Client-Side Mitigation of Remote Latency Side-Channel Attacks

Stefan Gast[1][0009−0007−1229−3962]✉, Simone Franza[1][0009−0008−8150−7749], Martin Heckel[2][0009−0006−9739−0587], Jonas Juffinger[3][0009−0002−0569−1704], Daniel Gruss[1][0000−0002−7977−3246], and Johanna Ullrich[4][0000−0003−0297−9614]

[1] Graz University of Technology, Graz, Austria
[2] Hof University of Applied Sciences, Hof, Germany
[3] Liebherr-Transportation Systems GmbH, Korneuburg, Austria*
[4] Interdisciplinary Transformation University, Linz, Austria**

Abstract. Remote latency side channels reveal sensitive information by only observing latency variations in the attacker's traffic with the victim. While these attacks are easy to mount and scale, there is no practical defense. Considering the more powerful attacker-in-the-middle scenarios, available mitigations require the cooperation of all communication partners for protection, and cause prohibitive overheads.
In this paper, we propose AckwardDelay, a new *unilateral, purely client-side, and lightweight* defense against remote latency side channels. In detail, we piece-wise apply constant-time principles on the latency side channel and computationally show that a fully remote attacker requires more than 250 samples to reduce the initial search space to below 1 % with our recommended parameters, even in a scenario favoring the attacker. Based on our proof-of-concept implementation with less than 1000 lines of code on Linux, we demonstrate that the accuracy of website- and video-fingerprinting attacks is reduced to random guessing in practice. With an increase of only 5.55 % on website-loading times and a reduction of only 0.51 % in transfer rates, we conclude that AckwardDelay is a practical, lightweight, and effective mitigation applicable to the vast number of client systems such as smart phones, tablets, and laptops.

1 Introduction

Network side channels leak sensitive information using packet timings, sizes, numbers and other information to any attacker able to sniff network traffic [82, 46, 63, 21, 75, 18, 29, 67, 44, 48]. Raising significant privacy concerns, these side-channel attacks reveal which video [74, 22] or website [28, 89, 78, 4, 56, 8, 81, 75, 22] is accessed, the content of voice calls [92], or even medical and financial secrets [16]. Most of these works focus on network packet analysis within the victim's network or attacker-controlled components on the path between victim

* Work was done while affiliated with Graz University of Technology, Austria
** Part of the work was done while affiliated with the University of Vienna, Austria

and communication partner (attacker in the middle). Similarly, most mitigations [64, 52, 50, 93, 19, 12, 61, 89, 91, 41, 51, 25, 55, 73, 11, 32] focus on scenarios with such powerful attackers, and require cooperating communication partners to apply the defense on both sides. In these cases, leakage can be reduced by padding packet sizes, varying the timings or the number of packets, aggregating packets, adding dummy packets, or splitting traffic over multiple network paths. These approaches are, however, not applicable to a scenario where only one side of the communication is under control of the user, aiming to protect their privacy, while the other side lacks incentives to implement defenses e.g., due to reduced performance or increased costs.

There has been less focus on fully remote latency side channels, where an attacker only observes latencies of their own traffic [57, 22], either in Tor [57] or regular internet connections [22], but does not sniff the victim's whole network traffic. Fully remote latency side-channel attacks are easy to mount and scale, as they do not require a privileged position for packet sniffing nor code running on the victim system. These attacks hide latency measurements within benign-looking traffic between the victim and an attacker-controlled server, e.g., a benign-but-slow resource on a website. Given the significant overhead of existing mitigations and the impracticality of having to control both communication ends for many mitigations, there are currently no mitigations against fully remote latency side channels and they are exploitable on most systems today.

In this paper, we propose AckwardDelay, a unilateral client-side mitigation with low performance overhead, specifically protecting against fully remote latency side channels. The core idea is that a side channel is mitigated by making the attacker-observed latencies statistically independent of the actual signal containing private information like the accessed website or video. Constant-time and constant-decrease padding, as known from cryptography, is however impractical for networking due to prohibitively high overheads, potentially even breaking applications, and lacking ground-truth knowledge on the connection between the attacker and the victim. Instead, AckwardDelay applies piece-wise constant-decrease on random-sized intervals to break the correlation between the original and the eventually observed latencies by the attacker. AckwardDelay groups responses into configurable and unpredictable time intervals, chosen independently of network activity, and returns all accumulated responses at the end of the interval. Beyond that, each response is sent with an additional random delay to hide the latency minima caused by an interval's last packet. Altogether, AckwardDelay transforms the raw trace, revealing potential private information like the accessed website or video, into a jigsaw pattern as shown in Figure 3.

For the design of AckwardDelay, we borrow from constant-time principles, as their attacker-optimistic/defender-pessimistic mathematical framework allows to compute a robust upper bound on leakage. Based on our computations, the attacker needs more than 150 samples to reduce the initial search space below 5 % with our recommended parameters, even in an unrealistically powerful attack assuming unique and deterministic latency sequences for every website. We implement AckwardDelay for TCP and ICMP in less than 1000 lines of code for

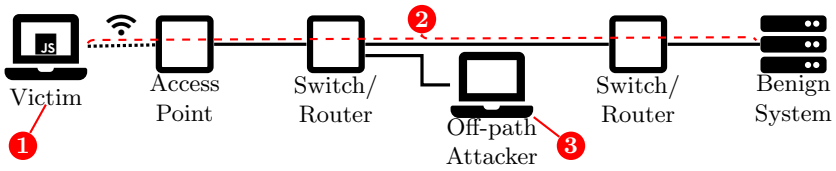


Fig. 1. Requirements for remote side-channel attacks vary from a victim running code in the browser ❶, an attacker-in-the-middle intercepting traffic or a wireless signal on the transport path ❷, to off-path fully remote attackers ❸.

Linux, and evaluate its security and performance in real-world measurements. Already for a conservative choice of parameters, AckwardDelay reduces an attacker's accuracy in website- and video-fingerprinting attacks to random guessing. Thereby, TCP download transfer rates are reduced by only 0.51 %, website loading times increased by 5.55 %.

AckwardDelay is a *unilateral, purely client-side and lightweight mitigation* protecting against fully remote latency side-channel attacks. In contrast to prior work, (a) it neither requires changes to servers, routers, or other components beyond the client, nor the availability of privacy-enhancing tunnels, and (b) impacts the networks transmission's performance only to a very limited extent. Concluding, AckwardDelay represents a unique type of privacy protection for the vast number of client systems, e.g., mobile phones, tablets, or laptops, especially when contrasting the gained protection with the defense's ease of implementation and low performance impact.

Contributions. In summary, our main contributions are:

- We present AckwardDelay, a unilateral, client-side mitigation against fully remote latency side channels, with a Linux proof-of-concept for TCP and ICMP.
- We compute an upper leakage bound and show that an attacker would need more observations than practically feasible, even in unrealistically favorable conditions for the attacker.
- We evaluate AckwardDelay in simulations and real-world measurements, highlighting that attack accuracies drop to random guessing for a conservative choice of parameters.
- Our performance evaluation emphasizes the low overhead of AckwardDelay. Transmission rates decrease by only 0.51 %, and website loading times increase by 5.55 %.

Outline. Section 2 provides background. Section 3 overviews our design. Section 4 provides a security analysis. Section 5 discusses our implementation. Section 6 and Section 7 evaluate security and performance of AckwardDelay. Section 8 discusses related work and limitations. Section 9 concludes.

2 Background

Remote Side-Channel Attacks. Remote side-channel attacks do not require physical access to the victim's device or native code on it, making them a larger security threat than non-remote attacks. Figure 1 gives an overview of attacker models commonly described as remote. One group ❶ runs JavaScript [62, 87, 85] or WebAssembly [72] code in the victim's browser. A second group ❷ observes network traffic from a privileged position, either from a network component on the transport path as an attacker-in-the-middle [34, 70, 5, 82] or by eavesdropping on wireless traffic nearby [15, 49]. In contrast, a third group ❸ neither requires attacker code on the victim system nor a privileged position, *i.e.*, these are *fully remote attacks*. In these, an off-path attacker sends packets to the victim and infers secrets from response content [53, 83, 7, 36] or latencies [27, 22, 71, 84].

Network Side Channels. The root cause of network side channels is due to data transfer over a network. Instead of (potentially remotely) observing a local effect on a victim system, e.g., CPU caching [62], the attacker exploits the standard-conforming data transfer between network components. Network side channels can be *content-based*, *shape-based* or both: In *content-based* side channels, secret information is inferred from individual packet content. For instance, multiple features of TCP packets, such as supported options and receive window sizes, allow an attacker to guess a device's system [53]. In *shape-based* side channels, secret information is inferred from the packet sizes, directions and timings, *i.e.*, the traffic shape. For example, website accesses [70] and video streaming [74] cause characteristic traffic patterns, exploitable in fingerprinting attacks. Inferring secret information from encrypted traffic shapes is known as *traffic analysis* [57]. Some attacks [20, 36] combine both approaches.

Remote Traffic Analysis. While the first traffic analysis attacks measured the traffic shape directly from a privileged, attacker-in-the-middle position [57], subsequent work has shown that the measurement can also be performed indirectly by an off-path attacker. These *remote traffic analysis attacks* rely on congestion of a network segment shared between packets of a benign connection and attacker packets, usually at the last mile of the victim's internet connection. Congestion on this shared segment leads to a bandwidth reduction [74] and increased latencies [27, 22], both exploitable by an off-path attacker. In *fully remote traffic analysis attacks*, an off-path attacker measures latency fluctuations, without any attacker-controlled code on the target system. Earlier work used ICMP Echo (*i.e.*, ping) messages [27, 42], while recent works exploited TCP packet acknowledgments [22] or error responses to connection requests [23].

Network-based Website- and Video-Fingerprinting Attacks. In network-based website- and video-fingerprinting, the attacker directly or indirectly observes a user's (encrypted) traffic and matches it against prerecorded traces. Table 1 shows an overview of prior attacks. Most of them assume strong attackers, e.g., intercepting network traffic or with code execution, resulting in strong mitigations burying the leaky *traffic shape* in noise, with significant overheads [54]. Only a few assume weaker, fully-remote attackers with only indirect traffic shape observations. Without an attacker-in-the-middle and without

Table 1. Website- and Video-Fingerprinting Attacks

Goal	Side Channel	Attacker Position	Victim Features	Attacks
website fingerprinting	traffic shape	❷	privacy-enhancing tunnel	[34, 33, 9, 21, 67, 76, 64, 63, 90, 31, 70, 79, 80, 8, 6, 17, 40, 75]
website fingerprinting	traffic shape	❷	–	[31, 10]
website fingerprinting	SSL/TLS metadata	❷	–	[46]
video fingerprinting	traffic shape	❷	privacy-enhancing tunnel	[66, 44, 88]
video fingerprinting	traffic shape	❷	–	[69, 30, 18, 74, 29, 44, 88]
video fingerprinting	available bandwidth	❶ ❸	run JavaScript	[74]
website fingerprinting	response latencies	❸	respond to unsolicited packets	[27, 42] AckwardDelay-mitigated
video fingerprinting	response latencies	❸	respond to unsolicited packets	[23] AckwardDelay-mitigated
website fingerprinting + video fingerprinting	response latencies	❸	download file from attacker	[22] AckwardDelay-mitigated

website fingerprinting · video fingerprinting · traffic shape · SSL/TLS metadata · available bandwidth · response latencies · ❶ attacker code · ❷ attacker-in-the-middle · ❸ off-path · privacy-enhancing tunnel · run JavaScript · respond to unsolicited packets · download file from attacker · AckwardDelay-mitigated

attacker code on the victim's system, these are stealthy and easy to mount, motivating a lightweight and easily deployable defense tailored towards them.

Website Fingerprinting. Network-based website fingerprinting typically assumes a passive attacker-in-the-middle, able to observe the traffic shape between benign servers and the victim. Most of them assume a privacy-enhancing tunnel, as an attacker-in-the-middle can otherwise trivially observe the unencrypted IP addresses of intercepted traffic. Hintz [34] demonstrated that the sizes of individual asset transfers form distinguishable fingerprints. Further works examined additional features of the traffic shape, e.g., packet sizes [9], the ratio between incoming and outgoing packets, the total number of packets, and more [64, 63, 10]. Korczyński and Duda [46] used unencrypted SSL and TLS metadata. Many works [33, 90, 31, 70, 8, 79, 80, 67, 75, 6, 17, 40, 76] improved classification, with state-of-the-art using neural networks (e.g., CNNs). In contrast, the fully remote website-fingerprinting attacks we mitigate assume an off-path attacker, unable to directly observe traffic shapes or inspect packet content. Instead, the off-path attacker observes response latencies, either from the victim system's responses to unsolicited packets (ICMP echo requests [27, 42]) or from a TCP download [22].

Video Fingerprinting. Video-fingerprinting attacks exploit that the traffic patterns from Dynamic Adaptive Video Streaming over HTTP (DASH) also form fingerprints, allowing an attacker to identify the video streamed [69, 74, 29]. As with website-fingerprinting, most of them also assume an attacker-in-the-middle, observing the traffic shape. While some of these also target privacy-enhancing tunnels [66, 44, 88], multiple attacks have also been demonstrated on regular connections [69, 30, 18, 74, 29, 44, 88]. The unencrypted IP addresses only leak the streaming platform, but not the encrypted video the user is watching. Schuster et al. [74] proposed an off-path attack variant, measuring the available network bandwidth to an off-path attacker server, with JavaScript code on the victim system. Attacks relying on attacker-code on the victim could be prevented

by (selectively) disabling the execution of untrusted code. Again, fully remote video-fingerprinting attacks from off-path attackers measure latencies from responses to unsolicited packets (TCP SYNs [23]) or from a TCP download [22].

3 Design of AckwardDelay

In this section, we define the threat model for AckwardDelay, and detail the design we propose as a new secure default.

Threat Model. We assume the typical fully remote network side-channel threat model [27, 42, 22, 23] ❸ to identify a website/video accessed by the victim: The attacker cannot directly observe the victim's network traffic from a privileged position, but the victim maintains a connection (e.g., TCP) with the attacker, e.g., loading an ad image in a website. The attacker has full knowledge of AckwardDelay and can adapt accordingly (cf. Section 4). Yet, the attacker wants to avoid any user-noticeable effect, *i.e.*, no suspiciously high activity, as the user may intervene otherwise. Attacks occupying the full bandwidth are out of scope. Mitigating such attacks requires intrusive and bilateral traffic shaping, for which many solutions have been proposed (see Section 8), orthogonal to our threat model and defense. Finally, we assume the victim runs no attacker code.

3.1 Challenges for Mitigation

We identify six challenges to mitigate fully remote network side channels: **$\mathcal{C}$1: Any connection is a potential attacker**, since any connection can perform a remote latency measurement through a benign data transfer (e.g., resources, cloud storage, or updates). **$\mathcal{C}$2: Every connection is a potential victim**, since it is unknown which connections leak sensitive information, *i.e.*, the existence of a connection to a specific website may be sensitive already. Modern websites open many connections to download their assets (e.g., JavaScript libraries, fonts, or advertisements), any of which may leak that the victim just visited this specific website. **$\mathcal{C}$3: Mitigation is only feasible on the client side**, since users only control their client system. Connecting parties cannot be trusted as they could one-sidedly not comply and mount an attack without the client's knowledge. Furthermore, web servers cannot predict sudden congestion caused by other traffic, *i.e.*, they cannot prevent the root cause. **$\mathcal{C}$4: The client can only decorrelate side channel (observed latency) and secret (ground truth latency)**, since full statistical independence is not feasible: packets already arrive at the client with a delay depending on the current bandwidth utilization of the last mile, *i.e.*, side-channel information is already introduced before the client receives the packet. As we discuss in Section 4, the client cannot undo the receive delay, it can only add further delays to the response, *i.e.*, increase the attacker-observed latency. **$\mathcal{C}$5: Constant-time latency is infeasible**, since the adaptive attacker chooses when to send packets, *i.e.*, the victim does not know when the attacker sent a packet and how long it traveled, since an adaptive adversary can freely choose when to send packets to exploit any weakness in

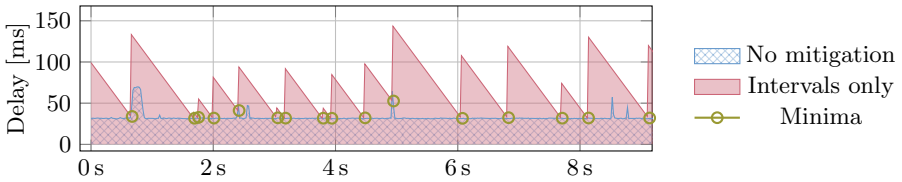


Fig. 2. Simulated traces without mitigation and with intervals only.

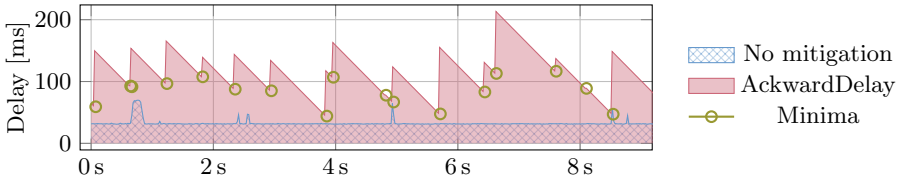


Fig. 3. Traces with and without AckwardDelay simulation.

the mitigation. ***C*6: Applications and network performance must remain largely unaffected**, since the mitigation needs to be a secure default given the ease of mounting such attacks and the current lack of **any** mitigations.

3.2 Basic Design

Based on the above challenges, we build AckwardDelay in **two components** with the goal of decorrelating the timing signal from secrets since a full constant-time solution is infeasible (***C*5**). The **first component** of AckwardDelay is a static response schedule with a fixed response interval length W, significantly larger than the regular latency and randomly sampled from a Gaussian distribution. AckwardDelay pads all packet responses to the next response interval boundary. Figure 2 shows the basic principle of this approach. The secret information ("no mitigation" trace) is hidden as the attacker-observed latency strictly follows the saw-tooth pattern generated by the intervals. The attacker-observed latency is exactly the time between sending the packet and the end of the current interval, effectively decorrelating response latency from actual latency. The attacker cannot predict, but only sees the post-hoc interval lengths. This prevents the attacker from targeting the end of an interval, *i.e.*, curve minima. To acknowledge multiple TCP packets simultaneously, AckwardDelay uses existing TCP mechanisms to send as few packets as possible by skipping packet numbers up to the highest number acknowledged. According to the TCP standard, this also acknowledges all lower packet numbers.

As visible in Figure 2, the minima of the saw-tooth pattern generated by the **first component** still touch the unmitigated trace. Latency peaks that push up (or exceed) the saw-tooth pattern and the minima at the end of the intervals still convey the unmitigated ground-truth signal to the attacker, albeit with a much lower resolution: As all responses are aligned to the next interval

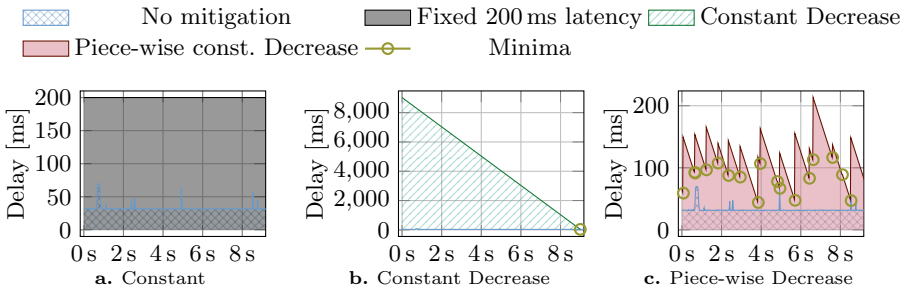


Fig. 4. Under the assumption that no latency spike is above 200 ms, a constant delay mitigation (red) hides all latency variations (blue), *i.e.*, there is zero information leakage (a). Similarly, a constant decrease mitigation (green) also hides all latency variations (b), effectively by grouping all packets to be sent into a single bucket. If the bucket is chosen independent of the secret and large enough to cover all packets, there is again zero information leakage as it effectively is a constant-time mitigation. Both (a) and (b) introduce prohibitively high latency overheads. A piece-wise constant decrease mitigation (red) groups packets into random-sized buckets, achieving a trade-off between latency overhead and information leakage (c).

boundary, the attacker can only obtain one timing observation per interval. This reduces the effective sampling rate to $f_s = 1/\mathbb{E}[W]$ where $\mathbb{E}[W]$ is the expected interval length. Instead of thousands of data points for a single website access, the attacker now has data for e.g., a dozen measurement points only. Even if the adaptive attacker performs more measurements, the number of minima only depends on the interval length. Summarizing, the client controls the number of observed data points per time.

We reduce leakage in these minima with a **second component**: a delay D, drawn randomly from $\mathrm{Unif}\,(0, D_{\max})$, additionally delays each response. As a result, shown in Figure 3, the attacker cannot observe precise latencies at the trace minima anymore. Instead, the attacker-observed latency can be anywhere between 0 and a full interval length W added on top of the actual latency. As a result, the curve's minima do not show correlation with the actual latency in most points. For a few points, e.g., latency peaks, the minima may still reach a point where the attacker-observed latency is the actual latency, but compared to Figure 2, the number of these points is again reduced by orders of magnitude.

We apply the two components of AckwardDelay to every connection to address challenges $\boldsymbol{\mathcal{C}1}$ and $\boldsymbol{\mathcal{C}2}$ as we do not know which connections are attackers and victims. Finally, challenge $\boldsymbol{\mathcal{C}6}$ is addressed by iteratively increasing interval lengths from a small initial length to the configured secure length. With this approach, AckwardDelay does not trigger any quality of service mechanisms.

4 From Constant-Time to a Practical Defense

In this section, we motivate Section 3 by an upper bound on leakage.

Motivation. Timing side channels, particularly in cryptographic algorithms [45], are eliminated by constant-time implementations. Independent of the secret, the algorithm terminates after a maximum time known as worst-case execution time (WCET) [24]. The likewise constant timing of network latencies would eliminate network-based timing side channels, as illustrated in Figure 4a. This naive approach, however, causes not only high latency overheads, but also assumes knowledge of the incoming packet latency to add adequate delay for each outgoing packet. As the client has no control over the sender and the transport path, it cannot reliably know for how long an incoming packet has travelled.

Another constant-time approach are packet delays that decrease by a constant rate of 1. This means that all packets are grouped into a single bucket and sent at connection termination, see Figure 4b. If the last latency is chosen to be higher than the maximum latency, this approach is constant-time as all observed latencies are eventually only a function of the last latency. An attacker observing the mitigated, green latency curve in Figure 4b can only infer information on the last point, but not on any other points. While this second approach requires no knowledge of a target latency, it introduces even higher latency overheads than the first, potentially even to the point that network protocols may time out [39].

General Idea for Networking Mitigation. For a practical trade-off between latencies and leakage, we propose a piece-wise constant decrease mitigation as illustrated in Figure 4c. Packets are grouped into buckets of random sizes, and sent after a delay that decreases at a constant rate. While having the advantage of constant decrease mitigation (no need tor target latency), an attacker can only infer the minima (*i.e.*, the last packet of the bucket), see the red curve in Figure 4c. Effectively, this reduces the attacker's sampling rate to one sample per bucket. To also avoid leakage in the minima, random noise might be introduced into the last packet's delay. The upper bound on leakage for a fully constant-time defense is clearly zero. For a bucket-based defense, the bound depends on the specific parameters and approach chosen. Yet, such a bound can often be mathematically derived [47] as also shown in the following.

Assumptions on Powerful Attacker. To derive an upper bound of leakage, we assume an overly powerful off-path fully remote attacker. We assume that the victim's unmitigated access to websites produces fully deterministic and unique raw latency traces with perfect alignment. In practice, this is unrealistic, as network latencies are subject to transport and server-side variations. Instead of relying on raw and mitigated traces from empirical measurements, we model raw and mitigated latency traces based on ground-truth latency distributions[5]. With the raw latency distribution, we model the set of possible, unmitigated websites with individual packet latencies identically and independently distributed. Similarly, we model the mitigated websites as traces of identically and independently distributed latencies, based on the distribution for mitigated latencies.

[5] Note that the existence of ground-truth distributions is assumed for modeling. For the computation of actual numbers, we later approximate them based on 9,000 measured traces. By the law of large numbers, increasing numbers of measurements would bring us arbitrarily close to the actual distribution.

The Sets of Raw and Mitigated Traces. We can express raw latency traces r as vectors $r = (r_1, \ldots, r_T)$ of length T where each entry r_i represents the latency of packet i. Likewise, the mitigated trace $m = (m_1, \ldots, m_T)$ represents the latencies in presence of a defense. The set of all possible mitigated traces M_r for a raw trace r consists of traces of the form $m_{r,i} = r_i + x_i$ where r_i is the raw latency and x_i represents the mitigation delay at packet i. The defender cannot speed up packets, only delay them up to a chosen maximum delay, *i.e.*, $0 \leq x_i \leq x_{\max} \forall i$. Hence, any mitigated latency $m_{r,i}$ for a raw trace r_i must be within these limits, *i.e.*, $\quad r_i \leq m_{r,i} \leq r_i + x_{\max} \qquad \forall i$.

Consequently, the attacker can infer that:

- Any raw trace r (*i.e.*, any website) that violates this condition for any index i cannot correspond to the observed trace m.
- Any raw trace r that satisfies this condition for all indices i remains feasible from the attacker's perspective.

The set of all raw traces r that remain feasible for an observed mitigated trace m is called the *indistinguishability set*. Following from our assumptions, samples r_i and r_j for $i \neq j$ are treated as uncorrelated, *i.e.*, each point adds the full probability that a trace is eliminated from the indistinguishability set, benefiting the attacker.[6] The mitigation randomness x_i is sampled from a random variable $\mathcal{X}$ independent of the random variable $\mathcal{R}$ representing the unmitigated latencies. The mitigated latencies are consequently sampled from a distribution $\mathcal{M}$.

The *indistinguishability set*, quantifying how many raw traces remain feasible after an observation m, consists of all raw traces that could have produced the observed mitigated latencies m_i: $F(m) = \{r \in \mathcal{R} : r_i \in [m_i - x_{\max}, m_i] \ \forall i\}$.

Fraction of Feasible Traces. We can quantify how strongly a single (fixed) observation m constrains the set of possible raw traces. Since we model raw traces as samples from $\mathcal{R}$, the fraction of feasible traces $f(m)$ is the probability that any trace $R' \sim \mathcal{R}$ is in the indistinguishability set $F(m)$:

$$\begin{aligned} f(m) &= \Pr_{R' \sim \mathcal{R}^T}[R' \in F(m)] \\ &= \Pr_{R' \sim \mathcal{R}^T}\left[R'_i \in [m_i - x_{\max}, m_i] \ \forall i\right] && \text{(by definition of } F(m)) \\ &= \prod_{i=1}^{T} \Pr_{R' \sim \mathcal{R}^T}\left[R'_i \in [m_i - x_{\max}, m_i]\right] && \text{(by independ. of } R'_i) \end{aligned}$$

We can define $q(m_i)$ as the probability that a single coordinate R'_i of a raw trace R' is within the feasible interval at position i:

$$q(m_i) = \Pr_{R' \sim \mathcal{R}^T}\left[R'_i \in [m_i - x_{\max}, m_i]\right] = \sum_{x=m_i-x_{\max}}^{m_i} \Pr[R'_i = x], \qquad \text{(in the discrete case)}$$

with $f(m)$ eventually resulting in $f(m) = \prod_{i=1}^{T} q(m_i)$.

[6] In reality there often are correlations between packet latencies, *i.e.*, not every point adds the full probability of elimination. However, by assuming independence, we obtain an upper bound, strictly higher than assuming correlations.

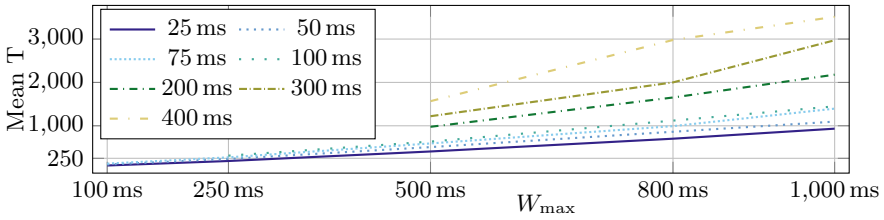


Fig. 5. Average number of samples T necessary to reduce the indistinguishability set to 1 % for different values of $W_{\max}$ and $D_{\max}$ (different lines).

Typical Fraction and Asymptotic Leakage. With $f(m)$ known for a *fixed* mitigated trace, we can analyze the *typical* fraction of feasible traces for *random* mitigated traces m and write the previous equation as $\log f(m) = \sum_{i=1}^{T} \log q(m_i)$.

Via the law of large numbers, which states

$$\frac{1}{T} \log f(m) \xrightarrow[T\to\infty]{\text{a.s.}} \mathbb{E}_{m\sim\mathcal{M}}\big[\log q(m)\big] = \mathbb{E}\big[\log q(m_1)\big],$$

the typical fraction of feasible traces is approximately

$$f_{\text{typ}} = \exp\Big(T \cdot \mathbb{E}\big[\log q(m_1)\big]\Big).$$

Based on this f_{typ}, we can compute the number of samples required so that the attacker can reduce the indistinguishability set to a desired fraction f_{target}:

$$T = \frac{\log f_{\text{target}}}{\mathbb{E}\big[\log q(m_1)\big]}.$$

where f_{target} is the desired fraction of feasible traces, which could assume values like 0.5 (attacker can eliminate half of the possible traces) or 0.01 (attacker can eliminate 99 % of the possible traces).

Figure 5 shows the number of samples T required to eliminate 99 % of the search space for the recommended window size W and uniform random delay D. The extremely powerful attacker of our mode requires more than 250 samples to reduce the indistinguishability set to 1 % of its original size. A realistic attacker requires even more samples as real-world traces are not perfectly aligned and fully deterministic for every website and every request, encouraging the practical implementation of our defense.

5 Implementation

We implement AckwardDelay in a simulation and as a Linux kernel module. In the simulation, we search the best parameter trade-offs between defense strength and impact on the network delay, which would take many weeks using real-world measurements. We then evaluate these parameters with our Linux module.

5.1 Implementation of the Simulation

With the simulation, we can select specific properties that simulate ideal conditions for an attacker, and thereby perform a worst-case estimation, with the goal

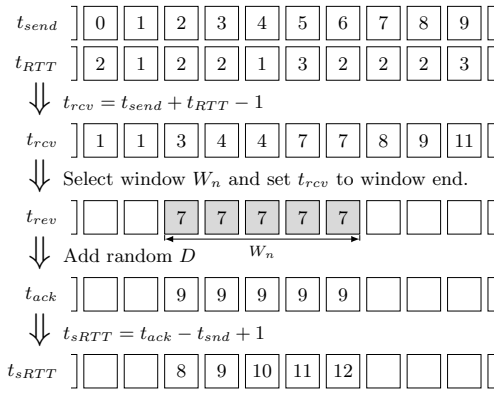


Fig. 6. The steps performed to transform a measured trace into a transformed trace with the mitigation simulated.

to get the fingerprinting accuracy as low as possible with small values for W and D. We simulate the mitigation on *unmitigated* traces as follows (cf. Figure 6:

Preparation. First, we split up the recorded round-trip times into its components. Loading a website causes only very little upload traffic. Therefore, we assume a constant upload time component $t_{up} = 1\,\text{ms}$ in our measured round-trip time. This means that, for every measured round-trip time, the packet was received by the victim at $t_{rcv} = t_{send} + t_{RTT} - t_{up}$.

First Component W. We infer the size of the current interval W_n within 1 and $W_{\max}$ using a Gaussian distribution. We select and group all round-trip times within the current interval. We set all the t_{rcv} to the end of the interval.

Second Component D. We select a random delay D between 0 and $D_{\max}$ for the current interval. The resulting timestamps are exactly when the victim would send out the acknowledgments t_{acksnd}. We ensure that all t_{acksnd} are within the next interval W_{n+1} to guarantee that all ACKs are sent in the correct order.

Final Step. Finally, we simulate sending ACKs by setting the receive time of the ACKs $t_{ackrcv} = t_{acksnd} + t_{up}$. The attacker uses $t_{simRTT} = t_{ackrcv} - t_{snd}$ as the simulated round-trip time.

Limitations. Our mitigation adds a delay to packets before sending acknowledgments for them. On an unmitigated trace as a baseline the simulation only adds delays. Hence, the simulated trace can never have a round-trip time below the original trace. On a real system, the mitigation is applied to all connections, which leads to modifications in the overall traffic sent and received by the system. Therefore, this limitation does not apply to a real system, leading to significantly stronger decorrelation from unmitigated traces than in our simulation.

5.2 Real-World Implementation in Linux

Our qdisc implementation for the Linux 6.8.0 kernel adds two packet queues: Q_{ACK} for ACK and ICMP packets, and Q_{FAST} for all other packets. In line with

the default on unmodified Windows clients [86], we disable the TCP Timestamp option [37] in our proof-of-concept implementation for Linux, preventing leakage of timing information from package content.

Interval Length. AckwardDelay delays the transmission of ICMP and ACK packets, based on the parameters interval length $W_{\max}$ and noise $D_{\max}$. After opening an interval at timestamp t_{open_int}, the qdisc computes the duration of the interval by randomly drawing $W \sim \mathcal{N}\left(\frac{W_{\max}}{2}, \left(\frac{W_{\max}}{3}\right)^2\right)$ and adding it to t_{open_int}. Thus, the end of the interval is scheduled for $t_{close\_int} = t_{open\_int} + W$. Finally, the qdisc computes t_{send_ack} to send all the packets in Q_{ACK} as $t_{send\_ack} = t_{close\_int} + D$, with $D \sim \text{Unif}\,(0, D_{\max})$.

Enqueuing. Packets ready to be sent are assigned to the corresponding queue. ICMP responses and TCP acknowledgment packets without payload (confirming the receipt of data) are enqueued in Q_{ACK}. Packets that are neither ICMP packets nor TCP packets, and packets containing a payload are considered to be non-sensitive information by AckwardDelay and enqueued in Q_{FAST}. In TCP, an acknowledgment of sequence number $N + 1$ means that all bytes up to N have been correctly received [39]. Thus, whenever a new acknowledgment packet is added to Q_{ACK}, all acknowledgment packets with a lower sequence number on the same connection, *i.e.*, matching destination IP and port, are considered obsolete, removed from the queue, and never sent. New packets on the same connection with the same acknowledgment number as another packet in the queue are enqueued for the fast retransmit algorithm [60].

Dequeuing. The qdisc accumulates acknowledgment and ICMP packets into Q_{ACK} over the interval. Once the interval closes, it dequeues all of them at t_{send_ack}. The packets from Q_{FAST} are not tied to the intervals and are sent in a FIFO fashion whenever there are no acknowledgment packets to be sent. Q_{ACK} has a higher priority than Q_{FAST}. Only after Q_{ACK} has been emptied, AckwardDelay resumes sending packets from Q_{FAST}.

Finishing TCP Slow Start. For new connections, TCP assesses the available network bandwidth using the slow start algorithm [60]. If AckwardDelay is applied as soon as a new connection starts, the server increases its send rate slower due to the delayed delivery of the acknowledgment packets. Consequently, the maximum download throughput is reached later, with a more pronounced effect for larger interval lengths $W_{\max}$. Thus, for new TCP connections, AckwardDelay uses an optional dynamic interval whose length gradually increases from 0 ms to $W_{\max}$ ms. The full level of protection is reached within < 400 ms, which is too fast to mount any realistic attacks. As we show in Section 7 (cf. Figure 10), with this technique, AckwardDelay reaches the maximum throughput as quickly as an unmitigated TCP connection.

6 Security Evaluation

In this section, we evaluate AckwardDelay in a simulation and with our real-world implementation. We also show that the leakage previously visible in the latency distribution is buried in the latency distribution with AckwardDelay.

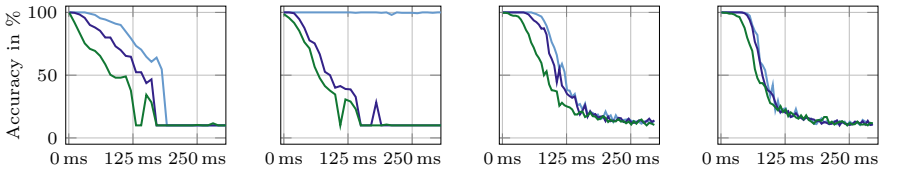


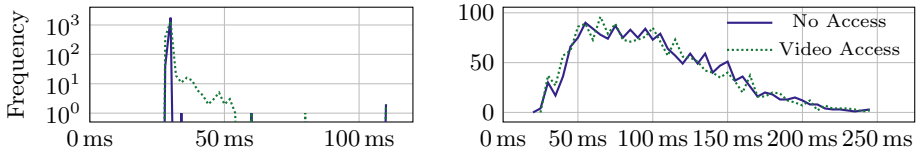
Fig. 7. Website- (a, b) and video-fingerprinting (c, d) maximum accuracy for 10 websites and videos with 150 simulated traces with $D_{\max}$ or $W_{\max}$ varied according to x axis, other value fixed. $D_{\max}$ and $W_{\max}$ should be $\geq$175 ms, while the other can be smaller, e.g., around 25 ms to 50 ms, for the accuracy to drop to random guessing.

Parameter Finding using our Simulation. High parameters result in stronger decorrelation from the secret ground truth but result in lower performance (as shown in Section 7). Hence, we search parameters with a good security-performance trade-off. For this purpose, we generate 150 mitigated (inherently randomized) traces for 10 websites and videos, based on a single unmitigated trace per website or video, for different combinations $W_{\max}$ and $D_{\max}$ (cf. Figure 7). This creates a **best-case scenario for the attacker**: The secret ground truth is expected to always be the same, which is not the case in reality as multiple measurements of the same website or video yield different traces [22]. Thus, any differences between simulated traces for a website or video are only from AckwardDelay. Like Gast et al. [22], we process the traces with an STFT and CNN, and report the maximum accuracy obtained, as shown in Figure 7. Reducing the attack accuracy to 10 % (1 in 10) is equivalent to random guessing.

For both website- and video-fingerprinting, we find values of $W_{\max}$ around 180 ms to be sufficient to drop the accuracy to random guessing. With $D_{\max}$ of 50 ms, the accuracy drops to random guessing slightly earlier. We can also see that for $D_{\max}$, without the interval mechanism, the accuracy essentially does not drop (cf. the blue line in Figure 7b). We use these parameters as guidelines for the following real-world evaluation.

Real-World Security Evaluation. We evaluate the security of our real-world implementation against the TCP-based website- and video-fingerprinting attacks presented by Gast et al. [22]. We record traces with different combinations of the parameters $W_{\max}$ and $D_{\max}$ and train a classifier on them, similar as in the simulation described before.

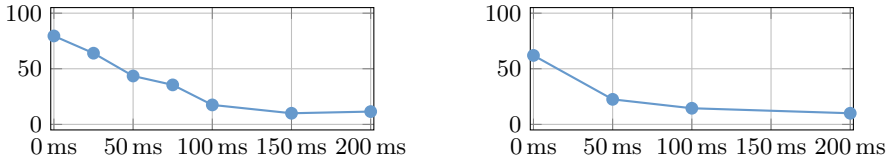
We collect the traces with a client on a 50 Mbit/s ADSL connection, and a remote TCP server. The server offers a large file to download over HTTP, sending TCP packets with a constant rate and measuring the latencies. The client has AckwardDelay enabled with the chosen parameters and starts downloading the file from the TCP server. After 2 seconds, the client opens the website or video to test. For the entire duration of the download, the client sends acknowledgment packets to the server according to the ACK interval mechanism specified above.



a. Latency distribution. Using no mitigation results in significant visual differences in the two distributions. Without video access the distribution has $\mu = 30.063$ and $\sigma = 2.760$, with video access $\mu = 30.084$ and $\sigma = 3.792$.

b. Latency distribution with AckwardDelay. Without video access ($\mu = 100.68$ and $\sigma = 43.277$) and with video access ($\mu = 98.294$ and $\sigma = 43.682$) fully overlap for $W_{\max} = 200\,\text{ms}$ and $D_{\max} = 25\,\text{ms}$.

Fig. 8. Distribution of latencies in the real-world measurement. Without AckwardDelay, the distributions are clearly distinguishable, with AckwardDelay, they fully overlap.



a. Classification accuracy for 10 websites over $W_{\max}$, $D_{\max} = 25\,\text{ms}$. At $W_{\max}$ of 100 ms to 150 ms, the accuracy drops to random guessing.

b. Classification accuracy for 10 videos over $W_{\max}$, $D_{\max} = 25\,\text{ms}$. At $W_{\max}$ of 100 ms to 200 ms, the accuracy drops to random guessing.

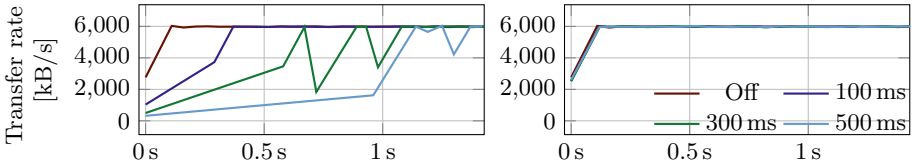
Fig. 9. Maximum accuracy of our model over $W_{\max}$ with 100 traces each. AckwardDelay quickly reduces the accuracy to random guessing in both cases.

Additionally, we investigate the latency distribution with and without AckwardDelay (cf. Figure 8). We observe that the latency without AckwardDelay has distinctive visual features with and without a video played. With AckwardDelay, these features are buried in the overall latency distribution.

To evaluate against website fingerprinting, we use the Alexa top 10 websites [3] with a download duration of 10 s and a server sampling rate of 0.2 ms. We perform 100 measurements per unique parameter set and website. For video fingerprinting, we use 10 random YouTube videos trending in the USA between October 2022 and September 2023 with at least 70 million views. We play the videos in 1080p resolution and measure latencies for 90 s with a server sampling rate of 50 ms and 100 measurements per unique parameter set and video.

Like with the simulation, we apply an STFT and a CNN to our real-world measurements. We observe similar behavior for website and video fingerprinting: The accuracy drops to random guessing, *i.e.*, $F_1 = 10\,\%$, with $W_{\max} >= 200\,\text{ms}$ (cf. Figure 9). Noteworthy, adding more websites would reduce the accuracy further, *i.e.*, for top 100 or top 1000 website fingerprinting it would drop even earlier and reduce to random guessing as well.

Additionally, we evaluated AckwardDelay against ICMP-based attacks [27, 42], see Appendix A. We conclude that with reasonable parameters, AckwardDelay mitigates real-world attacks based TCP ACK or ICMP message timings.



a. Transfer rate without dynamic intervals. **b.** Transfer rate with dynamic interval lengths.

Fig. 10. The download transfer rate with and without AckwardDelay and multiple maximum intervals $W_{\max}$. Without the dynamic interval lengths, the TCP slow-start mechanism has a longer ramp-up time. With dynamic interval lengths in AckwardDelay, there is no significant performance difference to an unmitigated connection.

7 Performance Evaluation of AckwardDelay

We evaluate the performance of our real-world AckwardDelay implementation using the same 50 Mbit/s connection as in Section 6. We evaluate its effects on TCP slow-start with and without dynamic intervals. We then study the performance impact of AckwardDelay observing only little performance loss.

Interaction with TCP Slow-Start. Delaying acknowledgment packets can have unintended side effects on the TCP slow-start mechanism, resulting in lower transfer rates on freshly established connections. We investigate the extent of this effect and show that it can be overcome with dynamically increasing interval lengths for new connections, as described in Section 5.2. For this, we first measure the download transfer rate over time with AckwardDelay enabled, without dynamic interval lengths implemented. As Figure 10 shows, the transfer rate requires significantly more time to reach the maximum. This effect is stronger for larger $W_{\max}$ values. Secondly, we repeat the measurement with dynamic interval lengths. Figure 10 shows that dynamic intervals thwart this performance effect, *i.e.*, no significant overhead to an unmitigated connection. Instead, even with a large $W_{\max}$ up to 500 ms, the fresh connection quickly finishes the slow-start phase within 100 ms, matching the behavior without AckwardDelay.

Finally, we measure the download transfer rate for a 1 GiB file from a remote web host. Without AckwardDelay, we measure a transfer rate of 5.84 MB/s. With AckwardDelay and $W_{\max} = 200\,\text{ms}$, $D_{\max} = 25\,\text{ms}$, we measure a transfer rate of 5.81 MB/s, resulting in a negligible performance loss of only 0.51 %.

Website Loading Speed Evaluation. We evaluate the loading speed of the top 25 websites from the Alexa top 1 million list [3] with and without AckwardDelay, on the same 50 Mbit/s ADSL connection as before. We evaluate 5 different configurations: one without mitigation and 4 configurations with different combinations of $W_{\max}$ and $D_{\max}$ for the mitigation. We repeat the measurements for a total of 100 iterations, automated with Firefox Selenium WebDriver. For each iteration and website we initially load the website once without mitigation and discard the measurements, to avoid influences from e.g., server-side caching. We then load the same website again 5 times, *i.e.*, once per configuration, and store the loading time. For each iteration and website, we randomize the order of the

configurations to avoid bias from order-dependent effects. Furthermore, we clear the browser cache before each website load, avoiding any bias from browser-level caching. To evaluate the performance loss caused by AckwardDelay, we compute the mean loading speed of each website over all iterations for each configuration. Measurements that are further than 2 times the standard deviation from the mean are removed as outliers. Then, we recompute the mean, and determine the percentage difference for each website from the baseline, *i.e.*, without mitigation. Finally, for each configuration, we compute the mean performance loss after removing outliers. We get a performance loss of 0.50 %, 1.26 %, 4.97 %, and 5.55 % for $W_{\max} \in [0\,\text{ms}, 100\,\text{ms}, 150\,\text{ms}, 200\,\text{ms}]$ respectively.

8 Related Work and Discussion

Mitigations against On-Path Attacks. Mitigating on-path attacks is more far-reaching than the mere modification of latencies. Traffic shaping changes the traffic's appearance, including packet lengths, timings, and the number of packets, to evade the attacker's classifiers. Panchenko et al. [64] and Luo et al. [52] use readily available capabilities (download of additional data, protocol features). Relying on a database of pre-captured traffic, the traffic shape is modified in a way to mimic the appearance of other traffic [93, 61, 89, 91, 2, 58, 65]. Zhang et al. [94] proposes adversarial machine learning- and differential privacy-based approaches to cause misclassification. Shen et al. [77] and Gong et al. [26] aim to evade correct classification by using k-anonymity and generative adversarial networks. Most approaches, however, make traffic look uniform and indistinguishable for the attacker. Liberatore and Levine [50] apply static per-packet padding. BuFlo and its extensions [19, 12, 13] send fixed-length packets at fixed intervals, DynaFlow [51] fixed burst patterns with dynamic intervals. To disrupt traffic shapes, Juarez et al. [41] use adaptive padding, Abusnaina et al. [1] the adversarial learning-based Deep Fingerprinting Defender (DFD), Gong and Wang [25] front obfuscation and trace gluing, and Holland and Hopper [35] traffic surging with successive decay. Pacer [55] introduces cloaked tunnels, and NetShaper [73] provides differential privacy-based traffic shaping. Hasselquist et al. [30] transfer front obfuscation [25] and RegulaTor [35] to video fingerprinting scenarios, introducing Scrambler, forming constant-rate bursts with random trail padding. On-path attacks are also mitigated by demultiplexing over multiple paths, e.g., distributing traffic over multiple entry relays or Tor paths[11], or over multiple Internet connections in multi-homed networks [32].

Mitigation Position. In Table 2, there are two approaches [43, 55] against remote traffic analysis by off-path attackers, one of which is specific to cloud computing [55]. As an attacker-in-the-middle scenario has stronger attacker capabilities, traffic shaping is also a viable mitigation, albeit with high overheads [54]. Conversely, multipath routing removes an attacker's sight on the traffic, but does not mitigate off-path attackers.

Unilateral, client-side mitigations, fully under user control, are preferable over solutions involving other parties. Multiple mitigations in this space [64,

52, 94], however, offer only limited protection (or again high overhead) [19, 14, 30]. The solution of Kadloor et al. [43] controls the last-hop router, which may not always be the case. Many non-unilateral approaches target privacy-enhancing tunnels like Tor. These mitigate remote traffic analysis, but their limited resources do not allow routing of all Internet traffic. To summarize, remote traffic analysis does not require an attacker at a privileged position and is thus more practical. However, no unilateral client-side defenses are tailored to this.

AckwardDelay with Other Protocols. As a purely client-side defense with only little overhead, AckwardDelay is effective against state-of-the-art fully-remote website- and video-fingerprinting attacks. With less than 1 000 lines of code for our Linux proof-of-concept, AckwardDelay can easily be adapted to other operating systems and request-and-response protocols. While we focused on TCP (and additionally evaluated on ICMP, cf. Appendix A), AckwardDelay could also be applied to e.g., the acknowledgment mechanism of *QUIC* [38].

AckwardDelay Limitations. We designed AckwardDelay specifically against fully remote latency side-channel attacks, which neither rely on malicious code execution nor on an attacker in the middle. Network side channels with stronger attackers might require more heavyweight, multilateral defenses [13, 51, 25, 73]. Finally, AckwardDelay might be detrimental for purposes with low-latency requirements like gaming or video conferencing. For connections to trusted servers with those requirements, AckwardDelay can be selectively disabled.

9 Conclusion

AckwardDelay is a lightweight unilateral client-side mitigation that should be deployed as a secure default in many systems and applications. We computed upper leakage bounds, showing that even in an extremely powerful attack scenario, an attacker requires more than 250 samples to reduce the search space below 1 % with our recommended parameters. In practice, we observe that the attacker's accuracy in website- and video-fingerprinting attacks is reduced to random guessing. Our mitigation, AckwardDelay, incurs practically no bandwidth reduction and only increases website loading times by 5.55 %. This makes AckwardDelay a practical mitigation against fully remote traffic analysis attacks.

Acknowledgments

We thank our anonymous reviewers for their valuable feedback. This research is supported in part by the European Research Council (ERC project FSSec 101076409), the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85), the Deutsche Forschungsgemeinschaft (DFG grant number 503876675) and the European Union (grant number ROF-SG20-3066-3-2-2). Additional funding was provided by generous gifts from Red Hat and Google. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

1. Abusnaina, A., Jang, R., Khormali, A., Nyang, D., Mohaisen, D.: DFD: Adversarial Learning-based Approach to Defend Against Website Fingerprinting. In: INFOCOM (2020)
2. Al-Naami, K., El-Ghamry, A., Islam, M.S., Khan, L., Thuraisingham, B., Hamlen, K.W., Alrahmawy, M., Rashad, M.Z.: Bimorphing: A bi-directional bursting defense against website fingerprinting attacks. IEEE Transactions on Dependable and Secure Computing **18**(2), 505–517 (2021)
3. Alexa Internet, Inc.: The top 1 million sites on the web (5 2023), `https://www.alexa.com/topsites`
4. Apthorpe, N., Reisman, D., Sundaresan, S., Narayanan, A., Feamster, N.: Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. arXiv:1708.05044 (2017)
5. Arp, D., Yamaguchi, F., Rieck, K.: Torben: A Practical Side-Channel Attack for Deanonymizing Tor Communication. In: Asia CCS (2015)
6. Bahramali, A., Bozorgi, A., Houmansadr, A.: Realistic Website Fingerprinting By Augmenting Network Traces. In: CCS (2023)
7. Beverly, R., Luckie, M., Mosley, L., Claffy, K.: Measuring and Characterizing IPv6 Router Availability. In: PAM (2015)
8. Bhat, S., Lu, D., Kwon, A., Devadas, S.: Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. PoPETS **4**, 292–310 (2019)
9. Bissias, G.D., Liberatore, M., Jensen, D., Levine, B.N.: Privacy Vulnerabilities in Encrypted HTTP Streams. In: PETS (2006)
10. Bushart, J., Rossow, C.: Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS. In: USENIX FOCI (2020)
11. De la Cadena, W., Mitseva, A., Hiller, J., Pennekamp, J., Reuter, S., Filter, J., Engel, T., Wehrle, K., Panchenko, A.: TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In: CCS (2020)
12. Cai, X., Nithyanand, R., Johnson, R.: CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In: WPES (2014)
13. Cai, X., Nithyanand, R., Wang, T., Johnson, R., Goldberg, I.: A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In: CCS (2014)
14. Cai, X., Zhang, X.C., Joshi, B., Johnson, R.: Touching from a distance: website fingerprinting attacks and defenses. In: CCS (2012)
15. Chen, B., Yenamandra, V., Srinivasan, K.: Tracking Keystrokes Using Wireless Signals. In: MobiSys (2015)
16. Chen, S., Wang, R., Wang, X., Zhang, K.: Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In: S&P (2010)
17. Deng, X., Yin, Q., Liu, Z., Zhao, X., Li, Q., Xu, M., Xu, K., Wu, J.: Robust Multi-tab Website Fingerprinting Attacks in the Wild. In: S&P (2023)
18. Dubin, R., Dvir, A., Pele, O., Hadar, O.: I Know What You Saw Last Minute—Encrypted HTTP Adaptive Video Streaming Title Classification. IEEE Transactions on Information Forensics and Security **12**(12), 3039–3049 (2017)
19. Dyer, K.P., Coull, S.E., Ristenpart, T., Shrimpton, T.: Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In: S&P (2012)
20. van Ede, T., Bortolameotti, R., Continella, A., Ren, J., Dubois, D., Lindorfer, M., Choffnes, D., van Steen, M., Peter, A.: FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In: NDSS (2020)

21. Feghhi, S., Leith, D.J.: A Web Traffic Analysis Attack Using Only Timing Information. IEEE Transactions on Information Forensics and Security (2016)
22. Gast, S., Czerny, R., Juffinger, J., Rauscher, F., Franza, S., Gruss, D.: SnailLoad: Exploiting Remote Network Latency Measurements without JavaScript. In: USENIX Security (2024)
23. Gast, S., Puntigam, N., Franza, S., Neela, S.R., Gruss, D., Ullrich, J.: Zero-Click SnailLoad: From Minimal to No User Interaction. In: ESORICS (2025)
24. Ge, Q., Yarom, Y., Chothia, T., Heiser, G.: Time Protection: The Missing OS Abstraction. In: EuroSys (2019)
25. Gong, J., Wang, T.: Zero-delay Lightweight Defenses against Website Fingerprinting. In: USENIX Security (2020)
26. Gong, J., Zhang, W., Zhang, C., Wang, T.: Surakav: Generating Realistic Traces for a Strong Website Fingerprinting Defense. In: S&P (2022)
27. Gong, X., Borisov, N., Kiyavash, N., Schear, N.: Website Detection Using Remote Traffic Analysis. In: PETS (2012)
28. Gong, X., Kiyavash, N., Borisov, N.: Fingerprinting Websites Using Remote Traffic Analysis. In: CCS (2010)
29. Gu, J., Wang, J., Yu, Z., Shen, K.: Walls Have Ears: Traffic-based Side-Channel Attack in Video Streaming. In: INFOCOM (2018)
30. Hasselquist, D., Witwer, E., Carlson, A., Johansson, N., Carlsson, N.: Raising the Bar: Improved Fingerprinting Attacks and Defenses for Video Streaming Traffic. PoPETS (2024)
31. Hayes, J., Danezis, G.: k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In: USENIX Security (2016)
32. Henri, S., Garcia-Aviles, G., Serrano, P., Banchs, A., Thiran, P.: Protecting against Website Fingerprinting with Multihoming. In: PoPETS (2020)
33. Herrmann, D., Wendolsky, R., Federrath, H.: Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In: CCSW (2009)
34. Hintz, A.: Fingerprinting Websites Using Traffic Analysis. In: PETS (2003)
35. Holland, J.K., Hopper, N.: Protecting against Website Fingerprinting with Multihoming. In: PoPETS (2020)
36. Holzbauer, F., Maier, M., Ullrich, J.: Destination Reachable: What ICMPv6 Error Messages Reveal About Their Sources. In: IMC (2024)
37. Internet Engineering Task Force: RFC 7323: TCP Extensions for High Performance (2014), `https://datatracker.ietf.org/doc/html/rfc7323`
38. Internet Engineering Task Force: RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport (2021), `https://datatracker.ietf.org/doc/html/rfc9000`
39. Internet Engineering Task Force: RFC 9293: Transmission Control Protocol (TCP) (2022), `https://datatracker.ietf.org/doc/html/rfc9293`
40. Jin, Z., Lu, T., Luo, S., Shang, J.: Transformer-based Model for Multi-tab Website Fingerprinting Attack. In: CCS (2023)
41. Juarez, M., Imani, M., Perry, M., Diaz, C., Wright, M.: Toward an Efficient Website Fingerprinting Defense. In: ESORICS (2016)
42. Kadloor, S., Gong, X., Tezcan, T., Borisov, N.: Low-Cost Side Channel Remote Traffic Analysis Attack in Packet Networks. In: IEEE ICC (2010)
43. Kadloor, S., Kiyavash, N., Venkitasubramaniam, P.: Mitigating timing based information leakage in shared schedulers. In: INFOCOM (2012)
44. Khan, M.U.S., Bukhari, S.M.A.H., Maqsood, T., Fayyaz, M.A.B., Dancey, D., Nawaz, R.: SCNN-Attack: A Side-Channel Attack to Identify YouTube Videos in a VPN and Non-VPN Network Traffic. Electronics **11**(3) (1 2022)

45. Kocher, P.: Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems. In: CRYPTO (1996)
46. Korczyński, M., Duda, A.: Markov chain fingerprinting to classify encrypted traffic. In: IEEE Conference on Computer Communications (2014)
47. Köpf, B., Dürmuth, M.: A Provably Secure And Efficient Countermeasure Against Timing Attacks. In: CSF (2009)
48. Lescisin, M., Mahmoud, Q.: Tools for Active and Passive Network Side-Channel Detection for Web Applications. In: WOOT (2018)
49. Li, M., Meng, Y., Liu, J., Zhu, H., Liang, X., Liu, Y., Ruan, N.: When CSI Meets Public WiFi: Inferring Your Mobile Phone Password via WiFi Signals. In: CCS (2016)
50. Liberatore, M., Levine, B.N.: Inferring the source of encrypted HTTP connections. In: CCS (2006)
51. Lu, D., Bhat, S., Kwon, A., Devadas, S.: DynaFlow: An Efficient Website Fingerprinting Defense Based on Dynamically-Adjusting Flows. In: WPES (2018)
52. Luo, X., Zhou, P., Chan, E., Lee, W., Chang, R., Perdisci, R.: HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In: NDSS (2011)
53. Lyon, G.: Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure (2009)
54. Mathews, N., Holland, J.K., Oh, S.E., Rahman, M.S., Hopper, N., Wright, M.: SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses. In: S&P (2023)
55. Mehta, A., Alzayat, M., Viti, R.D., Brandenburg, B.B., Druschel, P., Garg, D.: Pacer: Comprehensive Network Side-Channel Mitigation in the Cloud. In: USENIX Security (2022)
56. Msadek, N., Soua, R., Engel, T.: IoT device fingerprinting: Machine learning based encrypted traffic analysis. In: Wireless Communications and Networking Conference (WCNC) (2019)
57. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of Tor. In: S&P (2005)
58. Nasr, M., Bahramali, A., Houmansadr, A.: Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In: USENIX Security (2021)
59. Network Working Group: RFC 2784: Generic Routing Encapsulation (GRE) (2000), `https://datatracker.ietf.org/doc/html/rfc2784`
60. Network Working Group: RFC 5681: TCP Congestion Control (2009), `https://datatracker.ietf.org/doc/html/rfc5681`
61. Nithyanand, R., Cai, X., Johnson, R.: Glove: A Bespoke Website Fingerprinting Defense. In: WPES (2014)
62. Oren, Y., Kemerlis, V.P., Sethumadhavan, S., Keromytis, A.D.: The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications. In: CCS (2015)
63. Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., Wehrle, K.: Website Fingerprinting at Internet Scale. In: NDSS (2016)
64. Panchenko, A., Niessen, L., Zinnen, A., Engel, T.: Website Fingerprinting in Onion Routing Based Anonymization Networks. In: WPES (2011)
65. Rahman, M.S., Imani, M., Mathews, N., Wright, M.: Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. IEEE Transactions on Information Forensics and Security **16**, 1594–1609 (2021)
66. Rahman, M.S., Mathews, N., Wright, M.: Video Fingerprinting in Tor. In: CCS (2019)

67. Rahman, M.S., Sirinam, P., Mathews, N., Gangadhara, K.G., Wright, M.: Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. PoPETS (2020)
68. Ravaioli, R., Urvoy-Keller, G., Barakat, C.: Characterizing ICMP Rate Limitation on Routers. In: IEEE ICC (2015)
69. Reed, A., Kranch, M.: Identifying HTTPS-Protected Netflix Videos in Real-Time. In: CODASPY (2017)
70. Rimmer, V., Preuveneers, D., Juarez, M., Van Goethem, T., Joosen, W.: Automated Website Fingerprinting through Deep Learning. In: NDSS (2017)
71. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: CCS (2009)
72. Rokicki, T., Maurice, C., Botvinnik, M., Oren, Y.: Port Contention Goes Portable: Port Contention Side Channels in Web Browsers. In: AsiaCCS (2022)
73. Sabzi, A., Vora, R., Goswami, S., Seltzer, M., Lécuyer, M., Mehta, A.: NetShaper: A Differentially Private Network Side-Channel Mitigation System. In: USENIX Security (2024)
74. Schuster, R., Shmatikov, V., Tromer, E.: Beauty and the Burst: Remote Identification of Encrypted Video Streams. In: USENIX Security (2017)
75. Shen, M., Gao, Z., Zhu, L., Xu, K.: Efficient fine-grained website fingerprinting via encrypted traffic analysis with deep learning. In: International Symposium on Quality of Service (IWQOS) (2021)
76. Shen, M., Ji, K., Gao, Z., Li, Q., Zhu, L., Xu, K.: Subverting Website Fingerprinting Defenses with Robust Traffic Representation. In: USENIX Security (2023)
77. Shen, M., Ji, K., Wu, J., Li, Q., Kong, X., Xu, K., Zhu, L.: Real-Time Website Fingerprinting Defense via Traffic Cluster Anonymization. In: S&P (2024)
78. Shintre, S., Gligor, V., Barros, J.: Optimal strategies for side-channel leakage in FCFS packet schedulers. In: International Symposium on Information Theory (ISIT) (2015)
79. Sirinam, P., Imani, M., Juarez, M., Wright, M.: Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In: CCS (2018)
80. Sirinam, P., Mathews, N., Rahman, M., Wright, M.: Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning. In: CCS (2019)
81. Skowron, M., Janicki, A., Mazurczyk, W.: Traffic fingerprinting attacks on internet of things using machine learning. IEEE Access **8**, 20386–20400 (2020)
82. Song, D.X., Wagner, D., Tian, X.: Timing Analysis of Keystrokes and Timing Attacks on SSH. In: USENIX Security (2001)
83. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP Topologies with Rocketfuel. In: SIGCOMM (2002)
84. Ullrich, J., Weippl, E.: The Beauty or The Beast? Attacking Rate Limits of the Xen Hypervisor. In: ESORICS (2016)
85. Van Goethem, T., Pöpper, C., Joosen, W., Vanhoef, M.: Timeless Timing Attacks: Exploiting Concurrency to Leak Secrets over Remote Connections. In: USENIX Security (2020)
86. Vanderlinden, V., van Goethem, T., Vanhoef, M.: Time and Time Again: Leveraging TCP Timestamps to Improve Remote Timing Attacks. In: NDSS (2026)
87. Vila, P., Köpf, B.: Loophole: Timing Attacks on Shared Event Loops in Chrome. In: USENIX Security (2017)
88. Walsh, T., Thomas, T., Barton, A.: Exploring the Capabilities and Limitations of Video Stream Fingerprinting. In: S&P Workshops (2024)

89. Wang, T., Cai, X., Nithyanand, R., Johnson, R., Goldberg, I.: Effective Attacks and Provable Defenses for Website Fingerprinting. In: USENIX Security (2014)
90. Wang, T., Goldberg, I.: Improved Website Fingerprinting on Tor. In: WPES (2013)
91. Wang, T., Goldberg, I.: Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In: USENIX Security (2017)
92. White, A., Matthews, A., Snow, K., Monrose, F.: Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks. In: S&P (2011)
93. Wright, C., Coull, S., Monrose, F.: Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In: NDSS (2009)
94. Zhang, X., Hamm, J., Reiter, M., Zhang, Y.: Statistical Privacy for Streaming Traffic. In: NDSS (2019)

Table 2. Overview of network side-channel mitigations

Technique	Mitigation	Concept	Victim	Intended Scenario	Unilateral Defense
Scheduling	Kadloor et al. [43]	Accumulate-and-serve scheduling	end user	❸	◐
Traffic Shape Modification	Panchenko et al. [64]	Download of additional websites	end user[1]	❷	●
	Luo et al. [52]	HTTP(S) obfuscation	end user	❷	●
	Al-Naami et al. [2]	Bidirectional traffic parameter morphing (Bimorphing)	end user[1]	❷	○
	Nasr et al. [58]	Adversarial learning-based traffic shaping (Blanket)	generic	❷	○
	Nithyanand et al. [61]	Traffic shaping per website cluster	end user[1]	❷	○
	Rahman et al. [65]	Adversarial learning-based burst molding (Mockingbird)	end user[1]	❷	○
	Wang et al. [89]	Supersequence per anonymity set	end user[1]	❷	○
	Wang and Goldberg [91]	Half-duplex communication	end user[1]	❷	○
	Wright et al. [93]	Traffic parameter morphing	generic	❷	○
	Zhang et al. [94]	Adversarial machine learning causing misclassification	end users	❷	●
	Zhang et al. [94]	Differential privacy-based traffic shapping causing misclassification	end users	❷	●
	Gong et al. [26]	Traffic pattern adjustment by generative adversarial networks (Surakav)	end user[1]	❷	○
	Shen et al. [77]	Traffic cluster anonymization based on k-anonymity (Palette)	end user[1]	❷	○
	Liberatore and Levine [50]	Packet padding to uniform sizes	end user[1]	❷	○
	Cai et al. [12]	Congestion-sensitive and rate-adaptive BuFlo (CS-BuFLO)	end user[1]	❷	○
	Cai et al. [13]	Provably secure variant of BuFlo (Tamaraw)	end user[1]	❷	○
	Dyer et al. [19]	Fixed-length packets at fixed intervals (BuFlo)	end user[1]	❷	○
	Lu et al. [51]	Fixed-burst patterns at dynamic intervals	end user[1]	❷	○
	Juarez et al. [41]	Addition of dummy packets	end user[1]	❷	○
	Abusnaina et al. [1]	Adversarial learning-based addition of dummy packets per burst (DFD)	end user[1]	❷	○
	Gong and Wang [25]	Dummy packets for trace front obfuscation (FRONT) and trace gluing (GLUE)	end user[1]	❷	○
	Holland and Hopper [35]	Traffic surging with successive decay (RegulaTor)	end user[1]	❷	○
	Hasselquist et al. [30]	Repeated trace front obfuscation (Adapted FRONT)	end user	❷	○
	Hasselquist et al. [30]	Adapted RegulaTor against video fingerprinting	end user	❷	○
	Hasselquist et al. [30]	Constant-rate bursts with random trail padding (Scrambler)	end user	❷	○
	Mehta et al. [55]	Hypervisor-based packet shaping (cloaked tunnel)	virtual machine	❸	○
	Sabzi et al. [73]	Differential privacy-based traffic shaping	generic	❷	○
Multipath Routing	De la Cadena et al. [11]	Demultiplexing over Tor entry relays	end user[1]	❷	○
	De la Cadena et al. [11]	Demultiplexing over Tor paths	end user[1]	❷	●
	Henri et al. [32]	Demultiplexing by multi-homing	end user[1]	❷	◐

[1] with privacy-enhancing tunnel, [2] with or without privacy-enhancing tunnel, ❶ attacker-controlled code, ❷ attacker-in-the-middle, ❸ off-path

A Real-World Security Evaluation with ICMP Echo Attack

Numerous mitigations (cf. Table 2) tackle network side channels on lower protocol levels, whereas we focus on unilateral, client-side mitigations on the transport layer. Consequently, AckwardDelay has to be adapted for different protocols. For instance, instead of measuring latencies with TCP ACKs, Gong et al. [27] and Kadloor et al. [42] pinged the victim's gateway (*i.e.*, home router) with ICMP Echo Requests. While gateways usually disable ICMP Echo replies for hardening, we show that AckwardDelay can also mitigate leakage from ICMP messages in the following. By design, AckwardDelay prevents precise latency measurements with ICMP Echo messages, yet it still allows for basic reachability tests.

In contrast to TCP ACKs, ICMP Echo replies are sent by the gateway directly, without any involvement of the client computer behind. Consequently, to mitigate ICMP-based attacks, AckwardDelay has to run on the gateway. To simulate such a setup, we construct a GRE tunnel [59] between the attacker sending ICMP Echo requests and the victim client. This prevents the victim's gateway from answering the ICMP Echo requests and ensures that these are forwarded to the client. The client is configured to reply to ICMP Echo Requests, without rate limits. Other traffic, caused by the victim's website accesses, is not routed through the tunnel.

Similar to Section 6, we perform a website fingerprinting attack, using the same 50 Mbit/s ADSL connection and the same 10 websites as before. While the websites are loading, the attacker measures packet latencies, sending ICMP

Echo requests to the client through the tunnel and recording the resulting traces. We again record 100 traces per website, using a download duration of 10 s and a sample rate of 0.2 ms. Without AckwardDelay, we achieve a high baseline accuracy of 98.5 %. With AckwardDelay and $W_{\max} = 200\,\text{ms}$, $D_{\max} = 25\,\text{m/s}$, we again only achieve the random-guessing accuracy of 10 %.

However, in contrast to TCP, the client has to reply to each single ICMP message separately and cannot send cumulative single responses. With larger response intervals or higher attacker sample rates, this can yield a significant backlog of ICMP echo replies being released as a burst at the end of the response interval, potentially overwhelming the connection and resulting in packet loss. In particular, on the tested connection, for $W_{\max} \in [100, 200, 300]$, we observed a packet loss of 0.04 %, 16.46 % and 37.14 %, respectively. Thus, to prevent excessive bursts and potential information leakage from packet-loss patterns, we recommend rate-limiting outgoing ICMP packets in addition to AckwardDelay. Many routers already rate-limit certain ICMP packets [68]. Recent work [23] has shown that even traces rate-limited to 1 Hz by the home gateway carry enough information to perform video fingerprinting attacks. Yet, AckwardDelay combined with rate limiting eliminates the packet bursts and thus behaves similar to the TCP scenario, minimizing timing leakage from the remaining ICMP messages.