This is not the padding you are looking for! On the ineffectiveness of QUIC `PADDING` against website fingerprinting

Ludovic Barman[*] [‡] Sandra Siby[*] Christopher Wood[†] Marwan Fayed[†] Nick Sullivan[†] Carmela Troncoso[*]
[*]*EPFL* [†]*Cloudflare Inc.*

Abstract

Website fingerprinting (WF) is a well-know threat to users' web privacy. New internet standards, such as QUIC, include padding to support defenses against WF. We study whether network-layer padding can indeed be used to construct effective WF defenses. We confirm previous claims that network-layer padding cannot provide good protection against powerful adversaries capable of observing all traffic traces. In contrast to prior work, we also demonstrate that such padding is ineffective even against adversaries with partial view of the traffic. Network-layer padding without application input is ineffective because it fails to hide information unique across different applications. We show that application-layer padding solutions need to be deployed by both first and third parties, and that they can only thwart traffic analysis in limited situations. We identify challenges to deploy effective WF defenses and provide recommendations to reduce these hurdles.

1 Introduction

New standardization efforts have greatly increased the privacy of web traffic: *e.g.*, Encrypted Client Hello (ECH) [1] to encrypt Server Name Indication (SNI), or (Oblivious) DNS-over-HTTPS [2, 3] and DNS-over-TLS [4] to encrypt DNS queries. Yet, encryption alone cannot protect users' browsing history from traffic analysis. Traffic-analysis attacks such as *website fingerprinting* (WF), enable adversaries to infer which websites a user visits from the traffic patterns (*e.g.*, volume of packets exchanged or packets' size) [5, 6, 7, 8, 9, 10].

"To provide protection against traffic analysis [...]", the working group behind QUIC, the next transport layer standard for the Web, introduced a `PADDING` frame in the specification [11]. In this work, we demonstrate that defenses solely based on padding QUIC traffic are ineffective against WF attacks. This is because the most important feature is the total size of the websites, which is not known at the network layer and therefore cannot be effectively protected without application-layer information. We then study the potential of application-layer defenses and identify the obstacles that impede their deployment at large scale.

Our contributions are as follows:

- ✓ We show that, even against a limited adversary who only observes partial web traffic traces and has realistic computation capabilities, network-layer defenses that use the `PADDING` frame to hide the size of packets and inject dummy packets cannot prevent adversaries from inferring users' browsed web pages. These characteristics are sufficient to enable website identification with high performance (> 92% F1 score). We conclude that network defenses cannot effectively hide traffic global statistics unless there is communication with the application layer to provide this information (*e.g.*, the total number of packets, or the total incoming size).
- ✓ We study whether the capabilities of the adversary affects the attack performance. We show that traffic visibility is of paramount importance; and few ASes have vantage points to observe *full* traffic traces to run these attacks. These ASs would incur high costs to collect and transmit traces to a node with enough analysis capability. Yet, we show that even using limited information from typical network statistics (*e.g.*, NetFlow) are sufficient to perform high-performance traffic analysis.
- ✓ We show that the centralization of web resources on the Internet, in particular in the hands of Google, increases the traffic analysis threat: traffic analysis on solely the timing of Google resources fetched by a webpage achieves > 77% F1 score while reducing the adversary's cost by four orders of magnitude. Moreover, any AS between the client and Google, in addition to ASes between the client and the first-party domain host, can now perform attacks.
- ✓ We explore whether application-layer defenses, *i.e.*, resource manipulation and resource addition at the server and client, are a viable approach to thwart fingerprinting. We show that any defense has to be applied to *all* first- and third-party domain traffic. In most cases, application-layer

[‡] Ludovic Barman is currently at Google.

defenses suffer from the same issues as network-layer defenses due to difficulty in hiding global statistics. In the cases where application-layer defenses are more effective, they are impaired by deployment challenges due to current practices to develop and host web resources.

✓ We identify key deployment challenges associated with current web practices. We provide recommendations to guide future efforts in designing building blocks and defenses against website fingerprinting attacks.

Ethical considerations: We conduct traffic-analysis attacks against a deployed technology (QUIC). We do not perform any collection or analysis of real users' traffic. We only collect our own traffic, generated by an automated browser. We uncover vulnerabilities in the proposed defenses, which would put at risk network users if deployed. We believe that the benefits of our research which can guide current and future standardization efforts to outweigh these risks, by avoiding deployments that could give users a false sense of security. We have performed responsible disclosure of our findings to QUIC's IETF WG.

2 Background & Related Work

QUIC. QUIC is a connection-oriented protocol built on top of UDP that aims to provide low-latency, multiplexed, secure communication with less head-of-line blocking and faster connection migration [12]. QUIC was standardized in May 2021 and is currently being developed by the IETF. QUIC is the transport protocol for HTTP/3. Adoption of QUIC and HTTP/3 has been rising (as of Oct 2021, they are used by 21.3% of the top 10 million websites [13]). Of particular relevance for our work is the QUIC `PADDING` Frame. The IETF QUIC draft describes it as a frame with no semantic value, that can be used to increase packets size, and to provide protection against traffic analysis [11]. We investigate whether this frame is suitable to protect QUIC traffic against website fingerprinting attacks.

Website fingerprinting attacks. In website fingerprinting, an adversary aims to infer the website visited by a user by analyzing network traffic. The adversary builds a classifier trained on features obtained from website network traces. These features can be selected manually or via automatic extraction.

The most relevant attacks that rely on manual extraction are Wang et al.'s k-Nearest Neighbors (k-NN) classifier based on 3000 manually-selected features [14]; Panchenko et al.'s Support Vector Machines (SVMs)-based classifier, CUMUL, based on cumulative sums of packet lengths [15]; and Hayes and Danezis [16] *k*-fingerprinting method (*k*-FP), which models web fingerprints as the leaves of a random forest built on 150 manually-selected features.

Rimmer et al. [17] were the first in demonstrating that and found that automated feature extraction using deep learning neural networks (DNNs) results in attacks that perform as well as manual approaches and are more robust. Sirinam et al.'s [18] built on this observation to develop an attack that achieves high accuracy even in the presence of defended traces. Var-CNN by Bhat et al. [19] show that it is possible to achieve high accuracy with deep learning even in the presence of limited data, relying on ResNets trained on packet directions, packet times, and manually extracted summary statistics.

Website fingerprinting on QUIC traffic. Smith et al. [10] studied the impact of co-existence of TCP and QUIC on the performance of website fingerprinting using *k*-FP and Var-CNN. They concluded that, while QUIC traffic is not difficult to fingerprint, classifiers trained on TCP traffic do not perform well on QUIC traffic, and that jointly classifying both protocols is hard. They also show that *k*-FP outperforms Var-CNN in presence of QUIC. To enable comparison with the state-of-the-art on QUIC website fingerprinting [10], we also use *k*-FP and Var-CNN in our evaluation. We avoid more recent approaches [18, 20] that could yield better performance, as it comes at the cost of explainability. Since our aim is to understand the impact of the attack on padding defenses, we favor explainability over performance.

Website fingerprinting defenses. Dyer et al. [21] showed that *network-layer defenses* padding and morphing based countermeasures are ineffective in thwarting traffic analysis since they fail to hide coarse packet features. They proposed Buffered Fixed-Length Obfuscation (BuFLO) which pads packets to a fixed size at sends them at intervals of time. This results on a huge overhead, which was later reduced by CS-BuFLO [22] and Tamaraw [23], still being impractical. Works such as WTF-PAD [24] and FRONT [25] reduce this overhead by injecting dummies at appropriate positions in a trace. WTF-PAD injects dummy packets using a pre-defined distributions of inter-arrival times to detect gaps, and FRONT injects dummy packets in the front portion of traces, which are known to contain the most information for fingerprinting. Both approaches achieve low protection against DNN attacks [18, 19, 26].

Alternative approaches employ *adversarial perturbations* to cause DNN classifiers to misclassify traces. These defenses incur lower overhead than prior work. Mockingbird [27] applies perturbations to convert traces into a target. It converts traces into burst sequences of packets (where a burst is a set of contiguous packets in one direction), and perturbs these bursts rather than the raw trace. Because Mockingbird is designed for Tor traffic, it only consider packet directionality when building dummy bursts, and ignores packet sizes. Thus, it is unclear how to adapt it to QUIC traffic: a burst can correspond to many QUIC packet sequences. Moreover, to compute the perturbation, Mockingbird requires the defender to know the entire trace in advance, which is infeasible in practice.

Nasr et al. [28] tackle this issue by pre-computing blind perturbations for unseen traces. Shan *et al.* improve upon it with Dolos [26]: it apply adversarial patches or pre-computed sequences of dummy packets to protect network traces. Unfortunately, we could not run the code by Nasr et al. [29] and Shan et al. have yet to release their code. Thus, in our work, we resort to simple randomized dummies in line with WTF-PAD [24].

Other defenses, focus on the *application layer*. Luo et al. [30] developed HTTP Obfuscation (HTTPOS), a client-side defense that modifies features on the TCP and HTTP layers and uses HTTP pipelining to obfuscate HTTP outgoing requests. Randomized Pipelining [31] improves this defense by randomizing the order of the HTTP requests queued in the pipeline. Subsequent works have shown HTTPOS and Randomized Pipelining to be ineffective in protecting against traffic analysis attacks [14, 32]. Cherubin et al. [33] developed client- and server-side defenses, LLaMA and ALPaCA respectively, tailored towards onion services, i.e., scenarios with low third party content prevalence, lack of dynamic page content, and JavaScript disabled. In our work we propose defenses inspired by ALPaCa, and study its performance in web scenarios where these assumptions do not hold (see section 7).

3 Adversarial Model

We assume a local passive eavesdropper A located at some vantage point between an honest client and an honest Web host. A observes all network traffic passing through this vantage point and records some portion of it. The goal of the adversary is to infer the visited domain of HTTPS queries. A does not possess any decryption keys, and relies only on the size and timing of the observed packets.

The adversary A observes IP packets. A focuses on Web traffic and filters out packets that are not TLS or QUIC packets. Using the appropriate fields in the headers (IP addresses, ports, QUIC connection IDs), A is able to identify packets that belong to the same connection. We assume that DNS queries are done in a private manner (*e.g.*, via DoH [2] and appropriate padding [9]) and reveal no information to A.

We call A's *observation* a collection of flows corresponding to the network connections generated when the user browses a single website. Each flow contains $[\text{IP}_{\text{source}}, \text{IP}_{\text{dest}}, p_1, p_2, \dots]$, where packets p_i are (time,size)-tuples $(t_i, \pm s_i)$. Negative sizes indicate packets from the server to the client, and positive sizes packets in the opposite direction.

Vantage Points. Following prior work [34, 35, 36] we consider each AS on the path of the client traffic as a realistic adversary. Each AS' middlebox, router, or switch that routes traffic from a client is a potential vantage point for the adversary to collect this client's traffic.

In Figure 1, we depict a client located in AS X accessing two webpages hosted on an IP in AS W. The client traffic is represented by red lines. If the adversary controls AS X, they observe all the traffic related to the page visits. This is the adversary typically considered in the website fingerprinting literature [5, 6, 7, 8, 10, 14, 15, 16, 17, 18, 19]. If the adversary controls AS Y or AS Z, however, they would have limited visibility on the traffic, *i.e.*, they might not observe traffic from *all* clients' visited webpages, or for each observed webpage, they might only observe a portion of the traffic (*e.g.*, the loading

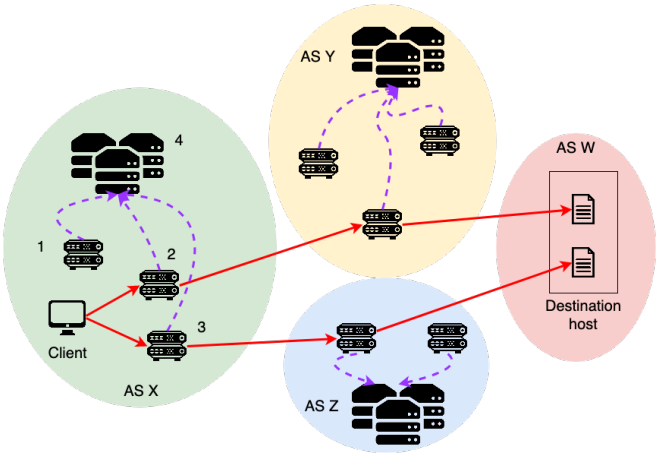

Figure 1: An adversary can be on any AS (X, Y, Z, or W) with vantage points on the client's traffic path (solid red arrows). The vantage points (*e.g.*, middleboxes or routers) transmit recorded data to a location that can perform traffic-analysis at scale (dotted purple arrows). If the adversary is AS X, vantage points 2 and 3 transmit recorded traffic to location 4.

of some resources). We note that it is possible for an adversary to control multiple ASes or an IXP (where traffic from multiple ASes can traverse) [35, 36].

Website Fingerprinting Attack. As in previous work, we implement website fingerpring attacks as a supervised learning problem. The adversary identifies the IP that contains the domains that they want to target. Then, the adversary enumerates all domains on that IP, and collects web traffic traces from these domains. The adversary extracts features from these traces and uses the feature vectors to train a classifier. Given a new trace, this classifier predicts to which website it belongs.

We implement the attack using a random forest classifier, a simple and effective model frequently used in website fingerprinting; and Var-CNN [19] to validate our results against the state of the art [10]. We use the comprehensive set of features proposed by Hayes *et al.* [16] for performing website fingerprinting on Tor since this is a comprehensive set of features from previous related works. To adapt them to the QUIC case, we add features about packet size frequencies. Since QUIC's maximum payload size (1400 B) is smaller than that of TLS (16 kB), we compute frequencies of packet sizes up to 16 kB to encompass both QUIC and TLS traffic.

Closed World. As opposed to the anonymous communications setting in which the adversary cannot observe the destination IP of clients request [37], in the QUIC setting the adversary observes these IP addresses. This information reduces the anonymity set of the websites visited by the user from the whole Internet to the websites hosted at a particular IP. Since the adversary can enumerate the domains hosted in one IP (*e.g.*, through DNS reverse lookups), we run our analysis in a *closed-world* scenario: the adversary can train their classifier on all the websites hosted on one IP.

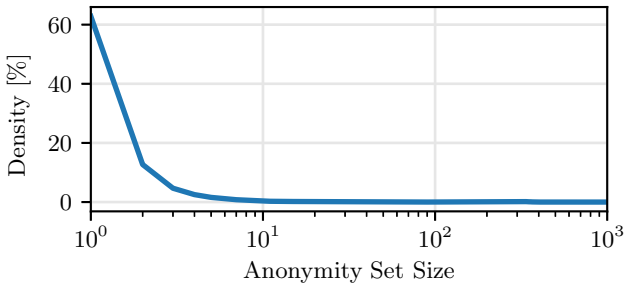

Figure 2: Distribution of the cluster sizes of 1.3M domains.

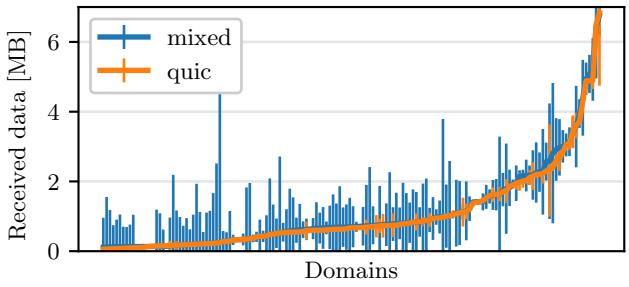

Figure 3: Comparison of the incoming total sizes between `mixed` and `quic`. `quic` was selected to have the same incoming total sizes as `mixed`.

To understand how large these anonymity sets are, we work with a large CDN provider. We identify 13′744′979 unique domain names hosted on the provider in March 2021, hosted on 593′338 IPs. To get the IP addresses corresponding to the domains, we run `zdns` in iterative mode. We query directly the DNS server of the provider, skipping the local cache. From the results, we filter out 2′641′100 errors (19.2%), 15973 `NXDOMAINS` (0.1%). Since we are interested in Domain Name → IP mappings, we ignore 1′024′234 `CNAME` answers (7.5%), and end up with 1′313′159 `A/AAAA` records.

We group the provider domains by IP address obtaining 593′338 anonymity sets whose distribution, depicted in Figure 2, coincides with that found by Patil *et al.* [38]. 60% of these domains are hosted on a unique IP address. For these domains, the adversary does not need to run any traffic analysis: when observing one of these IPs, the adversary is certain of which domain is being visited. Therefore, we leave these domains out of scope of our study. We describe in the next section how we use the rest of the domains from the CDN provider to build our evaluation datasets.

Adversary's Resources. We measure the adversary's cost to perform a website fingerprinting attack in terms of the bandwidth they require to collect and process the traffic traces. Bandwidth is a proxy for the required storage, as the adversary needs to store the transmitted information to query the machine learning model and possibly to retrain it. The computational cost is also proportional to the bandwidth, as the number and cost of operations needed to extract features depend on the length of the traces transmitted.

We assume that vantage points currently do *not* have the capability to run machine-learning tasks [39, 40]. They must mirror (part of) the traffic to a suitable location for analysis (purple dotted arrows in Figure 1). This location processes the traffic: it extracts features and performs classification to identify the page visited by the client.

We study two kind of adversaries. First, an unconstrained adversary who can afford to transmit full traces. In practice, mirroring all traffic is prohibitively expensive [41]. Thus, we also study a constrained adversary that only transmits summaries of locally computed statistics from sampled data [42, 43].

4 Datasets

Previous work mainly uses domains from Alexa Top 1M or Tranco. Using these lists, however, does not account for the fact that the adversary can observe the destination IPs. In the QUIC setting, an analysis based on these lists would represent a scenario in which the client uses a VPN, which is out of the scope of this work. We therefore create two new datasets.

`mixed` dataset. Our first dataset represents a current state of affairs. Currently, browsers learn that a website is QUIC-capable through an HTTP header, hence, after having established a TLS connection. Thus, TLS and QUIC co-exist in websites. We use the distribution of domains among IPs from our CDN provider collaborator to identify realistic website sets that could be targeted by an adversary.

First, we resolve IP addresses and group domains by IPs. Since the content of duplicate domains is often identical, website fingerprinting attacks often misclassify these websites among each other. The resulting low accuracy, however, does not translate into privacy of the browsing history: the website accessed is the same. We remove duplicate websites to avoid underestimating performance. We resolve `CNAME`s and filter out copies. We also remove IPs whose hosted domains are mostly aliases or copies of a single website (*e.g.*, `casino123.com`, `cazino132.com`); for this, we use the Jaccard similarity between the domain names to identify candidates. We manually examine candidates with a high similarity score and remove obvious duplicates. We also discard IPs that host similar sets of domains to support load balancing. Finally, we discard any IP for which the fraction of websites that return a successful HTTP response (when queried with HTTPS) is less than 80%.

After all the filtering, we end up with 593′338 M domains. Of these, only 50*k* IPs (8.5% of the dataset) host more than 150 domains, with one hosting as many as 56′319 domains. This leads us to choose a cluster of 150 websites for our experiments. This size represents the hardest scenario for the adversary among 91.5% of the IPs served by the CDN provider. Early in our work, we run experiments with other clusters of similar size obtaining comparable results.

Table 1: Mean classifier performance on full network traces for an unconstrained adversary.

Dataset	RF	Var-CNN
`mixed`	66.6 (std. dev. 0.5)	57.33
`quic` (Mar 21)	95.8 (std. dev. 0.4)	92.28
`quic` (Sept 21)	96.4 (std. dev. 0.2)	94.22

`quic` dataset. The `mixed` dataset is TLS-heavy: only 4% (std 1.7%) of its traffic is using QUIC. In our experiments in section 5, we observe that in the presence of a TLS/QUIC mixture, the classifier favors TLS-specific features. This prevents us from drawing meaningful conclusions regarding the vulnerability of QUIC, or the strength of potential defenses. However, as QUIC is not widely deployed yet, none of the IPs served by the CDN provider host a set of domains that primarily use QUIC for their main page and subresources.

We build `quic`, a QUIC-dominated dataset, following the example of Smith *et al.* [10]. We crawl Alexa 1M [44], Umbrella 1M [45], and Majestic 1M [46] and we perform HAR captures[1] to identify sub-resources and the protocols used by those websites. We select 150 domains that use QUIC; as the total size and number of packets have long been identified as a important and stable metric [21], we select these websites to match the distribution of total website sizes from the `mixed` dataset (we present the distributions in Figure 3). `quic` has 70% of all traffic over QUIC (std 3%).

Limitation. `mixed` and `quic` correspond to an anonymity set at a specific point in time. We did not evaluate how the anonymity set change over time. To remain accurate, the adversary must retrain on new labels if the set changes.

Data Collection We collect PCAP network traces from all the index pages of the domains of our two datasets. We use Firefox 88.0 isolated on its own network namespace (using `netns`), enabling HTTP3, and disabling telemetry and auto-update settings to minimize extraneous traffic. We record 40 samples for each website. For each sample, we recreate a fresh profile for Firefox; this ensures that all caches have been removed.

We filter the network traces and only extract well-formed TLS and QUIC packets. To avoid relying on plaintext markers, we follow the approach of Smith *et al.* [10] and only extract the time and size of the sent and received packets.

5 Unconstrained Adversary

We start by studying a powerful adversary that can observe all the traffic associated with a site visit, and who has the capability to store and process all the traffic that it observes. Such an adversary could be, for instance AS *X* in Figure 1, if it would not have bandwidth or storage constraints.

We run the random forest website fingerprinting attack on both the `mixed` and `quic` datasets. We use 10-fold cross validation; Table 1 shows the result. The classifier yields good performance compared to random guessing (0.67%), indicating that a powerful adversary can identify sites with high success. The performance for `mixed`is lower than results obtained by the state-of-the-art [10]. We inspect the sites in this dataset, and we discover that 22% of the sites are the same, despite having different domain names. This leads to sites being classified as one another. When we consider these as correct classifications, the performance increases to 76%. We also observe that the classifier's performance has higher variance against some websites in the `mixed` dataset (error bars in Figure 3). These domains often correspond to unoptimized, arcane sites, unlike the optimized websites of Alexa Top 1M used in previous work.

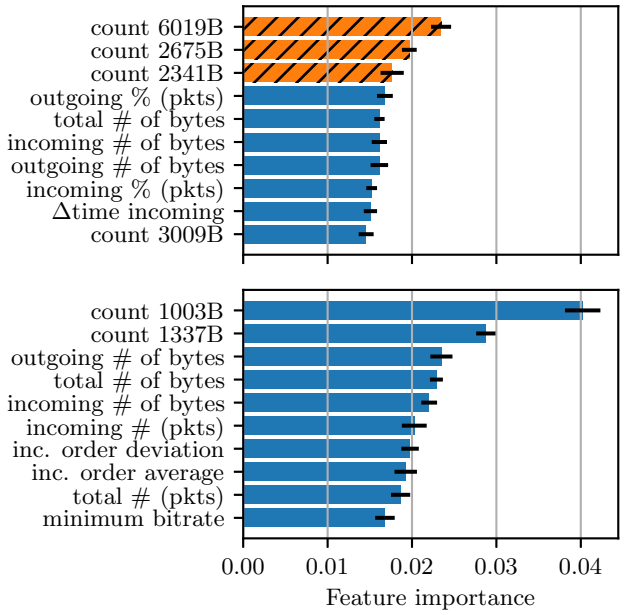

Figure 4: Feature importance for `mixed` dataset (top) and `quic` (bottom). The top three features for `mixed` (in orange, dashed) are specific to TLS.

We run a feature analysis on the `mixed` dataset. The most important features are TLS-specific (Figure 4, top). Since these features cannot be protected with a QUIC `PADDING` frames, in the remainder of this paper we focus on the `quic` dataset. When this dataset is unprotected, the most important features being the histograms of packets between 1000 B and 1400 B, and the total transmitted volumes (Figure 4, bottom).

Stability. To validate our results, we collect a second dataset for `quic` six months after the initial collection. We observe the same performance: 96% mean F1 score across 10 cross validations, with 0.2 standard deviation (see Table 1).

5.1 Protection against a powerful adversary

To understand how a defense could hamper fingerprinting, we explore defenses that hide local or global features, or both. We

[1] A copy of the "Network" tab of the Firefox Developer Tools console.

Table 2: `quic` dataset: Mean classifier performance on defended traces.

Variant	F1 Score	Std. dev.
undefended	95.8	0.4
hiding all timings	95.5	0.3
hiding individual sizes	93.9	0.4
+ hiding total transmitted sizes	85.0	0.5

ignore practical considerations and assume that there exists an implementation that perfectly protects these features. We report the results of the attack in Table 2.

Hiding local features. The top predictive features for `quic` are all sized-based (Figure 4, bottom). Thus, our first attempt aims at hiding individual packet size. We observe that padding individual packets poorly hides the total transmitted volumes, which becomes the top features once individual sizes are removed. The attacker performance slightly decreases to 93.9%. We try to also hide timings, which reduces the adversary's performance in one more percentage point.

Hiding global features. Next, we hide individual sizes and we pad the total transmitted size to the next megabyte, by increasing the size of all packets. The attacker simply switches its features to packets orderings (Figure 5), yielding only a small drop in performance which becomes 92.2%.

Injecting dummies. In order to hide individual packet orderings we augment the defense with dummy packets. As explained in section 2, we cannot use existing defenses based on adversarial perturbation to find the optimal placement of dummies. Instead, we inject dummy packets at random position in the padded trace, up to a maximum of 60 seconds after the last real packet (we observe 60s to be a conservative value; the maximum duration for loading a website being 33s). All packets and dummies are padded to the maximum size.

We vary the percentage P of dummies injected in the trace, and we plot their effect on F1-score in Figure 6. We see that dummies only achieve a significant reduction at a sharp increase in cost: to obtain a $\approx 10\%$ F1-score drop for the adversary, we must add +50% bandwidth overhead; in addition to the already large overhead in terms of packets to hide local and global features (mean cost of 612 kB per trace, with a large standard deviation: 440 kB).

Take-aways. Our experiments show that despite their high cost, network-layer defenses reduce the adversary's performance by, at most, 3.5 percentage points. The reason is that IPs are telling, anonymity sets are small, and the classifier is able to pick even small differences between the traces. We conclude that website fingerprinting needs to be tackled at the application layer to hide the total transmitted size and the total number of packets, which network-layer defenses cannot hide efficiently because they do not have the total size of objects to pad in advance. We explore application-layer defenses in section 7.

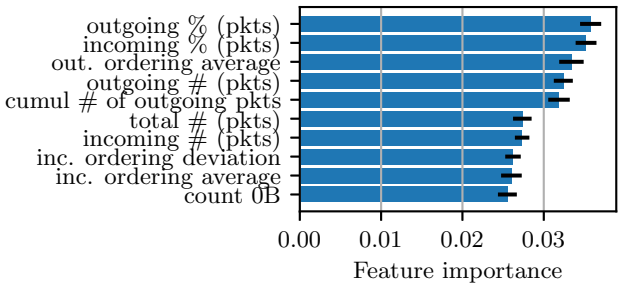

Figure 5: Feature importance when hiding hiding global features (last row of Table 2).

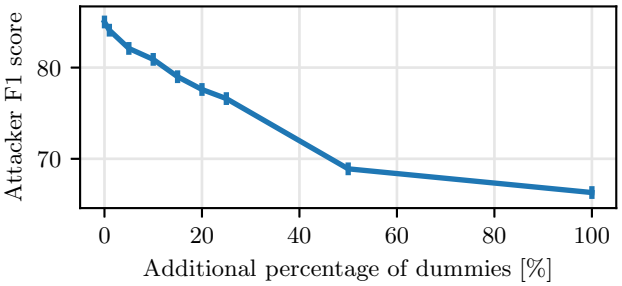

Figure 6: `quic` dataset: Mean classifier performance on traces with dummies.

6 Constrained Adversary

In section 5 we assumed a powerful adversary who can observe and process *all* of a victim's traffic. We showed that no network-layer defense can provide protection against this adversary. However, not all adversaries have such strong capabilities. As described in section 3, there are on-path adversaries who might not observe all traffic from a particular client, nor all traffic to a particular server of interest. Even if the adversary can observe all traffic, it may not have the capability to process this traffic or to transmit it to a location suitable for analysis.

We now study whether network-layer defenses could protect against adversaries that either have limited traffic visibility or limited processing capabilities. We find that sufficient traffic visibility is essential for a successful attack; and that using NetFlow, a standard technique for efficiently collecting statistics about network traffic [47, 48], is sufficient for the attack is the sampling rate is not too low. We also show that padding-based defenses bring limited benefits, in particular when contrasted with the performance drop due to the NetFlow sampling rate.

6.1 Limited traffic visibility

We simulate an AS adversary with partial view on the client's traffic. In order to identify which parts of the traffic an AS would see, we first study the routes taken by requests during a page load. We run HAR captures to identify all the resources that are queried for every page in our dataset. Then, as in prior work [49], we use traceroutes to record the path taken by all

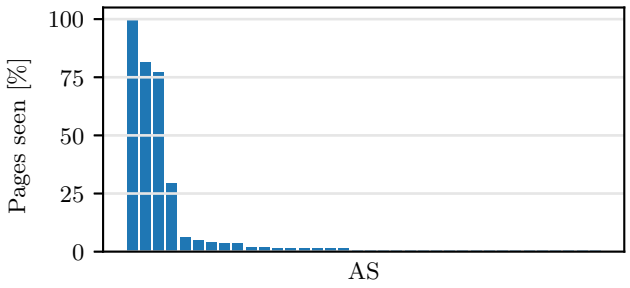

Figure 7: Distribution of webpages seen by each AS. Only three ASes (client's AS, Google, Cloudflare) can observe traffic from all the pages. Most ASes observe less than 10% of pages.

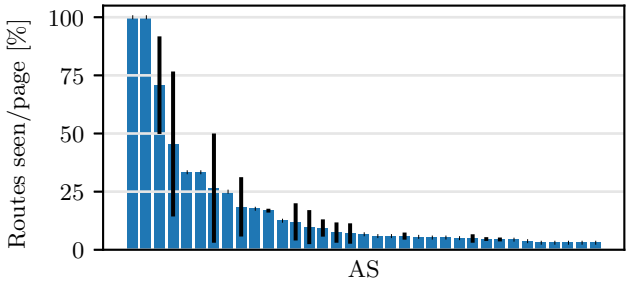

Figure 8: Distribution of routes per webpage seen by each AS. Only three ASes (client's AS, OVHcloud, Google) observe more than 50% of the traffic per site.

the resource requests and the ASs encountered on each route. We set `traIXroute` [50] to use `scamper` (configured with the Paris traceroute technique).

To capture the view of the adversary, we use the route information to filter our collected network traces, keeping only the packets associated to the resources visible from the adversarial AS vantage point. We filter the PCAPs by removing TLS/QUIC connections that do not correspond to the resources of interest. This filtering is based both on the destination IP address and the SNI, when it exists.

Our results below are from traceroutes collected from the same location as the `quic` dataset (in France). We also ran traceroutes from other vantage points (in Germany, UK, Singapore), and observed the same trends as described below. The additional experiments are described in Appendix A.

Results. We observe a total of 974 routes in the `quic` dataset. The distribution of routes per webpage is shown in the appendix. As done in [49], we discard route hops that do not have IP or AS information (asterisks in the traceroute). To avoid introducing inaccuracy in our analysis, we do not attempt to fill these gaps in routes via stitching as performed in [49]. Thus, our results provide a lower bound on the amount of traffic that an AS adversary sees.

Figure 7 shows how many webpages are *seen* by each AS; we consider the webpage is seen if the AS sees any traffic associated with this webpage visits (including its subresources). There are three ASes that observe traffic from more than 80% of webpages: one from Google, one from Cloudflare, and the AS where our client is located. The majority of the ASes, however, see only a small proportion of the sites (less than 10%). If any of these ASes were to be the adversary, they would not be able to fingerprint traffic from most websites hosted by this IP.

Table 3: Mean classifier performance on different AS views.

AS	Name	# Pages	F1 Score
15169	Google, LLC	67	89.5
13335	Cloudflare, Inc	50	92.9
3356	LEVEL3	2	81.7
32934	Facebook, Inc	2	92.3
45899	VNPT-AS-VN	1	100.0
20940	Akamai Technologies	1	100.0
62713	PubMatic, Inc	1	100.0

Seeing any traffic is a necessary condition to fingerprint but, to have a high attack success, an adversary also must observe a sufficient proportion a webpage traffic. For every web page an AS sees, we study what portion of this page they can observe (see Figure 8). The source AS sees 100% of the traffic. Another AS, 4367 (belonging to OVHcloud), sees 100% of page traffic as well, although the number of pages whose traffic it can observe is low. The Google AS is the second highest. It sees ≈ 70% of the routes for each page. All other ASes see less than 50% of the routes associated with a page.

We show in Table 3 the classifier performance if each of these ASes was the adversary. Few ASes have a substantial view of the client connections, *e.g.*, Google (67%) or Cloudflare (50%). On the page loads they observe, these entities can fingerprint the traffic with high F1 score. Adversaries that observe little traffic, however, cannot achieve significant performance (less than 6%). We conclude that, in order to successfully fingerprint, an AS adversary needs to observe a large proportion of the traffic, either by being the client's AS or by providing sub-resources on websites.

Inexpensive fingerprinting due to resource centralization. In subsection 7.1, we show that most websites query resources from a server hosted by Google. From the (incomplete) AS information on the network level, we found that at least 67 webpages were seen by Google. When looking at the HAR capture, we observe that 125 websites in `quic` (83%) request a resource from a Google-owned domain. Furthermore, even when the same set of resources are loaded, they are not loaded at the same time (with respect to the time of query of the home page) or in the same order.

This centralization of resources could be used by an adversary to perform traffic analysis at a fraction of the typical cost: instead of recording all traffic, an adversary can use the timings of `ClientHello`'s to Google IPs to efficiently fingerprint the traffic. To validate this hypothesis, instead of recording all traffic and deriving features from these traces, we only

Table 4: Mean classifier performance and median storage required per sample on traces filtered by connections to Google services. The last two rows use 125 samples.

Variant	**F1 score** (Std dev)	**Size [kB]**
Baseline (Full Traffic)	95.8 (0.4)	312.4
Full traffic to Google	78.4 (0.7)	112.1
`ClientHello`'s to Google	66.1 (0.4)	0.1

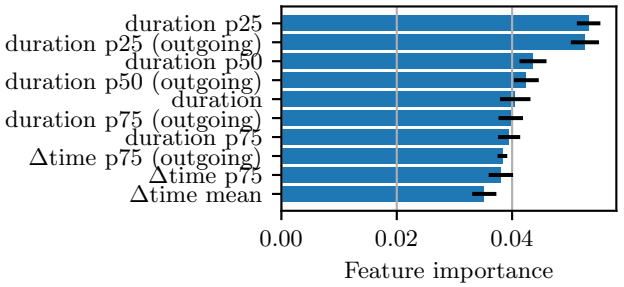

Figure 9: Feature importance for classifying websites based on the timings of their requests to Google services.

record the traffic towards to Google services by filtering the network traces for which the destination IP (or the SNI) belongs to Google. If this field is not present in the packet, we perform a reverse-mapping with the destination IP to confirm the destination. To list domains belonging to Google, we get the requested URLs using our HAR capture, and we check the ownership using tracker-radar [51]. In our attack, we use the following four Google-owned domains: `google.com`, `gstatic.com`, `youtube.com`, `doubleclick.com`, `ggpht.com`.

Finally, we extract the sending times of the packets containing a `ClientHello` record to these Google IPs and domains. For `quic`, this represents 7.6 floating-point values on average per loading of a website, with a maximum of 27 values. The size of the fingerprint is between 61B and 216B per loading of a website; in contrast, the mean .pcap size is 112 kB for the traffic towards Google, and 312 kB for all traffic.

Results. Table 4 shows that when using only the timing of requests to Google, the adversary achieves 77.9% F1 score for the 125 websites that use some Google resource. This can be achieved with only ≈ 61 B per connection, a saving of four to five orders of magnitude compared to recording a full network trace. The feature analysis confirms that the timings between sub-resources is what helps the attacker in this case (Figure 9). This experiment highlights two current practices that facilitate fingerprinting: resources are centralized in a few providers, and it is easy to observe connections to these services (*e.g.*, by logging the traffic to four domains/IPs); and websites query these resources in a unique way, which enables fingerprinting.

Take-aways. A few key large ASes (*e.g.*, Google, Cloudflare) can successfully run traffic-analysis attacks; yet, these entities are CDNs, and do not need to run such attacks: they terminate the encrypted connection and can observe the source page (*e.g.*, via `Referer` field). At the same time, other ASes observe too few packets to perform traffic analysis. In both cases, a defense against traffic analysis does not seem useful.

Interestingly, the fact that resources are centralized on a few CDNs can enable other entities to perform fingerprinting at a fraction of the usual cost: *e.g.*, an ISP can use the Google-filter to save bandwidth by 3 to 4 orders of magnitude (compared to running the attack on all traffic) and still efficiently fingerprint.

Table 5: Mean classifier performance and median storage cost per sample for Sampled NetFlow, `quic`.

Sampling	**F1 Score**	**Size [kB]**
Full traces	95.8	312.4
NetFlow 100%	90.5	25.9
NetFlow 10%	66.4	3.0
NetFlow 1%	41.7	0.9
NetFlow 0.1%	16.8	0.4

6.2 Limited processing power

Fingerprinting requires adversaries to have storage and computation capabilities, which middleboxes might not have. Also, mirroring the traffic (*e.g.*, to a location suitable for analysis) is expensive: typical network monitoring solutions only record aggregate statistics over sampled traffic [48]. Common tools for network sampling include *NetFlow* and *sFlow* [35]. More efficient techniques have been proposed in academic papers (*e.g.*, sketching [52, 53] or *skampling* [54]). To the best of our knowledge, these techniques are not yet widely deployed. We therefore focus on NetFlow, which is widely deployed on the Internet.

To investigate whether NetFlow statistics are sufficient to perform traffic-analysis attacks, we simulate *Sampled NetFlow*, a variation of NetFlow used in high-speed links where packets are first sampled in a deterministic fashion (1 out of every N packets) and flow statistics are computed on the sampled packets. First, we down-sample the PCAPs uniformly to the desired sampling rate, and we create NetFlow summaries from the PCAPs using `nfpcapd` and `nfdump`. We experiment with various sampling rates: 100%, 10%, 1% or 0.1% (common sampling rates range from 50% to 0.1% [43]). Second, we adapt the features used by our model to NetFlow summaries rather than packets. Our adaptation is straightforward: we consider a flow as a single packet whose size is the sum of all packets in a flow, and inter-packet timings become inter-*flow* timings. The number of packets per flow is recorded, but individual packet sizes, timings and directions are lost (they are not recorded by NetFlow). There are two flows per connection. We acknowledge there could be better, tailored features and thus our evaluation only provides a lower-bound on the attacker performance.

Results. We show the mean classifier performance on the NetFlow summaries in Table 5. Moving from full packet data to flow summaries leads to a reduction in the adversary's perfor-

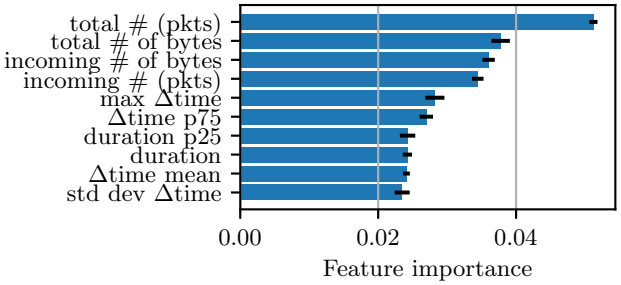

Figure 10: Feature importance for classifying NetFlow with 1% packet sampling rate.

Table 6: Mean attacker performance on defended NetFlow, `quic`.

Sampling	F1 Score
NetFlow 100%	53.1
NetFlow 10%	33.1
NetFlow 1%	21.6
NetFlow 0.1%	8.6

mance, *i.e.*, there is a trade-off between the adversary's success and their capabilities. The performance drops significantly with the sampling rate (29 percentage points lost when 10% of packets are sampled). All of these F1-scores are still much higher than random guessing (F1-score=0.6%).

Defenses. Regardless of the sampling rate, the most important features when using NetFlow are the total number of bytes and packets (Figure 10). We explore a costly defense that hides both per-flow metrics and overall statistics about the number of bytes and packets exchanged. When considering full traces the maximum transmitted size is 22 MB, and the maximum number of packets is 25K. We hide global statistics by padding the total transmitted bytes and the number of packets to these maximum values. For other sampling rates, the maximums diminishes linearly with the sampling rate, and we apply the same reduction in the padding function. This padding is added uniformly to all the flows of one sample.

Results & take-aways. These defenses do reduce the attacker performance, yet at an impractical cost; the median cost is ≈ 39 MB per trace (see Table 6). As seen in section 5, the adversary adapts the available features, such as flow orderings (11) or timings (Figure 12). We also note that although the defended traces have a low F1 score (*e.g.*, 21.6 at 1% sampling), most of the gain in privacy compared to the standard setting (95.8%) comes from the sampling (−54.1 percentage point) rather than the defense (−20.1).

7 Application-Layer Defenses

Our results in section 5 and 6 confirm, for QUIC, the findings of Dyer *et al.* [21] for HTTP over encrypted tunnels: network-

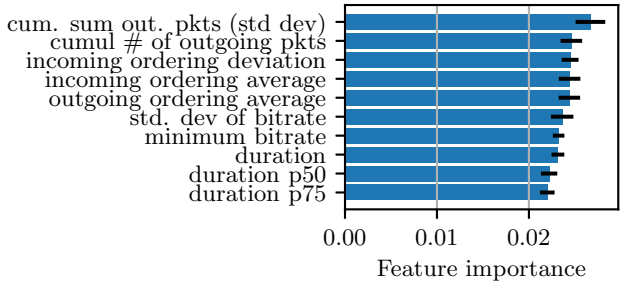

Figure 11: Feature importance for classifying defended NetFlow with 100% packet sampling rate.

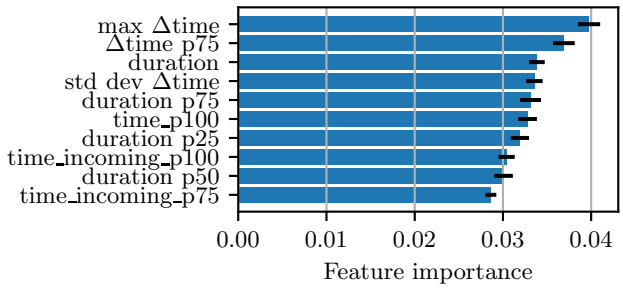

Figure 12: Feature importance for classifying defended NetFlow with 1% packet sampling rate.

layer defenses have limited ability to hide global traffic patterns. To efficiently hide global traffic patterns (*e.g.*, total incoming size), a defense needs to know in advance the size of objects. The only way to know in advance the size of objects is to know the resources, *i.e.*, to have the defense at the application layer.

7.1 Web page structure in the `quic` dataset

To design an application-layer defense, we need to understand how websites use resources. To study the websites in the `quic` dataset, we use OpenWPM [55], which logs the HTTP requests that occurred during the page load. Unlike HAR captures, OpenWPM also records the originator of a request. We collect page structures by crawling the pages in `quic` with OpenWPM (v0.17.0) and Firefox five consecutive times.

Resource dynamism. We first study how dynamic the pages in our dataset are, *i.e.*, how pages vary across crawls. Dynamism influences how easy it is to protect a page. The less dynamic pages are, the easier it is to protect them as one can select defense parameters tailored to the static resources. If pages vary overnight, defenses can only be configured to fit the average case.

Out of the 150 websites in `quic`, 149 were successfully visited across all crawls. For these pages, we calculate the proportion of resources that remain static across the runs. Sometimes, even if the resources fetched are the same, the URL parameters may vary. Hence, we strip the URL parameters, and plot the distribution of static resources in Figure 13a. The mean

proportion of static resources is 88.25% (Std: 22.46%) and the median is 100%. This indicates that there very little dynamism in our dataset. The majority of the resources remain static across runs. We note, however, that our measurements are taken over a short period of time, and dynamism could grow if webpages are observed over a longer time period.

Resource ownership. Next, we study the ownership, *i.e.*, which entity could modify the resources, of these resources. Understanding the variety of owners provides an idea of how much coordination is required among these owners to protect a page from website fingerprinting. We define ownership in terms of whether a resource is *first* party (shares the same eTLD+1 as the page) or *third* party (has a different eTLD+1 as the page). For example, on the page `www.example.com`, a resource `img.example.com` would be first-party and a resource `external.com` would be third-party. Using domains as a proxy for ownership is not perfectly accurate: for example, content for `facebook.com` can be served from `fbcdn.net`, and both domains come under the control of Facebook, but the latter would be identified as a third party. We tried to use services that provide entity-domain mappings [51] to validate our inference. Unfortunately, they do not have relevant ownership information for almost half of the resources in our dataset. So, we stick to domains to determine party for our analysis.

Figure 13b shows the proportion of first-party resources over the pages in our dataset. We observe that a majority of pages have a large proportion of first-party resources (Mean 61.18%, Std 31.61%, Median 66.67%). At the same time, the proportion of first-party resources can be as low as 3.92%, likely due to page content not being self-hosted. We observe that there are, on average, 5.95 unique third party domains on a page (Std: 7.64, Median: 3), with the number of domains going up to 44 for one of the pages in the set. This indicates that implementing a server-side defense is not straightforward since it would require coordination among multiple third parties. Visual inspection of the third-party domains shows a large number of domains commonly associated with Google. We map the domains to their owning entities [51] to measure Google's prevalence ([51] contains mappings for Google's domains). Figure 13c shows the proportion of Google resources per page. 24% of the pages have more than half their resources served by Google. Google's widespread prevalence hints that any application layer defense would need their cooperation to be successful.

7.2 Application-layer defense strategies

Now that we have an overview on the structure of webpages and the number of entities involved, we proceed to design defenses.

All parties must protect resources. As in the QUIC setting the adversary can observe IPs, the adversary can filter resources coming from different parties and perform the attack on traffic from different origins. Given the large proportion of third-party resources on websites, before diving into designing defenses, we first study whether these third parties need to be involved or it is sufficient that the first party protects resources.

Table 7: Mean classifier performance on traces filtered by 1st / 3rd party and Google CDN, `quic`.

Variant	F1 Score	Std. dev
All traffic	0.937	0.008
Only traffic to/from 1st parties	0.955	0.004
Only traffic to/from 3rd parties	0.915	0.005
Only traffic to/from Google CDN	0.914	0.006

We filter the resources by party to simulate cases in which either the first party, third parties, and the adversary only attacks the remaining undefended traffic. First-party defenses represent a scenario where webpages protect their content using some defense, but third-parties do not cooperate. Third-party defenses represent a scenario where third parties such as CDNs, which host a large number of resources, decide to implement a defense, but smaller first parties do not. We also study the case in which the adversary focuses on dominant third parties, as some CDNs represent a large portion of resources. We pick Google as dominant third party as 83% of the websites in `quic` query Google resources (see subsection 7.1).

Table 7 shows that, in all of these scenarios, the attacker can achieve a high performance just analyzing the partial, undefended traces. Thus, we conclude that for any defense to be effective, all parties involved in serving content must be coordinated and actively participate on the protection of resources.

Once it is clear that all parties must participate, we proceed to design defenses. We try four different strategies to protect local and global features.

1. Protecting local features with padding. We design a padding function $\texttt{pad}_{\text{resources}}$ to hide individual queries and resources sizes. Such a defense must be implemented both on the client and the server.

To configure this padding function, we use (1) the distribution of request sizes in the targeted set of websites and (2) one parameter N, which defines how many different sizes the defense allows for. Given the distribution of request sizes, we design the padding function to split the resources sizes into N groups of equal density. For $N = 1$, all resources are padded to the max resource size in `quic`. For $N = 2$, half of the resources are padded to the median size, half to the max size. Choosing a small N increases privacy: more resources will be padded to the same size and be indistinguishable. A small N also increases bandwidth usage.

2. Protecting global features with padding. Padding only individual packets' size cannot protect the overall transmitted volume. We design a padding function $\texttt{pad}_{\text{total size}}$ to pad the total incoming and outgoing traffic sizes. To evaluate the best case defense, we assume the ideal scenario in which the

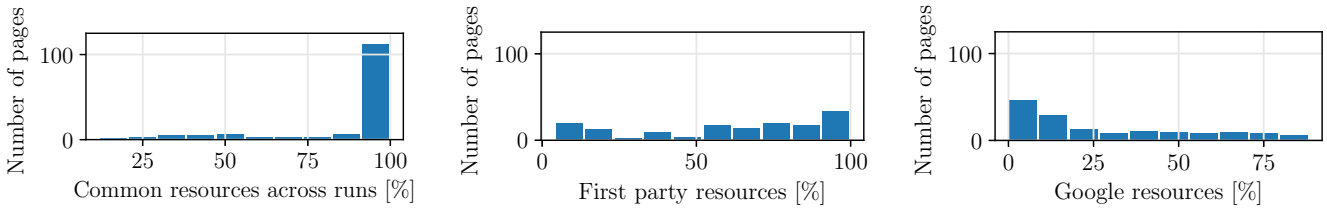

Figure 13: Resource dynamism and ownership for pages in the quic dataset. Figure a is the distribution of the proportion of resources that remain static across 5 runs. The majority of resources remain constant across runs, indicating low page dynamism. Figure b shows the proportion of first party resources. 18% of the pages have less than 20% of first-party resources. Figure c is the proportion of Google third party resources. 24% of the pages have more than half their resources served by Google.

padding effort is split evenly across all the parties queried on one webpage. This way, the adversary does not gain an advantage by dropping the traces from one party in particular. This strategy assumes the existence of a mechanism by which clients can ask third parties for a particular amount of padding per resource; how to design such a mechanism is outside the scope of this work.

The design of $\texttt{pad}_{\text{total size}}$ is similar to $\texttt{pad}_{\text{resources}}$. It has one parameter, N, which defines how many total incoming and outgoing traffic sizes are allowed. We first compute the maximum total incoming and outgoing traffic in our target dataset, quic. Across all websites, the maximum total size of queries in one website is 102 kB and the median is 14.4 kB; and the maximum total size of all downloaded resources is 8.19 MB, with median 750 kB. To apply the defense, we split the total incoming/outgoing sizes into N groups of traces with equal density. For instance, when $N = 1$, there is only one group of maximum size. Thus, we pad all websites' outgoing traffic would be padded to 102 kB, and the incoming traffic to 8.19 MB. For $N = 2$, the groups would correspond to the median and to the maximum total incoming and outgoing traffic. For $N = 3$, the groups would correspond to tertiles of the distribution, for $N = 4$ to quartiles, and so forth. We allocate every website to the group that is closest to its original total incoming and outgoing size, and we spread the padding evenly across all queries and resources of that website.

3. Protecting global features with dummies. An orthogonal, popular approach to hiding the total size is injecting dummy traffic in addition to padding individual packets [24]. Unlike in Tor, where dummies are indistinguishable from real cells due to the standard cell size, in QUIC, care must be taken that dummies' sizes do not enable the adversary to filter them. To evaluate the defense in ideal circumstances, we assume the existence of a dataset that contains the most popular queries and resources across all web pages of (in our case in the quic dataset). Then, we build dummies replicating the structure of those popular queries. This ensures that the queries are hard to filter for an adversary.

In our experiments, we select popular resources from Google (fonts, analytics, static assets). When a webpage is loaded, we choose a number of resources to inject. These resources can themselves trigger additional queries. The time at which the resource is injected is chosen uniformly at random over the duration of the connection, such that the adversary cannot use timing to identify and filter out dummies.

7.3 Application-layer defenses evaluation

Since we are interested in evaluating defenses at the application layer, we would like to directly evaluate their effectiveness by measuring how they perturb features at the application layer. The results by Hentgen [56], show that this is possible. Hentgen demonstrates that evaluating a defense at the application layer gives an upper bound on the capabilities of a network-layer adversary with respect to a set of features. This is because measuring resources features at the network layer is effectively a noisy version of the resources at the application layer, For instance, the number of incoming packets and the total size of incoming QUIC packets are a noisy version of the actual size of the downloaded resources, and the total duration of the connection is a noisy version of the total amount of bytes downloaded. Even features such as inter-packet timings are a noisy representation of the structure of the web page: they represent number and timing of resource loads.

Metrics. We use two different metrics to evaluate the attacker success. As in previous sections, we use the performance of the classifier. Since Var-CNN [19] relies on packet orderings and timings, but does not consider packet sizes, we limit our evaluation to the random forest classifier.

We study the attacker's performance in three scenarios: undefended traces with all features, undefended traces without timings, and defended traces without timings. The latter is a good estimate of the attacker performance (even with timings), as our baseline analysis shows (see subsubsection 7.3.1).

Dataset and features. For the undefended baseline, we use the HAR captures of quic; their format is a list of $[t_{\text{request}}, \text{size}_{\text{request}}, t_{\text{response}}, \text{size}_{\text{response}}]$. We derive the k-Fingerprinting features from these traces.

Our padding and dummy-injection defenses would affect the timing of packets. In practice, however, it is difficult to

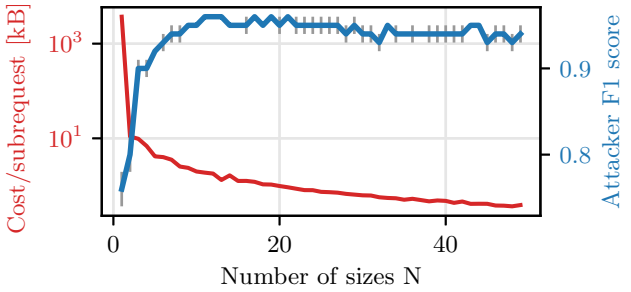

Figure 14: Number of allowed sizes, N, in $\texttt{pad}_{\text{resources}}$ versus attacker performance and median bandwidth cost per subrequest.

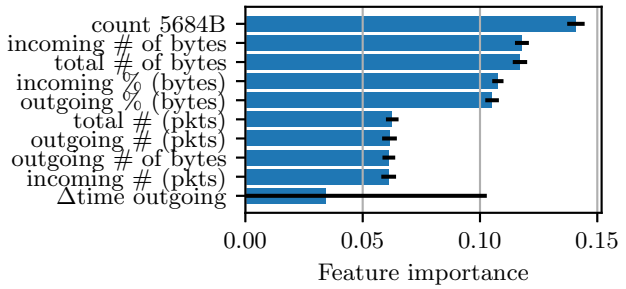

Figure 15: Feature importance with 3 padding sizes: 5.58 kB, 21 kB, 3.6 MB.

predict or measure how these changes would affect our traffic captures. Attempting to copy all websites of `quic` on a server we control to simulate the changes is insufficient, as we would not be able to simulate actions from third parties.

Fortunately, previous work and our own experiments (see Figure 19) show that timings are not among the most important features. While they can help, they are less stable (and hence useful) than sizes. This is corroborated in Hentgen's study, which shows that even without timings the analysis is still an upper bound over the network layer [56] Thus, when applying defenses, we discard timings and use with a list of $[\text{size}_{\text{request}}, \text{size}_{\text{response}}]$.

7.3.1 Defense Evaluation

Baseline on undefended traffic. When attacking full HARs, the adversary obtains very good performance (93% F1 score). When removing timing information, we obtain the same F1 score, confirming that timings are not so important in our scenario. All top features are based on sizes, `bytes_outgoing` being the most important features by a slight margin over `bytes_incoming` (`bytes_total` is the sum of the two).

Protecting local features with padding. We vary N (the number of allowed sizes) and evaluate the effectiveness of $\texttt{pad}_{\text{resources}}$. We pad the resources, run the attack, and plot the median cost and the attacker F1 score in Figure 14. Only large amounts of padding (small N) have an effect on the attacker

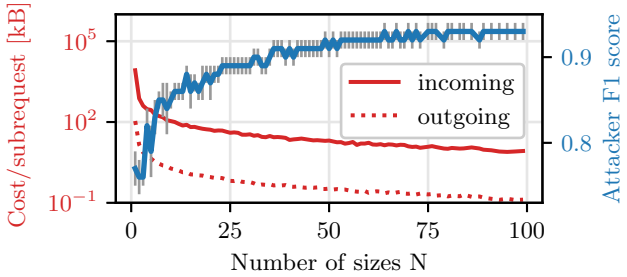

Figure 16: Number of allowed sizes N in $\texttt{pad}_{\text{total size}}$ versus attacker performance and median bandwidth cost per subrequest.

accuracy. For instance, padding with 3 large sizes (5.58 kB, 21 kB, 3.6 MB) decreases the accuracy adversary by 6% and incurs a median overhead of 9 kB per resource. This ineffectiveness stems from the fact that the adversary still has access to the number of requests and overall volume (see Figure 15), which are sufficient for the attack. The total sizes are too different to be efficiently hidden through the padding of individual resources.

Protecting global features with padding. We evaluate the effectiveness of padding the sum of requests and the sum of incoming resources independently using $\texttt{pad}_{\text{total size}}$. We plot the attacker F1 score and the median cost in Figure 16.

We observe that padding the total size again does not hamper the attack: for instance, to drop the adversary's accuracy by 10 percentage points, the defense incurs a median cost of 5.7 kB per request (outgoing traffic) and 300 kB per resource (incoming traffic). In the best case, it reduces the attacker's accuracy by 16 percentage point, with a median cost of 109 kB per request and 8.16 MB per resource.

Injecting dummies. We plot in Figure 17 the attacker F1 score for a varying number of dummies. This defense is more effective than the previous strategies. For instance, the parametrization ($p = 0.5, M = 10$), which injects on average 5 dummy requests, decreases the attacker performance from 93% to 54% F1 score, at a median cost of 137 kB per loading of a web page. We observe that setting p to 0.5 hinders the attacker performance more than the quantity of dummies (M): on average, the two parametrizations ($p = 0.5, M = 10$) and ($p = 1, M = 5$) inject 5 sequence of queries to a CDN; the former reduces the attacker adversary to 0.54 F1 score, and the latter to 0.43.

Limitations. In contrast to a network-layer defense, an application-layer defense interacts with HTTP traffic and applications; it is thus more complex to design a defense which has a minimal impact on existing systems (*e.g.*, client caching, pipelining of resources and other browser&website optimizations). Compared to a network-layer defense, padding and dummy requests incur an increased computation and bandwidth cost. On the client side, dummy requests affect how real requests are pipelined and might delay them.

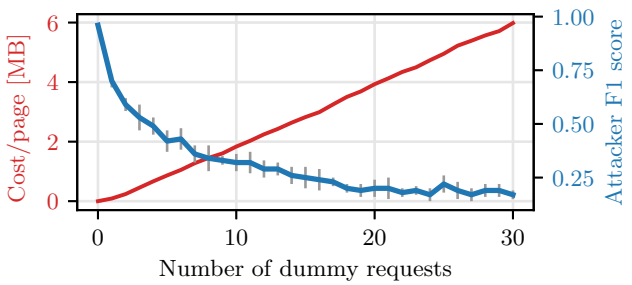

Figure 17: Attacker performance and cost when varying the number of dummies.

8 Discussion and Recommendations

Our work highlights the difficulties in developing an effective website fingerprinting defense for the Web. We carried out a comprehensive study of the ecosystem in which defenses are to be deployed, and provided evidence for the fundamental incompatibilities between today's Web communications and user privacy.

Network-layer challenges. First, we confirm that previous results which point out that network-layer defenses are not effective against website fingerprinting [21] also apply when the transport protocol changes from TCP to QUIC. The main problem stems notably from the differences in the total sizes of websites and result on identifying features [16, 57]. Hiding the total size is hard at the network layer, where the size of objects is not known in advance. Thus, without coordination with the application layer, a generic, application-agnostic defense using the QUIC's `PADDING` frame is an inadequate mitigation against traffic-analysis attacks. Effective mitigations require application involvement, either in the application code or as part of the browser's functionalities.

App-layer challenges & Next steps. While defenses at the application layer can obtain better protection at a smaller cost, our investigation shows that the current Web development practices hinder the effective deployment of any defense. For instance, a straightforward defense would be to pad the total size of websites to standard set of sizes. However, websites use resources hosted on different servers, and defenses must cover *all* resources to be effective (see subsection 7.2). The widespread use of third party resources means that achieving full coverage requires coordination between many different entities, which seems unlikely to happen organically. Without coordination between first and third parties, Web-oriented standard bodies (*e.g.*, W3C) and browser vendors could develop standard mechanisms for normalizing how third-party resources are requested and served to reduce the threat of traffic analysis. For instance, the definition of standard sizes for third-party served resources, and methods to request these resources such that all websites use the same order.

A solution to avoid coordination would be to rethink the trend of creating web development resources as a service, and go back to having first parties hosting and serving the resources. In this spirit, initiatives such as Web Bundles [58] (despite raising other privacy concerns) would remove the need for coordinating between third parties, simplifying the implementation of defenses, and removing vantage points.

IPs & Anonymity set sizes. In the QUIC setting, the adversary is largely helped by the IPs addresses; they can be used to turn the website fingerprinting problem into a closed-world classification problem, to dissect traffic based on first and third parties, and to link together a client's packets.

To address the easy linking of packets, clients could use techniques such as MIMIQ [59] to leverage QUIC's connection migration feature to change a their IP address; or Near-Path NAT [60] or MASQUE [61] to completely hide their IPs. This would force the adversary to probabilistically stitch packets together to form traces. If the source/destination IP/port are not identifying one client, simply rotating QUIC's connection ID might also prevent the adversary from linking together one client's packets.

To address the ease of separating traffic, CDNs that also host websites (*e.g.*, Cloudflare) could proxy the traffic to third parties, such that all traffic is served from a single IP. Finally, the closed-world size could be increased by co-hosting multiple websites on one server, or making a larger number of websites available behind load balancers; or even moved to open world if all web traffic would be downloaded via anonymous communication networks (*e.g.*, Tor [37]) or VPNs.

References

[1] TLS Encrypted Client Hello. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-12. Accessed: 2021-07-05.

[2] DNS Queries over HTTPS (DoH). https://datatracker.ietf.org/doc/html/rfc8484. Accessed: 2021-07-05.

[3] Sudheesh Singanamalla, Suphanat Chunhapanya, Marek Vavruša, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher Wood. Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS. *PETS*, 2021.

[4] Specification for DNS over Transport Layer Security (TLS). https://datatracker.ietf.org/doc/html/rfc7858. Accessed: 2021-07-05.

[5] Heyning Cheng and Ron Avnur. Traffic analysis of SSL encrypted web browsing. *Project paper, University of Berkeley*, 1998.

[6] Andrew Hintz. Fingerprinting websites using traffic analysis. In *International workshop on privacy enhancing technologies*, pages 171–178. Springer, 2002.

[7] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42, 2009.

[8] Brad Miller, Ling Huang, Anthony D Joseph, and J Doug Tygar. I know why you went to the clinic: Risks and realization of https traffic analysis. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 143–163. Springer, 2014.

[9] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted DNS–> Privacy? A traffic analysis perspective. In *NDSS*, 2020.

[10] Jean-Pierre Smith, Prateek Mittal, and Adrian Perrig. Website Fingerprinting in the Age of QUIC. *PETS*, 2021(2):48–69, 2021.

[11] RFC 9000, Section 19.1 PADDING Frames. https://datatracker.ietf.org/doc/html/rfc9000#section-19.1. Accessed: 2021-08-12.

[12] RFC 9000. https://datatracker.ietf.org/doc/html/rfc9000. Accessed: 2021-08-12.

[13] Usage statistics of HTTP/3 for websites. https://w3techs.com/technologies/details/ce-http3. Accessed: 2021-10-03.

[14] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, 2014.

[15] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website Fingerprinting at Internet Scale. In *NDSS*, 2016.

[16] Jamie Hayes and George Danezis. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, Austin, TX, August 2016. USENIX Association.

[17] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated Website Fingerprinting through Deep Learning. In *NDSS*, 2018.

[18] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1928–1943, 2018.

[19] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *PETS*, 2019.

[20] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1148, 2019.

[21] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[22] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 121–130, 2014.

[23] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238, 2014.

[24] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security*, pages 27–46. Springer, 2016.

[25] Jiajun Gong and Tao Wang. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 717–734, 2020.

[26] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. A Real-time Defense against Website Fingerprinting Attacks. In *ACM Workshop on Artificial Intelligence and Security (AISec'21)*, 2021.

[27] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *IEEE Transactions on Information Forensics and Security*, 16:1594–1609, 2020.

[28] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[29] BLANKET. https://github.com/SPIN-UMass/BLANKET. Accessed: 2021-10-08.

[30] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, Roberto Perdisci, et al. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *NDSS*, 2011.

[31] Mike Perry. Experimental Defense for Website Traffic Fin- gerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting. Accessed: 2021-10-03.

[32] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616, 2012.

[33] Giovanni Cherubin, Jamie Hayes, and Marc Juárez. Website Fingerprinting Defenses at the Application Layer. *Proc. Priv. Enhancing Technol.*, 2017(2):186–203, 2017.

[34] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 66–76, 2004.

[35] Steven J Murdoch and Piotr Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In *International workshop on privacy enhancing technologies*, pages 167–183. Springer, 2007.

[36] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348, 2013.

[37] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[38] Simran Patil and Nikita Borisov. What can you learn from an IP? In *Proceedings of the Applied Networking Research Workshop*, pages 45–51, 2019.

[39] Jiasong Bai, Menghao Zhang, Guanyu Li, Chang Liu, Mingwei Xu, and Hongxin Hu. FastFE: Accelerating ml-based traffic analysis with programmable switches. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pages 1–7, 2020.

[40] Diogo Barradas, Nuno Santos, Luıs Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. In *Proceedings of the 28th Network and Distributed System Security Symposium (San Diego, CA, USA*, 2021.

[41] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2053–2069, 2017.

[42] Davide Tammaro, Silvio Valenti, Dario Rossi, and Antonio Pescapé. Exploiting packet-sampling measurements for traffic characterization and classification. *International Journal of Network Management*, 22(6):451–476, 2012.

[43] Valentín Carela-Español, Pere Barlet-Ros, Albert Cabellos-Aparicio, and Josep Solé-Pareta. Analysis of the impact of sampling on NetFlow traffic classification. *Computer Networks*, 55(5):1083–1099, 2011.

[44] Alexa 1M. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip. Accessed: 2021-07-05.

[45] Cisco Umbrella Top 1M Domains List. https://www.trisul.org/devzone/doku.php/cisco_umbrella_top-1m_domains_list. Accessed: 2021-07-05.

[46] The Majestic Million. https://majestic.com/reports/majestic-million. Accessed: 2021-07-05.

[47] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064, 2014.

[48] João Marco C Silva, Paulo Carvalho, and Solange Rito Lima. Inside packet sampling techniques: exploring modularity to enhance network measurements. *International Journal of Communication Systems*, 30(6):e3135, 2017.

[49] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar. Defending Tor from Network Adversaries: A Case Study of Network Path Prediction. *Proceedings on Privacy Enhancing Technologies*, 2015(2):171–187, 2015.

[50] George Nomikos and Xenofontas Dimitropoulos. traIXroute: Detecting IXPs in traceroute paths. In *International Conference on Passive and Active Network Measurement*, pages 346–358. Springer, 2016.

[51] DuckDuckGo Tracker Radar. https://github.com/duckduckgo/tracker-radar. Accessed: 2021-07-05.

[52] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247, 2003.

[53] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[54] Paul Tune and Darryl Veitch. OFSS: Skampling for the flow size distribution. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 235–240, 2014.

[55] OpenWPM: a Web Privacy Measurement Framework. https://github.com/mozilla/OpenWPM. Accessed: 2021-10-04.

[56] Emily Hentgen. Measuring the Security of Website Fingerprinting Defenses. https://infoscience.epfl.ch/record/289258?&ln=en, 2019. Accessed: 2021-10-08.

[57] Rebekah Overdorf, Mark Juarez, Gunes Acar, Rachel Greenstadt, and Claudia Diaz. How unique is your. onion? an analysis of the fingerprintability of tor onion services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2021–2036, 2017.

[58] Web Bundles. https://wicg.github.io/webpackage/draft-yasskin-wpack-bundled-exchanges.html. Accessed: 2021-10-08.

[59] Yashodhar Govil, Liang Wang, and Jennifer Rexford. MIMIQ: Masking IPs with Migration in QUIC. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, 2020.

[60] Near-path NAT for IP Privacy. https://github.com/bslassey/ip-blindness/blob/master/near_path_nat.md. Accessed: 2021-10-08.

[61] Multiplexed Application Substrate over QUIC Encryption (MASQUE). https://datatracker.ietf.org/wg/masque/about/. Accessed: 2021-10-08.

A Traceroute experiments at additional vantage points.

The client location impacts the resources that might be fetched during a page load, and the paths taken by the network traffic to the destination servers. This, in turn, impacts the ASes that can view the traffic. In order to confirm that the trends we observe in our traffic visibility experiment (Section 6.1) hold at different locations, we collect additional traceroutes from three additional vantage points located in Germany, UK, and Singapore.

Figure 18 shows the distribution of webpages seen by different ASes, for our three vantage points. The number of total ASes we encounter on the traceroute are 36, 35, and 23. Out of these 13 ASes are common across all the vantage points. While the ASes that observe the traffic vary across locations, similar to Section 6.1 , only a small proportion of ASes that observe a large proportion of the traffic. Three ASes see more than 25% of the traffic for each vantage point: the client's AS, Google, and Cloudflare.

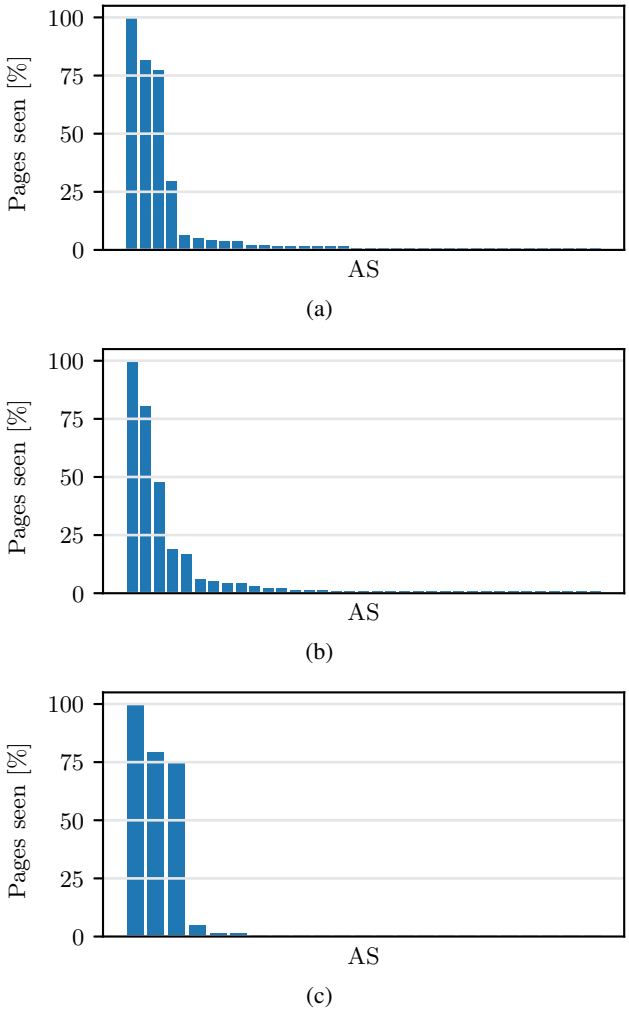

Figure 18: Distribution of webpages seen by each AS, at three vantage points. Only the client's AS, Google, and Cloudflare observe >25% of the traffic.

B Additional graphics

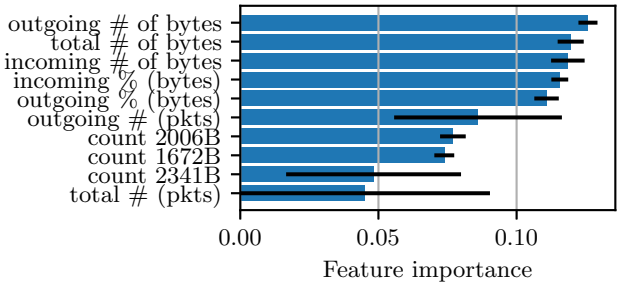

Figure 19: Feature importance for `quic` when using application-layer features (based on HAR captures).