Evaluating practical QUIC website fingerprinting defenses for the masses

Sandra Siby*
Imperial College London
s.siby@imperial.ac.uk

Ludovic Barman
EPFL
ludovic.barman@protonmail.com

Christopher Wood
Cloudflare Inc.
chriswood@cloudflare.com

Marwan Fayed
Cloudflare Inc.
marwan@cloudflare.com

Nick Sullivan
Cloudflare Inc.
nick@cloudflare.com

Carmela Troncoso
EPFL
carmela.troncoso@epfl.ch

ABSTRACT

Website fingerprinting (WF) is a well-known threat to users' web privacy. New Internet standards, such as QUIC, include padding to support defenses against WF. Previous work on QUIC WF only analyzes the effectiveness of defenses when users are behind a VPN. Yet, this is not how most users browse the Internet. In this paper, we provide a comprehensive evaluation of QUIC-padding-based defenses against WF when users directly browse the web, *i.e.*, without VPNs, HTTPS proxies, or other tunneling protocols. We confirm previous claims that network-layer padding cannot provide effective protection against powerful adversaries capable of observing all traffic traces. We show that the claims hold even against adversaries with constraints on traffic visibility and processing power. We then show that the current approach to web development, in which the use of third-party resources is the norm, impedes the effective use of padding-based defenses as it requires first and third parties to coordinate in order to thwart traffic analysis. We show that even when coordination is possible, in most cases, protection comes at a high cost.

KEYWORDS

Website fingerprinting defenses, QUIC, traffic analysis

1 INTRODUCTION

New standardization efforts have greatly increased the privacy of web traffic: *e.g.*, TLS Encrypted Client Hello (ECH) [11] to encrypt Server Name Indication (SNI) in connection handshakes, or (Oblivious) DNS-over-HTTPS [6, 64] and DNS-over-TLS [9] 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 communications' traffic patterns (*e.g.*, volume of packets exchanged or packets' sizes) [25, 37, 38, 50, 62, 67].

QUIC is the next transport layer standard for the web, and it is being rapidly adopted [17]. In order to combat traffic analysis, the working group behind QUIC introduced a `PADDING` frame in the specification [8]. Recently, Smith *et al.* [66] developed a client-side framework that can implement existing website fingerprinting defenses such as Tamaraw [22] and FRONT [33] using the `PADDING` frame. While they showed that it is possible to deploy website fingerprinting defenses solely from the client, their approach is restricted to users in a VPN scenario where IP addresses cannot be used as a signal to reduce the anonymity set of websites [39]. While a non-negligible amount of users browse the web through VPNs, this is not the case for the majority [14] who unfortunately can not benefit from Smith *et al.*'s tools. In this work, we investigate whether defenses built using the `PADDING` frame can be effective outside of the VPN setting, *i.e.*, for *the masses*.

Existing padding-based website-fingerprinting defenses differ in the layer they target and the information they use to inform the defense. There exist *application-agnostic* techniques at the transport and network layers that work independently of the website they are protecting [21, 22, 30, 33, 43]. On the other hand, there are *application-aware* techniques. These can be network- and transport-layer techniques that require prior knowledge (*e.g.*, resource size, resource order, or total size) of the website trace they aim to protect to tailor the defense [58, 60], or application-layer techniques that propose modifications directly on the resources [26, 47, 49]. We provide a comprehensive evaluation of both application-agnostic and application-aware defenses for QUIC traffic, to answer the question: can efficient website-fingerprinting defenses be implemented solely at the network/transport layers or does there need to be involvement of the application layer? This is motivated by the fact that QUIC is built in the user-space, thus allowing it to be more tightly-coupled with applications and making it a good candidate for application-layer-informed defenses. In contrast to prior work, we emphasize *practicality*, *i.e.*, we focus on the feasibility of deploying defenses widely so that the average user can be protected.

Outside of the VPN setting, the IP address can be a unique identifier for many websites [57]. However, users frequently encounter content delivery networks (CDN)-served resources while browsing – as of November 2022, ≈ 44% of the top million sites use CDNs [12]. In fact, there is an increased consolidation on the web due CDN penetration [29], with a consequent increase in size of clusters due to domains co-hosted behind the same IP [40, 61]. When using privacy-preserving protocols such as TLS ECH, this co-hosting provides a natural anonymity set for the websites hosted behind a CDN server's IP address. In such a scenario, website fingerprinting becomes the only way to identify websites hosted by CDNs.

* The majority of this work was completed while Sandra was at EPFL.

Proceedings on Privacy Enhancing Technologies 2023(4), 79–95

https://doi.org/10.56553/popets-2023-0099

We focus on this scenario, where the only available information to the adversary is the CDN server's IP address, IP addresses of a website's secondary resources hosted on non-CDN servers, and metadata such as the size of encrypted data, its timing, and its direction (sent by/to the server).

We answer two questions:

Q1: Can we build effective application-agnostic traffic-analysis defenses using the `PADDING` frame at the transport layer as envisioned by the standard?

- ✓ Transport-layer application-agnostic defenses which build on the QUIC `PADDING` frame to hide packets' sizes or inject dummy packets are not sufficient. Unless one accepts very large overhead, the adversary can use global trace information (*e.g.*, the total number of packets, or the total incoming size) to recognize websites (> 92% F1-score). Despite the use of padding, websites can be identified from their landing pages or from subpage visits, even if the adversary has not previously visited a specific subpage.
- ✓ The centralization of web resources on the Internet, in particular in the hands of Google, creates a favorable setting for the adversary. Traffic analysis solely on the timing of Google resources fetched by a web page achieves > 77% F1-score, requiring *four orders of magnitude less data than using full traces*. This increases the surface of attack from only vantage points in the ASes between the client and the first-party domain host to vantage points in any AS between the client and Google.
- ✓ We show that our findings above not only hold against powerful adversaries that can observe all communications between clients and servers, as those typically considered in the literature [25, 37, 38, 50, 62, 67], but also against weaker, more realistic, adversaries that can only observe partial traces and are restricted in their storage, computation, or bandwidth, *e.g.*, by using limited information from typical network statistics (*e.g.*, NetFlow) the F1-score is 66% at 10% sampling rate. Performance only decreases to close to random guessing when the adversary's vantage position in the network only enables them to observe a very small percentage of the page.

Q2: Can application-aware defenses effectively thwart traffic analysis?

- ✓ Websites can be fingerprinted by either solely their first party or their third party resources (33% of resources in our dataset). When users are not behind a VPN, if all parties do not participate in the padding strategy, an adversary can successfully identify pages (> 91% F1-score), leaving users vulnerable.
- ✓ Application-aware defenses can offer better trade-offs in terms of efficiency than application-agnostic defenses, but their deployment is hindered by the current reliance on third parties in web development practices. Unless those practices change, protecting users would require significant bandwidth overhead and coordination among many parties under very dynamic conditions. We provide recommendations to modify these practices in such a way that it would be easier to support effective defenses against website fingerprinting attacks.

Takeaways. Our results show that `PADDING`-based defenses are not sufficient to thwart traffic analysis. One of the main reasons they fall short stems from the fact that their proposal, rooted in previous website-fingerprinting defenses, is disconnected from the evolution of web practices.

On the one hand, the increased reliance on third parties by web developers implies that no effective defense can be deployed without the coordination of parties who may have different objectives and incentives to protect users. Furthermore, the popularity of a small set of resources used by many websites concentrates subrequests to a small number of service providers (such as Google). This increases the attack surface and eases the task of website-fingerprinting adversaries by decreasing the amount of traffic required to identify websites. On the other hand, the consolidation of services behind CDNs creates a natural reduced anonymity set for websites, which makes website fingerprinting a closed-world classification problem that is much harder to defend against than the open-world counterpart.

These two aspects need to be accounted for before the community can produce website-fingerprinting defenses that will work for the masses. In the last section of the paper, we identify some challenges to deploy effective defenses derived from these issues and provide recommendations to address them.

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, 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 [7]. QUIC was standardized in May 2021 by the IETF. QUIC is the transport protocol for HTTP/3. Adoption of QUIC and HTTP/3 has been rising (as of June 2023, they are used by 26% of the top 10 million websites [17]). Of particular relevance for our work is the QUIC `PADDING` Frame. The IETF QUIC standard describes it as a frame with no semantic value, that can be used to increase packets size and to provide protection against traffic analysis [8].

Website fingerprinting attacks. In website fingerprinting, an adversary analyzes network traffic to infer the website visited by a user. 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 relying on manual features are Wang *et al.*'s k-Nearest Neighbors (k-NN) classifier based on 4226 manually-selected features [71]; Panchenko *et al.*'s Support Vector Machines (SVMs)-based classifier using cumulative sums of packet lengths [56]; and Hayes and Danezis [35] *k*-fingerprinting method (*k*-FP), which models web fingerprints as the leaves of a random forest built on 150 manually-selected features.

On the automatic extraction side, Rimmer *et al.* [59] use deep learning neural networks (DNNs) to produce attacks that perform

as well as manual approaches. Sirinam *et al.* [65] build on Rimmer *et al.* to develop an attack that achieves high accuracy, even in the presence of defended traces. Last, Bhat *et al.* [20] propose Var-CNN, a hybrid strategy that achieves high accuracy with deep learning even in the presence of limited data. It does so by relying on ResNets trained on packet directions, packet times, and manually extracted summary statistics.

Website fingerprinting on QUIC traffic. Smith *et al.* [67] study the impact of co-existence of TCP and QUIC on the performance of website fingerprinting using *k*-FP and Var-CNN. They conclude 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. To enable comparison with the state-of-the-art, we also use *k*-FP and Var-CNN in our evaluation.

Website fingerprinting defenses. Dyer *et al.* [30] show that *network layer* padding- and morphing-based countermeasures are ineffective in thwarting traffic analysis because they fail to hide coarse packet features. They propose Buffered Fixed-Length Obfuscation (BuFLO), which pads packets to a fixed size and sends them at intervals of time. BuFLO results in a huge overhead. CS-BuFLO [21] and Tamaraw [22] are more efficient, but still impractical. Works such as WTF-PAD [43] and FRONT [33] provide a better trade-off by injecting dummies at appropriate positions in a trace. WTF-PAD injects dummy packets using pre-defined distributions of inter-arrival times to detect gaps, and FRONT injects dummy packets in the front portion of traces, which is known to contain the most information for fingerprinting. Both approaches achieve low protection against deep-learning-based attacks [20, 60, 65].

Other defenses employ *adversarial perturbations* to cause deep-learning classifiers to misclassify traces. These defenses incur lower overhead than prior work. Mockingbird [58] applies perturbations to convert traces into a target trace. It converts traces into sequences of bursts (where a burst is a set of contiguous packets in one direction), and perturbs these burst sequences rather than the raw trace. To compute the perturbation, Mockingbird requires the defender to know the entire trace in advance, which is infeasible in practice. Nasr *et al.* [52] tackle this issue with Blind, a defense that pre-computes blind perturbations that can be applied to live network traffic. Shan *et al.* [60] show that Blind [52] offers lower protection when the adversary trains on perturbed traces. They propose Dolos, which applies adversarial patches or pre-computed sequences of dummy packets to protect network traces. Dolos utilizes a user-side secret to generate patches, making it hard for an adversary to generate the same patch, thereby reducing the risk of adversarial training.

Mockingbird, Dolos, and Blind provide protection to Tor traffic. Tor cells have constant size, as opposed to QUIC packets. These defenses do not account for packet sizes, hindering their adoption to protect QUIC traffic. Mockingbird [58] only considers packet directionality when building dummy bursts. It is unclear how to adapt it to QUIC traffic; a burst can correspond to many QUIC packet sequences. Blind [52] does not use packet size information in their website fingerprinting evaluation. The paper contains a size perturbation technique (tailored to Tor) and uses it for their flow correlation experiments. However, even after communicating with the authors, we were unable to find where in their implementation [4] one can configure this technique, nor are there details about how to configure it for non-Tor traffic. Dolos [60] uses solely direction features to compute patches. Adapting it to QUIC would require integrating size information, for which there is no place in their implementation, with no guarantee that the patches would still be effective. Moreover, Dolos requires a prior connection to the website in order to pre-compute patches which can be used in future connections. Hence, it is predicated on information from the application layer, *i.e.*, which website is being visited. This is in line with our findings on the importance of having the application layer informing any network layer defense (Section 5). Due to the challenges associated with adapting these Tor-tailored systems to our QUIC scenario, in our work, we compare to FRONT [33], which Smith *et al.* show can be implemented for QUIC [66].

Finally, there are defenses at the *application layer*. Luo *et al.* [47] propose 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 [49] 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 against traffic analysis attacks [23, 71]. Cherubin *et al.* [26] developed client- and server-side defenses, LLaMA and ALPaCA respectively, tailored towards Tor onion services. These defenses only work well in 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 their performance in dynamic web scenarios where there are a large amount of third party resources (see Section 5).

3 ADVERSARIAL MODEL AND DATASETS

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 domain visited by the user.

The adversary A observes IP packets. We assume applications and websites hosted behind content delivery networks (CDNs) that use QUIC, and protocols to protect sensitive information, *e.g.*, TLS ECH. The adversary does not possess any decryption keys, and relies only on the size and timing of the observed packets. We assume that DNS queries are done in a private manner (*e.g.*, via DoH [6] with appropriate padding [62]) and reveal no information to A. 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 call A's *observation* a collection of flows corresponding to the network connections generated when the user browses a single website. Each flow contains $[\mathrm{IP}_{\mathrm{source}}, \mathrm{IP}_{\mathrm{dest}}, p_1, p_2, \dots]$, 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 indicate packets in the opposite direction.

Vantage Points. Following prior work [32, 42, 51], we consider each AS (autonomous system) on the path of the client traffic as a realistic adversary. Each AS' middlebox, router, or switch that routes

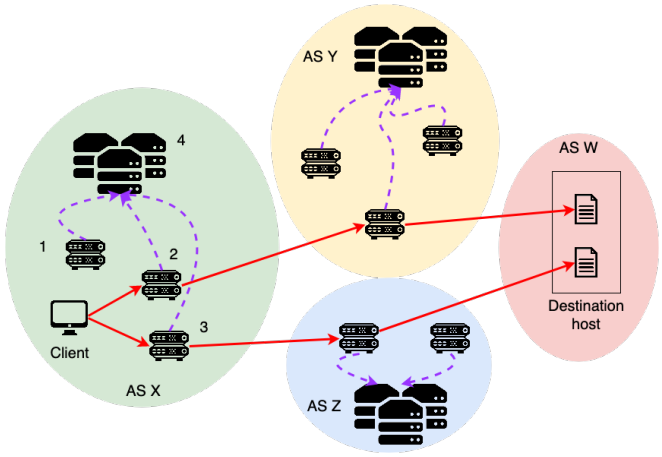

Figure 1: An adversary can be on any AS (X, Y, Z, or W) with vantage points on the client's traffic path (solid red arrows). The vantage points transmit recorded data to a location that can perform traffic-analysis at scale (dotted purple arrows). If the adversary is AS X, vantage points 2 and 3 transmit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

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 [20, 25, 35, 37, 38, 50, 56, 59, 65, 67, 71]. 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 web page, they might only observe a portion of the traffic (*e.g.*, the loading 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) [42, 51].

3.1 Website fingerprinting

As in prior work, we implement website fingerprinting attacks as a supervised learning problem. The adversary identifies the CDN IP containing the domains that they want to target. Then, the adversary enumerates all the 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 which domain it belongs to.

We implement the attack using a random forest classifier, a simple and effective model frequently used in website fingerprinting; and Var-CNN [20] to validate our results against the state of the art [67]. We use the comprehensive set of features proposed by Hayes *et al.* [35] for performing website fingerprinting on Tor. 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. We use 10-fold cross validation, and provide the average and standard deviation of the classifier's F1-score for all our results. We use the mean decrease in impurity to measure the feature importance when evaluating the random forest classifier [15].

Table 1: Overview of datasets. All datasets except `CHROMIUM` are collected using Firefox.

Experiment	Identifier	# webpages	# samples
Landing pages main set (Mar'21)	`MAIN`	150	40
Landing pages large set (Nov'22)	`LARGE`	350	40
Influence of time (Sept'21)	`TIME`	145	40
Influence of client (May'22)	`FIREFOX`	131	40
Influence of client (May'22)	`CHROMIUM`	131	40
Domains (Nov'22)	`HET`	60	35

3.2 Adversarial Capabilities and Goals

We assume applications and websites hosted behind CDNs that use QUIC, and protocols to protect sensitive information, *e.g.*, TLS ECH. The information available to the adversary comprises the server's IP address, which is determined by the content delivery network, and any application-specific metadata such as the size of encrypted data, which is determined by the application. As the IP is known, an adversary can enumerate the domains this IP hosts (*e.g.*, through DNS reverse lookups or from DNS scans using public name sources, including Certificate Transparency logs), collect pages associated with the domain, and train a classifier on these samples. This is a more tractable scenario than that of the open-world Internet where the assumption is that the user may visit websites outside of those monitored by the adversary. As tools such as ZDNS can enumerate all the domains in the IPv4 space in $\approx$ 12 hours [41], enumerating only the domains on the target IP can be done in a few hours and thus adversaries can easily incorporate this step in their attack pipelines.

We study two adversarial models: unconstrained and constrained, depending on the adversary's visibility on traffic and the resources they can dedicate to fingerprinting. We also study different adversarial goals and client setups that may impact the adversary's success. We summarize details of the datasets we collect in Table 1.

3.2.1 Unconstrained adversary. An *unconstrained* adversary can observe and process *all* the traffic associated with a web page visit. We assume this adversary can have one of two goals:

Fingerprinting landing pages (Homogeneous Closed World). In this scenario, the most common in the literature, we assume that users only visit landing pages [20, 35, 37, 59, 65]. Thus, the adversary only needs to collect landing pages to train their model. The training and testing sets contain the same web pages, for which the traces vary due to the page content changing when traces are collected at different points in time. As QUIC leaves IPs unprotected, the adversary can limit the anonymity set of a web page, and this corresponds to a classic *closed world* classification.

Dataset creation. Prior works rely on lists of most visited sites (Alexa, Umbrella, *etc.*) to build datasets. These domains are typically hosted on different IPs, and thus would not be in the same anonymity set in our adversarial model. This would only happen if the client uses a VPN, which is out of the scope of this work. We work with a CDN provider to identify realistic anonymity sets that could be targeted by the adversary. We identify 13, 744, 979 unique domain names on

the CDN in Mar 2021, and use `zdns` [41] to find that these domains are hosted on 593,338 IPs or anonymity sets for the adversary.

The anonymity set sizes follow the same distribution as that found by Patil *et al.* [57] (see Figure 12 in Appendix). We find that 60% of these domains ($\approx$ 8 million) are hosted on a unique IP address: when observing one of these IPs, the adversary is certain of which domain is being visited. Therefore, these domains are out of scope for our study. Only 50k IPs (8.5% of the dataset) host more than 150 domains, with one hosting as many as 56,319 domains. We choose a cluster of 150 websites for our experiments: which is a harder scenario to attack than the 91.5% of the IPs served by the CDN provider. We collect all the traffic generated when a client visits the websites on the cluster – this traffic consists of both connections to the CDN and to other hosts to fetch secondary resources on the websites. We observe that this cluster has a high percentage of TLS traffic: only 4% (std 1.7%) of the traffic is transmitted over QUIC. This results in the classifier focusing on TLS-specific features, preventing us from drawing meaningful conclusions about QUIC traffic's vulnerability or QUIC-oriented defenses (results in Appendix C). We also run experiments with other clusters of similar size and obtain comparable results.

To address this problem, following the example of Smith *et al.* [67], we crawl Alexa 1M [3], Umbrella 1M [5], and Majestic 1M [10]. We perform HAR captures[1] and we identify the protocols used by those websites. Unlike Smith *et al.* [67], we consider only domains that primarily use QUIC. We select 150 of these domains, and collect a dataset of traces: `MAIN` (collected in March 2021). `MAIN` has 70% of all traffic over QUIC (std 3%). We also collect a larger dataset of 350 domains, `LARGE` (collected in November 2022, with 50% QUIC traffic), to evaluate whether fingerprinting performance changes with an increase in the dataset size. We note that given our measurements, a larger dataset would be rare in a CDN scenario– only 2.2% of the IPs in our CDN dataset host more than 350 domains.

Fingerprinting domains (Heterogeneous Closed World). The previous scenario is somehow artificial because real-world users visit more than the landing pages of domains. Thus, only training on landing pages would not work well for an adversary in a realistic deployment. To model a domain, the adversary needs to train on both landing and subpages. Yet, due to the visited IPs being visible, the classification problem is still a closed world: the adversary has a finite set of domains to associate traffic traces to. We collect a dataset to evaluate how the adversary performs when also training on heterogeneous subpages of a domain.

Dataset creation. We collect `HET`, a dataset consisting of subpages for each domain hosted by the IP, instead of multiple samples of the landing page. We enumerate all pages that can be visited from the landing page (sharing the same domain as the landing page), and the pages that can further be visited from those subpages. Since the classifiers require a reasonable number of training samples, we limit our study to a set of 60 domains hosted by the target IP which have at least 35 subpages.

Website-fingerprinting robustness We collect data to study additional factors that may influence the performance of the adversary. To study the stability of website fingerprinting attacks over time, we collect a QUIC-dominated set of traces from the same domains as in `MAIN` 6 months later, in September 2021 (`TIME`). In order to evaluate the influence of the client's browser, we collect `FIREFOX` and `CHROMIUM`– two datasets collected during the same time period, with the Firefox and Chromium browsers.

Data collection. To build the datasets, we collect PCAP network traces from visits to each landing page of the domains in our list. We use Firefox isolated in 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 clean the caches by creating a fresh Firefox profile. We extract well-formed TLS and QUIC packets from the traces. To avoid relying on plaintext markers, we follow the approach of Smith *et al.* [67], and only extract the time and size of the sent and received packets. We use Firefox 88.0 for `MAIN` and `TIME`, Firefox 98.0 for `LARGE`, `FIREFOX` and `HET`, Chromium 101.0 for `CHROMIUM`.

3.2.2 Constrained adversary. We assume that vantage points do *not* have the capability to run machine-learning tasks [18, 19]. 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. In practice, mirroring all traffic is prohibitively expensive [53]. Thus, we also study a *constrained* adversary that only transmits summaries of locally computed statistics from sampled data [24, 68].

We measure the adversary's cost to perform the 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 evaluate the adversary's success using filtered versions of the datasets described above.

We follow some simplifying assumptions similar to prior work in website fingerprinting [20, 59, 67]: we assume that the adversary can identify the start and end of a trace, and that users visit webpages sequentially without background traffic.

4 APPLICATION-AGNOSTIC `PADDING`-BASED DEFENSES

In this section, we study whether `PADDING`-based transport-layer defenses, without the use of application-layer information, can thwart website fingerprinting.

4.1 Unconstrained Adversary

First, we evaluate the effectiveness of defenses against a powerful adversary that can observe *all* the traffic associated with a site visit, and can store and process *all* the traffic that it observes. Such an adversary could be AS X in Figure 1, if this AS would not have bandwidth or storage constraints.

4.1.1 Unprotected Traces. First, we determine the performance of the adversary on unprotected QUIC traces on which no `PADDING` frames are used. We study different scenarios according to the goals defined in Sect. 3.

Fingerprinting landing pages (Homogeneous closed world). A random forest classifier obtains a F1-score of 95.8% when users only

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

Table 2: Performance of website and IP fingerprinting on the LARGE and MAIN datasets (10 experiments).

Method	F1-score
Website fingerprinting (MAIN)	95.8 (std. dev. 0.4)
Website fingerprinting (LARGE)	93.7 (std. dev. 0.2)
IP fingerprinting with primary IP (LARGE)	70.6 (std. dev. 0.1)
IP fingerprinting without primary IP (LARGE)	37.5 (std. dev. 0.1)

browse landing pages (MAIN, Table 2, first row). The results are orders of magnitude better than random guessing (0.67%). We repeated this experiment with ten different 150-sites sets and obtained similar results.

These results hold when considering a large anonymity set for the CDN scenario (LARGE, second row). We also try Var-CNN [20] on MAIN and we obtain an F1-score of 92.28%. As the performance of random forest is better, and it gives us the advantage of interpretability, we use random forest for all our experiments. As we do not observe a large difference between MAIN and LARGE, we pick MAIN (which shows a greater advantage for the adversary) and datasets of equivalent sizes for our remaining experiments.

Unlike in prior work [67] which considered a VPN scenario, in our threat model, the adversary can observe the IP addresses of the communication end-points. Hoang *et al.* [39] showed that destination IPs can be used to identify the websites visited by a user. To understand whether the adversary can take advantage of the IPs, we conduct the IP fingerprinting process described in [39]. We perform repeated DNS resolutions of the resources loaded by each website in LARGE to build a database of IP fingerprints. For each website, we perform 20 resolutions per day over a period of 10 days. We perform these resolutions right after the trace collection of LARGE, so that IP-domain mappings are as close as possible to our traces.

We use the basic-IP fingerprinting methodology described in [39] to find the best match between our LARGE traces and the fingerprints. A domain fingerprint of a website consists of two parts: the *primary* domain which is the domain of the URL in the address bar (the domain of the first connection), and *secondary* domains which can be different from the primary domain and host other resources on the website. In this method, first the adversary matches the IPs of the primary domain to the fingerprints to get a set of candidate sites. Then, they match the IPs of secondary domains against the candidates' fingerprints to obtain a final match. We perform two experiments: in the first, we perform the IP fingerprinting as described. In the second, to match our threat model in which all sites are hosted on a single IP by a CDN, we perform the matching solely on secondary IPs. The results for the two experiments are shown in the third and fourth rows of Table 2.

The IP-fingerprinting performance we observe is worse than in Hoang *et al.*'s work. This difference could arise due to differences in crawling periods – Hoang *et al.* crawled 200k sites 24 times over 2 months, whereas we crawl 250 sites 200 times over 10 days. Longer crawls may have larger variation in fingerprints, leading to better performance. The websites themselves may lead to differences – sites sharing resources may have fewer variations in IP fingerprints, yielding worse performance.

Table 3: Fingerprinting Landing webpages vs. Website.

Scenario	Dataset	F1-score
Landing webpage (baseline)	FIREFOX	97.8 (std. dev. 0.4)
Website (all subpages known)	HET	96.4 (std. dev. 0.9)
Website (unknown subpages)	HET	94.1 (std. dev. 1.1)

We see that IP fingerprinting is less accurate than website fingerprinting, especially when the primary IP is not available. When the primary IP is used, IP fingerprinting presents a somewhat bi-modal behavior. It is very reliable on 32% of the sites for which it correctly classifies *all* the traces (vs. 18% fully correct classification by website fingerprinting). But, it performs very poorly on the remaining 68%. Website fingerprinting is not perfectly reliable for more websites, but overall performs better.

The advantage of website fingerprinting grows when the primary IP is unavailable. Then, website fingerprinting correctly predicts all samples for ≈ 35% of the sites as compared to ≈14% for IP fingerprinting. In other words, IP fingerprinting is useful when there is a stable primary IP to provide a strong signal. Given our threat model where the primary IP is not a distinguishing signal (since all domains are hosted on the same CDN), in the remaining sections, we will only use website fingerprinting as we are studying the effect of website fingerprinting defenses.

We note that an adversary could potentially combine the two attacks to improve their performance. For example, an adversary could use IP fingerprinting with the secondary IPs to further reduce the size of the closed world, and then perform website fingerprinting on the domains in the smaller world.

Fingerprinting domains (Heterogeneous closed world). Next, we study the case in which users visit any website within a domain. We use the FIREFOX (as the fingerprinting landing webpage baseline) and HET datasets. These datasets use the same browser and are collected during the same time period. We use only the domains found in HET when attacking the FIREFOX dataset (results in Table 3).

For HET, we evaluate two cases. In the first case (second row), the adversary has trained on all the subpages of a site, *i.e.*, the training set has the same distribution as the test set. In the second case (third row), the adversary has trained on a subset of subpages and has to classify unseen subpages, *i.e.*, there is a shift between the training and testing sets distribution. We use 30 subpage samples per domain to train the classifier, and 5 subpage samples to test (with different train and test samples per fold). We use one sample per subpage, *i.e.*, every subpage sample is different. We conclude that, even when the adversary has to classify unknown subpages, there is a small performance drop, maximum ≈ 3%. The adversary's success is most likely caused by subpages of the same site sharing many resources, leading to very similar traces.

4.1.2 Robustness. Next, we study factors that can influence the adversary's fingerprinting performance.

Influence of time. The results in Table 2 assume that the adversary collected the training set for their classifier close in time to their attack. We study the performance of the attack when trained on traces collected at a different time than testing in Table 4. We use the

Table 4: Influence of time. F1-score when training on the row dataset and testing on the column dataset.

F1-score	MAIN	TIME
MAIN	95.8 (std. dev. 0.4)	35.2 (std. dev. 1.2)
TIME	17.5 (std. dev. 2.1)	96.4 (std. dev. 0.2)

Table 5: Influence of client. F1-score when training on the row dataset and testing on the column dataset.

F1-score	FIREFOX	CHROMIUM
FIREFOX	95.6 (std. dev. 0.3)	35.6 (std. dev. 2.4)
CHROMIUM	22.9 (std. dev. 1.6)	92.9 (std. dev. 0.3)

MAIN and TIME datasets, collected 6 months apart. We note that TIME has 145 domains instead of 150, due to failures in data collection (unreachable sites and repeated timeouts), hence we consider only these 145 domains. For both datasets, the classifier performance remains consistent when it is trained on data collected close to the attack. However, the performance drops significantly when used on data collected at a different point in time. This indicates that for an adversary to be successful, they would need to update their training data with more recent samples to keep up with constantly evolving web pages.

Influence of client. We consider the influence of the client setup on the adversary's performance. We use two datasets, FIREFOX (Firefox 98.0) and CHROMIUM (Chromium 101.0), collected at the same time. Both datasets consist of 131 domains after accounting for collection failures. Table 5 shows that the classifier performs as expected when trained and tested on the same client setup. Similar to previous work [62], we see that when the setup changes, performance drops, indicating that an adversary would need a classifier tailored to the client setup. This is due to differences in traces caused by differences in parsing HTML and fetching resources – *e.g.*, Firefox traces are generally longer (due to activities such as contacting OCSP servers).

4.1.3 PADDING*-based defenses.* To study the effectiveness of defenses based on the QUIC PADDING frame, we focus on the worst case for the defender: fingerprinting undefended landing pages visited with the same client using fresh training datasets. We run our experiments on the MAIN dataset.

We explore defense strategies that hide local and global features, assuming the presence of an implementation that perfectly protects these features. Table 6 reports the results against different defenses (first row are undefended traces).

Hiding local features. The top predictive features for MAIN are all size-based (Figure 2). We first hide *local* features, *i.e.*, individual packet sizes and timings. When the defense hides individual packet sizes, the attacker performance slightly decreases with respect to undefended traces. Padding individual packets poorly hides the total transmitted volumes, which becomes the top feature once individual sizes are removed. Hiding timings also has minimal impact on the adversary's performance.

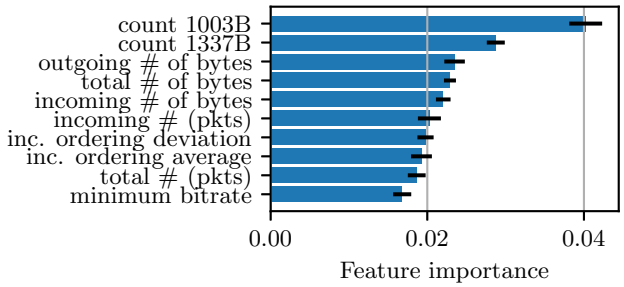

Figure 2: Feature importance for MAIN

Table 6: MAIN dataset: Mean classifier performance on defended traces.

Variant	F1-Score	Std. dev.
undefended	95.8	0.4
hiding individual sizes	93.9	0.4
hiding all timings	95.5	0.3
+ hiding total transmitted sizes	84.6	0.5

Hiding global features. To hide *global features*, *i.e.*, the total transmitted volume, we increase the size of all packets such that the total transmitted size is padded to the next megabyte. This yields another small drop in performance. The attacker simply starts using packets orderings as main feature (Figure 3), which are almost as informative as sizes.

Injecting dummies. We then include dummy packets (padded to the maximum size) to hide individual packet orderings. Since we cannot use existing defenses based on adversarial perturbation to find the optimal placement of dummies (see Section 2), we inject dummy packets based on FRONT [33], which [66] showed can be implemented as a client-side-only QUIC defense. FRONT has four parameters that can be adjusted: N_s and N_c, which are the maximum number of dummies that can be sent by the server and client respectively, and W_{min} and W_{max}, which indicate the time window within which the dummies must be sent.

We inject dummies using the FRONT parameters suggested by Smith *et al.* [66]: $N_s = N_c = 1300$, $W_{min} = 0.2s$, $W_{max} = 3s$. We calculate the bandwidth overhead using the same definitions as in [66], *i.e.*, the increase relative to the transmitted data. We find that FRONT achieves a significant reduction only at a sharp increase in cost: to obtain $\approx$ 70% F1-score drop for the adversary requires an overhead of 2.31. This is in addition to the already large overhead in terms of padding used to hide local and global features (mean cost of 612 kB per trace, with a large standard deviation: 440 kB). We also experiment with different FRONT parameters (as shown in Table 7), and find that reducing the overhead comes at the cost of defense effectiveness. (We note that we cannot have a direct comparison with Smith *et al.* [66] since their evaluation is in an open-world VPN scenario.)

Table 7: F1-score and bandwidth overhead when injecting dummies using FRONT. We vary the FRONT parameters N_s, N_c, and W_{max}.

F1-Score/Overhead		$N_c = N_s$		
		325	650	1300
W_{max}	0.5	64.9/0.55	60.6/1.15	58.3/2.24
	1	60.4/0.55	53.8/1.12	48.1/2.26
	2	53.3/0.56	43.3/1.12	34.9/2.22
	3	51.4/0.56	41.0/1.12	31.4/2.31

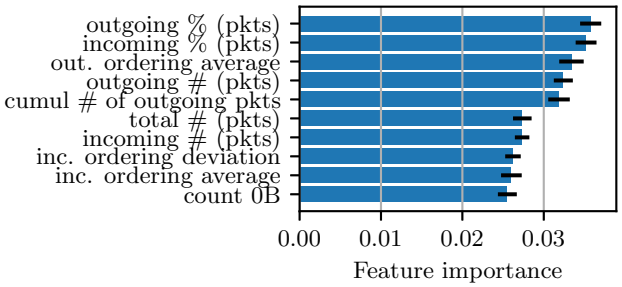

Figure 3: Feature importance when hiding global features (last row of Table 6).

4.2 Constrained Adversary

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; or that may not have the capability to process this traffic or to transmit it to a location suitable for analysis. We now study the performance of these adversaries, and whether PADDING-based defenses could protect against them. We run our experiments on the MAIN dataset.

4.2.1 Limited traffic visibility. To understand the impact of limited visibility on the adversary's performance, we simulate an AS adversary with partial view of the client's traffic. We determine which parts of the traffic an AS would see using HAR captures to identify resources requested during page loads. Then, as in prior work [44], we use traceroutes to record the ASes in the path taken by the resource requests. We set `traIXroute` [54] to use `scamper` (configured with the Paris traceroute technique). As Juen *et al.* [44], we discard route hops that do not have IP or AS information (asterisks in the traceroute). To avoid inaccuracy in our analysis, we do not attempt to fill these gaps in routes via stitching. Thus, our results provide a lower bound on the amount of traffic that an AS adversary sees.

We then simulate the partial view of the adversary by filtering out TLS/QUIC connections that do not correspond to the resources visible to the adversary. This filtering is based on the destination IP address, and the SNI when it exists. We observe a total of 974 routes in the MAIN dataset. These results are from traceroutes collected from the same location as the MAIN dataset (in France). We ran traceroutes from other vantage points (Germany, UK, and Singapore), and observed the same trends. We report on these additional experiments in Appendix A.

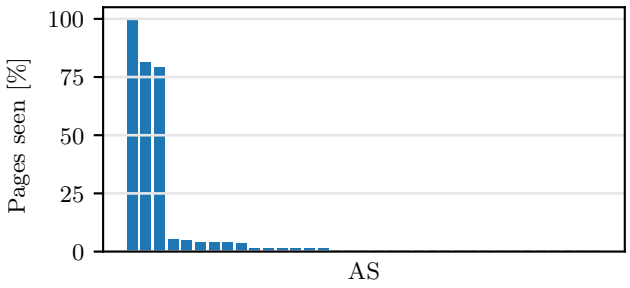

Figure 4: Distribution of web pages seen by each AS. Only three ASes (client's AS, Google, Cloudflare) can observe traffic from all the pages. Most ASes observe less than 10% of pages.

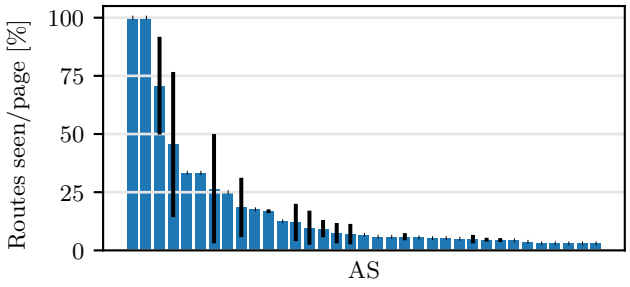

Figure 5: Distribution of routes per web page seen by each AS. Only three ASes (client's AS, OVHcloud, Google) observe more than 50% of the traffic per site.

Figure 4 shows how many webpages are *seen* by each AS. We consider the web page is seen if the AS sees any traffic associated with this web page visit (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 these ASes were to be the adversary, they would not be able to fingerprint traffic from most websites hosted by the IP that our attacks target.

We show the classifier performance for some ASes in Table 8. The few ASes that have a substantial view of the client connections, *e.g.*, Google or Cloudflare, obtain high F1-score. Most other ASes observe traffic from very few pages, *e.g.*, LEVEL3, Facebook, or VNPT-AS-VN observe less than 5% of the pages in the dataset, meaning that they cannot fingerprint the remaining 95%. For the pages that they observe, they obtain high F1-scores. VNPT-AS-VN observes traffic for just one page and, thus, always identifies it.

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. For every web page an AS sees, we study what portion of this page they can observe (see Figure 5). Our source AS, naturally, sees 100% of the traffic of all pages in our dataset. Another AS, 4367 (belonging to OVHcloud), sees 100% of page traffic for a very limited number

Table 8: Mean classifier performance on different AS views.

AS	Name	# Pages	F1-Score
15169	Google, LLC	118	89.5
13335	Cloudflare, Inc	115	92.9
3356	LEVEL3	7	81.7
32934	Facebook, Inc	5	92.3
45899	VNPT-AS-VN	1	100.0

of pages. The Google AS is the second highest, seeing ≈ 70% of the routes for 80% of the pages in the dataset. All other ASes see less than 50% of the routes associated with the pages for which they can observe traffic, and therefore are not much of a threat.

Given these results, we expect the effectiveness of PADDING-based defenses in this scenario to be similar to the unconstrained adversary. If the AS has high visibility on the traces, they are essentially unconstrained and defenses cannot significantly reduce the performance. If the AS observes little traffic, their performance will be already low and the gain provided by defenses can only be marginal, while still imposing high bandwidth overhead.

4.2.2 Limited processing power. To perform website fingerprinting, adversaries must have storage and computation capabilities, which middleboxes typically do not have. In fact, typical network monitoring solutions only record aggregate statistics over sampled traffic [63]. Common tools for network sampling include *NetFlow* and *sFlow* [51]. More efficient techniques have been proposed in academic papers (*e.g.*, sketching [45, 46] or *skampling* [69]), but to the best of our knowledge, they are not widely used.

We focus on NetFlow, which is widely deployed on the Internet. 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. We down-sample the PCAPs uniformly to the desired sampling rate, and 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 in the wild range from 50% to 0.1% [24]). Then, we adapt the features used by our classifier to NetFlow summaries. These summaries record the number of packets but not their sizes, timings, or directions. For classification, 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. We acknowledge there could be better features and that our evaluation only provides a lower-bound on the attacker performance.

We show the adversary's success on the NetFlow summaries in Table 9. Moving from full packet data to flow summaries leads to a significant reduction in the adversary's performance: 29 percentage points lost when 10% of packets are sampled. Yet, the F1-score for any sampling rate is much higher than random guessing (F1-score=0.6%).

Defending NetFlow traces Regardless of the sampling rate, the most important features when using NetFlow are the total number of bytes and packets (Figure 6). We explore a defense that hides both per-flow metrics and overall statistics about the number of bytes and packets exchanged. For full traces, we hide global statistics by padding the

Table 9: Mean classifier performance and median storage cost per sample for Sampled NetFlow, `MAIN`.

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

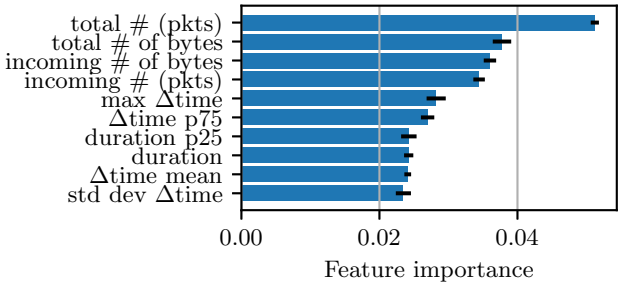

Figure 6: Feature importance for classifying NetFlow with 1% packet sampling rate.

Table 10: Mean attacker performance on defended NetFlow, `MAIN`.

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

total transmitted bytes to 22 MB and the number of packets to 25K (the maximum transmitted size and number of packets we observe in our dataset). This padding is added uniformly to all the flows of one sample. For sampled traces, we reduce the padding with the sampling rate.

The defense reduces the attacker performance (Table 10), introducing prohibitive cost (≈ 39 MB per trace in median). We also observe that most of the gain in privacy compared to the standard setting (95.8%) comes from the sampling rather than the padding (*e.g.*, for NetFlow 1%, −54.1% via sampling vs. −20.1 % via padding).

4.2.3 Inexpensive fingerprinting due to resource centralization. We show how the common use of Google resources by web developers can be used by constrained adversaries to bypass these limitations.

From the (incomplete) AS information on the network level, we found that at least 118 out of the 150 websites in our dataset were seen by Google, *i.e.*, the traffic traverses Google's AS. From the HAR captures, we observe that most websites request resources from a Google-owned domain (125 websites in `MAIN` (83% of the dataset)). We confirm this result in Section 5.1. While studying the traces, we also observe that the order in which these resources are loaded is

Table 11: Mean classifier performance and median storage required per sample on traces filtered by connections to Google services, MAIN. 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

website-particular, *i.e.*, even when two sites load the same set of resources, these resources are loaded at different times –with respect to the time of query of the home page– and in different orders. Such website-dependent behavior results in a fingerprint. This fingerprint, caused by the centralization of web resources at Google, can 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 validate this, we study the performance of an adversary that only records the traffic towards Google services. We filter 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 check the ownership of the requested URLs in our HAR capture using Tracker Radar [2]. We use the following Google-owned domains: google.com, gstatic.com, youtube.com, doubleclick.com, ggpht.com. We extract the sending times of the packets containing a ClientHello to these Google IPs and domains. For MAIN, 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.

We show in Table 11 that even when *using only the timing of requests to Google*, the adversary achieves a F1-score of 77.9% for the 125 websites that use some Google resource. To obtain this result, the adversary needs *only* ≈ 61 *B per connection*, a saving of four to five orders of magnitude compared to recording full network traces. The feature analysis confirms that the timings between sub-resources is what helps the attacker in this case (Figure 13 in Appendix).

4.3 Takeaways

The results in the previous section confirm that website fingerprinting is a threat for QUIC traffic [67] and that application-agnostic defenses are ineffective unless they incur an unreasonable overhead [30]. Additionally, we show that the threat persists even when the adversary has limited visibility or storage capabilities; that domains can be identified even when users browse subpages. While factors such as time and user client can influence the adversary's performance, these can be overcome by retraining the classifier with newer data, and by using classifiers tailored to specific client setups.

More interestingly, we find that the fact that resources are centralized in a few CDNs raises the threat in two ways: it enables ASes in the path to Google to attack a client even if they are not between the client and the server, and they can do so 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).

The failure of transport-layer PADDING-based defenses is because the anonymity sets behind IPs are usually small, and the classifier is able to pick even small differences between global statistics of the traces, *e.g.*, total transmitted size or the total number of packets. The defenses we study cannot hide these differences because, *at the transport layer, they lack information on these global statistics*. We note that these defenses are even less effective against more recent deep-learning attacks and incur high overheads [48, 65]. This further undermines their potential as defenses in a QUIC scenario.

5 DESIGNING APPLICATION-AWARE PADDING-BASED DEFENSES

In this section, we explore whether *application-aware defenses* can be effective against website fingerprinting. We consider any defense that requires knowledge of a page's resources to be an application-aware defense. For instance, knowledge of the number of resources, their size, and their order; or knowledge about the total (incoming or outgoing) size. This information can only be obtained from the application layer, e.g., Smith *et al.*'s [66] practical implementation of FRONT [33] and Tamaraw [22]. While this work demonstrates that QUIC can be used to implement existing network- or transport-layer defenses using a user-space library, it does not explore whether application-layer information could be used to adjust the defense configuration. In contrast, we aim to understand whether using application-aware defenses fare better than their uninformed network-layer counterparts. We study application-aware defenses assuming they are implemented at the application layer, i.e., padding resources directly. Our results hold if the defenses were to be implemented at the transport-layer by passing application-layer information to the middleware that would configure the PADDING frames (see Section 5.2.2).

5.1 Understanding web page composition

We first study different dimensions related to the structure and composition of websites that are relevant for configuring application-aware defenses. We collect page structures by crawling the pages in MAIN with OpenWPM (v0.17.0) [1] using Firefox five consecutive times. OpenWPM logs the HTTP requests that occur during the page load. It also records the originator of requests.

Resource dynamism. Dynamism influences the ease of protecting a page. Less dynamic pages are easier to protect, 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 MAIN, 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. We strip the URL parameters, and plot the distribution of static resources in Figure 7a. The mean proportion of static resources is 88.25% (Std: 22.46%) and the median is 100%. This indicates that pages in our dataset mostly contain static content. Our measurements, however, are taken over a short period of time and dynamism could be more prevalent when web pages are observed over a longer time period.

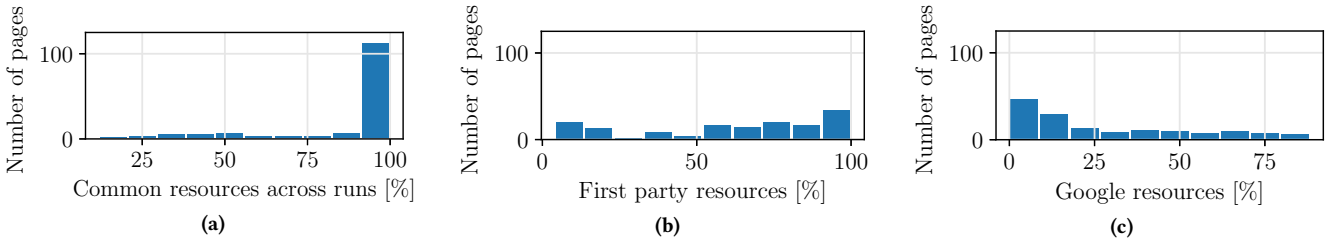

Figure 7: Resource dynamism and ownership for pages in the MAIN dataset. (a) Proportion of resources that remain static across 5 runs. The majority of resources remain constant across runs, indicating low page dynamism. (b) Proportion of first party resources. 18% of the pages have less than 20% of first-party resources. (c) Proportion of Google third party resources. 24% of the pages have more than half their resources served by Google.

Resource ownership. Resource owners can provide information about resources and modify them. Understanding ownership is important to get an idea of how much coordination is required to protect a page. We study whether resources belong to the *first* party (same eTLD+1 as the page) or to a *third* party (different eTLD+1 from the page). For example, on the page `www.example.com`, `img.example.com` would be first-party and `external.com` would be third-party. Using domains as a proxy for ownership is not perfectly accurate: content for `facebook.com` can be served from `fbcdn.net`. While both domains come under Facebook's control, the latter would be identified as a third party. Unfortunately, we need to rely on domains as existing services that provide entity-domain mappings [2] do not have relevant ownership information for almost half of our dataset.

Figure 7b shows the proportion of first-party resources for our dataset. The majority of the pages have a large proportion of first-party resources (Mean 61.18%, Std 31.61%, Median 66.67%), though some pages have as few as 3.92%. On average, there are 5.95 third party domains per page (Std: 7.64, Median: 3), with the number of domains going up to 44 for one of the pages. Visual inspection of the third-party domains shows several domains commonly associated with Google. We map the domains to their owning entities [2] to measure Google's prevalence ([2] contains mappings for Google's domains). Figure 7c shows the proportion of Google resources per page. 24% of the pages have more than half their resources served by Google.

5.2 Application-aware defense strategies

5.2.1 Party-based resource protection. The adversary filter resources from different origins (*e.g.*, only on Google resources as in Section 4.2.3). We assess whether third parties need to be involved to protect a page or it suffices with protecting first-party resources.

We build traces with only first- or third-party resources, and protect those using padding. First-party protection represents a scenario where web pages protect their content using some defense, but third-parties do not cooperate. Third-party protection represents 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 assume the adversary attacks the remaining undefended traffic. In order to keep the same set of websites across the experiments, we discard websites that do not contain third-party or Google resources. This leaves us with 100 websites. As shown in Table 12, the adversary can achieve a high performance by just analyzing partial, undefended traces, regardless of the origin. Thus, for any defense to be effective, all parties serving content for a page must coordinate and actively participate in the protection of resources. In particular, due to its prevalence, *Google must collaborate for any* `PADDING`*-based defenses to be efficient.*

Table 12: Mean classifier performance on traces filtered by 1st / 3rd party and Google CDN, MAIN.

Variant	F1-Score	Std. dev
All traffic	96.9	0.6
Only traffic to/from 1st parties	98.2	0.5
Only traffic to/from 3rd parties	96.8	0.8
Only traffic to/from Google CDN	96.9	0.6

5.2.2 Evaluating application-aware defense strategies. We directly evaluate perturbed application-layer traces as this gives an upper bound on the performance of a transport-layer adversary with respect to a set of features [36]. The reason is that transport-layer features are effectively a noisy version of application layer resources [36] (*e.g.*, the number or 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). We study 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 Section 5.2.3).

Metrics. We use two metrics to evaluate the defenses' success: the performance of the classifier and the overhead imposed in terms of kilobytes of data added per subrequest.

Dataset and features. For the undefended baseline, we use the HAR captures of `MAIN` to derive k-Fingerprinting features.

In practice, our padding and dummy-injection defenses would affect the timing of packets. However, without deploying the defenses, we cannot predict these changes would reflect on our traffic captures. Deployment is not a possibility, as even if we would copy all websites of `MAIN` on a server we control, we would not be able to simulate actions from third parties. Fortunately, timings are less stable (and hence, less useful) than sizes, and therefore, they are

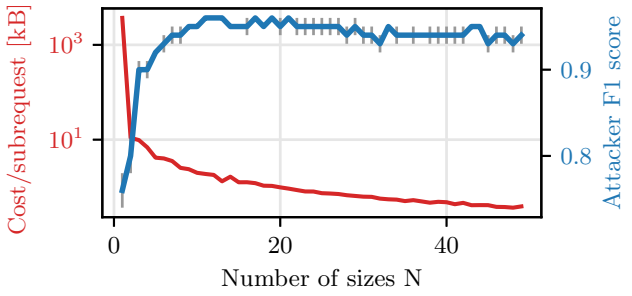

Figure 8: Number of sizes, N, in $\mathsf{pad}_{\text{resources}}$ versus attacker performance and median bandwidth cost per subrequest.

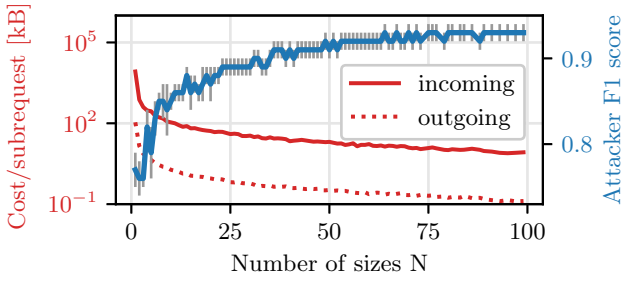

Figure 9: Number of sizes N in $\mathsf{pad}_{\text{total size}}$ versus attacker performance and median bandwidth cost per subrequest.

not among the most important features (Figure 15 in Appendix C). When attacking full HARs, the adversary obtains very good performance (93% F1-score) both with and without timing information. The most important features are size-related, being `bytes_outgoing` the most important feature by a slight margin over `bytes_incoming` (`bytes_total` is the sum of the two). This corroborates the findings by Hentgen: even without timings, evaluating at the application layer yields an upper bound over what happens over the network [36]. In the remaining experiments, we discard timings.

5.2.3 Defense strategies. We now evaluate possible strategies that use application-layer information.

Protecting local features with padding. We design a padding function $\mathsf{pad}_{\text{resources}}$ to hide individual queries and resources sizes. Such a defense must be implemented both on the client and the server. To configure the function, we use (1) the distribution of resource sizes in the set of websites and (2) a parameter N, which defines how many different sizes the defense allows for. The padding function splits the resources sizes into N groups of equal density. For instance, if $N = 1$, all resources are padded to the max resource size in `MAIN`; and if $N = 2$, 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; but also increases bandwidth usage.

We run this defense varying N, and plot the median cost and the attacker F1-score in Figure 8. Only large amounts of padding (small N) have an effect on the attacker accuracy. Padding with large sizes has little effect. For instance, $N = 3$, which results in resources of 5.58 kB, 21 kB, 3.6 MB, decreases the accuracy of the 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 16 in Appendix C), which are sufficient for the attack. The traces' total size are too different to be efficiently hidden through the padding of individual resources.

Protecting global features with padding. Padding only individual resource sizes cannot protect the overall transmitted volume. We design a padding function $\mathsf{pad}_{\text{total size}}$ to pad the total incoming and outgoing sizes. To evaluate the best case defense, we assume the ideal scenario in which the padding effort is split evenly across all the parties queried on one web page. This way, the adversary does not gain an advantage by filtering 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 *e.g.*, using the method by Smith *et al.* [66].

The defense has one parameter, N, which defines the total incoming and outgoing traffic sizes that are allowed. We first compute the maximum total incoming and outgoing traffic in our target dataset, `MAIN`. The maximum total size of queries per 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: 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, 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.

We run this defense varying N, and plot the median cost and the attacker F1-score in Figure 9. As in the transport layer, padding the total size does not mitigate 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.

Protecting global features with dummies. Injecting dummy traffic can also hide the total size [43]. Unlike in Tor, care must be taken that dummies' sizes cannot be easily identified and filtered. For simplicity, we assume dummies are sampled from a dataset that contains the most popular queries and resources across all pages to be defended.

In our experiments, we select popular resources from Google (fonts, analytics, static assets). When a web page is loaded, we choose a number of resources to inject (M). These resources can themselves trigger additional queries. We flip a coin and, with probability p, we inject a dummy resource. We inject these resources at random times over the duration of the connection, such that the adversary cannot use timing to identify and filter out dummies.

We plot in Figure 10 the attacker F1-score for a varying number of dummies. This defense is more effective than the previous ones. For instance, with ($p = 0.5, M = 10$), which injects on average 5 dummy

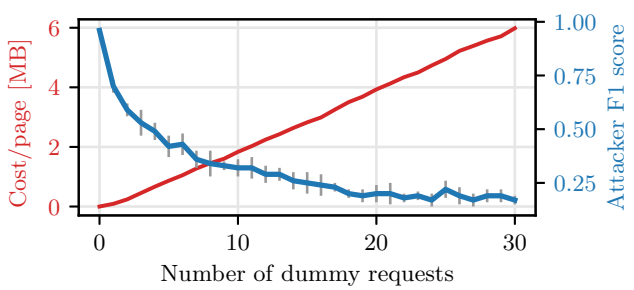

Figure 10: Attacker performance and cost when varying the number of dummies.

requests, the attacker's F1-score decreases from 93% to 54%, at a median cost of 137 kB per loading of a web page. In general, increasing p has better impact on the attacker's performance than increasing 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; but the former reduces the attacker's F1-score to 0.54, and the latter to 0.43.

5.3 Takeaways

Application-aware defenses, while showing better trade-offs than application-agnostic ones, suffer from deployment challenges due to the current state of the web. The presence of a large number of third-party resources on websites means that, contrary to application-agnostic defenses, first and third party need to coordinate to deploy effective defenses. Application-aware defenses need a-priori knowledge of the resources sizes and ordering. This may be hard to obtain when browser & website optimizations (*e.g.*, client caching or pipelining of resources) are in place. Similarly to application-agnostic defenses (see Section 4), it is likely that deep-learning attacks are also more effective than our attacks against application-aware defenses. Our k-fingerprinting features-based evaluation, thus, presents a lower bound for the adversary's success. To understand the extent to which an adversary could undermine `PADDING`-based defense future work is needed to adapt current attacks [26], primarily designed for application-agnostic defenses, to an application-aware scenario.

6 DISCUSSION AND RECOMMENDATIONS

We conducted a comprehensive study of website-fingerprinting defenses of an ecosystem where the majority user does not use Tor/VPNs. We provided evidence of fundamental incompatibilities between today's Web practices and the deployment of effective defenses.

Challenges in application-agnostic transport-layer defenses. First, we confirm that transport-layer defenses, which do not use information from the application layer, are not effective against website fingerprinting [30]. We show that this also applies when the transport protocol changes from TCP to QUIC. The main problem stems notably from the differences in the total sizes of websites, which result in identifying features [35, 55]. Hiding the total size is hard at the transport layer, where the size of objects is not known in advance. Without coordination with the application layer, using the QUIC's `PADDING` frame is unlikely to result in effective defenses.

Application-aware defenses challenges & Next steps. Effective mitigations require application involvement, either in the application code or as part of the browser's functionalities [27]. While defenses with application-layer involvement can obtain better protection at a smaller cost, our investigation shows that current use of third parties hinder the effective deployment of any defense as effectiveness requires *all* resources to be padded (see subsection 5.2). Achieving full coverage requires coordination among many different entities, which seems unlikely to happen organically.

To improve the situation without the need for coordination between first and third parties, Web-oriented standard bodies (*e.g.*, W3C) and browser vendors could develop mechanisms to standardize how third-party resources are requested and served. For instance, defining standard sizes for third-party served resources, and methods to request these resources such that all websites use the same order. Another option 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. Alternatively, CDNs could proxy the traffic to third parties, such that all traffic is served from a single IP. A factor to keep in mind is that unlike defenses for Tor, incentivizing the involved parties to contribute towards defenses for the majority user is a big challenge. Another promising avenue is to explore how client-side measures such as caching or the use of ad-blockers [62] could be used in conjunction with padding-based measures to avoid the need for coordination by eliminating the need to contact third parties.

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 [34] to leverage QUIC's connection migration feature to change their IP address; or privacy proxy [13] or MASQUE [16] to completely hide their IPs; or CoMPS [70] to hide IPs and split traffic across multiple paths. 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. Instead of focusing on a single defense technique, layering defenses (*e.g.*, combining padding and traffic splitting techniques) have shown promising results for Tor traffic and warrants further exploration [48].

Finally, the closed-world size could be increased by co-hosting multiple websites on one server, making a larger number of websites available behind load balancers, or even decoupling the bindings between IP addresses and hostnames [31]; or even moved to open world if all web traffic would be downloaded via anonymous communication networks (*e.g.*, Tor [28]) or VPNs.

ACKNOWLEDGMENTS

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

REFERENCES

[1] 2016. OpenWPM: a Web Privacy Measurement Framework. https://github.com/mozilla/OpenWPM. Accessed: 2023-06-08.

[2] 2020. DuckDuckGo Tracker Radar. https://github.com/duckduckgo/tracker-radar. Accessed: 2023-06-08.

[3] 2021. Alexa 1M. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip. Accessed: 2021-07-05.

[4] 2021. BLANKET. https://github.com/SPIN-UMass/BLANKET. Accessed: 2023-06-08.

[5] 2021. Cisco Umbrella Top 1M Domains List. https://www.trisul.org/devzone/doku.php/cisco_umbrella_top-1m_domains_list. Accessed: 2021-07-05.

[6] 2021. DNS Queries over HTTPS (DoH). https://datatracker.ietf.org/doc/html/rfc8484. Accessed: 2023-06-08.

[7] 2021. RFC 9000. https://datatracker.ietf.org/doc/html/rfc9000. Accessed: 2023-06-08.

[8] 2021. RFC 9000, Section 19.1 PADDING Frames. https://datatracker.ietf.org/doc/html/rfc9000#section-19.1. Accessed: 2023-06-08.

[9] 2021. Specification for DNS over Transport Layer Security (TLS). https://datatracker.ietf.org/doc/html/rfc7858. Accessed: 2023-06-08.

[10] 2021. The Majestic Million. https://majestic.com/reports/majestic-million. Accessed: 2021-07-05.

[11] 2021. TLS Encrypted Client Hello. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-12. Accessed: 2023-06-08.

[12] 2022. CDN Usage Distribution in the Top 1 Million Sites. https://trends.builtwith.com/cdn. Accessed: 2023-06-08.

[13] 2022. Near-path NAT for IP Privacy. https://developer.chrome.com/docs/privacy-sandbox/ip-protection/. Accessed: 2023-06-08.

[14] 2023. 3rd Annual VPN Market Report. https://www.security.org/resources/vpn-consumer-report-annual/. Accessed: 2023-06-08.

[15] 2023. Feature importance based on mean decrease in impurity¶. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html. Accessed: 2023-06-08.

[16] 2023. Multiplexed Application Substrate over QUIC Encryption (MASQUE). https://datatracker.ietf.org/wg/masque/about/. Accessed: 2023-06-08.

[17] 2023. Usage statistics of HTTP/3 for websites. https://w3techs.com/technologies/details/ce-http3. Accessed: 2023-06-08.

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

[19] Diogo Barradas, Nuno Santos, Luıs Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. 2021. 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)*.

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

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

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

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

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

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

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

[27] Alex Davidson, Matthias Frei, Marten Gartner, Hamed Haddadi, Adrian Perrig, J Subirà Nieto, Philipp Winter, and François Wirz. 2022. Tango or square dance? how tightly should we integrate network functionality in browsers?. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 205–212.

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

[29] Trinh Viet Doan, Roland van Rijswijk-Deij, Oliver Hohlfeld, and Vaibhav Bajpai. 2022. An Empirical View on Consolidation of the Web. *ACM Transactions on Internet Technology (TOIT)* 22, 3 (2022), 1–30.

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

[31] Marwan Fayed, Lorenz Bauer, Vasileios Giotsas, Sami Kerola, Marek Majkowski, Pavel Odintsov, Jakub Sitnicki, Taejoong Chung, Dave Levin, Alan Mislove, et al. 2021. The Ties that un-Bind: Decoupling IP from web services and sockets for robust addressing agility at CDN-scale. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 433–446.

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

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

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

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

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

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

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

[39] Nguyen Phong Hoang, Arian Akhavan Niaki, Phillipa Gill, and Michalis Polychronakis. 2021. Domain Name Encryption Is Not Enough: Privacy Leakage via IP-based Website Fingerprinting. In *Proceedings of the 21st Privacy Enhancing Technologies Symposium (PoPETs '21)*.

[40] Nguyen Phong Hoang, Arian Akhavan Niaki, Michalis Polychronakis, and Phillipa Gill. 2020. The web is still small after more than a decade. *ACM SIGCOMM Computer Communication Review* 50, 2 (2020), 24–31.

[41] Liz Izhikevich, Gautam Akiwate, Briana Berger, Spencer Drakontaidis, Anna Ascheman, Paul Pearce, David Adrian, and Zakir Durumeric. 2022. ZDNS: a fast DNS toolkit for internet measurement. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 33–43.

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

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

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

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

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

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

[48] Nate Mathews, James K Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2022. SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 344–361.

[49] Mike Perry. 2011. Experimental Defense for Website Traffic Fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting. Accessed: 2023-06-08.

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

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

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

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

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

[55] Rebekah Overdorf, Mark Juarez, Gunes Acar, Rachel Greenstadt, and Claudia Diaz. 2017. 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*. 2021–2036.

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

[57] Simran Patil and Nikita Borisov. 2019. What can you learn from an IP?. In *Proceedings of the Applied Networking Research Workshop*. 45–51.
[58] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. 2020. Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *IEEE Transactions on Information Forensics and Security* 16 (2020), 1594–1609.
[59] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *NDSS*.
[60] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. 2021. A Real-time Defense against Website Fingerprinting Attacks. In *ACM Workshop on Artificial Intelligence and Security (AISec'21)*.
[61] Craig A Shue, Andrew J Kalafut, and Minaxi Gupta. 2007. The web is smaller than it seems. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. 123–128.
[62] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. 2020. Encrypted DNS–> Privacy? A traffic analysis perspective. In *NDSS*.
[63] João Marco C Silva, Paulo Carvalho, and Solange Rito Lima. 2017. Inside packet sampling techniques: exploring modularity to enhance network measurements. *International Journal of Communication Systems* 30, 6 (2017), e3135.
[64] Sudheesh Singanamalla, Suphanat Chunhapanya, Marek Vavruša, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher Wood. 2021. Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS. *PETS* (2021).
[65] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1928–1943.
[66] Jean-Pierre Smith, Luca Dolfi, Prateek Mittal, and Adrian Perrig. 2022. QCSD: A QUIC Client-Side Website-Fingerprinting Defence Framework. In *31st USENIX Security Symposium (USENIX Security 22)*. 771–789.
[67] Jean-Pierre Smith, Prateek Mittal, and Adrian Perrig. 2021. Website Fingerprinting in the Age of QUIC. *PETS* 2021, 2 (2021), 48–69.
[68] Davide Tammaro, Silvio Valenti, Dario Rossi, and Antonio Pescapé. 2012. Exploiting packet-sampling measurements for traffic characterization and classification. *International Journal of Network Management* 22, 6 (2012), 451–476.
[69] Paul Tune and Darryl Veitch. 2014. OFSS: Skampling for the flow size distribution. In *Proceedings of the 2014 Conference on Internet Measurement Conference*. 235–240.
[70] Mona Wang, Anunay Kulshrestha, Liang Wang, and Prateek Mittal. 2022. Leveraging strategic connection migration-powered traffic splitting for privacy. *Proceedings on Privacy Enhancing Technologies* 1 (2022), 18.
[71] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*. 143–157.

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 4.2.1) hold at different locations, we collect additional traceroutes from three additional vantage points located in Germany, UK, and Singapore.

Figure 11 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 4.2.1 , 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.

B PRECISION AND RECALL FOR THE EXPERIMENTS

We provide the precision and recall numbers for our experiments in the tables below. Tables 13 to 6 shows the results of the experiments

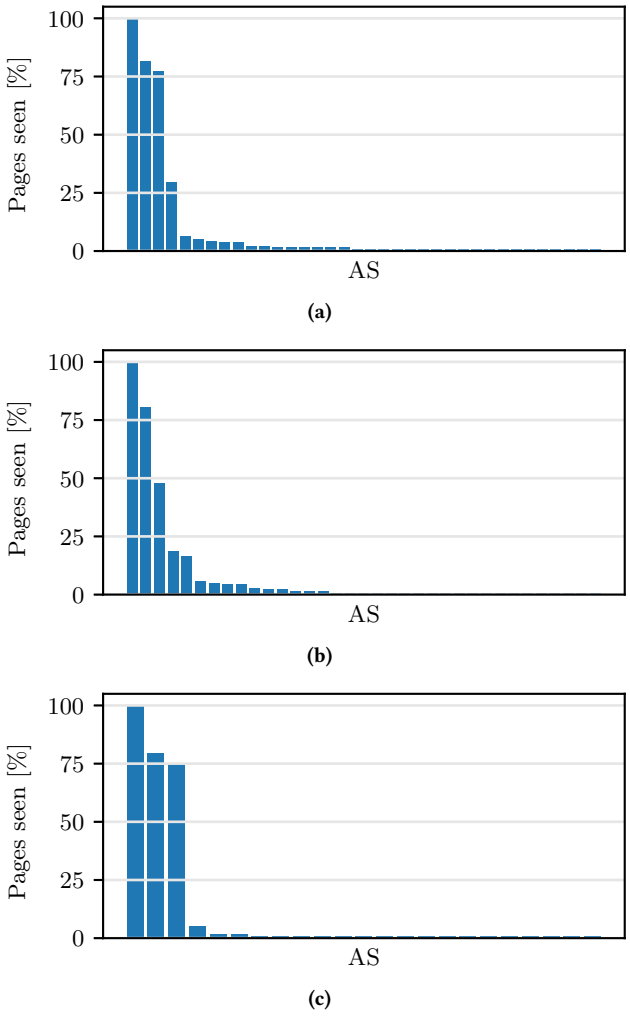

Figure 11: Distribution of webpages seen by each AS, at three vantage points. Only the client's AS, Google, and Cloudflare observe >25% of the traffic.

in Section 4. Table 13 shows the results for the website fingerprinting and IP fingerprinting experiments. Table 14 shows the comparison between the homogeneous and the heterogeneous closed-world experiments. Tables 15 and 16 shows the robustness experiments to evaluate the effects of time and client respectively. Table 17 shows the effect of various defenses on full traces.

Table 13: Performance of website and IP fingerprinting on the `LARGE` and `MAIN` datasets (10 experiments).

Method	Precision/Recall
Website fingerprinting (`MAIN`)	95.8/95.7
Website fingerprinting (`LARGE`)	93.7/93.9
IP fingerprinting with primary IP (`LARGE`)	71.7/71.5
IP fingerprinting without primary IP (`LARGE`)	40.3/40.3

Table 14: Fingerprinting Landing webpages vs. Website.

Scenario	Dataset	Precision/Recall
Landing webpage (baseline)	FIREFOX	98.0/97.8
Website (all subpages known)	HET	97.0/96.3
Website (unknown subpages)	HET	95.1/94.1

Table 15: Influence of time. Precision/Recall when training on the row dataset and testing on the column dataset.

Precision/Recall	MAIN	TIME
MAIN	95.8/95.7	35.6/35.2
TIME	21.6/17.5	96.5/96.4

Table 16: Influence of client. Precision/Recall when training on the row dataset and testing on the column dataset.

Precision/Recall	FIREFOX	CHROMIUM
FIREFOX	95.8/95.7	35.6/35.7
CHROMIUM	26.0/22.9	93.2/92.9

Table 17: MAIN dataset: Mean classifier performance on defended traces.

Variant	Precision/Recall
undefended	95.8/95.7
hiding individual sizes	94.1/93.8
hiding all timings	95.7/95.5
+ hiding total transmitted sizes	85.6/84.9

Tables 18 to 21 shows the results of the experiments in Section 4.1.3. Table 18 shows the classifier's performance when considering different AS views. Tables 19 and 20 shows the classifier's performance on undefended and defended NetFlows respectively. Table 21 shows the classifier's performance when filtering traces for connections only to Google.

Table 18: Mean classifier performance on different AS views.

AS	Name	# Pages	Precision/Recall
15169	Google, LLC	118	89.6/89.4
13335	Cloudflare, Inc	115	93.7/92.9
3356	LEVEL3	7	82.4/81.7
32934	Facebook, Inc	5	93.7/92.3
45899	VNPT-AS-VN	1	100.0/100.0

Table 22 shows the results of the experiment in Section 5. Table 22 shows the classifier's performance when filtering website traffic by first party, third party, and Google services.

Table 19: Mean classifier performance and median storage cost per sample for Sampled NetFlow, MAIN.

Sampling	Precision/Recall	Size [kB]
Full traces	95.8/95.7	312.4
NetFlow 100%	90.8/90.4	25.9
NetFlow 10%	66.5/67.4	3.0
NetFlow 1%	42.2/41.7	0.9
NetFlow 0.1%	18.8/16.8	0.4

Table 20: Mean attacker performance on defended NetFlow, MAIN.

Sampling	Precision/Recall
NetFlow 100%	54.1/53.0
NetFlow 10%	33.6/33.1
NetFlow 1%	21.6/21.6
NetFlow 0.1%	8.3/8.6

Table 21: Mean classifier performance and median storage required per sample on traces filtered by connections to Google services, MAIN. The last two rows use 125 samples.

Variant	Precision/Recall	Size [kB]
Baseline (Full Traffic)	95.8/95.7	312.4
Full traffic to Google	78.9/78.3	112.1
ClientHello's to Google	66.9/67.1	0.1

Table 22: Mean classifier performance on traces filtered by 1st / 3rd party and Google CDN, MAIN.

Variant	Precision/Recall
All traffic	97.4/96.9
Only traffic to/from 1st parties	98.4/98.5
Only traffic to/from 3rd parties	97.3/96.8
Only traffic to/from Google CDN	97.4/97.0

C ADDITIONAL GRAPHICS

Figure 12 shows the distribution of clusters for the 1.3M domains we obtained from our CDN partner. We find that 60% of the clusters have an anonymity set of one, i.e., the domains are hosted on a unique IP. These domains can be identified without conducting website fingerprinting. Only 8.5% of the IPs host more than 150 domains.

Figure 14 shows the feature importance when the classifier is run on the dataset from the cluster. We get an F1-score of 66.6% (std. dev. 0.5). On running a feature analysis, we find that the most important features are TLS-specific. Since these features cannot be protected with a QUIC PADDING frame, we discard this dataset in favor of QUIC-dominated datasets created from website lists.

Figure 13 shows the feature importance when using ClientHello timings to Google services as a feature. The most important features are timings between sub-resources

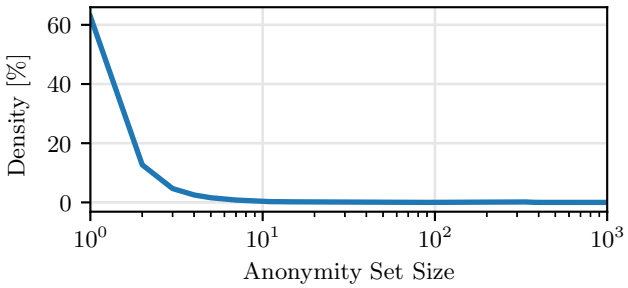

Figure 12: Distribution of the cluster sizes of 1.3M domains.

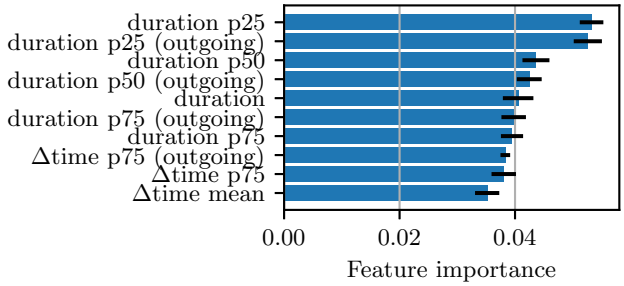

Figure 13: Feature importance for classifying websites based on the timings of their requests to Google services.

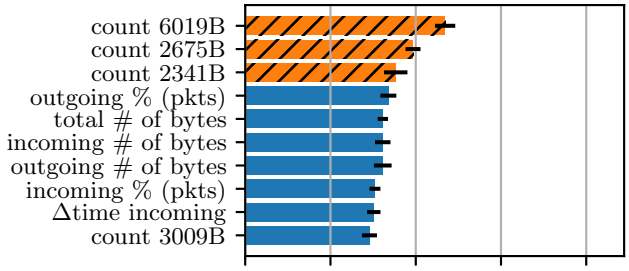

Figure 14: Feature importance for the cluster dataset. Due to low prevalence of QUIC, most of the features are TLS-specific (orange, dashed).

Figure 15 shows the feature importance of MAIN when using application layer features from HAR captures. We find that size based features are the most important, and time-based features do not play a large role.

Figure 16 shows the feature importance when we protect local features at the application layer with a padding function. The function uses a parameter, N, that indicates the number of sizes to which resources can be padded. We experiment with various values of N, and find that only large amounts of padding (small N) have an effect on the adversary's performance. This is because the adversary still has access to the global features such as number of requests and overall volume (as shown by Figure 16 for an example of $N = 3$.

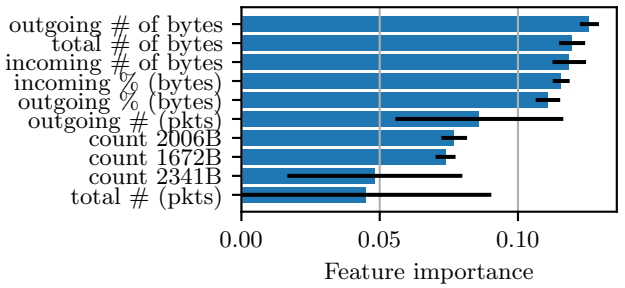

Figure 15: Feature importance for MAIN when using application-layer features (based on HAR captures).

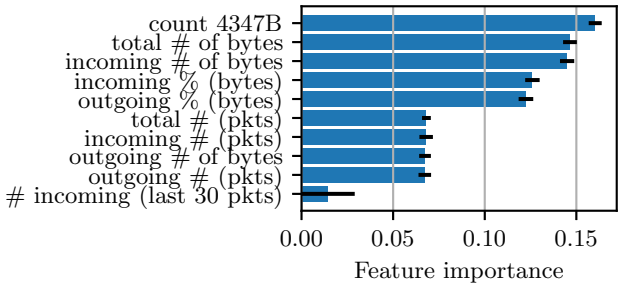

Figure 16: Feature importance with 3 padding sizes: 5.58 kB, 21 kB, 3.6 MB.