in response to increases in arrival rate reduced (so reducing the delay experienced by user packets) but also the delay in reducing the service rate in response to a fall in the arrival rate is also reduced (so reducing the number of dummy packets transmitted). Since it has better performance, in the rest of this section we will make use of this unsynchronised scheduler unless otherwise stated.

Fig. 11: Dummy overhead vs trace rate for a sample website. For each rate data is shown for traces of duration 1, 2, 4, 8 and 16 seconds. \( (\gamma = 4096) \).

Fig. 12: Illustrating the effect of trace rate and duration on the completion time of a sample web page. For each rate the data shown in (a) is for traces of duration 1, 2, 4, 8 and 16 second. Similarly, for each duration the data shown in (b) is for rates of 0.1, 0.2, 0.5, 1.0 and 2.0 pkts/ms. \( (\gamma = 4096) \).

Fig. 13: Illustrating the effect of trace duration on dummy overhead for a sample website. For each duration the data shown for rates of 0.1, 0.2, 0.5, 1.0 and 2.0 pkts/ms. \( (\gamma = 4096) \).

Fig. 14: Comparison between dummy and delay rates for CBR UDP traffic and different choices of \( \gamma \). Given the negligible variance, average over 5 tests are presented for each \( \gamma \). (duration: 5min, \( \bar{a} = 2800 \), \( \alpha = 1 \), \( P = 1682 \), \( n = 9615 \), \( c = 20 \)). Values for each \( \gamma \) are averaged over 5 tests (error bars are not shown since they are too small to be clearly visible).

Fig. 15: Comparison between delay and dummy rates vs arrival rate for CBR UDP traffic with \( \gamma = 1024 \). (duration: 5min, \( \alpha = 1 \), \( P = 1682 \), \( n = 9615 \), \( c = 20 \)). Values are averaged over 5 tests.
E. Performance with CBR UDP Traffic

In this section we study throughput efficiency and delay performance with constant bitrate (CBR) UDP arrivals. By throughput efficiency we mean the mismatch between the transmit rate of the scheduler and the arrival rate of client packets. Recall that when there are insufficient client packets buffered at the scheduler then the scheduler transmits dummy packets so that its transmissions can continue to follow the predefined trace pattern. These dummy packets provide resistance to traffic analysis attacks but increase the load on the network and so we would like to minimise these. There is a fairly direct trade-off between delay and the volume of dummy packets transmitted since by increasing the backlog of client packets buffered at the input to the scheduler we reduce the need for dummy packets but increase the delay experience by client packets. Conversely, reducing the number of client packets buffered tends to increase the need for dummy packets but decrease delay.

This trade-off between throughput efficiency and delay can be seen in Figure 14 which plots measurements of the delay and dummy rate vs scheduler parameter $\gamma$ which directly influences the queue backlog (with increases in $\gamma$ increasing the backlog). The dummy rate value shown is the ratio of the dummy packets sent to the total number of packets sent (dummy plus processed packets). Note that there is no packet loss within the scheduler in these tests. It can be seen from Figure 14 that as $\gamma$ is increased the delay rises but the rate at which dummy packets are sent falls. Observe, however, that even for relatively small values of $\gamma$ the dummy rate remains reasonably small, which is encouraging.

Figure 15 shows corresponding measurements of delay and dummy rate as the arrival rate is varied (scheduler parameter $\gamma$ is held constant at 1024 in these plots). It can be seen that the delay tends to increase with arrival rate and the dummy rate to fall. This can be understood by noting that as the arrival rate rises more user packets tend to be buffered at the scheduler. Hence, the delay rises but also user packets are more likely to be available when a transmission opportunity occurs and so the number of dummy packets needed falls. Importantly, observe that the delay is consistently reasonable, rising to no more than 275ms at higher arrival rates, despite the extensive traffic shapping being carried out. Also, the decrease in dummy rate with increasing arrival rate means that the throughput efficiency increases with increasing load, so mitigating the dummy packet cost incurred by the traffic shapping.

F. Performance with TCP Flows

We now present data on the scheduler performance with TCP traffic (for the default Linux Cubic TCP variant). Figure 16 plots measurements of delay and fraction of dummy packets vs design parameter $\gamma$ when fetching a 1024MB file, the average over 5 tests is presented for each $\gamma$ ($\alpha = 1$, $P = 1682$, $n = 9615$, $c = 20$).

![Fig. 16: Measurements of delay and fraction of dummy packets vs $\gamma$ for TCP traffic.](image)

Fig. 16: Measurements of delay and fraction of dummy packets vs $\gamma$ for TCP traffic. The data is measured fetching a 1024MB file, the average over 5 tests is presented for each $\gamma$ ($\alpha = 1$, $P = 1682$, $n = 9615$, $c = 20$).

![Fig. 17: Completion time vs file sizes when using TCP, $\gamma = 1024$.](image)

Fig. 17: Completion time vs file sizes when using TCP, $\gamma = 1024$. Results are shown both for the traffic shaped tunnel and for an unprotected link (no encryption, no traffic shaping). Values are averaged over 5 runs.

This experiment is conducted by fetching a file of size 8192MB, insensitive to $\gamma$ when using TCP and remains consistently below 20%.

Figure 17 plots the measured completion time vs file size fetched. For comparison, the corresponding data is also shown for a link with no traffic shaping. It can be seen that the cost, in terms of increase in completion time, is modest. Figure 18 provides additional detail, showing measured data on delay and dummy rate vs file size. It can be seen that the delay and dummy rate both tend to fall as the file size increases. The effect here is due to the time that it takes the scheduler to adapt to the arrival of a new flow: for longer flows this adaptation overhead gets washed out and amortised over many packets but for short flows its effect is more pronounced. This can be seen in Figure 19 which shows time histories of the number of active traces on both the downlink and uplink (uplink traces are carrying the TCP acks and so are fewer in number). The experiment is conducted by fetching a file of size 8192MB.
The link reaches steady state in about 20 seconds.

G. Web Traffic Privacy

We fetched the home pages from the Alexa top 100 finance and health web sites in Ireland. Figure 20 shows four example trace time histories recorded during these fetches. It can be seen that the traces time histories in Figures 20a and 20c are identical and so evidently these two web pages cannot be distinguished by an attacker. Figure 21 plots the number of distinct trace time histories measured on the uplink and downlink while fetching the 100 web pages and also the number of pages for which each trace time history is observed. It can be seen that certain trace time histories are generated by 10-30 different web pages and so these web pages are indistinguishable to an attacker.

It can also be seen in Figure 21 that around 20 trace time histories are generated by a single web page, and so potentially vulnerable to attack. An example is shown in Figure 20d. Evidently the trace time history in Figure 20d cannot be distinguished from two fetches of the web sites in Figures 20a and 20c. The trace time history in Figure 20b is more complex, and raises the question of whether there exists one or more combinations of web page fetches that yield the same trace time history and so cannot be distinguished from this web fetch.

As an illustrative example we collected 10 samples for each of the two following browsing behaviours (i) a web page from our data set is fetched on a single tab and (ii) on three concurrent tabs, the words “invisible”, “history” and “browsing” are queried from Google search with a 0.2 second gap between queries. The measured trace time histories are plotted in Figure 22. It can be seen that the time histories look very similar, thus allowing a user to plausibly deny that they fetched the single sensitive web page since the observed trace time history might equally have been generated by non-sensitive google queries.

To get more insight, we proceed as follows. Let \( H \) de-
note the set of measured trace time histories. For trace time history $h \in \mathcal{H}$ we determine (by exhaustive search) the combinations $C_{h,1}, C_{h,2}, \ldots, C_{h,m_h}$ of the other trace time histories that are indistinguishable from $h$. Combination $C_{h,i} = \{(h_{i,1}, n_1), (h_{i,2}, n_2), \ldots\}$ with $h_{i,j} \in \mathcal{H} \setminus \{h\}$ and $n_j$ the number of times that $h_{i,j}$ is repeated in the combination. Let $\mathcal{W}_h$ denote the set of web pages that generate trace time history $h$, so $|\mathcal{W}_h|$ is the number of web pages that generate $h$ (see Figure 24). Let the web page fetched $W$ be a random variable that takes values in $\cup_{h \in \mathcal{H}} \mathcal{W}_h$. Assume, for simplicity, that the web pages in $\mathcal{W}_h$ are equally likely to be fetched. Then when combination $C_{h,i}$ is observed the probability that page $\omega$ was fetched is,

$$P(W = \omega|C_{h,i}) = \sum_{(h',n') \in C_{h,i}} \sum_{\omega \in \mathcal{W}_h} \frac{n'}{|\mathcal{W}_h|} \left(\frac{1}{|\mathcal{W}_h|} \right) = \frac{m_h}{\sum_{i=1}^{m_h} P(W = \omega|C_{h,i})}$$

(12)

Let $C$ be a random variable which is the combination used. Assume, again for simplicity, that each combination $C_{h,1}, C_{h,2}, \ldots, C_{h,m_h}$ is equally probable when trace $h$ is observed, i.e $P(C = C_{h,i}) = 1/m_h$. Then the probability that web page $\omega$ was fetched given that trace time history $h$ was observed is,

$$P(W = \omega|h) = \frac{m_h}{\sum_{i=1}^{m_h} P(W = \omega|C_{h,i})}$$

(13)

When the traces $h \in \mathcal{H}$ are equally likely to be observed then $P(W = \omega) = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} P(W = \omega|h)$. Figure 23 plots $P(W = \omega)$ calculated using (12) for each of the 100 web pages, sorted in increasing order. It can be seen that this probability is less than 0.08 on the downlink and less than 0.05 on the uplink. We can conclude therefore that with the foregoing assumptions the user can reasonably deny that page $\omega$ was fetched despite an attacker observing the transmitted packet trace.

To get a sense of the sensitivity of these values to the assumption of equiprobability, for comparison, let $h^* \in \arg\min_{h \in \mathcal{H}} P(W = \omega|h)$ and suppose that this worst case trace $h^*$ is always observed i.e $P(W = \omega) = P(W = \omega|h^*)$. Figure 24 shows the calculated $P(W = \omega|h^*)$ for our measured web fetches. It can be seen that $P(W = \omega|h^*)$ is higher than in Figure 23 as expected, and indeed reaches one but only for a small number of web pages. Hence, even in this worst case a user has fairly level of strong deniability.

VI. CONCLUSIONS

In this paper we introduce a low overhead capacity-achieving tunnel that is resistant to traffic analysis in the sense that it provides deniability to users that any specified web page was fetched given that a specified packet trace is observed on the tunnel. We present a scheduler design for managing the transmission of traces to satisfy user traffic demand while maintaining reasonably low delay and throughput overhead due to dummy packets. Experimental results are also presented demonstrating the effectiveness of this scheduler under a range of realistic network conditions and real web page fetches.

REFERENCES

[1] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In G. Danezis and D. Martin, editors, Privacy Enhancing Technologies, volume 3856 of Lecture Notes in Computer Science, pages 1–11. Springer Berlin Heidelberg, 2006.

[2] X. Cai, X. Ch. Zhang, B. Joshi, and R. Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS ’12, pages 605–616, New York, NY, USA, 2012. ACM.

[3] Xiang Cai, Rishab Nithyanand, and Rob Johnson. Cs-buflo: A congestion-sensitive website fingerprinting defense. In Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES ’14, pages 121–130, New York, NY, USA, 2014. ACM.

[4] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS ’14, pages 227–238, New York, NY, USA, 2014. ACM.
[5] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In Security and Privacy (SP), 2012 IEEE Symposium on, pages 332–346, May 2012.

[6] S. Feghhi. Seculink Tunnel Project: Codes and Measurements, 2016. available at: [www.scss.tcd.ie/disciplines/stable/seculink](www.scss.tcd.ie/disciplines/stable/seculink).

[7] S. Feghhi and D. Leith. A Web Traffic Analysis Attack Using Only Timing Information. IEEE Transactions on Information Forensics and Security, 11(8):1747–1759, April 2016.

[8] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing web latency: the virtue of gentle aggression. ACM SIGCOMM Computer Communication Review, 43(4):159–170, 2013.

[9] Xun Gong, Negar Kiyavash, and Nikita Borisov. Fingerprinting websites using remote traffic analysis. In Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS ’10, pages 684–686, New York, NY, USA, 2010. ACM.

[10] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCS ’09, pages 31–42, New York, NY, USA, 2009. ACM.

[11] A. Hintz. Fingerprinting Websites Using Traffic Analysis. In Roger Dingledine and Paul Syverson, editors, Privacy Enhancing Technologies, volume 2482 of Lecture Notes in Computer Science, pages 171–178. Springer Berlin Heidelberg, 2003.

[12] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A Critical Analysis of Website Fingerprinting Attacks. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), pages 263–274, 2014.

[13] M. Liberatore and B. N. Levine. Inferring the Source of Encrypted HTTP Connections. In Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS ’06, pages 255–263, New York, NY, USA, 2006. ACM.

[14] L. Lu, E. C. Chang, and M. C. Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In D. Gritzalis, B. Preneel, and M. Theohariou, editors, Computer Security ESORICS 2010, volume 6345 of Lecture Notes in Computer Science, pages 199–214. Springer Berlin Heidelberg, 2010.

[15] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. HTTPS: Sealing information leaks with browser-side obfuscation of encrypted flows. In In Proc. Network and Distributed Systems Symposium (NDSS). The Internet Society, 2011.

[16] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar. I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis. In Emiliano De Cristofaro and Steven J. Murdoch, editors, Privacy Enhancing Technologies, volume 8555 of Lecture Notes in Computer Science, pages 143–163. Springer International Publishing, 2014.

[17] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES ’14, pages 131–134, New York, NY, USA, 2014. ACM.

[18] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society, WPES ’11, pages 103–114, New York, NY, USA, 2011. ACM.

[19] S. Souders. Velocity and the bottom line. In Velocity (Web Performance and Operations Conference), 2009.

[20] W. Tao, C. Xiang, N. Rishab, R. Johnson, and I. Goldberg. Effective Attacks and Provably Defenses for Website Fingerprinting. In 23rd USENIX Security Symposium (USENIX Security 14), pages 143–157, San Diego, CA, August 2014. USENIX Association.

[21] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I Goldberg. Effective Attacks and Provably Defenses for Website Fingerprinting. In Proceedings of the USENIX Security Symposium, pages 143–157, 2014.

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

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

[24] Wenxuan Zhou, Qingxi Li, Matthew Caesar, and P Godfrey. ASAP: A low-latency transport layer. In Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies, page 20. ACM, 2011.