Website fingerprinting on Nym: Attacks and Defenses

Eric Jollès
EPFL
eric.jolles@epfl.ch

Simon Wicky
Nym Technologies
simon@nymtech.net

Ania M. Piotrowska
Nym Technologies
ania@nymtech.net

Harry Halpin
Nym Technologies
harry@nymtech.net

Carmela Troncoso
MPI-SP & EPFL
carmela.troncoso@mpi-sp.org

Abstract

Website fingerprinting (WF) enables a passive eavesdropper to infer which web page a client is visiting, even when communications are encrypted or anonymized. In this paper, we study the vulnerability to website fingerprinting of Nym, a mix network based on the Loopix design that enables users to browse the Web. We show that although Nym adds delays and cover traffic to change packet patterns compared to Tor, it still leaks features that website fingerprinting attacks can exploit in both closed- and open-world settings. We carry out an in-depth analysis of the effectiveness of Nym's obfuscation mechanisms, originally designed to provide anonymity in messaging, in thwarting website fingerprinting. We show that mix delays, counterintuitively, not only fail to protect against website fingerprinting but actually make the attack more effective as the mix delays make it easier to distinguish incoming from outgoing packets. We also demonstrate that the current cover traffic strategy of Nym is not effective in thwarting website fingerprinting attacks unless it imposes a large overhead. To address these limitations, we design two new WF defenses based on Nym's existing obfuscation mechanisms that significantly reduce WF effectiveness. The first defense introduces cover traffic to match the bursty nature of real-world web traffic, reducing the F1 score to 0.39 (compared to 0.65 obtained by similar defenses applied on Tor) at moderate overhead increase. The second defense plummets the F1 score to 0.06 by channeling web traffic via Nym's constant traffic capabilities, at the cost of bandwidth.

Keywords

website fingerprinting, mixnets, anonymous communication

1 Introduction

Tor [12] has been the most popular anonymous communication system since 2004, attracting millions of daily users. It employs onion routing to conceal routing metadata such as IP addresses, but it does not obfuscate network traffic patterns. Consequently, Tor is susceptible to website fingerprinting attacks, which analyze features like packet size, timing, and traffic direction to infer the websites users visit [17, 19, 24, 25, 30, 33–35].

Proceedings on Privacy Enhancing Technologies 2026(2), 436–451

https://doi.org/10.56553/popets-2026-0055

An alternative to onion routing are mix networks (mixnets) [6]. Unlike onion routing, which sends all packets of a connection through a fixed circuit, mixnets route each packet independently along a randomly selected path comprising multiple 'mixnodes'. These nodes introduce random delays before forwarding packets, thereby disrupting temporal correlations between incoming and outgoing traffic and making traffic analysis attacks more difficult by a global passive adversary that can monitor the entire network. However, website fingerprinting attacks on mixnets have not been empirically studied before.

Researchers originally designed mixnets for messaging-oriented applications, where delays can be acceptable. However modern mixnets, such as Nym [11], a low-latency mixnet based on Loopix [36], also enable users to browse the web through a SOCKS5 proxy client. In this paper, we study Nym's vulnerability to website fingerprinting attacks. Our work makes the following contributions:

- We conduct the first empirical evaluation of website fingerprinting attacks on the Nym mixnet in both closed- and open-world settings. Our results show that Nym, in its default configuration, offers protection only marginally better than that of Tor despite use of cover traffic and various types of delays.
- We conduct an in-depth leakage analysis of the impact that cover traffic and mixing delays have on Nym's resistance against website fingerprinting attacks. Our results demonstrate that these obfuscation methods, originally designed to provide strong anonymity for messaging, offer fairly limited protection against website fingerprinting regardless of the configuration. In particular, we find that (1) mixing delays that were previously shown to thwart correlation attacks [32], actually benefit the website fingerprinting adversary because they create separation between incoming and outgoing traffic; and that (2) Nym's Poisson-based cover traffic does not hide highly relevant website fingerprinting features unless it imposes large overhead.
- We show that defenses designed for Tor can enhance Nym's website fingerprinting protection, but that defenses tailored to Nym's architecture offer the best protection. We introduce two novel Nym-specific defenses: WTF4Nym, which offers moderate protection at a moderate overhead increase; and POISS-OFF, which trades off latency and bandwidth overhead for strong protection.

Of independent interest, we also find that the way in which monitored and unmonitored sets are built in the open world setting from the BigEnough dataset [30] creates an unfair advantage for the website fingerprinting adversary. Thus, previous work might be overestimating the effectiveness of website fingerprinting attacks in the open-world. See Appendix A for more details.

2 Nym network overview

The Nym network is a decentralized overlay mixnet, based on the Loopix [36] architecture, that provides unobservability and third-party sender and receiver anonymity. It is formed by mixnodes that introduce delays in order to 'mix' incoming packets. These delays aim to thwart traffic analysis attacks by a global passive adversary watching the entire network, and so one could hypothesize these delays would help thwart attacks like website fingerprinting (WF) attacks [17, 19, 24, 25, 30, 33–35], traffic correlation [5, 31], or flow watermarking [21].

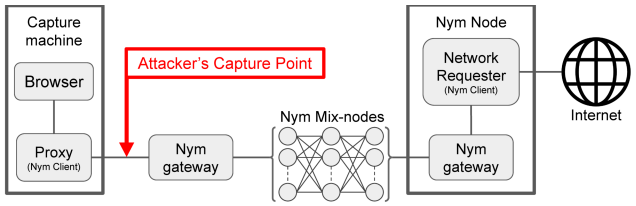


Figure 1: Network architecture and attack configuration: This figure illustrates the network configuration when browsing the Internet with Nym, showing the Nym components through which a network packet passes from the browser to the Internet.

2.1 Network architecture

The Nym mixnet consists of a set of *mixnodes*, which route encrypted data on behalf of Nym *clients* (see architecture in Figure 1). Nym mixes are organized into three layers and arranged in a *stratified topology* [13], i.e., each mix in a specific layer is connected with each mix in the previous and next layer. Traffic flows from the first to the last layer. Unlike Tor, which is based on circuits [12], Nym routes packets through independent communication paths, even between the same pair of clients. For each packet, the Nym client chooses a new route, composed of a random mix of each layer.

A set of gateways provides access to the Nym mixnet, serving as entry proxies that forward client packets to the mixnet and exit proxies that deliver packets from the final mix layer to their destination. Nym clients establish a long-lived secure channel with at least one gateway of its choice. Nym clients are not anonymous to that gateway as they must reveal their IP address and retrieve data from that gateway. But, because the traffic is encrypted and routed through multiple mixes, the gateway cannot determine the destination client. As the user communicates, the Nym client sends all requests to the associated gateway, which then forwards them to the mixnet.

The Sphinx packet format [7] encrypts Nym packets, guaranteeing bit-wise unlinkability and integrity protection. Each packet has four layers of encryption, one for each *mixnode* it traverses, plus an additional layer for the exit gateway.

2.2 Metadata obfuscation

To counter traffic analysis attacks, Nym implements several obfuscation techniques designed to disrupt temporal correlations and mask traffic patterns, summarized in Table 1.

Mixing. Each Nym mixnode independently delays packets before forwarding them to the next hop [26, 36]. These delays are selected by the sender: the Nym client chooses an individual delay for each mixnode along the packet's path, independently sampling from an exponential distribution with a publicly known parameter λ_M. The chosen delay values are embedded into the corresponding layers of the Sphinx-encrypted packet. This design ensures that even if an adversary observes packet flows at a mixnode, they cannot reliably correlate incoming and outgoing packets based on timing.

Poisson-rate traffic. When a user needs to send a packet, the Nym client does not immediately forward it to the mixnet. Instead, clients place Sphinx-encoded packets into a client's internal FIFO queue. Packets leave this queue at intervals determined by an exponential distribution with parameter λ_Q.

Whenever it is time for a packet to be sent, but there is no user-created packet in the queue, the Nym client will generate a cover packet indistinguishable from real traffic.

In a website browsing scenario, Nym distinguishes between the Poisson rate for the proxy (with parameter λ_{Q_P}) and for the network requester (with parameter λ_{Q_R}). By default, the proxy implements Poisson-rate traffic. To minimize latency during website browsing, the network requester, which retrieves responses from web servers, does not apply Poisson rate by default. As a result, incoming packets from the web server are sent to the Nym mixnet immediately upon arrival, potentially exposing temporal patterns that an adversary could exploit.

Loop cover traffic. Nym sends cover traffic in a loop. The cover packet traverses an independent, randomly chosen, mixnet route and then returns to the initiating client.

Cover traffic hides patterns of user activity, as there is always a stream of packets leaving the client regardless of the users' activity, and prevents active attacks [8]. Additionally, sending cover traffic on routes that loop back to the sender enables users to maintain locally updated knowledge of the network's health, including the ability to detect when nodes go down, become congested, or are under attack [8].

Nym generates two kinds of loop cover traffic. First, cover traffic used to complement user-created traffic to guarantee Poisson rate. Second, each Nym client also emits a separate stream of *loop cover traffic*, where packets exit the queue at intervals determined by an exponential distribution with parameter λ_L. This additional cover traffic further masks the actual timing of packets, which makes it more difficult for adversaries to deduce communication patterns through timing analysis.

3 Website fingerprinting on Nym

Website fingerprinting (WF) is a traffic analysis attack in which a passive *local* eavesdropper, who observes encrypted traffic at a single capture point between the client and a network, aims to learn which website the client is visiting [1, 2, 10, 17, 18, 30, 33, 35, 38–40]. A WF adversary observes packet sizes, timings, and directions (inbound or outbound), but they cannot decrypt the content of the network packets, and they do not alter or inject traffic. Additionally, a WF adversary does not control or compromise any network nodes.

To perform a WF attack, the adversary first creates a dataset of webpages' traffic traces by downloading these webpages multiple

Obfuscation	Description	Function
Mixing	Delay of packets at mixnodes	$\text{Exp}(\lambda_M)$
Poisson rate	Inter-packet delay for the cover stream that carries real traffic.	Proxy: $\text{Exp}(\lambda_{Q_P})$ Requester: $\text{Exp}(\lambda_{Q_R})$
Loop cover	Inter-packet delay for loop cover traffic.	$\text{Exp}(\lambda_L)$

Table 1: Nym obfuscation mechanisms and their implementation. Exp(λ) stands for exponential distribution with parameter λ, which defines the average rate of occurrence for events in the system.

times and capturing the traffic at the vantage point. The adversary extracts features from the metadata of these traces. The adversary uses these features to train a machine learning (ML) classifier that enables them to identify websites given a traffic trace. When a user visits a website, the attacker captures the user's network traffic and uses the classifier to determine which website the user accessed.

WF adversary. In this paper, we assume an adversary that monitors the network link between the user's device and the Nym gateway (see Figure 1). This monitoring can be performed by an internet service provider (ISP), a compromised router, or a malicious Wi-Fi hotspot. We do not consider entry gateways to be adversaries. This is contrary to the literature on Tor where the entry guard is typically adversarial. This is because, on Nym, the gateway is the only point where defenses can be implemented and thus gateways would see all traffic as sent from the client and server. This is because on Nym each packet follows a different path and therefore defenses cannot be implemented in mixnodes as they do not have a global view of the client traffic. As a result, our adversary model is weaker than in previous work [15, 25].

Adversary's goals. We evaluate the effectiveness of the attack in two settings often considered in the WF literature.

First, in a *closed-world* setting, the set of potential web pages a user can visit is limited and known in advance by the attacker. This scenario represents a strong adversary that has knowledge about the user's browsing habits and can be certain that the target user browsing happens within a finite set of suspect websites (e.g., by observing the DNS queries coming out of the network at the time of the observation [16, 37]).

Second, in an *open-world* scenario in which the adversary is interested only on a small subset of known web pages, referred to as the *monitored set*, and wants to know whether a user accessed a website that belongs to this set. This model reflects weaker adversary that cannot narrow down the websites visited by a user to carry out specific website identification, but still wants to monitor users' browsing habits with respect to given sites of interest. Such an adversary could be for example, a repressive government using the national ISP to identify users accessing forbidden content.

Following the approach used in WF literature on Tor [30], we assume that the adversary knows when traffic capture starts and ends. In the case of Nym, as shown in Figure 4, an adversary can clearly distinguish between periods of connection to a web server and inactivity, as incoming traffic bursts are significantly larger than cover traffic.

4 Evaluation methodology

In this section, we describe our data collection and evaluation methodology used to assess the resistance of the Nym mixnet to WF attacks. To facilitate reproducibility, all data collection scripts, evaluation code, and datasets are openly available in our GitHub repository: https://github.com/spring-epfl/WF4NYM-artifacts.

4.1 Data collection

Experimental setup. To facilitate our data capture, we use two Nym components that facilitate web browsing: a SOCKS5 proxy installed in the user's computer and a network requester in the Nym gateway (see Figure 1). Both the SOCKS5 proxy and the browser run within a separate network namespace[1], isolating proxy traffic from the rest of the system. To simulate an adversary capturing traffic between the user and the Nym network, we capture traffic between the SOCKS5 proxy and the gateway using dumpcap[2].

To access webpages, we employ a Python script to automate a Mozilla Firefox browser using Selenium[3] and Geckodriver[4] (version 0.34.0). To ensure consistency, we load pages without using caches. We also disable Firefox's tracking protection, as this service generates its own traffic that adds noise to the captures and prevent us from analyzing the protection against WF that the Nym network provides by itself.

To browse a web page, the user's browser sends a request to the SOCKS5 proxy. This proxy has a Nym client integrated which encrypts the request in a Sphinx packet and schedules its transmission to the Nym network. Once the encrypted packet reaches the egress Nym node, the gateway removes the Sphinx layered encryption and forwards the TLS-encrypted web request to the network requester. The network requester sends the message to the web server.

At the time we began this work, the available Nym release used a mechanism in which the proxy sent its Nym address (defined as user@gateway) to the requester, allowing the requester to send replies directly to that address. This interim design compromised the proxy's anonymity with respect to the network requester. However, it is important to note that this approach has since been replaced in the Nym mainnet by the use of Single Use Reply Blocks (SURBs), which enable anonymous replies without revealing the sender's address. Since our evaluation is based on the earlier version and does not focus on this aspect of anonymity, this limitation does not affect our results.

Target Nym network. For our evaluation, we collect data from two distinct instantiations of the Nym network.

Nym Mainnet (Mainnet). In this setup, the client uses Nym mixnodes that are part of the actual Nym network[5]. The SOCKS5 proxy runs on a Mac mini operating under Ubuntu 22, using a 1 Gb/s

[1] https://man7.org/linux/man-pages/man8/ip-netns.8.html
[2] https://www.wireshark.org/docs/man-pages/dumpcap.html
[3] https://www.selenium.dev
[4] https://github.com/mozilla/geckodriver
[5] https://explorer.nymtech.net/

upload/download connection. The setup reflects a typical user environment using consumer hardware and a standard ISP connection, making the collected traffic more representative of real-world scenarios. The same machine captures the traffic, providing the router owner or ISPs with a direct view of network activity. The network requester runs on a Linode[6] 4GB instance. The proxy, the gateway, and the network requester run Nym version 1.1.1. The Nym mixnet had 40 mixnodes arranged in 3 layers, for a total of 120 mixnodes, at the time of the experiment.

Nym-in-the-lab (Testnet). The Nym Mainnet is not stable enough [7] for controlled experiments (see Appendix B) and does not allow us to study different parameters of the obfuscation mechanisms (see Table 1). To address this limitation, we also set up a mixnet on a lab environment where we can study the influence of different configurations on Nym's resistance to WF attacks.

The Testnet mixnet consists of six mixnodes, forming a three-layer network, where each layer contains two mixnodes running Nym version 1.1.8. These mixnodes are located in Germany and the UK. Each mixnode operates as a separate Linode 2GB instance. In this setup, a SOCKS5 proxy, a gateway, and a network requester relay requests from the proxy to the Internet.

Nym network parameters. Unless specified differently, we use the default configuration[8] hardcoded in the Nym client. This configuration specifies that the inter-packet delay λ_{Q_P} is set to 20 milliseconds, λ_{Q_R} is disabled, the inter-loop delay λ_L is set to 200 milliseconds, and the mixing delay is sampled from an exponential distribution with a mean λ_M of 50 milliseconds (see Table 1 for the detailed description of the parameters).

Tor Network. To enable comparison with previous work, and across design principles behind anonymous communications networks, we also collected traffic traces over Tor. We used a Tor client (version 0.4.8.10) as a SOCKS5 proxy which, similarly to our setup on Nym, we connected to a Mozilla Firefox browser automated with Selenium and Geckodriver (version 0.34.0) to browse websites. The Tor client, used to handle the routing of traffic, runs on a Mac mini operating under Ubuntu 22, using a 1 Gb/s upload/download connection (same setup as Nym proxy).

Dataset composition. Using the setup described above, we collected a new dataset of website traces using Nym and Tor, using the URL list from the BigEnough dataset [30]. The foreground class of the original dataset consists of 95 websites, each with 10 sub-pages visited 20 times, for a total of 19,000 captures. The background class (list of unmonitored websites) comprises single captures of 19,000 unrelated websites.

We use two different list of monitored websites for our experiments: the Full List and the Reduced List. The specific websites included in these lists are detailed in Appendix A.

The Full List contains 79 websites with 10 sub-pages each. Compared to the original BigEnough dataset, we experienced access issues in 16 websites, either due to being blocked or timing out when accessed via Tor and Nym. We visit each sub-page 20 times, resulting in a dataset of 15,800 webpage download captures.

The Reduced List contains 58 websites with 10 sub-pages each. We use the Reduced List to evaluate the impact of modifying Nym parameters (as detailed in Section 5). Some parameter configurations led to timeouts, and certain pages became inaccessible over time, resulting in a final Reduced List of 58 websites. We visit each sub-page 20 times, resulting in a dataset of 11,600 webpage download captures.

To construct the background class, we select unmonitored websites from the URL list of the BigEnough dataset, matching the size of the Full List or Reduced List depending on the scenario. We do one capture per website. The first websites in the unmonitored list of BigEnough that did not result in timeouts or get blocked constitute the background class.

From network captures, we extract the direction, and timestamp of packets transmitted between the proxy and the Nym (or Tor) gateway, as done in the literature [30, 33]. We represent packet directions as +1 for incoming packets and −1 for outgoing packets. We filter out packets corresponding to HTTP ports or malformed entries, and exclude TCP ACKs that do not carry useful payload information.

4.2 Website fingerprinting attack

Previous studies [23, 30] identify *Tik-Tok* [38] as the most stable and effective website fingerprinting technique in the state of the art. Thus, we use it in our evaluation. (We also evaluated more recent attacks, such as Laserbeak [29], but they did not offer better performance than Tik-Tok on our dataset.)

Tik-Tok is a deep learning (DL)-based attack that uses a convolutional neural network (CNN) architecture for classification tasks. This network comprises several layers, including convolutional, batch normalization, activation, pooling and dropout layers. Its input is the sequence of incoming and outgoing cell directions multiplied by the corresponding timestamp. This integration merges directional and temporal information, improving attack scores compared to other ML attacks [17, 40].

In our evaluation, we use the Tik-Tok implementation by Jansen and Wails [23] adapted to the Nym network. Concretely, since Nym traces are longer than Tor traces studied in previous works, we extended the input size of the neural network to 10,000 (compared to 5,000 in the original and subsequent papers [23, 38, 40]). As a consequence, the model requires more time to train and test, but achieves better results for captures containing more than 5,000 packets.

To ensure correct input for the website fingerprinting attack, we filter the traffic captures before executing the attack. Indeed, both Nym and Tor pad all packets to fixed-size cells, 2413 bytes and 514 bytes, respectively. Lower-level protocols like TLS or WebSockets encapsulate each packet and transmit them over TCP, which in turn performs packet fragmentation or buffering to improve efficiency. These mechanisms cause multiple lower-layer packets to represent a single higher-layer packet or split higher-layer packets into multiple lower-level packets. For these reasons, we transform packets that contain multiple Tor or Nym cells into individual cells that have the same size. Therefore, we focus on timing and volume features instead of packet sizes.

[6] https://www.linode.com/

[7] Nym Mainnet is undergoing active development, with ongoing updates on both the node and client sides, and hence exhibits evolving network behavior. Our analysis reflects the network under these conditions, and future work can refine and extend these measurements as both the network and associated methodologies evolve.

[8] https://github.com/nymtech/nym/blob/1733aaa4ccb24eb6f0d156c8a84d876bbca1919a/clients/client-core/src/config/mod.rs#L26

Closed- vs. Open-world experiments. We conduct experiments in closed-world and open-world scenarios. We evaluate these scenarios using the Full List (15,800 captures) and Reduced List of webpages (11,600 captures). In the open-world scenario, following the approach of BigEnough [30], we match the number of captures in the unmonitored dataset to those in the monitored dataset.

In both scenarios, we split the dataset into 60% training and 40% testing, ensuring equal representation for each label in both sets. The ML model remains the same for both scenarios, but the labels change. In the closed-world setting, each website receives a unique label. In the open-world setting, we assign "1" to monitored websites and "0" to unmonitored websites.

4.3 Evaluation metrics

WF effectiveness. As in previous works [17, 25], we evaluate the effectiveness of WF attacks using the *F1 score*, which is the harmonic mean of precision and recall.

Feature importance. To better understand which network metadata makes a trace fingerprintable, we compute the importance of sets of different network characteristics (features). In this paper, we use the following feature sets inspired by the information leakage framework introduced by Li et al. [27]:

- Packet Count: Includes multiple statistics of incoming, outgoing and global packet counts.
- Time: Includes the 25th, 50th, 75th and 100th quartiles of packet transmission time, relative to the start of the capture.
- Inter-arrival Time: Includes the mean, variance, and standard deviation of packet inter-arrival times.
- N-gram: Includes the frequency of all possible sequences of n packets, with n between 2 and 6 (e.g., counts how many times the sequence [-1, +1, -1] or [+1, +1, +1, -1, -1, -1] appears on the trace).
- Transposition: Includes statistics of the precise ordering (position) of outgoing and incoming packets.
- Bursts: Includes statistics of burst of incoming data, which corresponds to the number of consecutive incoming packets received without outgoing packet in between.

Compared to the feature sets of Li et al. [27], we distinguish the temporal features by separating the transmission time from the inter-arrival time. This distinction reflects the fact that, in Nym, Poisson-rate is likely to reduce some of the utility associated with inter-arrival time, while leaving total capture time unchanged. We discarded the following feature sets that did not provide any additional information: Interval I, Interval II, Interval III, Packet Distribution, First 20, Last 20, First 30 and CUMUL. For instance, Interval I / II / III produce results similar to N-gram, so we exclude these feature sets from our analysis. Additionally, we discarded feature sets based on statistics of initial packets, such as First 20 and Last 30, because given the Nym cover traffic, they do not provide meaningful information. We group the Packet Count, Time and Inter-arrival Time feature sets into three subsets representing incoming traffic (Features In), outgoing traffic (Features Out) and both directions (Features In/Out).

To measure the importance of these feature sets for website fingerprinting, we use a technique inspired by the information leakage measurement presented by Mathews et al. [30]. For each features set, we run a Random Forest classifier trained only with those features, and use the accuracy of this feature-specific classifier as measure of importance. To ensure robustness, we apply 5-fold cross-validation (CV).

Latency and Bandwidth Overheads. The anonymity trilemma states that anonymity (in our case resistance to WF), cannot be obtained without increasing latency of communications or bandwidth consumption [9]. Reducing latency overhead is essential since high latency not only negatively impacts the user experience, but can lead to timeouts preventing users from accessing webpages (see timing analysis in Appendix C). High bandwidth overhead can make a defense unusable due to restricted network capabilities of the client or the limited processing/networking capabilities of intermediate nodes (e.g. Tor nodes or mixed Nym nodes). It can also affect the battery consumption of mobile devices [20].

To understand the impact of different parameters on the protection versus overhead trade-off, we compute the following metrics.

Latency Overhead (LO). It measures the additional time taken to fully load a webpage while using a defense mechanism compared to the direct loading of the same page without any defense. We compute the latency overhead of a webpage as:

$$\mathrm{LO}_{\mathrm{webpage}} = \frac{t_{\mathrm{defended}} - t_{\mathrm{original}}}{t_{\mathrm{original}}}$$

where t_{defended} represents the median time taken to fully load the webpage with the defense mechanism in place, and t_{original} the median time without any defense.

We compute the overall latency overhead (LO) incurred by a defense by taking the median of the latency overhead per website across all webpages.

$$\mathrm{LO} = \mathrm{median}(\mathrm{LO}_{\mathrm{webpage}_1}, \mathrm{LO}_{\mathrm{webpage}_2}, \cdots, \mathrm{LO}_{\mathrm{webpage}_n})$$

This method ensures that our measurement of the latency overhead remains robust to variations in individual webpage captures. The resulting LO is a positive number that approaches zero when there is no overhead.

Bandwidth overhead (BO). It represents the additional data transmitted by a defense mechanism compared to the undefended traffic:

$$\mathrm{BO}_{\mathrm{webpage}} = \frac{b_{\mathrm{defended}} - b_{\mathrm{original}}}{b_{\mathrm{original}}}$$

where b_{defended} represents the median number of bytes transmitted to fully load a webpage with a defense mechanism in place, and b_{original} denotes the median number of bytes transmitted to load the same webpage without any defense. Similarly to the computation of the latency overhead, we compute the total latency overhead as the median of all $\mathrm{BO}_{\mathrm{webpage}}$.

We note that our overhead measurements in this paper only consider the overhead *while the client is loading a webpage*. In practice, Nym clients send loop cover traffic continuously, also before and after the browsing session. This means that the total BO under continuous loop cover traffic is higher than the per-session BO we report, especially if there are long idle periods. We focus on per-session BO because the start and the end of a page load are observable events (see Figure 2), and thus, from a web privacy

URL List	Network	LO	BO	F1 score	
				Closed-World	Open-World
Full	Nym Mainnet	11.93	8.97	0.69	0.71
	Nym Testnet	3.69	3.73	0.87	0.83
	Tor	2.8	0.64	0.94	0.91
Reduced	Nym Mainnet	12.38	8.90	0.71	0.73
	Nym Testnet	3.78	3.54	0.87	0.87
	Tor	2.79	0.54	0.95	0.92

Table 2: Website fingerprinting effectiveness: Tik-Tok attack F1 score for Tor and Nym across Full List and Reduced List. (See Appendix A for details on open world scores)

	Nym Mainnet	Nym Testnet	Tor	WTF-PAD	Front	Tamaraw
Pkt. Count	30.5	41.8	52.6	27.6	13.1	11.9
Time	41.3	58.7	44.3	33.4	26.6	11.5
Inter-arrival Time	21.2	30.9	39.9	22.3	14.9	11.7
N-gram	8.2	58.7	68.6	48.5	33.7	13.3
Transposition	3.2	40.8	76.3	33.2	22.7	5.3
Bursts	24.0	26.9	44.0	30.0	25.5	12.3
Features In	53.9	72.1	70.2	54.7	40.0	10.4
Features Out	33.1	43.9	65.0	42.0	24.4	10.9
Features In/Out	51.6	70.0	71.9	53.5	39.3	11.5

Table 3: Information Leakage: Top-1 accuracy (in %) of 5-fold cross-validation using a random forest, evaluated on data from Nym, Tor, and Tor defenses with the Reduced List. Maximum standard deviation is under 1.5%. Cells are colored according to the severity of leakage as follows: red for $x > 60\%$, orange for $40\% < x \leq 60\%$, yellow for $20\% < x \leq 40\%$, and no color for $x \leq 20\%$.

perspective, loop cover traffic outside these intervals is superfluous. Yet, we note that keeping the loop active continuously can improve privacy by hiding small, sporadic communications, such as instant-messaging traffic (e.g., Signal) or cryptocurrency transactions happening in the Nym network. We do not include overhead measurements including continuous traffic as they can be directly computed from the loop cover rate λ_L.

5 Evaluation results

In this section, we study the level of protection against WF offered by Nym, and how it differs from the protection offered by Tor. We also perform an in-depth analysis of the influence of Nym's various obfuscation mechanisms on the trade-off between protection against website fingerprinting and overhead.

5.1 Protection against website fingerprinting

We run the Tik-Tok attack on traces collected over the Nym Mainnet and Testnet with the default Nym configuration, and on traces collected over Tor. We report the results for both the closed- and open-world scenarios in Table 2.

Resistance to Website Fingerprinting. We see in Table 2 that, in its current configuration, Nym is vulnerable to WF attacks in both closed- and open-world settings. This may seem counter-intuitive, as Nym's Poisson processes should theoretically hide traffic patterns. However, in the default Nym configuration, Nym's proxy only receives cover traffic that it generated from its Poisson-rate and Loop cover. But, packets transmitted from the network requester to the proxy are not sent at a Poisson rate nor have cover traffic, making traffic arriving at the proxy vulnerable to traffic analysis.

When running this experiment, we noticed that the F1 score in the open world is unusually high. Looking closer to the issue, we realized that the way in which the monitored and unmonitored sets are built in the BigEnough dataset [30] results in an advantage for the adversary. When fixing this issue (see appendix A), the open-world F1 score for the Nym Testnet is 0.61 instead of 0.83, which is similar to the performance of the typical open-world website fingerprint attack found in the literature.

Nym Mainnet performance. We observe that the attack performs significantly worse on Mainnet than on Testnet or Tor. We hypothesize that this difference comes from two factors. The first factor is geographical distribution: Mainnet nodes are spread across the world, whereas Testnet nodes remain close together. In Nym, every packet traverses a different path, causing the packets to experience different network delays depending on the location of the Nym nodes in the path. The second factor is packet retransmissions: traces collected from Mainnet show an unusually high retransmission rate (around 5% of packets). These retransmissions further contribute to packets arriving out of order. Both issues significantly affect order-specific patterns (see feature sets N-gram and Transposition in Table 3), which impacts the adversary's classifier ability to identify websites. However, this protection comes at the cost of high latency due to retransmissions. These retransmissions, in turn, increase bandwidth overhead, as the extended download time causes the system to send dummy packets for a longer duration in both Nym cover loops. Additionally, it is important to highlight that Nym's packet size is of size 2,413 bytes, while Tor cells are 514 bytes. Thus, any cover packet adds a factor of 4.7 of overhead compared to adding a cover packet in Tor, amplifying the bandwidth overhead required in to generate cover traffic in Nym.

In contrast, the Testnet setup operates in controlled conditions. Here, only our client sends traffic through the mixnodes, ensuring stable and reliable network links where retransmissions are rare. There are only eight different paths possible with mixing nodes spread across the UK and Germany. The lack of variety and proximity of the mixing nodes results in the end-to-end latency between the proxy and the network requester being smaller and less variable than in Nym's Mainnet or Tor network. Due to this lack of real-world network "jitter" that naturally obfuscates packets in both Tor and Nym Mainnet, this stability results in more predictable traffic patterns, making Testnet a worst-case scenario to study resistance to WF attacks.

Differences between Nym and Tor traffic captures. In both closed- and open-world settings, traffic captures from Nym and Tor reveal website patterns, even with Nym obfuscation mechanisms. We show traffic trace (number of incoming and outgoing packets) when accessing the website https://nym.com/ via Nym Testnet

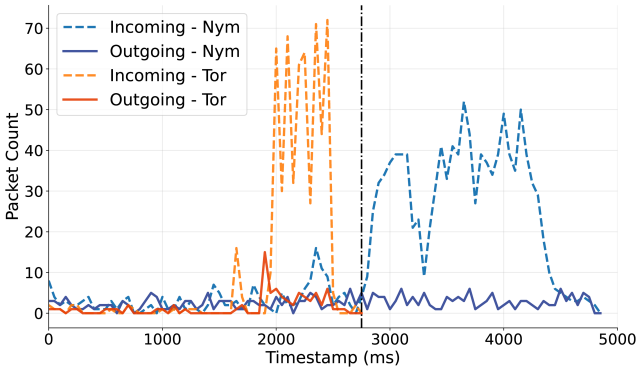


Figure 2: Comparison of incoming and outgoing packet counts over time in Nym Testnet and Tor for the query of https://nym.com/ for 50 ms time window. The vertical line indicates the point at which Tor capture ends.

(blue) and Tor (orange) in Figure 2. Tor, which does not implement obfuscation mechanisms, reveals the real timing and number of packets. This visibility results on features like Pkt. Count, Time, and Inter-arrival Time leaking information about the browsed website, each achieving an accuracy above 40% (see Table 3). In addition, because Tor uses TCP connections between different nodes and routes all packets from a browsing session along the same path, the order of packets remains largely consistent across different captures. This results in N-gram and Transposition being the most important features in Table 3 for Tor.

Nym introduces cover traffic and delays, which modify capture patterns. The delays added by Nym mixes impact the global download time in Nym, making it significantly longer than Tor. Yet, this added time per download does not vary significantly from capture to capture, so the Time feature set is still important to Nym despite the delay. In fact, on Nym's Testnet, which has very stable nodes and network connection, the global time is even more consistent between the captures than Tor and Nym Mainnet. As a result, the Time feature set leaks 10 percentage points more information in Nym Testnet compared to Tor.

The cover traffic continuously generated by the proxy adds outgoing and incoming packets to the link between client and gateway. These added packets introduce variability to the Packet counts and Inter-arrival Time, and influence the order of the packets reflected in N-Gram, Transposition and Bursts. Consequently, cover traffic decreases the amount of information leaked by these feature sets compared to Tor. Yet in Nym's default configuration, the network requester forwards packets from the web server towards the proxy as soon as they arrive, without generating cover traffic or enforcing a Poisson rate. As a result, these packets are only obfuscated by the mixing delay, leaving identifiable characteristics in the incoming traffic to the proxy that enable website fingerprinting. This is apparent in Figure 2: When using Nym, bursts of incoming traffic are visible and consist mainly of real data packets. Thus, most feature sets leak information as the incoming traffic is not sent at a Poisson rate.

Configuration	λ_{Q_P}	λ_M	λ_L	λ_{Q_R}	LO	BO	F1 score
Default	20	50	200	X	3.78	3.54	0.87
1	10	50	200	X	3.74	5.54	0.84
2	40	50	200	X	4.51	2.43	0.91
3	20	50	X	X	3.84	3.39	0.90
4	X	50	X	X	3.13	1.13	0.97
5	X	100	X	X	5.64	1.31	0.98
6	X	0	X	X	0.88	1.16	0.91
7	20	0	200	X	1.80	2.54	0.85
8	20	20	200	X	2.51	2.67	0.88
9	20	100	200	X	6.53	4.79	0.87
10	20	50	200	10	7.32	4.88	0.75
11	20	50	200	20	12.66	7.90	0.64
12	X	50	X	10	6.73	1.12	0.97

Table 4: Overheads and Tik-Tok attack F1 score for different configuration of Nym in closed-world with Testnet. LO stands for Latency Overhead, BO stands for Bandwidth Overhead. Parameters in blue means they are different from the default one, X corresponds to removal of a distribution. Results in red indicates an increase compared to the default configuration (first row), green indicates a decrease, and black indicates no change.

Incoming traffic leaks substantial information. This is reflected when we group features depending on the direction of the traffic. As shown in Table 3 (three last rows), outgoing traffic (*Features Out*) leaks less information in Nym than in Tor due to its constant transmission rate. The combined feature set (*Features In/Out*) does not enhance *Features In*, indicating that outgoing traffic does not provide independent information beyond incoming traffic. In contrast, incoming traffic leaks a lot of information, such as website size and inter-packet timing and appears exposed when examining Figure 2. This shows that defenses are required for incoming traffic.

5.2 Influence of obfuscation mechanisms

We now study whether there exist a configuration of Nym's obfuscation mechanisms (Table 1) that can result in a better overhead vs. WF protection trade-off. To this end, we collect traces from all URLs on the Reduced List on the Testnet under different configurations of the mixnodes. For all configurations, we run the Tik-Tok attack in the closed-world setting, measure the overhead (see Table 4), and conduct an information leakage analysis (see Table 5).

Impact of proxy-side Poisson rate (λ_{Q_P}). The proxy-side inter-message delay (parameter λ_{Q_P}) governs the rate at which traffic, both real packets and loop cover traffic, leave the Nym proxy.

Reducing the time between sent packets speeds up the browsing process but also generates more cover traffic, as additional loop cover packets are sent while waiting for webpage responses. We can see this effect in configuration 1, where lowering λ_{Q_P} to 10 ms increases bandwidth overhead to 5.54. This cover traffic disrupts the order of packets: the leakage of the *Transposition* feature set decreases significantly (40.8% → 7.4%), but this does not have large impact on the F1 score of the Tik-Tok attack that remains high

Configuration	Default	1	2	3	4	5	6	7	8	9	10	11	12
Changed parameter(s)		λ_{Q_P}=10	λ_{Q_P}=40	~~λ_L~~	~~λ_{Q_P}~~ & ~~λ_L~~	$\lambda_M = 100$ ~~λ_L~~ & ~~λ_{Q_P}~~	$\lambda_M = 0$ ~~λ_L~~ & ~~λ_{Q_P}~~	λ_M=0	λ_M=20	λ_M=100	λ_{Q_R}=10	λ_{Q_R}=20	λ_{Q_R}=10 ~~λ_{Q_P}~~ & ~~λ_L~~
Pkt. Count	41.8	33.5	45.5	40.2	55.2	57.8	51.4	33.9	39.1	37.7	32.5	27.9	48.0
Time	56.7	51.0	58.7	56.6	70.0	74.6	61.8	50.1	54.0	54.4	40.9	32.8	71.7
Inter-arrival Time	30.9	19.6	27.5	25.4	64.5	68.6	63.0	17.9	23.5	26.8	18.7	16.2	56.9
N-gram	58.7	47.0	53.2	51.6	74.4	78.1	68.8	42.9	49.2	49.8	40.9	37.7	71.9
Transposition	40.8	7.4	66.1	38.7	91.9	95.5	88.5	49.9	52.9	17.7	21.1	14.6	88.5
Bursts	26.9	24.0	23.2	22.1	73.2	87.1	71.0	23.0	23.2	23.8	19.4	18.7	72.5

Table 5: Information Leakage: Top-1 accuracy (in %) of 5-fold cross-validation using a random forest, evaluated on data from Testnet with varying configurations, using the Reduced List. ~~λ~~ corresponds to removal of a distribution. Maximum standard deviation is under 1.5%. Cells are colored according to the severity of leakage as follows: red for $x > 60\%$, orange for $40\% < x \leq 60\%$, yellow for $20\% < x \leq 40\%$, and no color for $x \leq 20\%$.

(0.84). The latency overhead is however not impacted too much. This indicates that while browsing, the browser is generally not sending more than one packet every 20 ms, so decreasing the value of this parameter does not bring an advantage in that sense.

When we increase λ_{Q_P} to 40 ms (configuration 2), we naturally observe a higher latency because packets are sent at a lower rate. But, less cover traffic is generated, and so there is a lower bandwidth overhead. Decreasing the amount of cover traffic leaves information about the order of the packet (*Transposition*) visible, increasing its leakage from 40.8% to 66.1%, resulting in an increase of 0.03 in the F1 score. When we fully remove the Poisson rate (configuration 4), the overhead drops on both dimensions. But, these unprotected traces lead to very high F1 score (0.97). We conclude that loop cover traffic does play a role in protecting network traces against WF attacks, by hiding features related to the order of packets (*Transposition*, and *Bursts*). However, its effectiveness in thwarting website fingerprinting is limited, and it comes at a high overhead cost.

Impact of network requester side cover traffic (λ_{Q_R}). The goal of the network requester side loop traffic (parameter λ_{Q_R}), is to hide incoming traffic bursts that might lead to website identification.

Configurations 10 and 11 in Table 4 show that enabling the inter-message delay on the network requester side such that the F1 score decreases, increases significantly the latency overhead (12.66 when $\lambda_{Q_R} = 10$). However, due to Nym's architecture, loop cover traffic generated by the network requester is transmitted via mixnodes back to the network requester and does not arrive to the proxy. As a result, this additional traffic does not strongly contribute to hiding the traffic patterns between the Nym gateway and the client-side proxy and while the amount of information leaked decreases (see Table 5), the attack still performs considerably well.

Removing the proxy-side cover traffic while adding the network requester Poisson rate in configuration 12 makes the system as vulnerable (F1 score of 0.97) as in configuration 4, which only has the mixing protection. Indeed, when there is no cover traffic generated by the proxy, all the packets in the trace contain webpage information, but they are transmitted with a delay. As a result, the Interarrival time feature set leaks less information than in configuration 4. When we add Poisson rate to both sides of the communication (configurations 10 and 11), the Inter-arrival time also leaks very little information, but this reduction comes at the expense of high bandwidth and latency overhead.

Impact of mixing delay. In order to understand the impact of the mixing delay, we test configurations without cover traffic (configurations 4, 5, and 6 in Table 4). We observe that the mixing delay has the opposite effect than expected: increasing the mixing delay results in lower protection. This counterintuitive outcome is due to two factors. First, larger delays make webpage download times consistent across captures. This is because as the mix delay increases, inconsistencies caused by network fluctuations become less noticeable.

Second, as the mixing delay increases, the separation between outgoing and incoming bursts of traffic increases. Recall that the website fingerprinting adversary observes traffic between the client and the gateway. Thus, when there is no cover traffic, the outgoing traffic is unprotected and the adversary sees outgoing bursts as they are produced. Requests must then traverse the mixnet before reaching the server, while responses follow a separate return path through the mixnet. As mix delays increase, responses take longer to arrive back at the proxy, creating a visible time gap between outgoing and incoming bursts in the network traffic.

We illustrate this in Figure 3, where we plot the incoming and outgoing traffic for https://nym.com/ for the mixing delay $\lambda_M = 0ms$ (blue) and $\lambda_M = 100ms$ (orange). When there is no mixing delay, incoming and outgoing bursts overlap in time (see blue lines, around 800 ms into the trace). This overlap varies depending on network conditions, affecting the order in which incoming and outgoing packets are observed. This reduces the information carried by Transposition and N-gram feature sets and thus reduces the effectiveness of the attack.

When the mixing delay increases, we observe that the outgoing bursts (orange solid line, around 1500 ms) and the incoming bursts (orange dashed lines, starting after 2000 ms) barely overlap. Thus, small changes in network conditions will not affect the trace representation: there will be no transpositions and n-grams remain consistent across captures. Thus, the corresponding feature sets carry more information and the attack performs better overall.

When cover traffic is added (configurations 7, 8, and 9), this phenomenon is less apparent, as the continuous rate of outgoing traffic

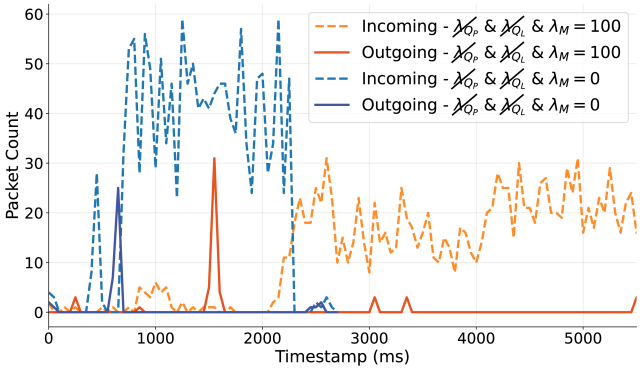


Figure 3: Comparison of incoming and outgoing packet counts over time in Nym Testnet without cover traffic for the query of https://nym.com/ within a 50 ms time window. A higher mixing delay causes incoming and outgoing traffic bursts to become more separated, making them more recognizable to an adversary.

from the proxy ensures that N-gram and Transposition features vary across captures, drastically reducing the information in these feature sets (28.3 percentage points reduction for N-gram, and 78.4 percentage points reduction for Transposition when $\lambda_M = 100$ ms).

This analysis leads to an important hypothesis: That it is cover traffic from the end-points that are more effective in defeating website fingerprinting attacks than mix delays applied to traffic while the traffic is en route. This makes sense as delays are only effective insofar as they cause incoming and outgoing traffic to obfuscate each other at the point of the adversary's observation. Therefore, a closer inspection of cover traffic strategies is necessary.

5.3 Tor defenses

Prior work on Tor has proposed and evaluated a range of defenses against WF attacks [3, 14, 24, 28, 35, 42]. These mechanisms aim to hide traffic patterns in order to reduce the information available to an adversary, while keeping latency and bandwidth overheads low. In this section, we examine three of the most commonly studied defenses to understand how they hide traffic patterns and what trade-offs they offer in terms of protection versus overhead.

WTF-PAD [25]. WTF-PAD hides real traffic bursts of network traces by inserting fake bursts of incoming and outgoing traffic. A burst refers to a sequence of packets transmitted within a relatively short period, indicating high-bandwidth activity. WTF-PAD creates fake bursts by applying predefined statistical distributions to inter-arrival times (the time between consecutive packets) and inter-burst times (the interval between consecutive traffic bursts). WTF-PAD derives these statistical models from real traffic captures to ensure that the generated bursts mimic real network behavior.

Because it only adds traffic, WTF-PAD does not introduce delays in the actual application data, ensuring that the user experience remains unaffected. WTF-PAD effectively protects traffic patterns against classical ML-based WF attacks, but it struggles against DL-based attacks [30].

Defense	LO	BO	F1 score
Tor	2.79	0.54	0.95
WTF-PAD	2.79	1.27	0.86
FRONT	2.79	1.84	0.65
Tamaraw	11.72	2.69	0.06

Table 6: Overheads and Classifier performance with Tik-Tok attack for Tor and Tor defences.

FRONT [15]. FRONT is a zero-latency, low-bandwidth defense that targets the first part of a website's trace with randomized cover traffic. FRONT ensures that traces of the same web page look different per trace. FRONT has proven highly effective against classical ML and DL attacks while maintaining minimal overheads, making it an effective privacy-preserving technique [30].

Tamaraw [4]. Tamaraw hides traffic characteristics such as packet timings and volume by sending packets at fixed intervals and padding the capture time to a multiple of a padding parameter L. This implies that some timing metadata is still leaked, as the total trace length is still an indicator of the length of the undefended traces. The constant packet flow offers strong and almost ideal protection against traffic analysis attacks due to packets being sent at a constant rate, but this comes at a very high bandwidth and latency overhead [30].

Evaluation. We simulate Tor defenses traces using FRONT's source code [15] and show the results in Table 6.

WTF-PAD and FRONT manage to reduce the F1 score (by up to 30%) without adding significant latency and only slightly increasing bandwidth overhead. Tamaraw makes the F1 score drop to 0.06, severely limiting the amount of information that is leaked (see Table 3), but the latency overhead (11.72) is really high.

When we compare these results with the ones of different Nym parametrization (see Table 4), we notice that Nym does not manage to defend against WF attacks when compared with state of the art Tor defenses. The three defenses leak less information than Nym Testnet (see Table 3), while WTF-PAD and FRONT also introduce lower overhead.

This leads to the question of whether or not new cover traffic designs can be created that can take advantage of Nym's sending traffic at a Poisson rate while taking into account the bursty nature of web traffic. In the next section, we propose new defenses tailored to Nym's design that provide more optimal trade-offs than its current default obfuscation mechanisms.

6 Mitigating website fingerprinting attacks on Nym

Our analysis of the effectiveness of Nym's obfuscation mechanisms against website fingerprinting shows that no combination of these mechanisms can effectively thwart the attack. In this section, we explore three alternatives: directly applying defenses designed for Tor to Nym with disabled Poisson rate (Nym+WTF-PAD and Nym+FRONT), adapting Tor-specific defenses to Nym's characteristics (WTF4Nym), and a constant-traffic defense based on Nym's existing obfuscation mechanisms (POISS-OFF).

We evaluate all defense strategies on the Reduced List-based dataset in the closed-world. For each defense, we modify the traces in the dataset to simulate the effect the defense would have on the traffic observed by the adversary. We show the performance results for all defenses in Table 7, and the result of the information leakage analysis in Table 8.

Defense	LO	BO	F1 score
Nym	3.78	3.54	0.87
Nym+WTF-PAD	3.13	2.27	0.87
Nym+Front	3.13	2.93	0.63
WTF4Nym	3.78	4.22	0.39
POISS-OFF	7.32	8.06	0.06

Table 7: Overheads and Tik-Tok attack F1 score for different defenses on top of Nym in closed world with Testnet. We provide timing measurements in Appendix C.

	Default	Nym+ WTF-PAD	Nym+ FRONT	WTF4Nym	POISS-OFF
Pkt. Count	41.8	26.0	10.9	14.1	7.2
Time	56.7	49.4	39.0	26.6	8.4
Inter-arrival Time	30.9	33.0	19.8	11.3	3.6
N-gram	58.7	46.8	27.2	29.0	9.5
Transposition	40.8	22.3	18.7	11.8	1.9
Bursts	26.9	28.9	19.7	15.2	9.1

Table 8: Information Leakage: Top-1 accuracy (in %) of 5-fold cross-validation using a random forest, evaluated on data from Testnet protected with different defenses, using the Reduced List. Maximum standard deviation is under 1.5%. Cells are colored according to the severity of leakage as follows: red for $x > 60\%$, orange for $40\% < x \leq 60\%$, yellow for $20\% < x \leq 40\%$, and no color for $x \leq 20\%$.

6.1 Applying Tor-specific defenses on Nym

Since our analysis reveals that Nym's cover traffic (λ_{Q_P}, λ_{Q_R}, λ_L) is not particularly effective against WF, we test what happens when we deactivate cover traffic and have the proxy and Nym's entry gateway implement two Tor website fingerprinting defenses: WTF-PAD [25] and FRONT [15]. We refer to these approaches as Nym+WTF-PAD and Nym+FRONT, respectively.

Evaluation. We simulate the effect of applying Nym+WTF-PAD and Nym+FRONT on traces without cover traffic from configuration 4 in Table 4.

Tik-Tok on Nym+WTF-PAD and Nym+FRONT obtains F1 scores similar to those of the same defenses on Tor (see Table 6). Nym+WTF-PAD obtains a marginal improvement on latency and halves the bandwidth overhead with respect to the default Nym configuration. It adds some protection over Nym without any obfuscation (it reduces the F1 score of configuration 4 from 0.97 to 0.87) but it does not provide better protection than the default Nym configuration (which also has an F1 score of 0.87). This is due to the fact that WTF-PAD does not provide good protection against deep learning attacks [38, 40]. While WTF-PAD leaks less information than Nym's default configuration (see Table 8), the two most important features, Time and N-gram, still reveal enough information to enable website identification.

Nym+FRONT, provides a significant improvement in protection, reducing both latency and bandwidth overhead. Similarly to Tor, Front on Nym effectively reduces the information leaked by all feature sets. Yet, it does not fully remove information from Time and N-gram features. With an F1 score of 0.63, it still leaves Nym vulnerable to website fingerprinting.

6.2 WTF4Nym

While analyzing the impact of applying WTF-PAD to Nym we observe that its burst detection algorithm does not work very well on Nym traffic. This is because the continuous cover traffic is not accounted for in the state machine underlying WTF-PAD.

To address this issue, we design WTF4Nym, a new defense with an architecture similar to WTF-PAD, in which the entry gateway sends bursts of traffic to the Nym proxy. As discussed in Section 3, we do not consider adversaries that control the entry gateway, since in that case the gateway has the same view of the traffic as it would without the defense. We design WTF4Nym to account for the loop cover traffic the proxy sends to hide ongoing traffic when detecting bursts.

An effective burst-based defense requires knowledge of three parameters to be able to create bursts that are indistinguishable from real bursts, and therefore cannot be discarded trivially by the adversary: the inter-burst time, to initiate fake bursts at intervals that reflect genuine traffic patterns; and the burst duration together with the packet inter-arrival time, so that it can create bursts that match the volume and temporal characteristics of real bursts.

Burst definition. To learn the parameters of bursts in real traces, we first divide captures into 50 ms time windows and compute the total number of bytes within each window. Each window value is a data point that the burst detection algorithm uses as input to identify bursts.

We define an incoming burst as a continuous period of time for which the quantity of incoming bytes is above a pre-defined burst threshold. For a given trace, we calculate the burst threshold θ as follows:

$$\theta = \mu + k \times \text{MAD}$$

where μ is the median number of bytes per 50 ms in the trace, k is a scaling factor, and MAD is the median absolute deviation. MAD quantifies how much the byte count of each time window deviates from the median μ. We experimented with scaling factors k from 2 to 7 and found that using a factor of 4 provided the best trade-off between capturing true bursts and avoiding false positives caused by noise.

We illustrate the meaning of the threshold in Figure 4, where we depict the incoming bytes for https://nym.com/. Each point in this trace represents the number of bytes observed in a 50 ms time window. In this trace, the median number of bytes per 50 ms window is 6 kB, and the burst threshold is $\theta = 23$ kB. Consecutive points exceeding the threshold (marked with colored ×) are considered

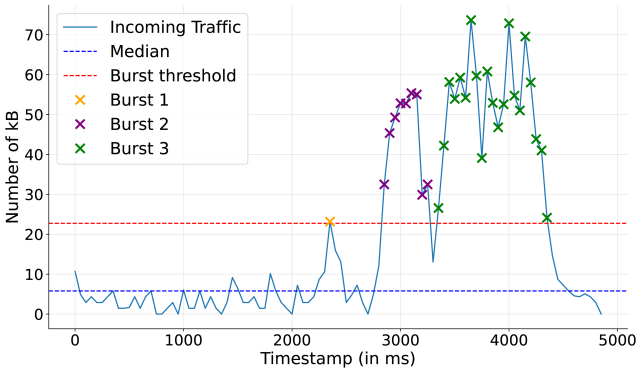


Figure 4: Incoming traffic bytes for "https://nym.com/" with Nym Testnet. The incoming traffic is grouped into 50 ms time windows. In this graph, there are 3 distinct bursts, defined as sequences of consecutive time windows where the incoming traffic exceeds the burst threshold θ, calculated with $k = 4$. For this trace, the median incoming traffic is $\mu = 6$ kB, and the burst threshold is $\theta = 23$ kB.

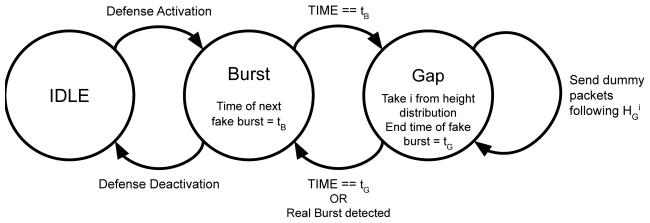


Figure 5: Finite State Machine Algorithm for WTF4Nym in the Nym Gateway. The diagram displays the state transitions of the WTF4Nym defense mechanism. When a proxy starts a connection to the Nym gateway, the defense mechanism is activated, it stops when the download of the page finishes. After a random time t_B, drawn from the inter-burst time histogram, the Gap mode (burst creation) is activated, triggering the transmission of dummy packets based on the height of the burst i and the inter-arrival time histogram (H_{G_i}). The system remains in the burst state until t_G, when either the burst concludes or a real burst is detected, leading to go back to Burst mode and draw a new t_B.

part of a burst. In this trace there are three bursts, one very small (orange), one mid-sized (purple), and one large (green).

Learning the parameters. We learn the distribution of burst duration, inter-burst time, and packet inter-arrival time using a frequentist approach, where we compute histograms of these three parameters based on our dataset. To build these histograms, we analyze additional captures of websites in our dataset which we refer to as the learning set. This dataset is disjoint from the training and testing sets. For each capture in the learning set, we calculate the burst threshold, and identify distinct bursts. Then, we compute the total burst duration, the inter-arrival times of packets within each burst, and the inter-burst delay time (if more than one burst is present in the trace). We use these values to create the histograms to learn the burst parameters.

We observed that inter-arrival times during a burst are much smaller than those in Tor [25], by a factor of 1000. This is due to the constant traffic generated by Nym's loop cover traffic. When using this histogram, the fake bursts created contain a lot of packets and are much larger (in bytes) than real bursts. As a result, they not only introduce significant overhead but also become easily distinguishable by deep learning attacks such as Tik-Tok.

To address this issue, instead of a single histogram, we create multiple inter-arrival time histograms, denoted as H_G^i, one per maximum burst height i, where the maximum burst height refers to the highest number of packets received within any given time window during the burst. Additionally, we compute a histogram of maximum burst heights on learning set to represent the distribution of burst maximum peaks in real traffic.

Finally, we compute the distribution of burst duration across all traces in the learning set; and we calculate the overall burst threshold as the median of all θ in the learning set. This overall threshold is what we use to detect whether a real burst is occurring when running the defense.

Creation of a burst. To create bursts according to WTF4Nym, the gateway implements the state machine described in Figure 5.

The gateway starts in IDLE mode, during which it remains inactive. When the user browses a website, the proxy sends an activation message to the entry gateway. The proxy uses a loop cover traffic packet for this purpose so that the activation message cannot be detected by the adversary. When the gateway receives the activation message, the gateway moves from the IDLE to Burst Mode.

In Burst Mode, the gateway draws a random value t_B from the inter-burst distribution to determine when it switches to Gap Mode.

In Gap Mode, the gateway first determines the maximum burst height i and separately selects the burst duration, both drawn from previously learned distributions. The gateway generates cover traffic by repeatedly sampling an inter-packet interval from H_{G_i} and transmitting a dummy packet as each interval elapses.

At any time during Gap mode, if the gateway detects an incoming burst of real data directed to the proxy in the last 50 ms (i.e., a stream of traffic above the threshold learned on the learning sets) it exits Gap mode and immediately schedules a new duration for the next fake burst. If no real burst happens by the time the fake burst ends, the system returns to Burst mode and schedules the next false burst by drawing a new sample t_B from the inter-burst distribution.

When the webpage is fully loaded, the proxy sends a deactivation message to the gateway, which returns to IDLE mode.

In Figure 6, we present the results of applying WTF4Nym to the undefended capture. The original bursts are preserved, and the newly introduced bursts look like real ones, obfuscating the traffic without adding extra latency.

Evaluation. We evaluate WTF4Nym under Nym's default configuration. To learn the parameters, we collect a new learning set of 11,600 captures from the Testnet list using Nym's default configuration. We collected this eight months later than the training and testing sets, showing that it is not necessary to constantly update

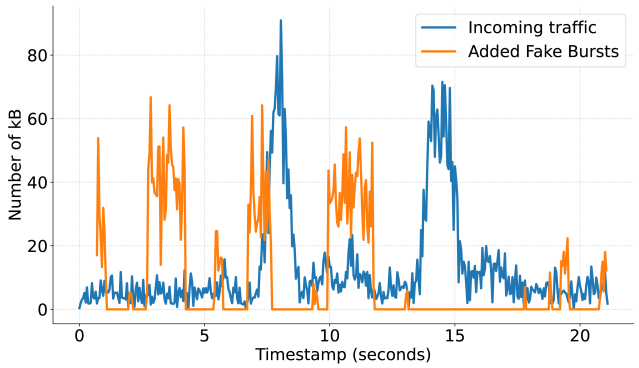


Figure 6: Incoming Bytes count for the query of https://nym.com/ with Nym (in blue) with the bursts added by WTF4Nym (in orange).

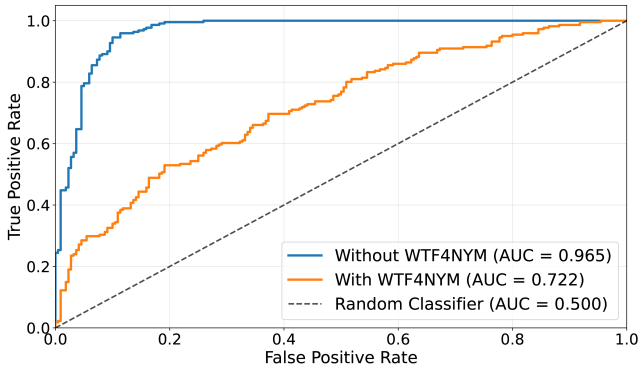


Figure 7: ROC curves (TPR vs. FPR) for the traffic-correlation attack without defenses and with the WTF4Nym defense. The WTF4Nym defense shifts the curve towards the lower-right, indicating a reduced attack success rate, especially at low FPR values.

these histograms to maintain performance. For evaluation we simulate WTF4Nym on the testing set collected using the Reduced List of the Testnet.

As shown in Table 7, the WTF4Nym defense yields a substantial gain in protection, with an F1 score to 0.39. This is 48 percentage points less than Nym with its default configuration, with a similar latency overhead and moderate increase in bandwidth. We also evaluate an implementation of WTF4Nym in Appendix D.

Impact of WTF4Nym on traffic correlation attacks. WTF4Nym departs from the current protection mechanisms in Nym to hide patterns specific to web traffic. To understand whether its benefits extend beyond website fingerprinting protection, we evaluated its impact on flow correlation attacks.

In a flow correlation attack, a passive adversary that observes traffic in the links between the endpoints and the anonymity network tries to determine which endpoints are communicating with each other. We implemented an attack using the deep learning model of Oldenburg et al. [32], which operates directly on packet-level traffic rather than on Sphinx packets. In the experiments of Oldenburg et al. [32], the adversary monitored the Sphinx packets at the entry and exit gateways. In contrast, our implementation considers an attacker who observes both ends of the Nym network: one vantage point between the proxy and the entry gateway, and another between the network requester and the targeted web server. As for Nym defenses (see Section 3), in our attack implementation, we do not consider adversaries with the same vantage point as the entry gateway, since their network visibility would be equivalent with or without the defense. Unlike the approach of Oldenburg et al. [32], we did not modify the code of the Nym gateways, making our experiment closer to what a powerful adversary (e.g., an ISP or government), without access to a gateway, could realistically monitor.

The correlation model takes as input traffic captures from both ends of the connection. We process these captures using the same feature extraction method described in Section 4.1. We then apply the machine learning model in Oldenburg et al. [32] and consider the classification outcome as correct when a capture from the network requester is paired with a capture from the proxy side corresponding to the same website.

We compare the performance of the traffic correlation attack using the current Nym configuration, and when applying WTF4Nym. We conducted all experiments in Testnet using Reduced List. We show the result in Figure 7. As Oldenburg et al. [32], we find that traffic correlation attack is highly effective on vanilla Nym, with an AUC of 0.965: traffic patterns can be reliably matched between proxy and requester connections. With WTF4Nym the AUC is reduced to 0.722, showing that this defense also improves protection against traffic correlation attacks.

6.3 POISS-OFF

One main reason why Nym offers only limited protection against website fingerprinting, despite multiple metadata obfuscation mechanisms, is that the network requester does not send its loop cover traffic to the Nym proxy. As a result, incoming data remains unmasked on the gateway-proxy link, which is precisely where the WF adversary can observe traffic.

We design another defense, POISS-OFF, which runs a Poisson flow of traffic between the network requester and the proxy, but only while the proxy is actively browsing a website in order to be efficient. Outside of these browsing periods, neither the proxy nor the network requester transmits dummy traffic.

To initiate the sending of dummy traffic, the proxy sends a signaling message to the requester. For the dummy traffic sent by the requester to arrive at the proxy, we modified the loop cover packets to resemble the drop cover packets in the Loopix design [36]. Instead of looping back to the sender, a Nym client chosen by the packet sender drops the packet.

To hide the total time of a capture, inspired by Tamaraw [4], the network requester pads the download with dummy packets to a multiple of σ seconds. The parameter σ trades off protection and bandwidth overhead: a longer σ results in more websites having the same padded download time, but increases bandwidth consumption. Yet, the time to load the webpage remains unchanged.

Evaluation. We evaluate POISS-OFF on top of configuration 10 from Table 4, with an inter-message delay on the proxy side of

λ_{Q_P} = 20 ms, as in Nym's default configuration, and λ_{Q_R} = 10 ms on the network requester side. This asymmetric choice follows the design of Tamaraw [4], whose optimal configuration also applies a lower inter-packet delay on the proxy side than on the network requester side.

We configure POISS-OFF with σ = 7 s, which provides the best protection vs. bandwidth overhead trade-off compared to other σ values between 0 s and 20 s. Values of σ smaller than 7 resulted in less uniformity among padded download times, reducing anonymity; while higher values increased bandwidth consumption without significant additional privacy gains.

POISS-OFF minimizes information leakage and achieves an F1 score of 0.06, comparable to the results obtained with Tamaraw on Tor. However, due to the larger size of Nym packets compared to Tor cells, POISS-OFF results in significantly higher bandwidth overhead than Tamaraw. Therefore, POISS-OFF can also be thought of as an ideal case of using Nym's Poisson rate with cover traffic from end-to-end, but the trade-offs in WTF4Nym could be considered more optimal for real-world web-browsing.

6.4 Implementing defenses on Nym

SURBs. Since the beginning of our experiments, Nym has introduced single-use response blocks (SURBs) to enable anonymous responses without revealing the sender's address. This change does not affect the effectiveness of our proposed defenses.

For Nym+WTF-PAD, Nym+Front, and WTF4Nym, packet transmission is independent of the reply routing mechanism, since the added flow is created between the entry gateway and the proxy, so the deployment of SURBs does not impact these defenses.

The POISS-OFF defense is also compatible with SURB-based reply routing. In Nym, three Sphinx packets can contain up to 10 SURBs. Empirical measurements show that POISS-OFF generates approximately 40 packets per second in the form of cover traffic. This value corresponds to the measured difference between the outgoing packet rate under POISS-OFF and the equivalent configuration without cover traffic (configuration 12). These unused packets can be repurposed to deliver SURBs. At this rate, the system can provide roughly 130 SURBs per second to the network requester through unused packets. Since the network requester must send about 100 packets per second to the proxy (with λ_{Q_R} = 10 ms), this provisioning rate is sufficient to support all responses during the browsing session. The network requester can store the remaining 30 SURBs per second for later use within their 24-hour validity period.

7 Conclusion

The Nym mixnet, based on the low-latency Loopix design, implements obfuscation mechanisms that are specialized to provide anonymity in asynchronous message-based communication such as instant messaging and cryptocurrency transactions. In this paper, we study whether those mechanisms can also protect against website fingerprinting attacks when Nym is used to browse the Web.

Our results show that Nym's defense mechanisms, that have been proven extremely effective at protecting anonymity in messaging, are weak when it comes to hiding the burst-oriented nature of web traffic. On one hand, mix delays, which are effective in breaking the sender-receiver relationship in messaging [36], and at mitigating the accuracy of correlation attacks [32], are counterproductive in website fingerprinting scenarios. In website fingerprinting, delays introduce a separation between incoming and outgoing traffic, unintentionally making website fingerprints more stable across multiple visits. On the other hand, Nym's cover traffic reduces the information available to the adversary, but cannot hide the characteristics of bursts that make websites' traffic identifiable.

These results highlight critical limitations of the default strategies employed by the Nym mixnet when it comes to protecting web traffic. To address these limitations, we have proposed two defenses designed specifically to mask incoming traffic patterns in Nym's architecture. Equipped with these defenses, Nym can offer improved website fingerprint protection, although this comes with a moderate increase in overhead. Our defenses implement a possible cover-traffic strategy tuned to deal with the bursty nature of web traffic. As a result of our work, Nym is implementing both WTF4Nym and POISS-OFF in the Nym network, targeted for the first half of 2026. Future work should explore whether different noise strategies on the link between the proxy and the gateway can provide a better protection vs. overhead trade-off; and whether splitting outgoing traffic among different gateways [10, 41] could bring further protection.

Our work demonstrates that traffic analysis defenses do not necessarily transfer well across applications, even when shown to be secure in one particular scenario. Yet, we also show that mixnets, which have been traditionally relegated to only be used in messaging, can become anonymous browsing tools if they implement tailored protection mechanisms. However, it is yet to be seen whether there exists a mechanism that can, at reasonable cost, provide good privacy protection to traffic across a wide range of applications with different traffic types. We hope that our results inspire others to work in this direction so that next-generation anonymous communications networks can offer strong protection for all user needs.

References

[1] Kota Abe and Shigeki Goto. 2016. Fingerprinting attack on Tor anonymity using deep learning. In *Proceedings of the Asia-Pacific Advanced Network*.

[2] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. In *Proceedings on Privacy Enhancing Technologies*.

[3] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*.

[4] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.

[5] Sambuddho Chakravarty, Marco V. Barbera, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. 2014. On the Effectiveness of Traffic Analysis against Anonymity Networks Using Flow Records. In *Passive and Active Measurement*.

[6] David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* (1981).

[7] George Danezis and Ian Goldberg. 2009. Sphinx: A Compact and Provably Secure Mix Format. In *2009 IEEE Symposium on Security and Privacy (SP)*.

[8] George Danezis and Len Sassaman. 2003. Heartbeat traffic to counter (n-1) attacks: red-green-black mixes. In *Proceedings of the 2003 on Workshop on Privacy in the Electronic Society*.

[9] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2018. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two. In *2018 IEEE Symposium on Security and Privacy (SP)*.

[10] Wladimir De La Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2020. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*.

[11] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. 2021. The Nym Network: The Next Generation of Privacy Infrastructure. (2021). https://nymtech.net/nym-whitepaper.pdf

[12] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*.

[13] Roger Dingledine, Vitaly Shmatikov, and Paul F. Syverson. 2004. Synchronous Batching: From Cascades to Free Routes. In *Privacy Enhancing Technologies*.

[14] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *2012 IEEE Symposium on Security and Privacy (SP)*.

[15] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. In *USENIX Security Symposium*.

[16] Benjamin Greschbach, Tobias Pulls, Laura M. Roberts, Philipp Winter, and Nick Feamster. 2017. The Effect of DNS on Tor's Anonymity. In *NDSS Symposium 2017*.

[17] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *USENIX Security Symposium*.

[18] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Cloud Computing Security Workshop*.

[19] Andrew Hintz. 2002. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies*.

[20] Daniel Hugenroth and Alastair R. Beresford. 2023. Powering Privacy: On the Energy Demand and Feasibility of Anonymity Networks on Smartphones. In *USENIX Security Symposium*.

[21] Alfonso Iacovazzi, Sanat Sarda, and Yuval Elovici. 2018. Inflow: Inverse Network Flow Watermarking for Detecting Hidden Servers. In *INFOCOM 2018 - IEEE Conference on Computer Communications*.

[22] Rob Jansen, Justin Tracey, and Ian Goldberg. 2021. Once is Never Enough: Foundations for Sound Statistical Inference in Tor Network Experimentation. In *USENIX Security Symposium*.

[23] Rob Jansen and Ryan Wails. 2023. Data-Explainable Website Fingerprinting with Network Simulation. In *Proceedings on Privacy Enhancing Technologies*.

[24] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.

[25] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. 2016. Toward an Efficient Website Fingerprinting Defense. In *Computer Security – ESORICS 2016*.

[26] Dogan Kesdogan, Jan Egner, and Roland Büschkes. 1998. Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System. In *Information Hiding*.

[27] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring Information Leakage in Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.

[28] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. 2011. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *NDSS Symposium 2011*.

[29] Nate Mathews, James K. Holland, Nicholas Hopper, and Matthew Wright. 2024. Laserbeak: Evolving Website Fingerprinting Attacks With Attention and Multi-Channel Feature Representation. In *Transactions on Information Forensics and Security*.

[30] Nate Mathews, James K. Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2023. SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses. In *Symposium on Security and Privacy*.

[31] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.

[32] Lennart Oldenburg, Marc Juarez, Enrique Argones Rúa, and Claudia Diaz Martinez. 2024. MixMatch: Flow Matching for Mixnet Traffic. In *Proceedings on Privacy Enhancing Technologies*.

[33] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. In *Network and Distributed System Security Symposium*.

[34] Andriy Panchenko, Asya Mitseva, Martin Henze, Fabian Lanze, Klaus Wehrle, and Thomas Engel. 2017. Analysis of Fingerprinting Techniques for Tor Hidden Services. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*.

[35] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 2011 on Workshop on Privacy in the Electronic Society*.

[36] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *USENIX Security Symposium*.

[37] Tobias Pulls and Rasmus Dahlberg. 2020. Website Fingerprinting with Website Oracles. In *Proceedings on Privacy Enhancing Technologies*.

[38] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. In *Proceedings on Privacy Enhancing Technologies*.

[39] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *NDSS Symposium 2018*.

[40] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.

[41] Mona Wang, Anunay Kulshrestha, Liang Wang, and Prateek Mittal. 2022. Leveraging strategic connection migration-powered traffic splitting for privacy. *Proceedings on Privacy Enhancing Technologies*.

[42] Charles V. Wright, Scott E. Coull, and Fabian Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *NDSS Symposium 2009*.

Acknowledgments

This work was partially funded by the Swiss National Science Foundation under the MinWeb project (grant 10000923). The SPRING lab lead by Prof. Troncoso received an unrestricted gift from Nym in 2022. The authors used generative AI-based tools to revise the text, improve flow and correct typos, grammatical errors, and awkward phrasing.

Appendices

A List of websites

We list the websites used in our experiments in Table 10. We started with the complete list in the BigEnough dataset [30]. Some websites blocked Tor or Nym connections (shown in red), resulting in a reduced-size Full List (blue and green). The Reduced List (blue) is even smaller, as some sites restricted access to content over time, while others had excessively long load times for certain Nym settings, creating timeouts that prevented us from capturing traffic.

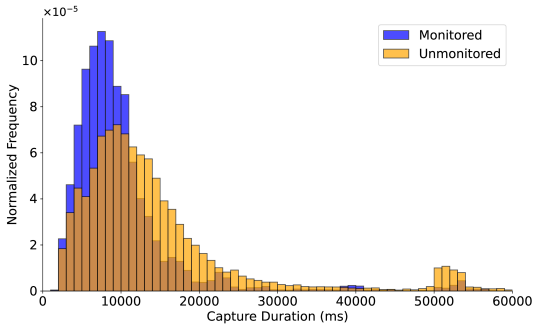
The creators of the BigEnough dataset selected the 100 websites in the monitored set among the most popular sites based on rankings from the Open PageRank Initiative. The unmonitored dataset includes 19,000 unrelated index pages randomly drawn from the remaining highest-ranked websites.

This "arbitrary" choice actually results on an advantage of the classifier in the open-world setting. This is because the monitored and unmonitored dataset show different characteristics in their temporal distribution that makes them distinguishable. We illustrate this difference in Figure 8a, where we see that captures from the monitored set in the Reduced List (in blue) have a shorter overall capture time than the unmonitored set (in orange).

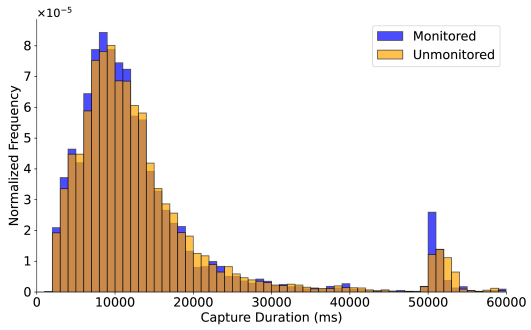
To demonstrate that this difference is important, we conduct an experiment with a new list of monitored websites, that we call Fair List. This monitored dataset comprises 580 websites chosen at random from the Reduced List, each with 20 captures. The background set contains elements from the original unmonitored list from the BigEnough dataset that share the same temporal distribution. We report the results in Table 9, where we see that when the monitored and non-monitored list follow the same time distribution, the F1-score and all information leakage diminishes significantly in comparison to the Reduced List.

List of URL	Reduced List			Fair List	
Scenario	Tor	Nym Mainnet	Nym Testnet	Tor	Nym Testnet
F1-score	0.92	0.73	0.87	0.77	0.61
Pkt. Count	60.6	60.2	72.9	53.6	56.6
Time	70.7	64.9	78.1	57.4	60.4
Inter-arrival Time	79.1	61.7	74.7	63.7	59.1
N-gram	75.9	64.6	78.8	62.4	64.5
Transposition	80.8	50.2	71.5	63.3	56.9
Bursts	71.8	62.1	75.2	61.3	57.6

Table 9: F1 scores and Information Leakage for the open-world scenario: Top-1 accuracy (in %) with RF trained using the Information Leakage features individually in the open-world scenario. Cells are colored according to the severity of leakage as follows: red for $x > 70\%$, orange for $60\% < x \leq 70\%$ and yellow for $50\% < x \leq 60\%$.



(a) Reduced List time distribution



(b) Fair List time distribution

Figure 8: Comparison of time distributions between monitored and unmonitored captures in the Testnet

Future work should focus on building monitored and unmonitored sets that better reflect the choices an adversary would make to produce more meaningful results.

B Setup limitation

Our preliminary experiments on the Nym Mainnet showed practical issues that made it unsuitable for controlled WF studies. As also reported by Oldenburg et al. [32], we observed an unusually high packet retransmission rate (approximately 5% of packets), which significantly increased page load latency. These retransmissions caused unpredictable delays and packet reordering, making the system much slower than it should be (see Appendix C). While such instability may accidentally reduce the accuracy of the WF, it also makes systematic evaluation impossible, as network conditions vary from run to run and cannot be reproduced.

To overcome these constraints, we adopted an isolated Testnet environment, following the methodology of Oldenburg et al. [32]. This setup allowed us to precisely control per-mix delays, enable or disable cover traffic at will, and ensure loss-free transmission.

addtoany.com	gov.uk	soundcloud.com
adobe.com	gravatar.com	sourceforge.net
amazon.com	harvard.edu	spotify.com
apache.org	huffingtonpost.com	springer.com
apple.com	ibm.com	stanford.edu
baidu.com	imdb.com	surveymonkey.com
bbc.co.uk	imgur.com	techcrunch.com
berkeley.edu	instagram.com	ted.com
bing.com	issuu.com	telegraph.co.uk
bitly.com	latimes.com	theguardian.com
blogger.com	linkedin.com	theverge.com
bloomberg.com	medium.com	time.com
cdc.gov	microsoft.com	tumblr.com
cnet.com	mit.edu	twitter.com
cnn.com	mozilla.org	un.org
creativecommons.org	msn.com	unsplash.com
dailymail.co.uk	myspace.com	usatoday.com
datatracker.ietf.org	mysql.com	vimeo.com
debian.org	nationalgeographic.com	vk.com
dropbox.com	nature.com	w3.org
ebay.com	nih.gov	washingtonpost.com
etsy.com	npr.org	weebly.com
europa.eu	nytimes.com	weibo.com
eventbrite.com	office.com	who.int
facebook.com	opera.com	wikipedia.org
flickr.com	paypal.com	wired.com
forbes.com	php.net	wix.com
foxnews.com	pinterest.com	wordpress.org
free.fr	qq.com	wsj.com
gc.ca	reddit.com	yahoo.com
github.com	researchgate.net	yelp.com
gnu.org	reuters.com	youtube.com
google.com	slideshare.net	zoom.us

Table 10: Mathews et al. [30] selected websites for BigEnough. Blue identifies websites appearing in both Reduced List and Full List experiments. Green highlights websites also included in Full List. Red shows websites not captured in our dataset but present in the original BigEnough [30] list.

We did not use some simulation-based framework for our captures, because none of them provided real browser interactions with access to real websites [23]. By using our setup, we preserve realistic traffic patterns while retaining experimental control.

However, the Testnet also has limitations that may affect the generalization of our findings. It contains only six mixnodes in two geographic regions (Germany and the UK), operating under stable conditions with no other background traffic. This reduces natural network jitter and packet reordering factors that in real deployments can obscure traffic patterns. As a result, classifiers may benefit from more consistent traces, which can exaggerate the effectiveness of WF attacks. This concern echoes Jansen et al. [22], which warns against the risk of bias and overfitting in results obtained from a single static snapshot of the network.

Since our Testnet setup remains highly idealized and may not reflect real-world variability, we investigated a complementary baseline for comparison. To this end, we launched Tor nodes on the same machines as the Nym nodes in our lab experiments and captured traffic while browsing the websites from Reduced List. We call this configuration Tor Testnet. In this scenario, there is no difference between Tor and Nym Testnet in terms of network conditions or node load (as in our Nym Testnet experiments, Tor Testnet nodes carry only our traffic).

In our captures from the main Tor network, the path changes with each capture, resulting in high variability: sessions can be very short or very long depending on the circuit taken. On the Tor Testnet, paths are identical and contains only nodes in Germany and the UK, resulting in much lower latency and more predictable trace lengths. Under these controlled conditions, compared to the main Tor network, Tor Testnet latency decreases due to the geographical proximity of the nodes (LO = 0.29), while bandwidth overhead remains unchanged (0.54). The F1 score of Tik-Tok increases to 0.96, reflecting the resulting consistency of captures.

This comparison highlights both the strengths and weaknesses of our methodology. On the one hand, Tor Testnet confirms that our experimental framework produces results consistent with Tor's known behavior. On the other hand, the artificially stable conditions of Tor Testnet and Nym Testnet may inflate attack performance by eliminating the natural variability present in real-world deployments. Thus, while our findings provide valuable insights under controlled conditions, they should be interpreted with caution.

C Timing analysis

As a complement to the bandwidth overhead discussed in Section 4.3, we also report the distribution of page load times across different configurations (see Figure 9). Tor and Nym Testnet have similar load time patterns, which shows that both operate in stable network conditions. Nym Mainnet shows more variation, taking longer to load pages. This likely results from retransmissions and network dynamics discussed in Appendix B. The POISS-OFF defense leads to noticeably higher latency compared to WTF4Nym, reflecting its design choice to prioritize privacy over speed.

D WTF4Nym implementation

The results presented in Section 6.2 are obtained by simulating the defenses on top of the original captures without protection.

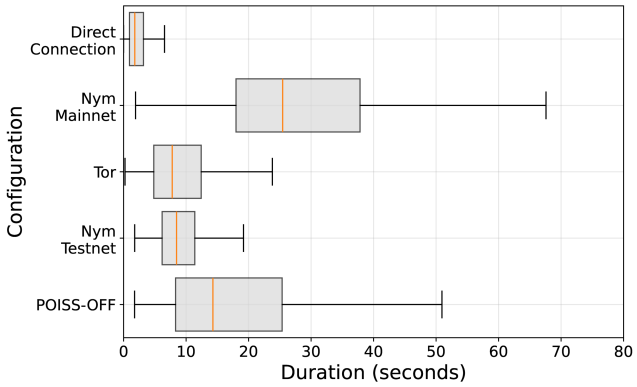


Figure 9: Distribution of website capture (Reduced List) durations across different privacy network configurations. Each box shows the inter-quartile range (25th-75th percentile), with the median indicated by the central line. Whiskers extend to the most extreme data showing the 5th and 95th percentiles.

This approach is common in prior work to evaluate defense performance [15, 30]. To assess how closely these simulations reflect real network conditions, we added a proof-of-concept WTF4Nym implementation to the entry gateway of our Testnet. We run a Python script on the entry gateway that sends bursts to the proxy according to the WTF4Nym parameters. The use of an external script changes the packet delays of the website, so we recalculated WTF4Nym parameters on the modified histograms. We then collect traces and run the attack again. The resulting F1 score (0.37) and latency overhead (LO = 3.68) match those observed in the simulation.

In our proof-of-concept, the defense is implemented by an external Python script that injects packets independently of the Nym process, over a separate TCP socket. This setup lacks fine-grained timing control. In particular, we notice that we cannot enforce precise delays between packets in Python when they are below 3 ms, resulting in bursts with reduced amplitude. In order to reach burst amplitudes at least as high as in the simulation, our script sends packets without inter-packet delay when the chosen one is below 3 ms. This limitation results in a higher bandwidth overhead (BO = 5.52), since the absence of fine-grained inter-packet delays causes some bursts to complete more quickly, leading to more bursts, and thus to a higher total number of packets. We regard this as a limitation of our prototype implementation rather than an inherent property of the defense. Given these results, we conclude that WTF4Nym can operate as intended, but further integration within the Nym gateway process is necessary to reduce the bandwidth overhead and enable finer timing control.