You get PADDING, everybody gets PADDING! You get privacy?
Evaluating practical QUIC website fingerprinting protections for the masses.

Sandra Siby ⋆ Ludovic Barman ⋆† Christopher Wood † Marwan Fayed † Nick Sullivan † Carmela Troncoso ⋆
⋆EPFL † Cloudflare Inc.

Abstract—Website fingerprinting (WF) is a well-know threat to users’ web privacy. New internet standards, such as QUIC, include padding to support defenses against WF. Previous work 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. We confirm previous claims that network-layer padding cannot provide good protection against powerful adversaries capable of observing all traffic traces. We further demonstrate that such padding is ineffective even against adversaries with constraints on traffic visibility and processing power. At the application layer, we show that defenses need to be deployed by both first and third parties, and that they can only thwart traffic analysis in limited situations. We identify challenges to deploy effective WF defenses and provide recommendations to address them.

1. Introduction

New standardization efforts have greatly increased the privacy of web traffic: e.g., TLS Encrypted Client Hello (ECH) [13] to encrypt Server Name Indication (SNI), or (Oblivious) DNS-over-HTTPS [6, 59] and DNS-over-TLS [13] to encrypt DNS queries. Yet, encryption alone cannot protect users’ browsing history from traffic analysis. Traffic-analysis attacks such as website fingerprinting (WF), enable adversaries to infer which websites a user visits from the traffic patterns (e.g., volume of packets exchanged or packets’ size) [25, 36, 35, 46, 57, 62].

QUIC is the next transport layer standard for the web that is seeing rapid adoption [16]. In order to combat traffic analysis, the working group behind QUIC introduced a PADDING frame in the specification [12]. Recently, Smith et al. [61] developed a client-side framework that can implement existing website fingerprinting defenses such as Tamaraw [22] and FRONT [31] using the PADDING frame. While they showed that it is possible to deploy website fingerprinting defenses solely from the client, their system is restricted to users in a VPN scenario where IP addresses cannot be used by an adversary to distinguish chaff traffic. While a non-negligible amount of users browse the web through VPNs, this is not the case for the majority [11] who unfortunately will not benefit from such tools.

In this work, we investigate how to build defenses using the PADDING frame for the masses, i.e., outside of the VPN setting. Existing padding-based website-fingerprinting defenses differ on the layer they target and the information they use to inform the defense. There exist application-agnostic network-layer techniques that work independently of the website they are protecting [29, 21, 22, 40, 31]; application-layer-informed network-layer techniques that require prior knowledge (e.g., resources size, resources order, or total size) of the website trace they aim to protect to tailor the defense [54, 56]; and purely application-layer techniques that propose modifications directly on the resources [44, 45, 26]. We provide a comprehensive evaluation of both network- and application-layer defenses, to answer the question: can website-fingerprinting defenses be implemented solely at the network layer 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. Moreover, in contrast to prior work, we emphasize practicality, i.e., we focus on how feasible the widespread deployment of these defenses are, so that normal users can be protected.

We assume a common scenario where websites are hosted behind content delivery networks (CDNs) and use privacy-preserving protocols such as TLS ECH. Users frequently encounter CDN-served resources while browsing – as of November 2022, ≈ 44% of the top million sites use CDNs [4]. In this scenario, the only available information to the adversary is the CDN server’s IP address, and metadata such as the size of encrypted data, its timing, and its direction (sent by/to the server). We answer two questions:

Can we build effective traffic analysis defenses at the network layer using the PADDING frame? We study whether QUIC padding configured at the transport layer as envisioned by the standard can, on its own, protect users from website fingerprinting attacks. We study adversaries with a wide range of capabilities: from powerful adversaries that can observe all communications between clients and servers, and have infinite storage and computation capability; to weaker, more realistic, scenarios in which the adversary can only observe partial traces and is restricted in its storage, computation, or bandwidth [49]. We find that:

✓ Network-layer defenses which build on the QUIC PADDING frame to hide packets’ sizes or inject dummy
Avoiding deployments that could give users a false sense of confidence in the benefits of our research, which can guide current strategies, would put at risk the network users if deployed. We believe that the centralization of web resources on the Internet, in particular, in the hands of Google, creates a favorable setting for the adversary. Traffic analysis on solely the timing of Google resources fetched by a web page achieves a >77% F1 score, requiring four orders of magnitude less data than using full traces. Moreover, the surface of attack is increased from only ASes between the client and the first-party domain host to any AS between the client and Google.

**Can application-layer defenses effectively thwart traffic analysis?** We reveal that network-layer defenses based on padding QUIC traffic are ineffective because they cannot efficiently hide the most important feature for the adversary: the total size of a website. This is because this information is not known at the network layer. We explore whether this information can be effectively collected at the application layer, and whether it can be used to inform padding algorithms in an effective manner. We find that:

- Websites contain a vast amount of resources. These resources are served, in a significant number of cases, by third parties (33% in our dataset). We demonstrate that if all parties do not participate in defending against WF, an adversary can successfully identify pages (>91% F1-score), leaving users vulnerable.

- We identify current web development practices that hinder the deployment of effective website fingerprinting defenses if users are not behind a VPN. We show that, unless those practices do not change, protecting users is close to impossible as it would require significant bandwidth overhead and coordination among parties under very dynamic conditions. We provide recommendations to guide future efforts and pave the way for the existence of effective defenses against website fingerprinting attacks.

**Ethical considerations:** We conduct traffic-analysis attacks against a deployed technology (QUIC). We do not perform any collection or analysis of real users’ traffic. We only collect our own traffic, generated by an automated browser. We uncover vulnerabilities in the proposed defenses, which would put at risk network users, if deployed. We believe that the benefits of our research, which can guide current and future standardization efforts, 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 [11]. QUIC was standardized in May 2021 and is currently being developed by the IETF. QUIC is the transport protocol for HTTP/3. Adoption of QUIC and HTTP/3 has been rising (as of November 2022, they are used by 25.5% of the top 10 million websites [16]). Of particular relevance for our work is the QUIC Padding Frame. The IETF QUIC draft describes it as a frame with no semantic value, that can be used to increase packets size and to provide protection against traffic analysis [12].

**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 that rely on manual features are Wang et al.’s k-Nearest Neighbors (k-NN) classifier based on 4226 manually-selected features [66]. Panchenko et al.’s Support Vector Machines (SVMs)-based classifier, based on cumulative sums of packet lengths [52]; and Hayes and Danezis’ 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. [52] use deep learning neural networks (DNNs) to produce attacks that perform as well as manual approaches. Sirinam et al. [69] 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. [62] 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. [29] 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 [40] and FRONT [31] 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 \cite{60,20,56}.

Other defenses employ adversarial perturbations to cause deep-learning-based classifiers to misclassify traces. These defenses incur lower overhead than prior work. Mockingbird \cite{54} applies perturbations to convert traces into a target trace. It converts traces into bursts (where a burst is a set of contiguous packets in one direction), and perturbs these bursts 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. \cite{48} tackle this issue with Blind, a defense that pre-computes blind perturbations that can be applied to live network traffic. Shan et al. \cite{56} show that \cite{48} 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 \cite{54} 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 \cite{48} 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 they can configure this technique, nor are there details about how to configure it for non-Tor traffic. Dolos \cite{56} 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 3). Due to the challenges associated with adapting these Tor-tailored systems to our QUIC scenario, in our work, we compare to FRONT \cite{31}, which Smith et al. show it can be implemented for QUIC \cite{61}.

Finally, there are defenses at the application layer. Luo et al. \cite{33} propose HTTP Obfuscation (HHTPOS), a client-side defense that modifies features on the TCP and HTTP layers and uses HTTP pipelining to obfuscate HTTP outgoing requests. Randomized Pipelining \cite{45} improves this defense by randomizing the order of the HTTP requests queued in the pipeline. Subsequent works have shown HHTPOS and Randomized Pipelining to be ineffective against traffic analysis attacks \cite{23,66}. Cherubin et al. \cite{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).
Vantage Points. Following prior work \cite{30, 47, 39}, we consider each AS on the path of the client traffic as a realistic adversary. Each AS’ middlebox, router, or switch that routes traffic from a client is a potential vantage point for the adversary to collect this client’s traffic. In Figure 1, we depict a client located in AS $X$ accessing two webpages hosted on an IP in AS $W$. The client traffic is represented by red lines. If the adversary controls AS $X$, they observe all the traffic related to the page visits. This is the adversary typically considered in the website fingerprinting literature \cite{25, 36, 35, 46, 66, 52, 33, 55, 60, 20, 62}. 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) \cite{47, 39}.

3.1. Website fingerprinting

As in previous work, we implement website fingerprinting attacks as a supervised learning problem. The adversary identifies the IP that contains the domains that they want to target. Then, the adversary enumerates all 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 website 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 \cite{20} to validate our results against the state of the art \cite{62}. We use the set of features proposed by Hayes et al. \cite{53} for performing website fingerprinting on Tor since this is a comprehensive set of features from previous related works. To adapt them to the QUIC case, we add features about packet size frequencies. Since QUIC’s maximum payload size (1400 B) is smaller than that of TLS (16 kB), we compute frequencies of packet sizes up to 16 kB to encompass both QUIC and TLS traffic.

3.2. Adversarial Capabilities and Goals

We assume applications and websites hosted behind content delivery networks 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.

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 \cite{35, 33, 55, 60, 20}. 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 \cite{38} 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. \cite{53} (see Figure 1 in Appendix). We find that 60% of these domains 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 (8.3% 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 hard scenario among the 91.5% of the IPs served by the CDN provider. 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 \cite{54}). We also run experiments with other clusters of similar size and obtain comparable results.

| 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 |
To address this problem, following the example of Smith et al. [62], we crawl Alexa 1M [8], Umbrella 1M [5], and Majestic 1M [14]. We perform HAR captures and we identify the protocols used by those websites. Unlike [62], 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 in 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 in the sense that users visit more than the landing pages of domains. Thus, only training on landing pages would not work well 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 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 Chrome 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. [62], 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 [49]. Thus, we also study a constrained adversary that only transmits summaries of locally computed statistics from sampled data [63, 24].

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.

**4. Network-layer PADDING-based defenses**

In this section, we study whether PADDING-based network-layer defenses 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 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%. Since the performance of random forest is better and they give 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 [62], which considered a VPN scenario, in our threat model, the adversary can observe...
the IP addresses of the communication end-points. Hoang et al. [37] 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 [37]. We perform repeated DNS resolutions of the resources loaded by each IP fingerprinting process described in [37]. 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 [37] to find the best match between our LARGE traces and the fingerprints. 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.

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 for 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 for the 68% remaining. 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 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, in the remaining sections, we will only use website fingerprinting as we are studying the effect of website fingerprinting defenses.

**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. Table 3 shows the results.

| 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) |

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

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) |

**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) |

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 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 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 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 [37], we see that when the setup changes, performance drops, indicating that an adversary would need a classifier tailored to the client setup. A reason may be that Firefox traces are generally longer, possibly due to activities such as contacting OCSP servers.
Table 5: Influence of client. F1-score when training on the row dataset and testing on the column dataset.

|               | 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) |

Figure 2: Feature importance for MAIN.

4.1.3. PADDING-based network defenses. To study the effectiveness of defenses based on the QUIC PADDING frame, we focus on the worst case: 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 7 reports the results against difference defenses (first row are undefended traces).

Hiding local features. The top predictive features for MAIN are all size-based (Figure 2). 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 reduces the adversary’s performance.

Hiding global features. To hide 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 [61], which 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 $W_{\text{min}}$ and $W_{\text{max}}$, which indicate the time window within which the dummies must be sent.

Table 6: F1-score and bandwidth overhead when injecting dummies using FRONT. We vary the FRONT parameters $N_s$, $N_c$, and $W_{\text{max}}$.

| Variant F1-Score/Overhead | $N_c = N_s$ |
|---------------------------|------------|
|                           | 325        | 650        | 1300       |
|                           | 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 |

Table 7: 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| 92.2     | 0.5       |

We inject dummies using the FRONT parameters suggested by Smith et al. [61]: $N_s = N_c = 1300$, $W_{\text{min}} = 0.2s$, $W_{\text{max}} = 3s$. We calculate the bandwidth overhead using the same definitions as in [61], 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 6), 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. [61] since their evaluation is in an open-world VPN scenario.)

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

Figure 3: Feature importance when hiding global features (last row of Table 7).
Concretely, we set `traIXroute` to record the ASes in the path taken by the resource requests. page loads. Then, as in prior work [41], we use traceroutes using HAR captures to identify resources requested during We determine which parts of the traffic an AS would see late an AS adversary with partial view of the client’s traffic. limited visibility on the adversary’s performance, we simu-

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 [41], we use traceroutes to record the ASes in the path taken by the resource requests. Concretely, we set `traIXroute` [50] to use `scamper` (configured with the Paris traceroute technique). As Juen et al. [41], 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 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 visits (including its subresources). There are three ASes that observe traffic from more than 80% of webpages: one from Google, one from Cloudflare, and the AS where our client is located. The majority of the ASes, however, see only a small proportion of the sites (less than 10%). If these ASes were to be the adversary, they would not be able to fingerprint traffic from most websites hosted by the IP 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 of pages. The Google AS is the second highest, seeing \( \approx 70\% \) 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 network-layer 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 [58]. Common tools for network sampling include NetFlow and sFlow [47]. More efficient techniques have been proposed in academic papers.
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.

(e.g., sketching [42, 43] or skampling [64]), 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 then we create NetFlow summaries from the PCAPs using nfpcapd and nfdump. We experiment with various sampling rates: 100%, 10%, 1% or 0.1% (common sampling rates 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. 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 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, introducing prohibitive cost (≈ 39 MB per trace in median, see Table 10). 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 now 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]. While studying the traces, we also observe that the order in which these resources are loaded is 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 behaviour 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 [7]. 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
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       |

Figure 7: Feature importance for classifying websites based on the timings of their requests to Google services.

\[ \approx 61 \text{ B per connection}, \] 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.

4.3. Take-aways

Our experiments confirm that website fingerprinting is a threat for QUIC traffic [62], both when users browse landing pages and when they visit 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. We show that the threat persists even when the adversary has limited visibility or storage capabilities. Interestingly, the fact that resources are centralized in a few CDNs can enable other entities to perform fingerprinting at a fraction of the usual cost: e.g., an ISP can use the Google filter to save bandwidth by 3 to 4 orders of magnitude (compared to running the attack on all traffic).

We find that network-layer PADDING-based defenses are not very effective at thwarting website fingerprinting attacks. Despite their high cost, they can only reduce an unconstrained adversary’s success by, at most, 3.5 percentage points without dummies. Dummy injection increases protection, at a prohibitive cost: a 2.31 overhead to achieve a 70% performance reduction. Network-layer PADDING-based defenses can help against bandwidth-constrained adversaries, but the bulk of the privacy gain stems from the adversary’s need to sample rather than from the padding.

The failure of network-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 such as total transmitted size or the total number of packets that will be sent. The defenses we study cannot hide these differences because, at the network layer, they lack information on these global statistics.

5. Designing application-aware PADDING-based defenses

The results in the previous section confirm that the findings of Dyer et al. [29] for HTTP over encrypted tunnels hold for QUIC traffic: network-layer defenses have limited ability to hide global traffic patterns. In this section, we explore whether application-aware defenses can be more effective against website fingerprinting.

Application-aware defenses. We consider any defense that requires knowledge of a page’s resources to be an application-aware defense. This can be knowledge of the number of resources, their size, and their order; or knowledge about global information such as the total (incoming or outgoing) size. The only way to obtain this information is to extract it from the application layer, as Smith et al. [61] do to implement FRONT [31] and Tamaraw [22] in practice. However, while their work demonstrates that QUIC being implemented in the user space makes it possible to implement existing network-layer defenses using a user-space library, they did not explore whether application-layer information could be used to tailor how dummies are injected into a trace. In contrast to Smith et al. [61], we aim to understand whether using application-layer information defenses fare better than their uninformed network-layer counterparts, and what would it take to deploy such defenses.

In this section, we analyze the structure of websites to gain understanding about what information can be extracted and used to inform defenses. Second, we study different ways in which this information can be used to build defenses, and evaluate how these defenses perform.

5.1. Understanding web page composition

We study different dimensions related to the structure and composition of websites that are relevant for configuring PADDING-based defenses. We collect page structures by crawling the pages in MAIN with OpenWPM (v0.17.0) [10] using Firefox five consecutive times. OpenWPM logs the HTTP requests that occur during the page load. Unlike HAR captures, OpenWPM also records the originator of a request.

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

Out of the 150 websites in 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 8a. 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. We note, however, that our measurements are taken over a short period of time and...
dynamism could become more prevalent when web pages are observed over a longer time period.

Resource ownership. Resource owners are the ones that can provide information about resources and modify them. Understanding ownership is important to get an idea of how much coordination among owners 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 do not have relevant ownership information for almost half of our dataset.

Figure 8b 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 a large number of domains commonly associated with Google. We map the domains to their owning entities to measure Google’s prevalence (contains mappings for Google’s domains). Figure 8c 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 can perform the attack by filtering resources from different origins (e.g., only on Google resources as in Sec 4.2.3). We assess whether third parties need to be involved or if only protecting first-party resources suffices.

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. 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.

5.2.2. Evaluating application-aware defense strategies. Application-aware defenses can be implemented both at the application layer or at the network layer (if information is passed to the middleware implementing padding). We directly evaluate perturbed application-layer traces as this gives an upper bound on the performance of a network-layer adversary with respect to a set of features. The reason is that network-layer features are effectively a noisy version of application layer resources (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.

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

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

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 not among the most important features (in Figure 15 in Appendix C). In fact, 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 bytesIncoming (bytesTotal is the sum of the two). This is corroborated by the findings by Hentgen, who shows that even without timings evaluating at the application layer yields an upper bound over the network layer. In the remaining experiments, we discard timings.

5.2.3. Defense strategies. We now evaluate possible strategies that use application-layer information to decide how to configure padding frames.

Protecting local features with padding. We design a padding function padResources to hide individual queries and resource 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 \text{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 9. 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 on packets of 5.58 kB, 21 kB, 3.6 MB, decreases the accuracy adversary by 6% and incurs a median overhead of 9 kB per resource. This ineffectiveness stems from the fact that the adversary still has access to the number of requests and overall volume (see Figure 10 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 packet sizes cannot protect the overall transmitted volume. We design a padding function $\text{pad}_{\text{total size}}$ 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. [61].

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, \text{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 10. As in the network 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 [40]. Unlike in Tor, care must be taken that dummies’ sizes do not enable the adver-
sary to filter them. Considering ideal circumstances in which the adversary cannot filter dummies, 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 11 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 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. Take-aways.

Padding resources at the application layer incurs large overhead and does not necessarily eliminate website fingerprinting attacks. Dummy injection, on the other hand, shows better trade-offs. On the downside, contrary to pure network-layer defenses, when information is protected at the application-layer, there is a need for coordination between first and third parties. Moreover, 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. Thus, the deployment of these defenses may require the collaboration of web pages and web browsers developers besides the coordination of parties that serve web content.

6. Discussion and Recommendations

We carried out a comprehensive study of the ecosystem in which defenses are to be deployed, considering the majority user that does not use Tor/VPNs. We provided evidence of fundamental incompatibilities between today’s Web practices and the deployment of effective defenses.

Network-layer challenges. First, we confirm that network-layer defenses are not effective against website fingerprinting [29], and 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 [33, 51]. Hiding the total size is hard at the network layer, where the size of objects is not known in advance. Thus, without coordination with the application layer, using the QUIC’s PADDING frame is unlikely to result on an effective defense against traffic-analysis attacks. Effective mitigations require application involvement, either in the application code or as part of the browser’s functionalities [27].

App-layer challenges & Next steps. While defenses at the application layer can obtain better protection at a smaller cost than network-layer defenses, our investigation shows that the current Web development practices hinder the effective deployment of any defense. Websites use resources hosted on different servers, and defenses must cover all resources to be effective (see subsection 5.2). The widespread use of third party resources means that 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. In this spirit, initiatives such as Web Bundles [17] (though raising other privacy concerns) would remove the need for coordinating between third parties, simplifying the implementation of defenses, and removing vantage points.

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

To address the easy linking of packets, clients could use techniques such as MIMIQ [62] to leverage QUIC’s connection migration feature to change their IP address; or Near-Path NAT [9] or MASQUE [8] to completely hide their IPs; or CoMPS [65] 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.

To address the ease of separating traffic, CDNs that also host websites could proxy the traffic to third parties, such
that all traffic is served from a single IP. Finally, the closed-world size could be increased by co-hosting multiple websites on one server, or making a larger number of websites available behind load balancers; or even moved to open world if all web traffic would be downloaded via anonymous communication networks (e.g., Tor [28]) or VPNs.

References

[1] “3rd Annual VPN Market Report,” https://www.security.org/resources/vpn-consumer-report-annual/, accessed: 2022-11-29.
[2] “Alexa 1M,” http://s3.amazonaws.com/alexa-static/top-1m.csv.zip, accessed: 2021-07-05.
[3] “BLANKET,” https://github.com/SPIN-UMass/BLANKET, accessed: 2022-05-25.
[4] “CDN Usage Distribution in the Top 1 Million Sites,” https://trends.builtwith.com/cdn, accessed: 2022-12-02.
[5] “Cisco Umbrella Top 1M Domains List,” https://www.trisul.org/devzone/doku.php/cisco_umbrella_top_1m_domains_list, accessed: 2022-10-08.
[6] “DNS Queries over HTTPS (DoH),” https://datatracker.ietf.org/doc/html/rfc8484, accessed: 2021-07-05.
[7] “DuckDuckGo Tracker Radar,” https://github.com/duckduckgo/tracker-radar, accessed: 2022-07-05.
[8] “Multiplexed Application Substrate over QUIC Encryption (MASQUE),” https://datatracker.ietf.org/wg/masque/about, accessed: 2021-10-08.
[9] “Near-path NAT for IP Privacy,” https://github.com/belassey/ip-blindness/blob/master/near_path_nat.md, accessed: 2022-10-08.
[10] “OpenWPM: a Web Privacy Measurement Framework,” https://github.com/mozilla/OpenWPM, accessed: 2021-10-04.
[11] “RFC 9000,” https://datatracker.ietf.org/doc/html/rfc9000, accessed: 2021-08-12.
[12] “RFC 9000, Section 19.1 PADDING Frames,” https://datatracker.ietf.org/doc/html/rfc9000#section-19.1, accessed: 2021-08-12.
[13] “Specification for DNS over Transport Layer Security (TLS),” https://datatracker.ietf.org/doc/html/rfc7858, accessed: 2021-07-05.
[14] “The Majestic Million,” https://majestic.com/reports/majestic-million, accessed: 2021-07-05.
[15] “TLS Encrypted Client Hello,” https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-12, accessed: 2021-07-05.
[16] “Usage statistics of HTTP/3 for websites,” https://w3technics.com/technologies/details/ce-http3, accessed: 2022-05-27.
[17] “Web Bundles,” https://wicg.github.io/webpackage/draft-yasskin-wpack-bundled-exchanges.html, accessed: 2021-10-08.
[18] J. Bai, M. Zhang, G. Li, C. Liu, M. Xu, and H. Hu, “FastFE: Accelerating ml-based traffic analysis with programmable switches,” in Proceedings of the Workshop on Secure Programmable Network Infrastructure, 2020, pp. 1–7.
[19] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, P. M. Ramos, and A. Madeira, “FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications,” in Proceedings of the 28th Network and Distributed System Security Symposium (San Diego, CA, USA, 2021.
[20] S. Bhat, D. Lu, A. Kwon, and S. Devadas, “Var-CNN: A data-efficient website fingerprinting attack based on deep learning,” PETS, 2019.
[21] X. Cai, R. Nithyanand, and R. Johnson, “CS-BuFLO: A congestion sensitive website fingerprinting defense,” in Proceedings of the 13th Workshop on Privacy in the Electronic Society, 2014, pp. 121–130.
[22] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, “A systematic approach to developing and evaluating website fingerprinting defenses,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 227–238.
[23] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, “Touching from a distance: Website fingerprinting attacks and defenses,” in Proceedings of the 2012 ACM conference on Computer and communications security, 2012, pp. 605–616.
[24] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta, “Analysis of the impact of sampling on NetFlow traffic classification,” Computer Networks, vol. 55, no. 5, pp. 1083–1099, 2011.
[25] H. Cheng and R. Avnur, “Traffic analysis of SSL encrypted web browsing,” Project paper, University of Berkeley, 1998.
[26] G. Cherubin, J. Hayes, and M. Juárez, “Website Fingerprinting Defenses at the Application Layer,” Proc. Priv. Enhancing Technol., vol. 2017, no. 2, pp. 186–203, 2017.
[27] A. Davidson, M. Frei, M. Gartner, H. Haddadi, A. Perrig, J. S. Nieto, P. Winter, and F. Wirz, “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, 2022, pp. 205–212.
[28] R. Dingleline, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” Naval Research Lab Washington DC, Tech. Rep., 2004.
[29] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, “Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail,” in 2012 IEEE symposium on security and privacy. IEEE, 2012, pp. 332–346.
[30] N. Feamster and R. Dingledine, “Location diversity in evaluating website fingerprinting defenses,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 717–728.
[31] J. Gong and T. Wang, “Zero-delay lightweight defenses against website fingerprinting,” in 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 717–734.
[32] Y. Govil, L. Wang, and J. Rexford, “MIMIQ: Masking IPs with Migration in QUIC,” in 10th USENIX
[33] J. Hayes and G. Danezis, “k-fingerprinting: A Robust Scalable Website Fingerprinting Technique,” in 25th USENIX Security Symposium (USENIX Security 16). Austin, TX: USENIX Association, Aug. 2016, pp. 1187–1203.

[34] E. Hentgen, “Measuring the Security of Website Fingerprinting Defenses,” https://infoscience.epfl.ch/recor d/289258?&ln=en, 2019, accessed: 2021-10-08.

[35] D. Herrmann, R. Wendolsky, and H. Federrath, “Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier,” in Proceedings of the 2009 ACM workshop on Cloud computing security, 2009, pp. 31–42.

[36] A. Hintz, “Fingerprinting websites using traffic analysis,” in International workshop on privacy enhancing technologies. Springer, 2002, pp. 171–178.

[37] N. P. Hoang, A. A. Niaki, P. Gill, and M. Polychronakis, “Domain Name Encryption Is Not Enough: Privacy Leakage via IP-based Website Fingerprinting,” in Proceedings of the 21st Privacy Enhancing Technologies (PoPETs ‘21, 2021).

[38] L. Izhikevich, G. Akiwate, B. Berger, S. Drakontaidis, A. Ascheman, P. Pearce, D. Adrian, and Z. Durumeric, “Zdns: a fast dns toolkit for internet measurement,” in Proceedings of the 22nd ACM Internet Measurement Conference, 2022, pp. 33–43.

[39] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, “Users get routed: Traffic correlation on Tor by realistic adversaries,” in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 337–348.

[40] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, “Toward an efficient website fingerprinting defense,” in European Symposium on Research in Computer Security. Springer, 2016, pp. 27–46.

[41] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar, “Defending Tor from Network Adversaries: A Case Study of Network Path Prediction,” Proceedings on Privacy Enhancing Technologies, vol. 2015, no. 2, pp. 171–187, 2015.

[42] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, “Sketch-based change detection: Methods, evaluation, and applications,” in Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, 2003, pp. 234–247.

[43] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, “One sketch to rule them all: Rethinking network flow monitoring with univmon,” in Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 101–114.

[44] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, R. Perdisci et al., “HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows.” in NDSS, 2011.

[45] Mike Perry, “Experimental Defense for Website Traffic Fingerprinting,” https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, accessed: 2021-10-03.

[46] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, “I know why you went to the clinic: Risks and realization of https traffic analysis,” in International Symposium on Privacy Enhancing Technologies Symposium. Springer, 2014, pp. 143–163.

[47] S. J. Murdoch and P. Zielinski, “Sampled traffic analysis by internet-exchange-level adversaries,” in International workshop on privacy enhancing technologies. Springer, 2007, pp. 167–183.

[48] M. Nasr, A. Bahramali, and A. Houmansadr, “Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations,” in 30th USENIX Security Symposium (USENIX Security 21), 2021.

[49] M. Nasr, A. Houmansadr, and A. Mazumdar, “Compressive traffic analysis: A new paradigm for scalable traffic analysis,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 2053–2069.

[50] G. Nomikos and X. Dimitropoulos, “traXroute: Detecting IXPs in traceroute paths,” in International Conference on Passive and Active Network Measurement. Springer, 2016, pp. 346–358.

[51] R. Overdorff, M. Juarez, G. Acar, R. Greenstadt, and C. Diaz, “How unique is your. onion? an analysis of the fingerprintability of tor onion services,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 2021–2036.

[52] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, “Website Fingerprinting at Internet Scale.” in NDSS, 2016.

[53] S. Patil and N. Borisov, “What can you learn from an IP?” in Proceedings of the Applied Networking Research Workshop, 2019, pp. 45–51.

[54] M. S. Rahman, M. Imani, N. Mathews, and M. Wright, “Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces,” IEEE Transactions on Information Forensics and Security, vol. 16, pp. 1594–1609, 2020.

[55] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, “Automated Website Fingerprinting through Deep Learning,” in NDSS, 2018.

[56] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, “A Real-time Defense against Website Fingerprinting Attacks,” in ACM Workshop on Artificial Intelligence and Security (AISec’21), 2021.

[57] S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, and C. Troncoso, “Encrypted DNS—Privacy? A traffic analysis perspective,” in NDSS, 2020.

[58] J. M. C. Silva, P. Carvalho, and S. R. Lima, “Inside packet sampling techniques: exploring modularity to enhance network measurements,” International Journal of Communication Systems, vol. 30, no. 6, p. e3135, 2017.

[59] S. Singanamalla, S. Chunhapanya, M. Vavrusa, T. Verma, P. Wu, M. Fayed, K. Heimerl, N. Sullivan,
and C. Wood, “Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS,” PETS, 2021.

[60] P. Sirinam, M. Imani, M. Juarez, and M. Wright, “Deep fingerprinting: Undermining website fingerprinting defenses with deep learning,” in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1928–1943.

[61] J.-P. Smith, L. Dolfi, P. Mittal, and A. Perrig, “{QCSD}: A {QUIC}{Client-Side}{Website-Fingerprinting} defence framework,” in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 771–789.

[62] J.-P. Smith, P. Mittal, and A. Perrig, “Website Fingerprinting in the Age of QUIC.” PETS, vol. 2021, no. 2, pp. 48–69, 2021.

[63] D. Tammaro, S. Valenti, D. Rossi, and A. Pescapé, “Exploiting packet-sampling measurements for traffic characterization and classification,” International Journal of Network Management, vol. 22, no. 6, pp. 451–476, 2012.

[64] P. Tune and D. Veitch, “OFSS: Skampling for the flow size distribution,” in Proceedings of the 2014 Conference on Internet Measurement Conference, 2014, pp. 235–240.

[65] M. Wang, A. Kulshrestha, L. Wang, and P. Mittal, “Leveraging strategic connection migration-powered traffic splitting for privacy,” Proceedings on Privacy Enhancing Technologies, vol. 1, p. 18, 2022.

[66] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, “Effective attacks and provable defenses for website fingerprinting,” in 23rd USENIX Security Symposium (USENIX Security 14), 2014, pp. 143–157.

Appendix

1. 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 12 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.

2. IP anonymity sets

Figure 13 shows the distribution of the cluster sizes of 1.3M domains. Figure 13 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.

3. Additional feature importance graphics

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 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 14](image1.png)

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](image2.png)

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

![Figure 16](image3.png)

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