QCSD: A QUIC Client-Side Website-Fingerprinting Defence Framework

Jean-Pierre Smith
ETH Zurich

Luca Dolfi
ETH Zurich

Prateek Mittal
Princeton University

Adrian Perrig
ETH Zurich

Abstract

Website fingerprinting attacks, which analyse the metadata of encrypted network communication to identify visited websites, have been shown to be effective on privacy-enhancing technologies including virtual private networks (VPNs) and encrypted proxies. Despite this, VPNs are still undefended against these attacks, leaving millions of users vulnerable. Proposed defences against website fingerprinting require cooperation between the client and a remote endpoint to reshape the network traffic, thereby hindering deployment.

We observe that the rapid and wide-spread deployment of QUIC and HTTP/3 creates an exciting opportunity to build website-fingerprinting defences directly into client applications, such as browsers, without requiring any changes to web servers, VPNs, or the deployment of new network services. We therefore design and implement the QCSD framework, which leverages QUIC and HTTP/3 to emulate existing website-fingerprinting defences by bidirectionally adding cover traffic and reshaping connections solely from the client. As case studies, we emulate both the FRONT and Tamaraw defences solely from the client and collected several datasets of live-defended traffic on which we evaluated modern machine-learning based attacks. Our results demonstrate the promise of this approach in shaping connections towards client-orchestrated defences, thereby removing a primary barrier to the deployment of website-fingerprinting defences.

1 Introduction

Despite its appearance in 1948 in the Universal Declaration of Human Rights, "No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence", privacy is difficult to achieve on today's Internet. Instead, government and private surveillance can determine Internet users' web-browsing activities by observing their unencrypted traffic and the metadata leaked by standard encrypted communication. The increasing popularity of privacy-enhancing services, such as virtual private network (VPN) connections and anonymity networks like Tor [1], as well as of privacy-focused browsers, such as Brave and Firefox, indicates the importance of privacy to web users. Indeed, according to the GlobalWebIndex, an estimated 25% of the world's Internet users utilise VPNs at least once a month [2], with 34% of those utilising it for privacy reasons [3].

Nonetheless, researchers have shown that despite the encryption and proxying performed by these privacy-enhancing technologies, it is still possible to identify the visited website [4]–[17]. In website fingerprinting, a passive observer identifies the website by statistically analysing observed packet sizes and timings, undeterred by encryption. This class of attacks has been shown to successfully identify encrypted websites while being undetectable by the web user [12]–[15].

Numerous defences against website fingerprinting have been proposed, which reshape the communication between the endpoints to obscure traffic features to and from the client [8], [18]–[27]. However, these defences require changes to both local endpoints, such as web clients, and remote endpoints, such as web servers or proxy gateways. This requirement to deploy or change endpoints outside of a client's control has hindered the deployment of website-fingerprinting defences. With the growing demand for VPNs provided as browser extensions, and with browser vendors targeting the VPN market with customised solutions, such as Mozilla VPN or Brave Firewall + VPN, protection for VPN users against website-fingerprinting attacks has become more relevant. We therefore investigate whether we can instead place the deployment and evolution of website-fingerprinting defences within the purview of the client, that is, without the explicit cooperation of servers or proxies.

Contributions We observe that since enacting a website-fingerprinting defence involves reshaping the communication both from and to the client, then doing so without modifying remote endpoints must be done through manipulation of the network protocols already being used to communicate. In this regard, we explore the recent development, and already broad deployment, of the QUIC transport protocol as an exciting

opportunity to enhance user privacy. QUIC's independent, multiplexed byte-streams and control-message variety provide rich features on which to base our framework, and its location in user space allows defences based upon it to be independently deployed for individual applications. Furthermore, QUIC is being widely adopted, as HTTP/3 – the upcoming HTTP standard – will only be implemented atop the QUIC protocol and notably does not use TCP.

In this work, we design and evaluate a client-side framework for implementing various website-fingerprinting defences *without requiring any changes to the remote server or additional network services*. Our QUIC client-side defence framework (QCSD) is built upon QUIC and HTTP/3 and uses these protocols to induce the server into cooperating with a client-orchestrated defence, specified as either a packet schedule or state machine. QCSD achieves this cooperation using two observations. First, HTTP requests are idempotent and web servers often host resources beyond those needed by any single web page. These resources can thus be repeatedly requested to provide a constant supply of chaff traffic from the server to the client. Second, QUIC connections multiplex individual byte streams, each of which may be independently flow-controlled. This flow control allows an endpoint to indicate to its peer when it is ready to receive data. We leverage this flow control to communicate to the server when to send bursts of application data, chaff data, or both. Through careful management of these chaff resources and individual streams, we create a framework that can orchestrate many website-fingerprinting defences in the literature from the client.

As case studies, we enacted both the FRONT [18] and Tamaraw [22] defences in QCSD, collected *live-defended* datasets totalling over 100,000 web-page loads, and evaluated the efficacy of our QCSD-enacted defences against the Deep Fingerprinting [12], Var-CNN [15], and *k*-fingerprinting [9] website-fingerprinting attacks. Our comparisons between the simulated defences and those crafted by QCSD show that our framework is able to emulate defences solely from the client, with privacy benefits akin to their conceptualised forms, and offers further examples of the differences that arise between live-defended and simulated website-fingerprinting defences, highlighting a need for more accurate simulations.

2 Background and Related Work

In this section, we introduce website fingerprinting, existing attacks and defences, our threat model, the QUIC and HTTP/3 protocols, and the requirements for enacting a defence.

2.1 Website Fingerprinting

In website fingerprinting, the goal of an observer is to identify the website fetched over an encrypted channel based solely on side-channel information, such as those derived from packet sizes and timestamps [4]–[17], [28]. Website fingerprinting has been investigated both in the setting of anonymity networks, such as Tor [1], as well as in the setting of encrypted proxies and VPNs, our focus in this work. To identify the website, the observer classifies a feature vector that is derived from the trace of loading a web page of the website. This classification is based on finding similarities between this feature vector and features of previously observed and labelled traces in the attacker's training set. In this process, a label corresponding to the most likely website is assigned, or no label if previously seen traces are too dissimilar.

Closed and open worlds The efficacy of website fingerprinting is evaluated in either the closed- or open-world settings [4], [8]–[17]. In the *closed-world* setting, the trace is known to be from one of *n monitored* websites, whereas in the *open-world* setting, it belongs to either one of the *n* monitored websites or to some *unmonitored* website. In this latter setting, used in our evaluations, the task to identify the monitored website (or if it is an unmonitored website) better emulates the real world where a trace may be from an unknown website.

Attacks Early website-fingerprinting attacks utilised statistical analyses [5], [6], and analyses using only the inferred sizes of HTML pages and resources [7]. Subsequent attacks evolved to leverage machine learning classifiers. These include a k-nearest neighbour (k-NN) based-classifier [8], the *k*-fingerprinting (*k*-FP) classifier based on k-NN and random forests [9], hidden Markov models [10], and stream-matching algorithms [13], [17]. These attacks are further enhanced through features extracted from the traces using detailed feature analyses [9], [29]–[31]. The most recent attacks [11], [12], [14]–[16] utilise neural networks and have achieved accuracy rates above 94% in closed-world settings of 900 websites with encrypted, padded network traffic [11]. Our prior work has also confirmed the efficacy of these attacks in the QUIC-over-VPN setting [32].

To evaluate our framework, we employ the *k*-FP [9], Deep Fingerprinting [12], and Var-CNN [15] classifiers as examples of traditional and deep learning attacks, since they have been shown to be among the most effective attacks [14], [18], [32].

Defences Numerous website-fingerprinting defences have been proposed, particularly for the Tor anonymity network, which differ in their means of defending the communication, their overhead, and their requirements for infrastructure or pre-existing knowledge. Table 1 provides an overview of these defences. The first category, chaff-only defences, defend by solely adding padding and chaff traffic to the communication and thus do not delay the loading of the web page. For example, the WTF-PAD defence uses chaff traffic to obscure long breaks between bursts of incoming or outgoing packet [21], whereas the FRONT defence adds a random amount of chaff packets in each direction, with an emphasis

Table 1: Website-fingerprinting defences categorised by whether their mechanism operates on a static predetermined schedule, a static schedule with a transmission-dependent stop point, or a dynamic schedule. Also indicated is whether the defence would be emulatable within our framework. GLUE is not emulatable as it operates between connections.

Defence	Schedule	Emulatable
Chaff-only		
Traffic Morphing [27]	dynamic	✓
WTF-PAD [21]	dynamic	✓
Cui 2018 [19]	static	✓
FRONT [18]	static	✓
GLUE [18]	dynamic	no*
Chaff and delay		
Glove [33]	static	✓
Supersequence [8]	static, dyn. end	✓
Walkie Talkie [20]	static	✓
CS-BuFLO [23]	dynamic	✓
BuFLO [24]	static, dyn. end	✓
Tamaraw [22]	static, dyn. end	✓
HTTPOS [25]	–	no
Other		
Decoy [26]	–	no
TrafficSliver [34]	–	no
Henri 2020 [35]	–	no

on obscuring the feature-rich head of the communication [18]. As a promising step forward, Tor has since implemented a derivative of WTF-PAD [21], [36] along with a framework for experimenting with other padding approaches [37]. Unfortunately, WTF-PAD has already been shown to be vulnerable to website-fingerprinting attacks [12], [15].

The second category, chaff-and-delay defences, shape the communication towards a target pattern by adding padding, chaff traffic, splitting, and delaying the packets sent to and from the client. Among these defences, BuFLO [24], CS-BuFLO [23], and Tamaraw [22] offer high levels of privacy at the cost of high bandwidth and delay overhead. Tamaraw, for example, transmits fixed-size packets at a constant rates from the client and from the server. Defences such as Supersequence [8] and Walkie Talkie [20] group web pages into anonymity sets such that a common target pattern can be found that reduces the shaping overhead.

All of the above defences rely on shaping the traffic both to and from the client, and have thus been thought to be reliant on client-side and server/proxy-side deployment. However, our framework, QCSD, enables the use of such defences with only client-side deployment. We investigate QCSD's ability to emulate chaff-only and chaff-and-delay defences, with the FRONT and Tamaraw defences respectively as exemplars. FRONT is one of the few chaff-only defences still effective against modern website-fingerprinting attacks [18]; whereas Tamaraw, a chaff-and-shape defence, offers similar protection to transmitting at constant rate with lower overhead. Appendix A describes these defences in more detail.

The HTTPOS [25] defence adds random chaff and delays to the communication through TCP and HTTP manipulation. Similarly to our work, it does so primarily from the client but requires proxies and intermediaries to enable shaping TCP, which is implemented in the kernel. Our work however, unlike HTTPOS, does not create a specific defence strategy but rather provides the tools with which website-fingerprinting defences can be enacted from the client. Finally, Decoy by Panchenko *et al.* [26] is unique in that it neither adds chaff nor delay but instead loads another web page in the background as a decoy.

Other related work The TrafficSliver defence [34] and the use of multihoming by Henri *et al.* [35] send traffic over different network paths to limit the data observed by an adversary. These approaches could be combined with QCSD, should they be deployed in a manner supporting QUIC.

Other QUIC-based privacy efforts include the MASQUE mechanisms [38], which allow multiple proxied stream- and datagram-based flows inside QUIC-HTTPS connections. This is similar to the HTTP CONNECT method that allows proxying via an HTTP server over TCP, and does not provide active deterrence against website-fingerprinting attacks.

2.2 Threat Model

We consider an adversary who attempts to determine the website associated with a traffic trace, through analysing only packet sizes and timings. This setting occurs when the client uses privacy-enhancing technologies, such as anonymity networks (e.g., Tor) or VPN tunnels (e.g., Wireguard, OpenVPN, or IPSec), encrypted wireless communication (e.g., IEEE 802.11i WPA), or when the adversary wants to identify a visit to a particular web page of interest on a website (e.g., the page related to a medical condition on a medical website).

Additionally, we inherit the simplifying assumptions used throughout the attack literature [11], [14]–[16], [39] to remain consistent in our use of website-fingerprinting attacks in the evaluation of our framework. That is, we conservatively assume that (1) an observer can identify the start and end of a web-page load, and can thus extract its trace from the overall traffic; (2) web pages are loaded sequentially without background noise such as media or file transfers; and (3) the page of interest is the index page of the domain. Wang and Goldberg [4] and Cui *et al.* [40] provide evaluations of these assumptions; however, as we focus on defending against an empowered adversary capable of realising these assumptions, inheriting them allows us to both remain consistent with prior works as well as to highlight any differences between conceptualised defences and their implementations in QCSD.

2.3 QUIC and HTTP/3

QUIC is a new connection-oriented transport protocol layered upon UDP. Originally designed by Google with the intent of speeding up the Web [41], it was rapidly deployed by large content providers such as Cloudflare [42], Akamai [43], and Facebook [44] before being standardised by the Internet Engineering Task Force [45]. QUIC provides reliable, in-order delivery of data similar to TCP, but differs in several ways:

Versioned, user space protocol The QUIC protocol is layered upon UDP and implemented in user space. This, along with its use of versioning, version negotiation, and extensions allows the protocol to evolve and avoid ossification.

Always encrypted QUIC packets are always encrypted. This encryption encompasses both the payload and header with the exclusion of the connection identifiers and flags necessary for decoding and decrypting the packet.

Framed data and control Transmitted data and control messages, such as acknowledgements and flow control, are encapsulated in frames which are bundled together and placed in packets. QCSD utilises the QUIC padding frame (`PADDING`), which is a repeatable, single-byte frame that is discarded upon receipt, and the ping frame (`PING`), which forces the server to acknowledge the packet containing it.

Multiplexed streams QUIC connections multiplex multiple in-order byte streams over a single connection. The data for each stream is first encapsulated with a `STREAM` frame before being placed in a packet. A single packet can therefore contain data corresponding to multiple streams. A `STREAM` frame with the `FIN` bit set marks the end of the stream.

Independent flow control Each QUIC stream is independently flow controlled. The maximum-stream-data value is an offset from the start of the stream until which the remote endpoint can send data. It can be increased by sending a `MAX_STREAM_DATA` frame with a higher offset to the remote endpoint. The amount of data that an endpoint is allowed to send on the stream is called its *flow-control credits*.

The standardisation of QUIC brought with it the standardisation of the HTTP/3 protocol [46]. HTTP/3 is the next generation of the Hypertext Transfer Protocol that underlies the web and will be solely transported via QUIC. It defines the semantics for using HTTP with QUIC, such as each QUIC stream transporting a single HTTP request and response. Furthermore, HTTP/3 demarcates HTTP headers and payloads by encapsulating them in frames with prefixed lengths [46].

2.4 Defence Preliminaries

Designing a generalised framework for enacting the aforementioned defences requires a representation of network traffic and understanding the building blocks of a defence. In requesting a web page, a browser contacts one or more web servers to request and receive the HTML of the main page as well as any web resources required to properly display the web page. The resulting ordered sequence of packets seen on the network, across an individual or multiple connections, is called a *trace*.

Definition 2.1. A *packet trace*, or simply *trace*, P, of length n can be uniquely described as a sequence of packet timestamps, lengths, and directions:

$$P = \big((t_1, l_1, d_1), \ldots, (t_n, l_n, d_n)\big)$$

where $t_i \in \mathbb{R}_{\geq 0}$ is the relative timestamp of the i-th packet, with $t_1 = 0$; $l_i \in \mathbb{Z}_{\geq}$ is a non-negative integer denoting the size of the packet, and $d_i \in \{0, 1\}$ is the direction of the packet (either from or to the client, i.e., outgoing or incoming).

Additionally, a trace may be represented as a pair of one-dimensional *aggregated time series*, one for each direction,

Definition 2.2. An *aggregated time series*, $T_{d\in\{0,1\}}$, of the direction d of trace P with granularity I, is a sequence of packet lengths:

$$T_d = (x_1, x_2, \ldots, x_m)$$

where x_i is the sum of all the packet lengths with timestamps t_i within the half-open time interval $[(i-1)\cdot I, i\cdot I)$ and with the specified direction, or 0 if no packets fell within that interval.

Building blocks of a defence QCSD aims to emulate existing defences in the literature, thus we must establish the functionality required to do so. A defence $\mathcal{M}$ is a mechanism that takes an input trace, P, and alters it to create a new trace P'. We can understand the necessary functionality of $\mathcal{M}$ by investigating the ways in which P and P' may differ. Consider w.l.o.g. x and x' at the same index in incoming time series representations $T_{d=1}$ and $T'_{d=1}$. There are three cases. First, when $x' = x$ the defence $\mathcal{M}$ does not perturb the input sequence at that time interval. Second, when $x' > x$ the packet to be transmitted must be padded by $x_\Delta = (x' - x)$ bytes. Furthermore, if $x = 0$ this equates to sending an additional $x_\Delta = x'$ bytes when no packet was present in the input sequence. And third, when $x' < x$, then the input packet is too large, and must reduced in size, sending x' bytes in that interval. If $x' = 0$ this equates to preventing the transmission of any data at that time interval.

From the above formulation, the requirements for an expressive defence framework are clear. A defence framework that emulates a defence $\mathcal{M}$ must be able to (1) send chaff packets, (2) pad packets, (3) split packets, and (4) delay the sending of packets from the client and server. Below we describe our formulation of such a framework, QCSD.

3 A QUIC Client-Side Defence Framework

We design a QUIC client-side defence framework (QCSD) that enables implementing complete website-fingerprinting defences solely from the client.

3.1 Overview

QCSD allows shaping transmissions on HTTP/3-QUIC connections in both directions, whether to or from the client, according to the configured website-fingerprinting defence. Additionally, QCSD's design aims to (1) provide website-fingerprinting resilience similar to that of the stipulated defence, (2) avoid adding additional data and delays to the connection beyond required by the defence, and (3) allow server interoperability by being standards compliant with the QUIC [45] and HTTP/3 [46] specifications.

To enact arbitrary defences, QCSD utilises standardised features of QUIC and HTTP/3 to provide the building blocks of website-fingerprinting defences, identified in Section 2.4.

3.1.1 Sending Chaff and Padding Packets: `PADDING`, `PING`, and Chaff Streams

QCSD performs padding in the client-to-server direction using QUIC's `PADDING` and `PING` control frames. These frames allow sending arbitrary amounts of null-valued data from the client – from a few bytes padding a packet to entire packets of only chaff data – which is discarded by the receiver. As a result of QUIC's pervasive encryption, these control frames are bitwise indistinguishable from normal application data.

In order to pad and chaff in the server-to-client direction, QCSD leverages the facts that HTTP GET requests are idempotent [47]–[49] and that servers host numerous resources beyond those required for the loading of any single web page. Additionally, as per the HTTP/3 standard [46], a single HTTP/3 request-response cycle is conducted upon a single stream and QUIC may place data from individual streams of a connection into the same packet [45]. Together, these allow us to construct, track, and manage streams of chaff data from the server to the client, on which padding and chaff data can be requested independently of application data. We term these streams *chaff-streams* to differentiate them from the application streams carrying the HTTP data of the web page.

3.1.2 Splitting and Delaying: Stream Flow Control

QUIC's implementation as a user-space protocol allows us to control transmissions from the client to the server, without modifying the operating system or making changes that would impact other applications at the client. We can therefore choose how much data and when to send packets originating from the client in accordance with the defence.

For data originating from the server, QCSD utilises the fact that QUIC multiplexes individual streams within a connection,

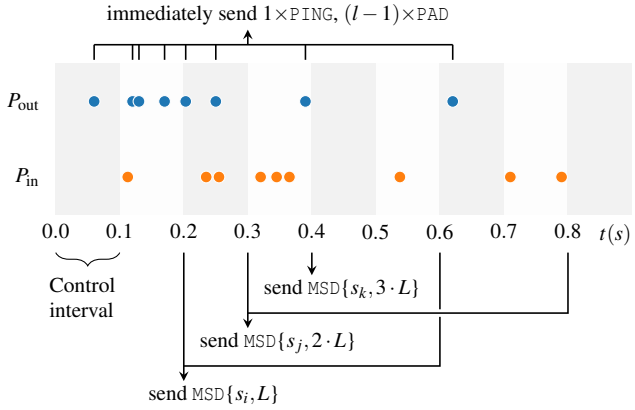


Figure 1: A simplified example of the events generated by QCSD to add chaff traffic to a connection according to the chaff-only defence trace $P = P_{\text{out}} \cup P_{\text{in}}$, with packets of length L, length $l \leq L$, and chaff streams s_i, s_j, and s_k.

where each stream is individually flow-controlled. Delaying packets is accomplished through individual, fine-grained manipulation of this stream-based flow control. A server will only send data for a stream if it has flow control credits for that stream. QCSD uses QUIC's `MAX_STREAM_DATA` (`MSD`) frame to incrementally provide flow control credits to the server on individual streams, which allows delaying or releasing data according to the defence. QCSD splits data to be received as a consequence of this approach, since it releases exactly how much data it would like to receive.

This results, however, in manipulating bursts of bytes which are delivered in one or more packets, instead of in individual packets. Frequently releasing small amounts of flow-control credit would enable finer-grained shaping, perhaps at sub-packet level bursts. However, servers may choose to aggregate received flow credit, thereby limiting such fine-grained manipulation. For example, a server that receives two releases of 600 bytes of flow credit within a short span of time may aggregate its available flow credit and send a single 1200-byte response, instead of two 600-byte responses. QCSD therefore does not send these releases at microsecond frequencies, but instead aggregates the releases at the client and signals the server every few milliseconds. As we show in Section 5, despite aggregating releases QCSD is able to effectively emulate defences from the literature.

3.1.3 Putting it All Together

Figure 1 shows a simplified example of the QUIC packets and events generated by QCSD to add chaff traffic to a connection according to a trace P. These events occur in parallel with the regular communication between the client and the server. Trace events for the outgoing direction are immediately satisfied by the client by sending additional packets, padded with `PING` and `PADDING` frames to create packets of the desired

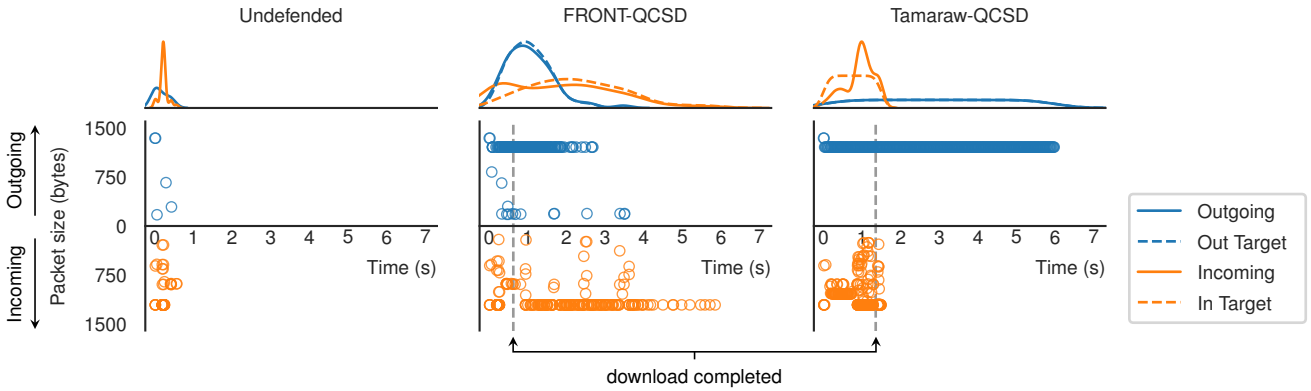


Figure 2: Packet times and sizes for https://www.pcgamingwiki.com/ and dependent QUIC resources when collected using a simple, undefended client vs. shaped with QCSD towards the FRONT and Tamaraw defences. Packet time distributions compared to the target defences are shown above. Packets below 150 bytes (250 bytes with Tamaraw) are not shown.

length, L. Events for the incoming direction are aggregated at the client for a control interval (0.1 s in this example but around 0.005 s in practice) and the data is pulled on an appropriate chaff stream at the end of each control interval by sending `MAX_STREAM_DATA` from the client to the server. For a chaff-and-shape defence, the outgoing packets would also contain the HTTP data required for regular communication, and the streams selected for incoming data would include application streams in addition to chaff streams.

Examples of QCSD shaping the index page and resources of https://www.pcgamingwiki.com/ towards the FRONT [18] and Tamaraw [22] defences are shown in Figure 2 compared to the original traffic pattern and the target schedules generated by the defences. In the next section we describe the design of QCSD and its shaping algorithm in more detail.

3.2 Orchestrating the Defence

QCSD accepts a target trace P corresponding to a defence, a mode of operation m, and a control time interval I. The target trace P may be either a static schedule of packet timestamps and sizes, or may be dynamically generated by the defence at runtime. While QCSD supports both types of targets, for simplicity we describe its operation in terms of a static schedule. The mode m is one of two modes as indicated by the type of defence: "chaff-only" or "chaff-and-shape". In chaff-only mode, the trace P defines the cover traffic to be added to the incoming and outgoing communication. In chaff-and-shape mode, P is the target trace of the overall communication and the traffic is delayed, padded, and otherwise reshaped towards this trace. Table 1 shows an overview of the modes associated with various website-fingerprinting defences. Finally, the control interval I defines how frequently flow-control credits may be released to the server. It balances the granularity of shaping against the overhead of the control messages.

3.2.1 Preparing the QUIC Connection

During initialisation, QCSD modifies the transport parameters of the underlying QUIC connection, which are negotiated between the client and server. It reduces the initial flow-control credit from the server to the client for all bidirectional streams to only 16 bytes. Bidirectional streams transport HTTP/3 requests and responses and thus setting a low initial flow-control credit prepares the streams for shaping by preventing unscheduled transmissions from the server. This is required for chaff-streams in the case of chaff-only mode, and for both chaff and data streams in chaff-and-shape mode. Consequently, each time the client opens a new application (non-chaff) stream to the server in chaff-only mode, QCSD immediately increases the flow control to their non-restricted levels with a `MAX_STREAM_DATA` frame.

3.2.2 The Core Control Loop

Algorithm 1 outlines the central logic for shaping a connection towards the target trace P for the defence and is called with the current time after every control interval I, or at the time indicated by the next event in P. It operates on the *events* of P, that is, the tuples of time, size, and direction of data to be sent or received as indicated by the defence.

Shaping begins by determining whether to pull data from the server to the client for any previously unsatisfied incoming events, pulling as much data as needed or available across open application and chaff streams. Next, it processes each event with a timestamp within the last control interval. QCSD uses data from application streams and chaff streams to fulfil these events. Application streams are only manipulated by QCSD when reshaping the connection (chaff-and-shape) and are shaped in the same manner as chaff streams. Where possible, QCSD prioritises data from application streams over chaff streams to reduce the impact of the defence on the load-

Algorithm 1 Orchestrate the defence corresponding to trace P with mode m at the current time now. The variable ci_end is the end of the scheduled incoming control interval, $astreams$, $cstreams$ are the set of application and chaff streams resp.; and $backlog \geq 0$ is the previously unfulfilled incoming data.

```
1: procedure RunDefence(P, m, now, ci_end, backlog, astreams, cstreams)
2:     pull ← 0
3:     prev_ci_end ← now − (now mod I)
4:   if ci_end ≤ now then
5:       ci_end ← prev_ci_end + I
6:       pull ← backlog
7:       backlog ← 0
8:   end if
9:   while P has an event with a time before now do
10:      event ← next event of P
11:    if event.dir = outgoing and m = chaff-and-shape then
12:        SendPacketOfSize(event.size)
13:    else if event.dir = outgoing and m ≠ chaff-and-shape then
14:        SendChaffPacketOfSize(event.size)
15:    else if event.time ≤ prev_ci_end then
16:        pull ← pull + event.size
17:    else
18:        backlog ← backlog + event.size
19:    end if
20:  end while
21:  if pull > 0 then
22:    if m = chaff-and-shape then
23:        pull ← pull − PullOn(astreams, pull)
24:    end if
25:     pull ← pull − PullOn(cstreams, pull)
26:     backlog ← backlog + pull
27:  end if
28:  if m = chaff-and-shape, IsDone(P), and backlog = 0 then
29:    CloseConnection( )
30:  end if
31:  return backlog, ci_end
32: end procedure
```

ing time of the web page. Next, QCSD prepares to pull data to satisfy events for the incoming direction if their times were within the last control interval, otherwise they are aggregated to be pulled at the end of the current control interval.

Pushing data and chaff For outgoing events of size x bytes, QCSD uses either pending data on open application and chaff streams or QUIC `PADDING` and `PING` frames to satisfy them. For a chaff-and-shape defence, QCSD constructs a QUIC packet of at most x bytes with any pending application data or control frames. If the data that was pending to be sent on these streams was insufficient, the remaining data is sent using QUIC's `PADDING` and `PING` frames. In the case of chaff-only defences, QCSD constructs and sends a QUIC packet of x bytes with *only* `PADDING` and `PING` frames. Each frame is 1 byte in length and may be repeatedly added to a packet. While `PADDING` frames would be sufficient to send the required chaff, placing a `PING` frame in the packet ensures that the packet will be acknowledged by the server (packets with only `PADDING` frames are not acknowledged), thereby mimicking application data from the client.

Pulling data and chaff For incoming events of size x bytes of a chaff-and-shape defence, QCSD requests data from the server on open application streams with pending data, followed by open chaff streams. In chaff-only defences, data is requested only on chaff streams as application streams freely send their data. For each stream i, QCSD tracks the current value of the maximum stream data allowed to the server, msd_i, as well as the total amount of data known to be available on the stream, $limit_i$, through observing received headers and QUIC frames (see Appendix B.3). The difference between these two values, $a_i = limit_i - msd_i$, is the available capacity of that stream to provide data. QCSD selects n application and chaff streams with available capacities $a_1, \ldots, a_n$, to provide the required amount of data, that is $a_1 + \cdots + a_n \leq x$. Next QCSD allows the server to send data on each of these streams by sending a `MAX_STREAM_DATA` frame with the new value $msd_i + a_i$. The server then responds with the data from the various streams aggregated across multiple packets.

Since all outgoing messages are regulated in chaff-and-shape mode, if the defence does not currently also send an outgoing packet, then a small packet of 150 bytes is constructed to transmit the control message. To have the arrival of the data accurately match the defence schedule, the `MAX_STREAM_DATA` frames would need to be sent 1 round-trip-time (RTT) before they are expected. The RTT tracked by QUIC's congestion control algorithm could be used to determine when to send the control frame. However, for the sake of simplicity we assume zero-RTT in our implementation.

Unlike the outgoing direction with its `PADDING` frames, the incoming direction cannot ensure that it will be able to pull all of required data. The amount of data that remains to be pulled is recorded and pulled in a subsequent control interval.

Finishing shaping Finally, once there are no more events for shaping either direction of a chaff-and-shape defence the connection is closed, as expected of defences like Tamaraw [22] and Supersequence [8]. For chaff-only defences, the application continues transmitting until it completes.

3.3 Chaff Streams

A *chaff stream* is a QUIC stream opened by the client over which an idempotent resource, tangential to the loading of the web page, is requested and delivered using an HTTP/3 GET request-response cycle. Chaff streams leverage two properties of QUIC: stream-based flow control and bundling of stream data. Individual stream-based flow control allows an endpoint to specify how much data it is willing to receive over a stream,

and to prioritise data from particular streams. By manipulating the stream-based flow control, QCSD requests data from an individual chaff stream as necessary. QUIC's framing allows frames from multiple streams to be bundled in a single packet.

Our motivation to use chaff-streams within our framework derives from the limitations of the other potential sources of chaff in the HTTP/3-QUIC stack – namely arbitrary, control, or retransmitted data – and is discussed in Appendix B.1.

3.3.1 Chaff Stream Management

Although chaff streams provide a potentially large of amount of chaff, they required overcoming a number of difficulties including identification of potential chaff resources, discovery and tracking of the amount of data available on a stream, and maintaining a pool of chaff streams to provide sufficient chaff.

We envisage that the client caches historical information regarding available resources for a given origin, possibly with their sizes. This information aids in the crafting of the defence at the start of the connection, and thus improves the user's privacy, but is not required for QCSD to operate. Each time the client sends a GET request QCSD records the associated URL and headers. Once the fetch of the URL has been fulfilled, QCSD is aware of the amount of HTTP data that was received on that stream. Using this information, it estimates how much data would be available on a subsequent request for that URL. Previously requested URLs are ranked according to their HTTP payload length and are re-requested to provide available chaff data. Additionally, outgoing requests for chaff resources specify the identity encoding HTTP header ("`Accept-Encoding: identity`"), which ensures that returned content is uncompressed, thereby maximising the amount of data received for each request.

At the beginning of the connection and each time a chaff stream closes, QCSD opens 5–20 streams or as many as will allow it to have around 1 MB of chaff available to request. Appendix B.3 provides more details on the tracking and controlling of chaff streams.

3.4 Shaping Multiple Connections

The mechanisms used by QCSD naturally extend to shape the multiple connections required to download a web page. Consider a defence target P to be applied across n connections $c_1, \ldots, c_n$, each with an arbitrary number of streams. Shaping these connections equates to assigning each event E generated by P to a single connection. That is, for each event E of bytes that should be sent or pulled, QCSD selects a connection c_i to satisfy that event. Since QCSD tracks the amount of pending incoming and outgoing data on each stream (Section 3.2.2, Appendix B.3), it can select the connection based on the availability of data (e.g., the total pending across all streams for a given connection is more than required by the event).

Given multiple connections with sufficient pending bytes to satisfy the event, the question arises as to how to prioritise the assignment of the event to a connection. While we consider the selection and evaluation of such scheduling algorithms beyond the scope of this work, Yu and Benson observed production servers multiplexing QUIC streams in a round-robin fashion [50] and thus we round-robin schedule connections. For a given event, the scheduler checks the pending bytes of each connection in turn and assigns the event to the first connection c_i with sufficient capacity. For the next event, it begins its checks from connection c_{i+1} thereby giving each connection an equal chance of being assigned an event and allowing simultaneous transfer across all connections. We discuss future work on scheduling approaches in Section 7.

4 Dataset Collection

To evaluate QCSD, we implemented a prototype in an HTTP/3-QUIC library and used it to collect datasets of webpage loads defended with FRONT and Tamaraw.

4.1 Implementation

We implemented a prototype of QCSD in Mozilla's Neqo HTTP/3-QUIC client library, which is used in the Firefox browser and written in the Rust programming language [51]. Additionally, we implemented a test client that uses the library to emulate the loading of a web page. It downloads the web page's HTML and resources over one or more QUIC connections while using a pre-generated resource dependency graph to maintain dependency and timing relationships among resources. Emulating the dependencies this way enabled mimicking the complex loading of modern web pages while simplifying our evaluation (see Appendix C for details).

Since QCSD is solely client side and QUIC is a user-space transport protocol, we were able to restrict our implementation to the Neqo user-space library. Along with the implementations of the FRONT and Tamaraw defences in QCSD, each under 200 lines of code, our prototype adds around 3,500 lines of code to the Neqo HTTP/3-QUIC client library. It is available from our repository (`https://github.com/jpcsmith/neqo-qcsd`).

4.2 Datasets

To evaluate QCSD, we collected several datasets with our test client across two settings. In the first setting, *undefended*, we downloaded the web pages without any defence applied. In the second setting, *defended*, we configured QCSD to shape connections towards either the FRONT or the Tamaraw defence. We collected the following datasets in the undefended, FRONT-defended, and Tamaraw-defended settings:

- **Distance datasets ($\mathcal{D}_{dist}$)** For distance-based evaluations, we collected datasets of single traces in each setting for each of 500 unique domains.
- **ML single-connection datasets ($\mathcal{D}_{conn}$)** For machine-learning evaluations, we collected single-connection open-world datasets of 20,000 samples, split across 100 monitored domains and 10,000 unmonitored domains using our test client in each setting.
- **ML multi-connection datasets ($\mathcal{D}_{mc}$)** Additionally, we collected open-world multi-connection datasets of 11,000 samples, split across 100 monitored domains and 1000 unmonitored domains in each setting.
- **ML full-page datasets ($\mathcal{D}_{full}$)** Finally, we collected a set of datasets on full web-page loads. These datasets are of the same composition as $\mathcal{D}_{conn}$ but were collected in the undefended and FRONT-defended settings only.

The datasets consist of packet sizes and timings of web pages from domains in the Alexa top 1m list [52] that support QUIC. We collected the traces through Wireguard VPN gateways, deployed in New York City, USA; Frankfurt, Germany; and Bengaluru, India, to provide a variety of network conditions. For the full-page datasets, $\mathcal{D}_{full}$, we orchestrated the FRONT defence on the full web page by loading the given web page using the Chromium browser and, over the same encrypted VPN tunnel, we added chaff traffic using QCSD with chaff resources from the primary web page.

To apply the FRONT defence, QCSD randomly generated the chaff-schedule using the parameters $N_s = N_c = 1000$ (1300 in $\mathcal{D}_{mc}$, $\mathcal{D}_{full}$), $W_{\min} = 0.5\,\text{s}$, $W_{\max} = 7\,\text{s}$ (0.2 s and 3 s in $\mathcal{D}_{full}$), and chaff-packets of 1200 bytes (1250 bytes in $\mathcal{D}_{mc}$, $\mathcal{D}_{full}$). These were adapted from the original paper [18] to account for reduced latency in our VPN setting and high attack performance against the simulated setting. For Tamaraw, QCSD used parameters similar to the original paper: $\rho_{\text{out}} = 0.020$, $\rho_{\text{in}} = 0.005$, $L = 300$, and packets of 750 bytes in $\mathcal{D}_{dist}$ [22] and 1200 bytes–1250 bytes otherwise. Appendix C provides further collection details.

Simulated setting Finally, we generated a third setting, *simulated*, which simulates the defence as proposed in the literature on undefended traces, for comparison with QCSD's live-defended traces. FRONT simulations combined the chaff-trace generated in the defended setting with the corresponding undefended web-page trace. Tamaraw simulations are the packet schedules generated in the defended setting, as chaff-and-shape defences specify the expected final trace. We accounted for the RTT between the client's transmission of a command and our observation of the server's response by shifting the times of the chaff-schedule (FRONT) or defended trace (Tamaraw) in the server-to-client direction. This shift was between 0 ms and 200 ms and minimised the Euclidean distance between the simulated and defended traces.

5 Shaping Case Studies: FRONT & Tamaraw

We evaluated QCSD's ability to accurately pad or shape HTTP-QUIC connections to two website-fingerprinting defences from the literature: FRONT [18] and Tamaraw [22]. Using the distance datasets $\mathcal{D}_{dist}$, we measured the similarity between each defended trace and its simulated counterpart, as well as the incurred bandwidth and latency overheads.

Trace similarity We measured the similarity between the aggregated time series (Definition 2.2) of the defended and simulated settings using Pearson's correlation coefficient r and the longest common subsequence measure (LCSS). Pearson's r measures linear correlation between two sets of data, and has been used in flow correlation attacks on Tor [53], [54]. LCSS was designed to compare noisy time series data and thus fits well to our use case [55]. Being less common, we describe LCSS below and assume familiarity with Pearson's r.

Definition 5.1. *Longest Common Subsequence (LCSS)* [55] is a measure of similarity between two time series that is robust to noise. It is given by

$$S_{\delta,\varepsilon}(\mathbf{x},\mathbf{y}) = \frac{LCSS_{\delta,\varepsilon}(\mathbf{x},\mathbf{y})}{\min\{|\mathbf{x}|,|\mathbf{y}|\}}$$

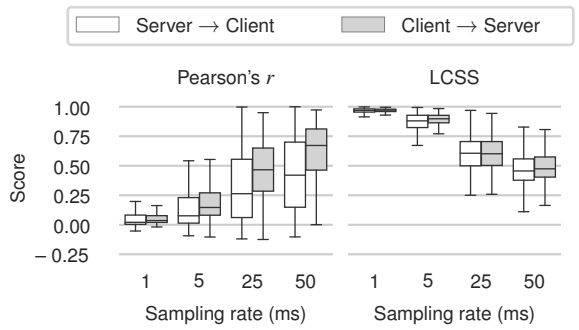
where $LCSS_{\delta,\varepsilon}$ denotes the length of longest matching subsequence between each direction of the simulated and defended aggregated time series $\mathbf{x}$ and $\mathbf{y}$, of lengths $|\mathbf{x}|$ and $|\mathbf{y}|$, and when allowing some elements to be unmatched. $LCSS_{\delta,\varepsilon}$ pairs entries at most δ entries forward or backward in time, and considers them a match if their difference is at most ε. The measure $S_{\delta,\varepsilon}$ ranges from 0 to 1 with 1 representing the greatest similarity. We set ε to 150 bytes to disregard server control packets and framing and left δ unconstrained.

Bandwidth and latency overheads We used the definition of bandwidth and latency overheads present in the literature [8], [18], [20]. They are calculated as the increase relative to the number of transmitted bytes or duration of the undefended communication and are thus unitless. In the case of the latency overhead, the duration of the defended communication is until the last application packet, as the user's experience of delay is unaffected by trailing chaff traffic.

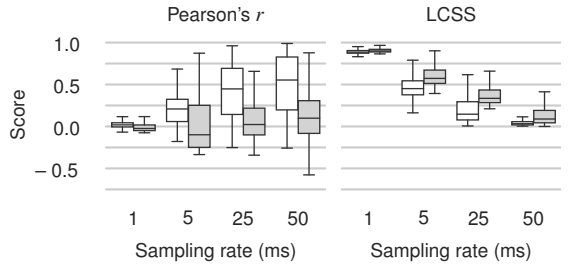
5.1 Chaff-Only Defence 'FRONT'

We evaluated the FRONT defence as an exemplar of chaff-only defences, that is, those that obscure the web page by solely adding chaff traffic to the trace.

QCSD closely emulates FRONT Pearson's r and LCSS between the QCSD-defended and simulated time-series, aggregated at various granularities, are shown in Figure 3a. Pearson's r shows increasing correlation strength with larger granularities, from a weak correlation ($r < 0.3$) at 5 ms to medium



(a) FRONT



(b) Tamaraw

Figure 3: Pearson's r and LCSS between QCSD-collected and simulated time series, aggregated at various sampling rates.

($0.3 < r < 0.5$) and strong ($r > 0.5$) correlations in the incoming and outgoing directions respectively at 50 ms granularity. This increasing trend arises from scheduling and network conditions causing small differences in time between the schedule and when a packet is sent or observed, and thus becomes less pronounced at higher granularities. The overall pattern of sent and received chaff therefore matches the target schedule.

LCSS reports high similarity between the defended and simulated time series (Figure 3a), as it is better tailored to our noisy environment. At both 1 ms and 5 ms granularities LCSS indicated that both the defended and simulated time series were transmitting similar amounts of bytes (within $\varepsilon = 150$) for over 85% of the intervals. The decrease at higher granularities is likely due to the aggregation of multiple packets within a single interval in the calculation. For example, two control packets aggregated in an interval expected to be empty would no longer be within ε of 0 bytes. Nevertheless, the ability of QCSD to closely emulate FRONT is apparent in the high LCSS score at 5 ms, the control interval used to pulled chaff.

FRONT emulation incurs small overheads Next, we measured the latency and bandwidth overhead incurred by QCSD when emulating FRONT. The bandwidth overhead, shown in Table 2, slightly increased from 1.17 in the simulated setting to 1.43 in the defended setting. There are two reasons for this increase. First, unlike in the simulated setting we send control packets from the client and receive acknowledgements from the server, in addition to other control messages necessary for maintaining the connection. We could reduce this overhead by bundling control messages with outgoing chaff as opposed to sending them as normal application traffic. Second, since we cannot construct the packets at the server, we pull the full amount of the desired chaff as stream data, which is additionally encapsulated with QUIC and `STREAM_FRAME` headers. This overhead could be reduced by estimating header lengths and reducing the amount of chaff requested accordingly.

Table 2: Median overheads and their lower and upper quartiles for FRONT and Tamaraw. Simulation overheads marked with * were computed using Cai *et al.*'s Tamaraw simulation [22].

	FRONT	Tamaraw
Bandwidth		
Defended	1.43 (0.58–5.86)	4.16 (1.43–10.6)
Simulated	1.17 (0.48–4.99)	3.09 (0.90–8.35)
	–	*0.58 (0.20–2.44)
Latency		
Defended	−0.12 (−0.22–0.05)	8.43 (3.61–17.3)
Simulated	0.00	*2.34 (0.69–6.98)

In the case of the latency overhead, Table 2 shows that both the simulated and QCSD approaches have overheads near zero, as the addition of chaff packets does not affect the original transmission duration. The small negative latency is likely due to network variations across the defended and undefended settings as the interquartile range spans zero.

5.2 Shaping Defence 'Tamaraw'

At the other end of the defence spectrum lies heavy-weight defences such as Tamaraw [22], which add chaff and shape the traffic from the client and the server to constant rates.

QCSD successfully emulates Tamaraw In Figure 3b, we can see both Pearson's r and LCSS scores for the simulated and defended time series for Tamaraw. Pearson's r reports median correlations of medium strength (∼0.5) in the direction from client to server at 25 ms and 50 ms granularities. The negative Pearson's r in the outgoing direction is indicative of a phase-shift between the compared time series, and can occur, for example, when an odd number of packets are aggregated in an interval in one series but an even in another.

In contrast, LCSS reports high similarities in excess of 0.87 in both the outgoing and incoming directions at 1 ms granularity and around 0.5 at 5 ms granularity. Insufficient data at the server, server aggregation of responses into unequal packets, such as of 1300 bytes and 200 bytes as opposed to two 750-byte packets, as well as network jitter resulting in multiple packets arriving in the same interval likely all contribute to the reduction in the similarity score.

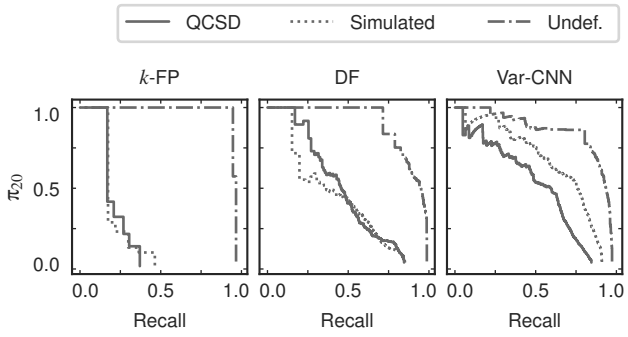
Large bandwidth overheads are even larger Tamaraw's bandwidth overheads are shown in Table 2. The already large bandwidth overhead of 3.09 times the undefended transmission size in the simulated setting is further increased to 4.16 when emulated with QCSD. This increase of around 30% above that of the simulated setting is similar to the increase encountered in FRONT and results from the addition of the control messages necessary to coordinate with the server. In the case of Tamaraw however, the control messages that are sent every 5 ms to the server cannot be bundled with outgoing scheduled packets, as those occurring every 20 ms. At these intervals, QCSD sends QUIC packets of 150 bytes to ensure that control messages are delivered to the server. Since `MAX_STREAM_DATA` control frames have variable length but are at most 17 bytes, and as multiple such frames may be sent in a control packet, profiling for frequently used control frame sizes or bounding the number of streams used to request data in each interval could reduce this overhead. We discuss the overhead difference between the live-generated schedule and the simulated overhead [22] below with the latency overhead.

Tamaraw's latency overhead diverges from the literature The latency overhead, shown in Table 2 for QCSD's emulation of Tamaraw, is a surprising 8.43 times as long as in the undefended case. By contrast, the latency overhead computed using Cai et al.'s Tamaraw simulation reports an expected overhead of only 2.34 in the median.[1] A similar divergence is also seen in the bandwidth overhead. There are two primary reasons for this. First, some web pages do not declare resource lengths *and* fragment them across multiple HTTP/3 data frames. In this case, QCSD cannot determine the length of the resource a-priori and must cautiously releases data on the stream. This primarily affects non-chaff resources in chaff-and-shape defences, as chaff-resource lengths can be cached and chaff-only streams do not throttle other resources. Second, incoming and outgoing directions were simulated independently. In reality however, delays in downloading the HTML of the web page then delays requests for dependent resources which themselves delay the fulfilment of those resources by the server: delays ripple throughout both directions. Therefore, higher transmission rates than used in simulations would be required to achieve lower latency overheads.

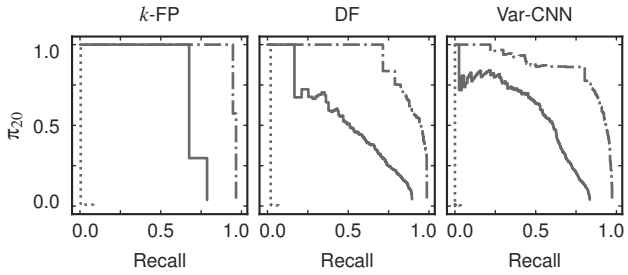
6 Defending against WF Attacks at the Client

Similarity measures cannot capture whether QCSD defences deter website-fingerprinting attacks. We therefore evaluated QCSD's ability to defend against modern attacks – k-fingerprinting (k-FP) [9], Deep Fingerprinting (DF) [12], and Var-CNN [15] – in the open-world setting.

[1] We use Cai et al.'s simulation as a simulation derived from the Tamaraw schedule would have the same latency overhead as the defended setting.



(a) QCSD-FRONT and simulated FRONT



(b) QCSD-Tamaraw and simulated Tamaraw

Figure 4: r_{20}-precision-recall curves of attacks on single-connection datasets defended with simulations or QCSD.

We trained these attacks on datasets from two scenarios: single connections and full web-page loads. For each dataset, we trained on an 80 % stratified split with modest tuning of the classifier hyperparameters from the original papers (see Appendix D). We additionally supplied them with vectors of signed packet sizes (DF and Var-CNN), packet timestamps (Var-CNN), and 165 engineered size- and time-related features [9] (k-FP). We then tested on the remaining 20 % of the dataset and measured recall and precision.

Definition 6.1. *Recall* or *true-positive rate* is the fraction of monitored websites labelled as the correct monitored website.

Definition 6.2. The *precision* of a classifier on a given dataset is the fraction of correct monitored website claims. As this is implicitly coupled to the number of positive and negative samples in the dataset, we use Wang's r-precision (π_r) [39] which adjusts precision to an explicit ratio of monitored to unmonitored websites. We use $r = 20$ corresponding to 1 monitored website visit for every 20 unmonitored websites, consistent with Wang [39].

6.1 Defending Single Connections

To determine QCSD's capability in defending against these attacks, we evaluated them on single-connection datasets $\mathcal{D}_{conn}$.

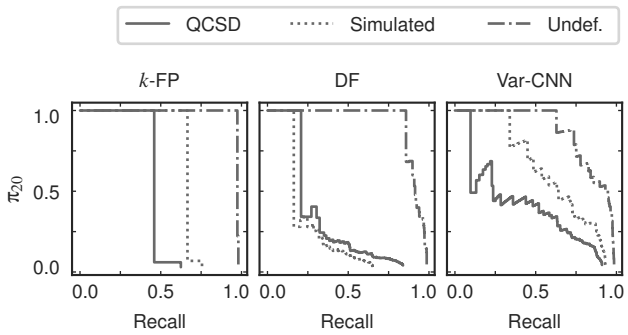
QCSD effectively defends with FRONT Figure 4a shows the precision and recall of the attacks against FRONT in the single-connection setting. Against all three attacks, the level of defence provided by QCSD-orchestrated FRONT is at least that of simulated FRONT. Against an adversary prioritising recall, QCSD reduced precision below 0.2 for recall above 0.75; whereas for an adversary prioritising precision, it reduced recall to below 0.3 for precision above 0.75. Furthermore, in all cases, QCSD provided significantly better defence when compared to the undefended setting, where for 0.75 recall it reduced the precision from over 0.85 to under 0.20. QCSD's improved performance against Var-CNN when compared to the simulation is likely due to the volatility of time feature vectors in the presence of the added control packets.

Inexact server control hinders Tamaraw In contrast to the chaff-only defence FRONT, QCSD fails to match the chaff-and-shape defence Tamaraw in its ability to render the attacks ineffectual. When orchestrated with QCSD, and despite reducing either precision or recall to below 0.4 (DF and Var-CNN) and 0.7 (k-FP) when prioritising the other, QCSD did not achieve Tamaraw's featureless transmission. With a constant outgoing transmission rate, the obvious source of this discrepancy is the inexact shaping from the server. Refinements of the approaches used in QCSD are possible and could improve the performance of a near-constant rate defence such as Tamaraw. Even so, QCSD-Tamaraw provides protection when compared to the undefended setting, reducing precision by 0.6 at recall values in excess of 0.75, and could be bolstered with support to ensure exact-length packets from the server, as discussed in Section 7.

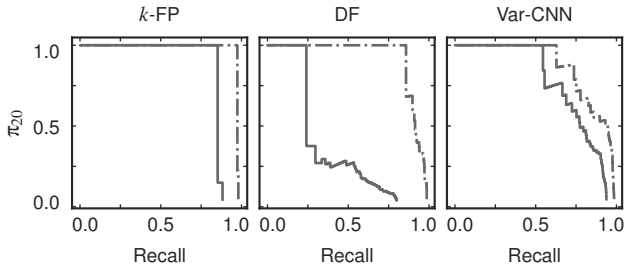
6.2 Defending Full Web-Page Loads

When loading a web page, the browser may open multiple connections to different web servers to request the various resources that constitute the web page. In this section, we evaluate the ability of QCSD to defend full web-page loads, consisting of resources downloaded on multiple connections from different servers, in two scenarios. First, we evaluated the ability of our simple multi-connection QCSD client to defend against attacks when shaping multiple QUIC connections towards the FRONT and Tamaraw defences ($\mathcal{D}_{mc}$). Second, we evaluated QCSD's ability to defend against attacks when loading the entire web page through the browser (QUIC and TCP resources), by crafting FRONT on a single-connection towards the same web server ($\mathcal{D}_{full}$).

QCSD defends multi-connection loads Figure 5 shows attack precision and recall against traces defended with QCSD across multiple connections. Similarly to the single-connection setting, when defending with FRONT, QCSD was able to significantly reduce attack precision and recall scores.



(a) QCSD-FRONT across multiple connections.



(b) QCSD-Tamaraw across multiple connections

Figure 5: r_{20}-precision-recall curves on the undefended and QCSD-defended multi-connection datasets.

By contrast, Tamaraw effectively defended against DF but did not offer comparable performance against k-FP and Var-CNN.

Examination of feature importances in k-FP's underlying random-forest indicated that variance in incoming packet sizes and total incoming data were among the top features used for classification of QCSD-defended Tamaraw. There are two primary reasons for this. The first is discrepancy between the resource sizes recorded in the dependency graph, which was used to determine chaff resources (Appendix B.2), and their size during the evaluation. When the lengths of chaff resources in the cache were overestimated (e.g., a resource unexpectedly delivering an HTTP 404) the difference would leak size information. This discrepancy arose from selecting resources requiring cookies, javascript, being dynamic, etc., and could be addressed with better identification of chaff resources when integrating with a browser. The second is due to accommodating servers that do not respond with data unless they have been provided with some minimum amount of flow credits (Appendix B.3). This leads to QCSD, for example, providing 1500-bytes of flow credit and the server responding with only 800-bytes at the end of the stream. This leakage could be potentially mitigated by identifying these servers and using coarse grained shaping with them and more fine-grained shaping with well-behaving servers. We discuss further improvements in Section 7.

Overall, the clear reductions in attack performance confirm the feasibility of extending QCSD to shaping the multiple con-

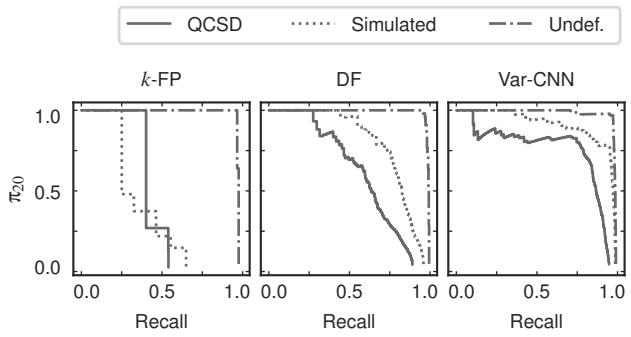


Figure 6: r_{20}-precision-recall curves for classifiers trained on browser web-page loads defended with QCSD-FRONT.

nections required to load full web pages, and exhibits similar overheads to the single-connection setting (see Appendix E).

QCSD defends browser web-page loads When applied to browser web-page loads, QCSD reduced attack efficacy as seen in Figure 6. QCSD's emulation of FRONT reduced the recall of k-FP from 0.95 to only 0.4 at high precisions. Against the deep-learning classifiers, at 0.90 precision it reduced the recall from around 0.95 to 0.10 against Var-CNN and to 0.35 against DF. For an adversary interested instead in high recall, such as 0.90, it reduced the precision from over 0.90 to under 0.35 against both DF and Var-CNN, although Var-CNN was able to maintain higher levels of precision at high recall values. Given the high performance of the deep-learning attacks on the simulated defence, however, better selection of parameters for FRONT could not only improve the defence, but also QCSD's orchestration of it; and further supports the feasibility of using QCSD to defend web-page loads against website-fingerprinting attacks.

7 Discussion

Our QUIC client-side website-fingerprinting defence framework, QCSD, has shown great promise in defending web pages loaded in the VPN setting. In this section, we discuss further potential improvements to QCSD, its use with other protocols, suggested extensions to QUIC, and limitations.

Shaping multiple connections Although QCSD naturally extends to the multi-connection setting, future work is needed to explore algorithms for scheduling the defence across the connections. For example, Cloudflare has been observed sequentially scheduling QUIC streams as opposed to the round-robin approach used by Google and Facebook [50]. Furthermore, prioritising connections that deliver non-terminal resources (those resulting in more HTTP requests) or that speed up rendering of the web page could improve user experience despite the presence of a website-fingerprinting defence.

Embracing the noise QCSD shaping of incoming traffic is fundamentally inexact and exhibits better performance for the noise-based defence, FRONT, when compared to the regularized defence, Tamaraw. For regularized defences, noised variants which perturb the regularized streams, for example, by reducing or increasing the requested amount of chaff or dropping chaff requests, could help to reduce information leakage in these settings when used with QCSD.

Use with other protocols QCSD's approach to shape the traffic from the server could be used with other application protocols besides HTTP/3. The primary requirements would be a common and idempotent method for the client to request data from the server and the possibility for the client to determine the volume of incoming data on a stream.

Suggested extensions to QUIC Many of the challenges in this work revolved around getting the server to send chaff at the client's behest. Despite TLS 1.2 already supporting chaff data on connections, to date there has been no means for an endpoint to request chaff from its remote peer. It is also not possible to request that the peer pads packets to a specified length. With the advent of QUIC and its ability for rapid deployment of extensions, these two features could be feasibly implemented as QUIC extensions. Together, they would provide core building blocks for a client to defend itself against network-based website-fingerprinting attacks.

A need for more accurate simulations Numerous defences in the website-fingerprinting literature use simulation to evaluate their overhead [8], [18], [22]–[24]. For defences that only add chaff traffic, such simulations can be an accurate means of determining the overhead. However this approach is inaccurate for defences that delay the traffic. Without considering the dependency between incoming and outgoing packets it is not possible to accurately estimate the impact caused by the delay of a single packet on subsequent packets, and thus the total time and bandwidth overhead of the defence. More accurate estimations could be found by requiring either all preceding packets, or those within some threshold to have been transmitted under simulation before scheduling a packet, thereby more accurately simulating the introduced delays.

Limitations Our approach has a few limitations. First, we rely upon servers' behaviour of adding multiple stream frames to the same packet, and were otherwise unable to strictly pad individual server packets, as opposed to bursts. Second, servers may limit the number of streams that a client can open, thus restricting the number of chaff streams. This, however, can be mitigated by caching resource lengths across web-page loads to most effectively utilise the available streams. Third, we do not currently shape unidirectional streams from

the client or server. Unidirectional streams are used to satisfy HTTP/3 push promises and to transfer HTTP/3 control messages between the client and server. While we simply disabled HTTP/3 push promises and allowed the settings to be transmitted as normal, these streams could also be shaped in a similar approach to bidirectional streams. And finally, we did not shape resources that are loaded over TCP, but we expect a general transition to and preference of QUIC for all encrypted communication as more servers deploy QUIC.

8 Conclusions

In this work, we designed a QUIC client-side defence framework (QCSD) that enables emulating website-fingerprinting defences without requiring any changes to servers or the deployment of new services. We implemented and evaluated QCSD, which shows great promise in enabling clients to enact defences from the browser or application, without needing to make any changes to existing server stacks. Our evaluations of QCSD show that not only can it orchestrate chaff defences such as FRONT, matching both the chaff-specification and the levels of protection provided by the conceptualised defence, but also to orchestrate chaff-and-shape defences such as Tamaraw. We anticipate that QCSD represents a promising direction for future work on deployable defences.

Acknowledgements

We thank the anonymous reviewers for their many useful suggestions, and in particular our shepherd, Matthew Wright, whose insightful feedback greatly helped to improve the paper. We gratefully acknowledge support from the ETH4D and EPFL EssentialTech Centre Humanitarian Action Challenge Grant. This work was also supported by the National Science Foundation under grants CNS-1553437 and CNS-1704105, and by the United States Air Force and DARPA under Contract No. FA8750-19-C-0079. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force, DARPA, or any other sponsoring agency.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *13th USENIX Security Symp. (USENIX Security 04)*, Aug. 2004. [Online]. Available: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router.

[2] O. Valentine. "VPN usage around the world in 2018," GlobalWebIndex. (Jul. 2, 2018), [Online]. Available: https://blog.gwi.com/chart-of-the-day/vpn-usage-2018/ (visited on Sep. 17, 2021).

[3] S. Feldman. "Entertainment is the main motivator for VPN use," GlobalWebIndex. (Nov. 19, 2018), [Online]. Available: https://www.statista.com/chart/16142/vpn-use-world/ (visited on Sep. 17, 2021).

[4] T. Wang and I. Goldberg, "On realistically attacking Tor with website fingerprinting," in *Proc. Privacy Enhancing Technologies*, Oct. 2016. DOI: 10.1515/popets-2016-0027.

[5] H. Cheng and R. Avnur, "Traffic analysis of SSL encrypted web browsing," University of California, Berkeley, Tech. Rep., 1998. [Online]. Available: http://people.eecs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps.

[6] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Proc. 2002 IEEE Symp. on Security and Privacy*, May 2002. DOI: 10.1109/secpri.2002.1004359.

[7] A. Hintz, "Fingerprinting websites using traffic analysis," in *Privacy Enhancing Technologies*, 2003. DOI: 10.1007/3-540-36467-6_13.

[8] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *23rd USENIX Security Symp. (USENIX Security 14)*, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang_tao.

[9] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symp. (USENIX Security 16)*, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes.

[10] Z. Zhuo, Y. Zhang, Z.-l. Zhang, X. Zhang, and J. Zhang, "Website fingerprinting attack on anonymity networks based on profile hidden Markov model," *IEEE Trans. Information Forensics and Security*, May 2018. DOI: 10.1109/TIFS.2017.2762825.

[11] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. 2018 Network and Distributed Systems Security Symp.*, 2018. DOI: 10.14722/ndss.2018.23105.

[12] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep Fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. 2018 ACM SIGSAC Conf. Computer and Communications Security*, 2018. DOI: 10.1145/3243734.3243768.

[13] R. Attarian, L. Abdi, and S. Hashemi, "AdaWFPA: Adaptive online website fingerprinting attack for tor anonymous network: A stream-wise paradigm," *Computer Communications*, Dec. 2019. DOI: 10.1016/j.comcom.2019.09.008.

[14] S. E. Oh, S. Sunkam, and N. Hopper, "P1-FP: Extraction, classification, and prediction of website fingerprints with deep learning," *Proc. Privacy Enhancing Technologies*, Jul. 2019. DOI: 10.2478/popets-2019-0043.

[15] S. Bhat, D. Lu, A. Kwon, and S. Devadas, "Var-CNN: A data-efficient website fingerprinting attack based on deep learning," *Proc. Privacy Enhancing Technologies*, 2019. DOI: 10.2478/popets-2019-0070.

[16] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet Fingerprinting: More practical and portable website fingerprinting with N-shot learning," in *Proc. 2019 ACM SIGSAC Conf. Computer and Communications Security*, 2019. DOI: 10.1145/3319535.3354217.

[17] A. Qasem, S. Zhioua, and K. Makhlouf, "Finding a needle in a haystack: The traffic analysis version," *Proc. Privacy Enhancing Technologies*, 2019. DOI: 10.2478/popets-2019-0030.

[18] J. Gong and T. Wang, "Zero-delay lightweight defenses against website fingerprinting," in *29th USENIX Security Symp. (USENIX Security 20)*, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/gong.

[19] W. Cui, J. Yu, Y. Gong, and E. Chan-Tin, "Realistic cover traffic to mitigate website fingerprinting attacks," in *2018 IEEE 38th Int. Conf. Distributed Computing Systems (ICDCS)*, Jul. 2018. DOI: 10.1109/ICDCS.2018.00175.

[20] T. Wang and I. Goldberg, "Walkie-Talkie: An efficient defense against passive website fingerprinting attacks," in *26th USENIX Security Symp. (USENIX Security 17)*, Aug. 2017. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tao.

[21] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Computer Security – ESORICS 2016*, 2016. DOI: 10.1007/978-3-319-45744-4_2.

[22] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proc. 2014 ACM SIGSAC Conf. Computer and Communications Security*, 2014. DOI: 10.1145/2660267.2660362.

[23] X. Cai, R. Nithyanand, and R. Johnson, "CS-BuFLO: A congestion sensitive website fingerprinting defense," in *Proc. 13th Workshop Privacy in the Electronic Society*, 2014. DOI: 10.1145/2665943.2665949.

[24] 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 *Proc. 2012 IEEE Symp. on Security and Privacy*, May 2012. DOI: 10.1109/SP.2012.28.

[25] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci, "HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows," in *Proc. 2011 Network and Distributed Systems Security Symp.*, Feb. 6–9, 2011. [Online]. Available: https://www.ndss-symposium.org/ndss2011/httpos-sealing-information-leaks-with-browser-side-obfuscation-of-encrypted-flows.

[26] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proc. 10th Workshop Privacy in the Electronic Society*, 2011. DOI: 10.1145/2046556.2046570.

[27] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic Morphing: An efficient defense against statistical traffic analysis," in *Proc. 2009 Network and Distributed Systems Security Symp.*, Feb. 8–11, 2009. [Online]. Available: https://www.ndss-symposium.org/ndss2009/traffic-morphing-efficient-defense-against-statistical-traffic-analysis/.

[28] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted HTTP streams," in *Privacy Enhancing Technologies*, 2006. DOI: 10.1007/11767831_1.

[29] J. Yan and J. Kaur, "Feature selection for website fingerprinting," *Proc. Privacy Enhancing Technologies*, 2018. DOI: 10.1515/popets-2018-0039.

[30] S. Li, H. Guo, and N. Hopper, "Measuring information leakage in website fingerprinting attacks and defenses," in *Proc. 2018 ACM SIGSAC Conf. Computer and Communications Security*, 2018. DOI: 10.1145/3243734.3243832.

[31] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright, "Tik-Tok: The utility of packet timing in website fingerprinting attacks," *Proc. Privacy Enhancing Technologies*, 2020. DOI: 10.2478/popets-2020-0043.

[32] J.-P. Smith, P. Mittal, and A. Perrig, "Website fingerprinting in the age of QUIC," in *Proc. Privacy Enhancing Technologies*, Jan. 29, 2021. DOI: 10.2478/popets-2021-0017.

[33] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proc. 13th Workshop Privacy in the Electronic Society*, 2014. DOI: 10.1145/2665943.2665950.

[34] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko, "TrafficSliver: Fighting website fingerprinting attacks with traffic splitting," in *Proc. 2020 ACM SIGSAC Conf. Computer and Communications Security*, 2020. DOI: 10.1145/3372297.3423351.

[35] S. Henri, G. Garcia-Aviles, P. Serrano, A. Banchs, and P. Thiran, "Protecting against website fingerprinting with multihoming," *Proc. Privacy Enhancing Technologies*, 2020. DOI: 10.2478/popets-2020-0019.

[36] M. Perry and G. Kadianakis. "Tor padding specification," The Tor Project. (Jul. 6, 2020), [Online]. Available: https://github.com/torproject/torspec/blob/main/padding-spec.txt.

[37] ——, "Circuit padding developer documentation," The Tor Project. (Sep. 14, 2020), [Online]. Available: https://github.com/torproject/tor/blob/main/doc/HACKING/CircuitPaddingDevelopment.md.

[38] D. Schinazi and L. Pardue. "Multiplexed application substrate over QUIC encryption (masque)." (Jun. 14, 2020), [Online]. Available: https://datatracker.ietf.org/wg/masque/about/.

[39] T. Wang, "High precision open-world website fingerprinting," in *Proc. 2020 IEEE Symp. on Security and Privacy*, May 2020. DOI: 10.1109/SP.2020.00015.

[40] W. Cui, T. Chen, C. Fields, J. Chen, A. Sierra, and E. Chan-Tin, "Revisiting assumptions for website fingerprinting attacks," in *Proc. 2019 ACM Asia Conf. Computer and Communications Security*, 2019. DOI: 10.1145/3321705.3329802.

[41] J. Roskind, *Experimenting with QUIC*, 2013. [Online]. Available: https://blog.chromium.org/2013/06/experimenting-with-quic.html (visited on Oct. 22, 2019).

[42] A. Ghedini and R. Lalkaka. "HTTP/3: The past, the present, and the future." (Sep. 2019), [Online]. Available: https://blog.cloudflare.com/http3-the-past-present-and-future/ (visited on Oct. 22, 2019).

[43] M. Yakan and A. Jayaprakash. "Introducing QUIC for web content." (Oct. 31, 2019), [Online]. Available: https://developer.akamai.com/blog/2018/10/10/introducing-quic-web-content.

[44] M. Joras and Y. Chi. "How Facebook is bringing QUIC to billions," FACEBOOK Engineering. (Oct. 21, 2020), [Online]. Available: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

[45] J. Iyengar and M. Thomson, *QUIC: A UDP-based multiplexed and secure transport*, RFC 9000, May 2021. DOI: 10.17487/RFC9000.

[46] M. Bishop, "Hypertext transfer protocol version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-27, Feb. 2020. [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-http-27.

[47] R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Semantics and content," RFC Editor, RFC 7231, Jun. 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7231.txt.

[48] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (HTTP/2)," RFC Editor, RFC 7540, May 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7540.txt.

[49] R. T. Fielding, M. Nottingham, and J. Reschke, "HTTP semantics," IETF Secretariat, Internet-Draft draft-ietf-httpbis-semantics-16, May 2021. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-httpbis-semantics-16.txt.

[50] A. Yu and T. A. Benson, "Dissecting performance of production quic," in *Proceedings of the Web Conference 2021*, 2021. DOI: 10.1145/3442381.3450103.

[51] Mozilla Corporation. "Neqo, an implementation of QUIC written in Rust." (Oct. 9, 2020), [Online]. Available: https://github.com/mozilla/neqo.

[52] *Alexa top 1M sites*, Alexa Internet, Inc., Jul. 18, 2021. [Online]. Available: http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

[53] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *Financial Cryptography*, 2004.

[54] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *Computer Security – ESORICS 2006*, 2006.

[55] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Proc. 18th Int. Conf. Data Engineering*, Feb. 2002. DOI: 10.1109/ICDE.2002.994784.

[56] P. Probst, M. N. Wright, and A.-L. Boulesteix, "Hyperparameters and tuning strategies for random forest," *WIREs Data Mining and Knowledge Discovery*, 2019. DOI: https://doi.org/10.1002/widm.1301.

A Website-Fingerprinting Defences

In this section we define the defences used throughout the paper in more detail.

FRONT The FRONT defence by Gong and Wang [18] obfuscates the feature rich front segment of communication without otherwise delaying the traffic sent by the endpoints. It does so by adding n_c and n_s chaff packets from the client and server respectively with timestamps selected according to the Rayleigh distribution, so as to prioritise adding the packets at the beginning of the communication. These timestamps are given by

$$f(t;w) = \begin{cases} \frac{t}{w^2} e^{-t^2/2w^2} & t \geq 0, \\ 0 & t < 0 \end{cases}$$

for $w_c, w_s \sim \mathcal{U}(W_{\min}, W_{\max})$, $n_c \sim \mathcal{U}\{1, N_c\}$, and $n_s \sim \mathcal{U}\{1, N_s\}$ sampled from continuous and discrete uniform distributions with defence parameters $N_c, N_s, W_{\min}, W_{\max}$.

Tamaraw The Tamaraw defence by Cai *et al.* [22] sends traffic in fixed-size packets (750 bytes) and at fixed intervals. Tamaraw sends packets from the client to the server at a rate of ρ_{out} seconds per packet, and from the server to the client at a rate of ρ_{in} seconds per packet. The values ρ_{in} and ρ_{out} are chosen such that $\rho_{\text{in}} < \rho_{\text{out}}$, since web servers often have more data to transmit than web clients. Additionally, in Tamaraw, the number of packets sent in each direction is padded to a multiple of L packets. This helps mask the total transmission time of web pages, and partitions the set of all web pages into anonymity sets based on their transmission time.

B Selecting and Using Chaff

In this section, we motivate QCSD's use of chaff resources and describe the selection of chaff resources and the process behind tracking and controlling streams.

B.1 Rejected Sources of Chaff

There are three possible sources of data that a server may transmit as chaff or padding: arbitrary (e.g., random or constant-valued), control, or application data. QUIC provides `PADDING` frames of null data but QUIC endpoints are unable to request that these frames be sent by their remote peer. Additionally, although QUIC has an assortment of control frames that may be sent from the server in response to a message from the client, none of these control frames invoke a sizeable response from the server without an equally sized message from the client. We therefore leverage application data to provide chaff traffic from the server to the client.

Application data can take two forms, either new data or retransmissions. Retransmissions are performed by the remote endpoint if data is considered lost, and is employed in HTTPOS [25] as a means of adding chaff to a connection. Each loss event, however, results in a reduction of the transmission rate from the server. Not only does this rate reduction negatively impact the ability to generate more chaff packets, and thus to shape the connection, but an adversary may also be able to identify the chaff due to the change in transmission rate. Furthermore, QUIC detects losses at the granularity of packets, not bytes of data, we would therefore need to induce packet losses that sum towards our desired amount of chaff. Consequently, we leverage new data in form of *chaff streams* to pad bursts and add chaff traffic originating from the server.

B.2 Selecting Chaff Resources

In our evaluations, we utilised the knowledge of the resources from the collected dependency graphs to select the initial resources used to provide chaff, namely, the resource type and observed length. Given a potential set of chaff resources, we prioritised images, as they are relatively static and can provide large amounts of data when uncompressed, followed by fonts, scripts, and style-sheets, and finally HTML documents, as they may be dynamically generated. In a real-world deployment, information about the potential chaff resources located at a domain, along with HTTP cache-control details, could be cached from previous connections to the domain, and would thus be available to QCSD at the start of a new connection.

B.3 Tracking and Controlling Streams

Accurately shaping a connection requires knowledge of the amount of data available at the remote server. QCSD therefore tracks information about application and chaff streams that are opened by the client and the framework. It considers a stream to be either throttled or unthrottled. A throttled stream is a stream for which QCSD controls the server's release of data, such as a chaff stream or an application stream being manipulated. An unthrottled stream is one that is not being manipulated, but for which we would like to track the resource length, such as all application streams in chaff-only mode.

Figure 7 shows the state transitions of the receiving side of a stream tracked by QCSD. QCSD begins tracking a stream once an HTTP-GET request has been prepared for a resource, and a new QUIC stream has been created at the client to transfer the request. Tracking ends once the `FIN` bit has been received or an error is encountered on that stream.

Throttled streams When the first byte is sent on a throttled stream, QCSD considers it to be receiving HTTP/3 headers. In this state, data is slowly released until the length of the resource is discovered. QCSD discovers the length of the resource when the stream encounters the length of an HTTP/3 data frame which signifies the start of the HTTP payload. At this point the stream is considered to be throttled but receiving data. The resource of a throttled stream that never arrives at the receiving data state has no length and is thus avoided when re-requesting resources on future chaff streams.

To receive data on a throttled stream while shaping the connection, QCSD increases the amount of flow-control credit provided to the server on that stream by sending a `MAX_STREAM_DATA` (`MSD`) frame with the new limit. In the ideal world, the client would know exactly how much data is available at the server to be pulled. Unfortunately, in reality the size of the stream is often unknown, which can result in requesting more data than the stream is able to provide. The difference between the amount requested and the data provided leaks information about the length of a stream. To avoid this, for each throttled stream QCSD tracks the current amount of flow control that has been released to the server, *msd*, the maximum amount of data it knows to be available on the stream, *limit*, and the amount of data consumed, *c*. As data is read from the QUIC stream at the receiver QCSD increases *c*; and each time that an HTTP/3 frame header is parsed indicating the length of the HTTP/3 frame, QCSD updates the known length of the stream, *limit*, accordingly. This knowledge is further supported by the HTTP `content-length` header that, if provided in the response header from the server, identifies the length of the resource that will be transmitted. The difference between the amount we know to be available and the current max stream data offset, i.e., $limit - msd$, is the available capacity of that stream to be used for shaping.

At times, servers may refuse to send any data for a stream unless some minimum amount of flow credit is available. We therefore specify a value, *excess*, that defines an amount of data that QCSD assumes the server will have available on the stream, beyond the known outstanding volume of data. QCSD ensures that *limit* is always at least $c + excess$, thereby allowing the client to provide `MSD` to the server to send any remaining data. Discovery and tracking of the amount of data to be delivered on the stream allows us to limit the role this plays in transmission, as for most of the transmission $limit \gg c + excess$.

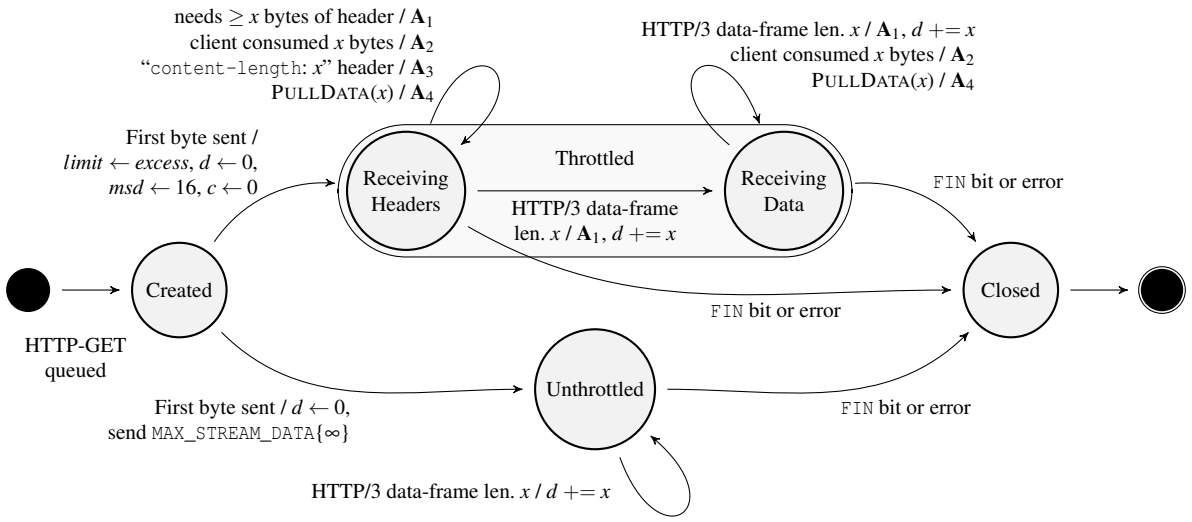


Figure 7: State transitions for chaff and application streams in QCSD. Denoted are the current max stream data offset, *msd*, the amount of bytes believed to be available at the server, *limit*, the amount of data consumed by the client on the stream, c, and the amount of data observed on the stream, d. Annotations are of the form "triggering event / resulting action", with the actions marked as $\mathbf{A}_1$–$\mathbf{A}_4$ denoting $\mathbf{A}_1$: $limit \leftarrow \max\{limit, c+x\}$; $\mathbf{A}_2$: $c \leftarrow c+x$, $limit \leftarrow \max\{limit, c+excess\}$; $\mathbf{A}_3$: $limit \leftarrow \max\{limit, x\}$; and $\mathbf{A}_4$: $msd \leftarrow msd + \max\{limit - msd, x\}$, send `MAX_STREAM_DATA`$\{msd\}$.

Unthrottled streams When the first byte is sent on an unthrottled stream, QCSD immediately sends a `MAX_STREAM_DATA` frame to increase the low initial max stream data offset set during initialisation. The max stream data offset is increased to allow the server to send as much data on the stream as in an unmodified QUIC connection. Further releases of max stream data is then handled by the original QUIC client logic. QCSD also begins tracking the amount of HTTP payload seen on that stream. Once this has been determined, the resource that was loaded over that unthrottled stream can be re-requested on a chaff stream.

C Further Details on Dataset Collection

QCSD shapes live QUIC connections and thus required a collection of QUIC-enabled web pages on which to be evaluated. We therefore identified a set of domains from the Alexa top 1m list [52] that supported QUIC and which version of the protocol each supported. To do so, we requested the web page in Python using HTTPS over TCP and recorded the HTTP alt-svc record with which web servers identify other supported protocols. We then filtered the results to web pages that advertised the "h3-29" tag, i.e. HTTP/3 over IETF QUIC draft-version 29, as this was the latest version supported by the underlying QUIC library that we used. Certain domains, such as `*.blogspot.com` and `*.appspot.com`, proved prevalent in our results and were therefore downsampled. Next, we requested these web pages using QUIC in Chromium (commit 870763) and logged the resources requested during the loading of the page. From these logs, we created a dependency graph of requested resources. We considered a URL A a dependency of URL B if A was an HTTP referrer of B, if A initiated the request of B (such as through include statements in CSS), or if B's request was the result of a sequence of JavaScript function calls that included the script at URL A. We filtered this graph to only URLs with the same origin of the final redirected URL (and thus be requested over the same connection) and removed graphs that redirected to the same page. Finally, these dependency graphs were passed to our test client to collect the actual network trace; and for each trace, we recorded the packet sizes and timestamps at the client using the tcpdump network monitoring utility.

D Hyperparameter Optimisation

For each of the three machine-learning evaluation scenarios (single-connection, multi-connection, and full web-page loads), we performed modest hyperparameter tuning to explore the potential for improvements in the attacks against QCSD defended traces. For each attack and dataset we performed a grid search over the selected hyperparameters and measured the mean F_1-score (using r_{20}-precision and recall) for each parameter combination using 3-fold cross-validation.

For the k-fingerprinting attack the parameters selected were the number of nearest-neighbours $\{2,3,6\}$ used to vote on a class label and, in the underlying random forest, the number

Table 3: Median overheads and their lower and upper quartiles for FRONT and Tamaraw in the multi-connection setting. Simulation overheads marked with * were computed using Cai *et al.*'s Tamaraw simulation [22].

	FRONT	Tamaraw
Bandwidth		
Defended	1.29 (0.60–3.45)	6.13 (3.17–13.12)
Simulated	0.90 (0.39–2.40)	4.88 (2.47–10.92)
	–	*0.86 (0.36–2.26)
Latency		
Defended	−0.05 (−0.15–0.08)	6.64 (2.98–11.86)
Simulated	0.00	*0.91 (0.17–2.92)

of estimators $\in \{100, 150, 200, 250\}$, the number of sampled features per estimator $\in \{2, 12, 20, 30\}$, whether to use out-of-bag scoring, and the fraction of samples on which to train each estimator $\in \{0.5, 0.75, 0.9, 1.0\}$, as they have been shown to provide the greatest benefit [56]. Scoring all 384 parameter combinations under 3-fold CV required around 40 minutes for each defended and undefended dataset on a server equipped with 2 Intel Xeon Gold 6242 CPUs (2.80 GHz, 64 cores).

For the time and size classifiers of the Var-CNN attack and for the Deep Fingerprinting attack, we tuned the number of packets used as features to these classifiers, as the defences increased the number of packets transmitted and thus may have shifted important features. We therefore evaluated each classifier with 5000 (original), 7500, and 10000 packets. Scoring all 3 parameter combinations under 3-fold CV for each defended dataset and classifier required from 2 to 5 hours on an NVIDA GeForce RTX 3060 (2021, 12 GB). Undefended datasets were evaluated on parameters taken from the original attack papers and so the measured efficacy of the attacks on these datasets are more conservative.

E Overhead in the Multi-connection Setting

We evaluated the bandwidth and latency overhead in the multi-connection setting by collecting 500 additional web pages. For each defended version of the web page, an undefended sample was collected immediately afterwards and the relative increase in latency and bandwidth was calculated as described in Section 5. The results shown in Table 3 are in accordance with the shift from the single to multi-connection. For FRONT, increasing the application data while sampling the amount of chaff from the same distribution resulted in a slight decrease in the relative bandwidth overhead. For Tamaraw, an increase in the bandwidth overhead was observed with a decrease in the latency overhead.

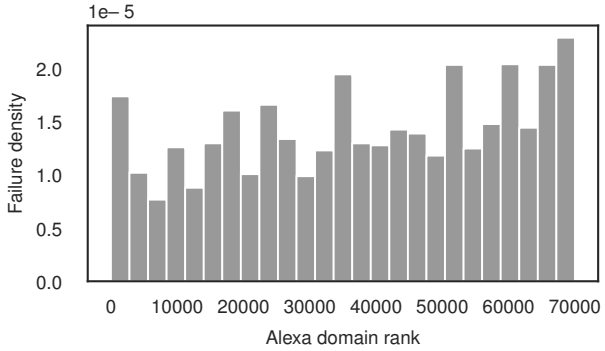


Figure 8: Distribution of the 586 failures across the ∼10,800 Alexa domains requested with FRONT. Samples in each bin were weighted according to the frequency of QUIC domains observed in that bin, and show a trend towards less popular domains having more failures.

F Server Compliance with Shaping

Despite being complaint with the QUIC standard [45], QCSD may encounter web servers whose parameters or configuration prevents shaping of the connection. This may take the form of, for example, insufficient stream numbers allocated by the server, or connection closure due to server disagreement with the QUIC transport parameters. We therefore recorded the failure rate of web-page downloads when collecting the datasets used for the single-connection machine-learning evaluations from Section 6.1.

Across the 10,790 web pages downloaded with FRONT, only around 5.43 % of the attempted downloads failed that did not also fail when not shaping the traffic. Meanwhile, among the around 13,200 web pages that loaded with Tamaraw, 5.81 % failed after accounting for timeouts in our collection procedure (22.70 % without), which were due to downloads exceeding 2-minutes as a result of the cumulative delays introduced by Tamaraw, as discussed in Section 5. Furthermore, these failures had a higher density among the less popular Alexa domains, as seen in Figure 8.

With a failure rate less than 6 %, QCSD is accepted by most servers that already deploy QUIC; and has greater potential for deployment than existing website-fingerprinting defences. Furthermore, since it utilises standard-compliant features of the protocols, we anticipate that success rates will increase as implementations transition towards the standard.