MixMatch: Flow Matching for Mixnet Traffic

Lennart Oldenburg
COSIC, KU Leuven
lennart.oldenburg@esat.kuleuven.be

Marc Juarez
School of Informatics, University of Edinburgh
marc.juarez@ed.ac.uk

Enrique Argones Rúa
COSIC, KU Leuven
enrique.argonesrua@esat.kuleuven.be

Claudia Diaz
COSIC, KU Leuven, and Nym Technologies, SA
cdiaz@esat.kuleuven.be

ABSTRACT

Mixnets provide communication anonymity against network adversaries by routing packets independently via multiple hops, delaying them artificially at each hop, and introducing cover traffic. We show that these features (particularly the use of cover traffic) significantly diminish the effectiveness of state-of-the-art flow correlation techniques developed to link the two ends of a Tor connection. In this work, we propose novel methods to determine whether a set of endpoints exchanges packets via a mixnet and demonstrate their effectiveness by applying them to the Nym mixnet. We consider Nym in both an *idealized lab setup* and the official *live network*, and propose and compare three classifiers to conduct *flow matching* on it. Our *statistical* classifier tests whether egress packet timestamps are consistent with ingress timestamps and the (known) routing delay characteristic of the mixnet. In contrast, our two *deep learning* (DL) classifiers learn to distinguish matched from unmatched flow pairs from collected datasets directly, rather than relying on priors that describe the delay distribution. All three classifiers use our *flow merging* technique, which enables testing a match for sets of communicating endpoints of any cardinality. Considering a use case where two observed endpoints communicate exclusively to exchange a file through Nym, we find that flow matching is fast and accurate in the idealized lab setup. If flow pairs are aligned using all network observations in a download, we achieve a TPR of circa 0.6 (DL) and 0.47 (statistical) at an FPR of 10^{-2} after only processing 100 observations. We evaluate classifier performance under key variations of this setup: the absence of loop cover traffic, an increased or decreased average per-mix delay, larger communicating sets (three endpoints) with faster responders, and the presence of realistic network effects (live network). The classifiers' matching performance diminishes on the live network where packet losses and variable propagation delays exist, reducing DL TPR to circa 0.26 and statistical TPR to circa 0.28 at an FPR of 10^{-2}. Informed by the insights of our analyses, we outline countermeasures that can be deployed in mixnets such as Nym to mitigate flow matching threats.

KEYWORDS

mixnets, anonymity, flow correlation, flow matching, deep learning

Proceedings on Privacy Enhancing Technologies 2024(2), 276–294
© 2024 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2024-0050

1 INTRODUCTION

With every packet we send or receive over the Internet, we generate *metadata*—data *about* the packets, such as timestamp, size, and direction. If left unprotected, an adversary able to collect and analyze these metadata can determine who is talking to whom, enabling privacy-invasive insights into our increasingly online lives. *Anonymous communication networks* such as *mixnets* [6] aim to protect metadata even against adversaries observing all network links, e.g., by artificially delaying packets and shaping traffic to be of constant bandwidth. Mixnets provide strong protection to individual packets, which are hard to trace as they are routed through the network. Once two endpoints exchange a large number of packets in a short period of time, however, an adversary may be able to match the flows of packets entering and leaving the network. While this issue has long been researched in the context of onion routing networks such as Tor, the extent to which mixnets may be vulnerable to this threat remains an open question.

Two recent developments bring renewed urgency to this question. First, the recent deployment of *Nym* [10, 22], a mixnet that aims to provide a general-purpose anonymous routing layer. The Nym mixnet is based on Loopix [25] and as such relies on source routing, continuous-time mixes, and a packet sending schedule following a Poisson process and where clients send "loop" cover packets to themselves in addition to the payload packets transmitted to others. Second, researchers have recently shown that adversaries employing *deep learning methods* are effective at linking the endpoints engaged in packet exchanges through Tor [11], the longest-running and most popular anonymity network. DeepCoFFEA [23] and its predecessor DeepCorr [19] strikingly demonstrate how well supervised learning methods are able to correlate network flows despite obfuscation from having been routed through Tor. While Tor is not designed to prevent the correlation of flows by an adversary observing both ends of a connection [35], this attack is squarely within the threat model of mixnets. Additionally, when mixnets like Nym use fixed packet sizes that enable general-purpose use, modern application workloads (e.g., email, messaging, web browsing), their data-intensive protocol parts (e.g., authentication, encryption), and interactivity can quickly produce large flows—prompting us to revisit the threat of matching flows on mixnets.

In this work, we develop novel methods for evaluating flow matching in mixnets and empirically investigate their effectiveness on the deployed Nym mixnet. Flow matching is a special case of *flow correlation* where sets of endpoints communicate *exclusively* among themselves (*endpoints closed set condition*), such that any packet sent by an endpoint to the mixnet is received by itself or another endpoint from its set. We design three classifiers for this flow matching setting that consider an adversary who observes communications between the endpoints and the mixnet (in the case of Nym, this can be achieved by running *gateways*). Through our *flow merging* operation (Section 3.3), we accommodate Nym's loop cover traffic in our model. Our *statistical classifier* (Section 4) uses the fact that the drift between ingress and egress timestamps follows a stationary distribution only for matched flows (but deviates for unmatched flows). We also propose two *deep learning classifiers* (Section 5), called *drift* and *shape*, that automatically extract the relevant information from the training data. Our drift network exploits the temporal drift between egress and ingress packets similar to how our statistical classifier does, while the shape network follows DeepCoFFEA's approach in learning embeddings where matched inter-arrival sequences are located closer to each other than unmatched sequences.

We evaluate our classifiers' performance on six datasets that represent distinct setups (Section 6). This allows us to identify and assess key factors in the effectiveness of Nym's metadata protection (Section 7). Dataset `baseline` captures file download trace pairs across default Nym over an idealized network (near-instant and loss-free transmissions). After aligning using all observations of a completed download and processing the first 100 observations per flow, our statistical and our drift classifier are fast and accurate in matching flows here (circa 0.47 and 0.6 TPR at 10^{-2} FPR, respectively), while our shape model lags behind at about 0.26 TPR. We examine flow matching in the absence of loop cover traffic (dataset `no-cover`, idealized network) and find that the effectiveness increases further to an FPR slightly higher than 10^{-6} for the same TPR. Reducing the average delay μ that packets stay at a mix for (`low-delay`, idealized network) increases TPR to 1.0 at 10^{-2} FPR, while increasing μ (`high-delay`, idealized network) leads TPR to fall to circa 0.13. Our simulated dataset `two-to-one` (idealized network) shows that our classifiers retain utility even when positive and negative samples share endpoints in larger sets. The sixth dataset (`live-nym`, real-world network) evaluates the threat in the deployed mainnet Nym in default configuration. Network effects like packet losses violate our classifiers' assumptions and cause performance to degrade significantly: roughly 0.28 (statistical), 0.26 (drift), and 0.03 (shape) TPR at 10^{-2} FPR. We outline countermeasures to our attacks and highlight our approach's shortcomings (Section 8). We summarize our main contributions as follows:

- We propose a *flow merging* technique that enables evaluating flow matching in mixnets, given a pair (or closed set) of endpoints communicating exclusively with each other.
- We empirically assess flow matching in different settings of state-of-the-art mixnet Nym. We examine three classifiers (statistical, drift, shape) that can be used after applying our flow merging technique. We find that all three classifiers perform well in a variety of evaluated Nym configurations, but degrade once communication is not loss-free or exclusive. We propose countermeasures to mitigate flow matching threats based on the insights derived from the analysis.
- We develop a method to collect metadata on Nym in an ethical way as well as six datasets of the same file download scenario in different Nym and network configurations. We make this work's source code and datasets available [1].

2 BACKGROUND AND RELATED WORK

2.1 Flow Correlation Against Tor

Fixed-sized data packets and per-hop cryptographic transformations prevent adversaries observing *both ends* of a packet exchange over connection-based anonymity systems like Tor (we give a brief overview of relevant aspects of Tor in Appendix A) from trivially deanonymizing it based on packet content or size. There remain, however, observable features of traffic flows that can be exploited: the number, direction, and timings of transmitted packets. *Flow correlation* attacks consider a passive adversary that observes traffic in the links between the endpoints and the anonymity network, without having visibility of traffic between network intermediaries. The adversary records traffic features observed in the available flows and tries to determine which endpoints are communicating with each other by analyzing correlations between flows. The susceptibility of connection-based anonymity systems to this type of attack has long been known [4, 12, 26, 27, 32] and, in fact, Tor is explicitly not designed to withstand such an attack [11]. Nonetheless, flow correlation represents a relevant threat to users of these systems and has thus been studied extensively.

We review early flow correlation attacks that rely on traditional statistical methods to identify correlated flows in Appendix B. State-of-the-art methods rely on deep learning techniques, first introduced by DeepCorr [19]. In contrast to other approaches, deep learning methods do not require feature engineering by human experts, but instead extract the dataset's features on their own. This allows to identify useful features potentially indiscernible to humans. DeepCorr processes packet timestamps (as *inter-packet delays*) and packet sizes from both connection ends in a *convolutional neural network* (CNN) architecture. Impressively, for a *false positive rate* (FPR) of 10^{-2}, DeepCorr achieves a *true positive rate* (TPR) of roughly 0.9 compared to circa 0.3 of RAPTOR [31], the previous state-of-the-art attack based on traditional statistical methods.

Oh et al. [23] introduced flow pair correlator DeepCoFFEA, which surpasses DeepCorr's performance and practicality by way of two main improvements. First, DeepCoFFEA transforms both traces of a candidate pair into a low-dimensional space where correlated traces are located closer together (according to a distance metric) than uncorrelated traces, and comparison is fast. This is learned by minimizing a *triplet loss* [15] between embeddings of an initiator trace (*anchor*), a correlated responder trace (*positive*), and an uncorrelated responder trace (*negative*). Second, DeepCoFFEA partitions each trace of a candidate pair into windows and arrives at the pair's correlation decision by aggregating window-by-window embedding distances in order to lower the FPR. As a result, DeepCoFFEA outperforms DeepCorr significantly, with a TPR of roughly 0.98 compared to DeepCorr's circa 0.3 at an FPR close to 10^{-3}.

2.2 Mixnets and Nym

In contrast to onion routing networks like Tor, mixnets [6] aim to provide anonymity against a *global passive adversary* who observes *all* network links, including both ends of a communication, and is thus capable of conducting flow correlation attacks. Mixnets differ in two key ways from onion routing networks. First, mixnets route each packet independently, using fresh cryptographic key material and a randomly chosen path of intermediaries through the network each time. They are thus *packet-oriented* rather than *connection-oriented* (where packets are explicitly associated to a circuit for all intermediaries in the path), meaning that an adversarial intermediary cannot tell whether or not two packets belong to the same flow. Second, packets are artificially delayed at each intermediary (called *mix node*) before being passed on. The delays alter the flow of packets routed by mix nodes so that their inputs and outputs cannot be linked based on packet order, and may also cause packets within a flow to be received out of order. Per-node packet reordering can be achieved in different ways. Many mixnet designs have nodes assemble incoming packets into *batches* before shuffling and forwarding them [5, 6, 8, 14, 18, 36]. On the other hand, *continuous-time mixes* delay packets individually, typically by a duration randomly drawn from an exponential distribution [16, 25], to achieve a similar effect.

The Nym Network. Nym [10, 22] is a recently deployed system based on continuous-time mixnet Loopix [25] intended for general-purpose traffic relaying (e.g., messaging, cryptocurrency access). We use Nym as the reference mixnet for this work as it is a state-of-the-art design and a deployed system with available source code [21]. We analyze Nym in git repository versions `nym-binaries-1.0.2` and `nym-binaries-v1.1.13`, with all described Nym behavior in this work as of the latter version unless otherwise noted. Endpoints communicating via Nym are associated to *gateways* that act as interfaces with the mixnet. Application data sent through Nym is fragmented and padded into one or more Sphinx [9] packets of fixed size (2413 bytes). Packet paths start at the sender's gateway, traverse three randomly selected mixes, and end in the receiver's gateway. For each intermediate mix, the sending endpoint samples a delay from an exponential distribution with mean $\mu = 50$ ms and encodes the value in the packet header, such that the mix will retain the packet for that amount of time before forwarding it along the path. Data packets contain an *acknowledgment* packet (386 bytes in size) that is extracted by the receiving gateway and sent to the mixnet, which routes it back to the sender via a dedicated fresh path independent of its data packet's path. Acknowledgments are delayed at each intermediate mix also following an exponential distribution with mean μ (i.e., 50 ms). Given that the sender selects all the per-mix delays for the packet and the acknowledgment, it can estimate when the acknowledgment should be received. If either the packet or the acknowledgment is lost in transit, the sending endpoint retransmits the data after a timeout.

With the goal of achieving unobservability [24], Nym endpoints regulate the times at which they send packets to their gateways. The emission schedule is modeled as a Poisson process that on average sends 50 packets per second, with inter-packet delay drawn from an exponential distribution with average $\lambda_R = 20$ ms. Each time the emission schedule indicates that a packet should be sent, the endpoint checks if an application-level packet is available for sending, and if so, sends it out. If no application-level packet is ready, the endpoint instead generates a *loop cover packet* with throwaway payload addressed to itself. Loop cover packets are routed like regular packets and are indistinguishable for all entities except the sender. This way, an adversary monitoring the link between endpoint and gateway is unable to tell whether the endpoint is sending any "real", application-level packets or not. In addition to these "gap-filling" loop cover packets, endpoints also send loop cover packets at all

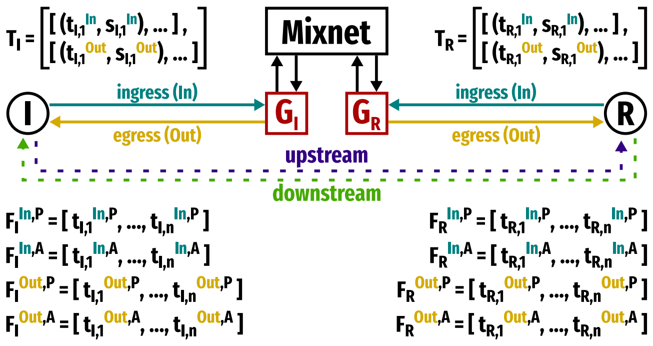

$T_I = \begin{bmatrix} [(t_{I,1}^{In}, s_{I,1}^{In}), \dots], \\ [(t_{I,1}^{Out}, s_{I,1}^{Out}), \dots] \end{bmatrix}$
Mixnet
$T_R = \begin{bmatrix} [(t_{R,1}^{In}, s_{R,1}^{In}), \dots], \\ [(t_{R,1}^{Out}, s_{R,1}^{Out}), \dots] \end{bmatrix}$
I
ingress (In)
egress (Out)
G_I
G_R
ingress (In)
egress (Out)
R
upstream
downstream
$F_I^{In,P} = [t_{I,1}^{In,P}, \dots, t_{I,n}^{In,P}]$
$F_I^{In,A} = [t_{I,1}^{In,A}, \dots, t_{I,n}^{In,A}]$
$F_I^{Out,P} = [t_{I,1}^{Out,P}, \dots, t_{I,n}^{Out,P}]$
$F_I^{Out,A} = [t_{I,1}^{Out,A}, \dots, t_{I,n}^{Out,A}]$
$F_R^{In,P} = [t_{R,1}^{In,P}, \dots, t_{R,n}^{In,P}]$
$F_R^{In,A} = [t_{R,1}^{In,A}, \dots, t_{R,n}^{In,A}]$
$F_R^{Out,P} = [t_{R,1}^{Out,P}, \dots, t_{R,n}^{Out,P}]$
$F_R^{Out,A} = [t_{R,1}^{Out,A}, \dots, t_{R,n}^{Out,A}]$

Figure 1: Traces recorded by an adversary that controls gateways G_I and G_R and wants to determine whether endpoints I and R are communicating.

times (independent of application traffic), according to a second Poisson process with average inter-packet delay $\lambda_C = 200$ ms.

2.3 Flow Correlation Against Mixnets

Danezis [7] presents an attack on the anonymity provided by a simulated network of continuous-time mixes with exponential delay, assuming that packets are associated to flows as they traverse the mixnet. The problem tackled in this work considers a more general and harder attack setting where all packets are independently routed and not associated to flows. We discuss how our results extend prior work in Section 4. Zhu et al. [38] evaluate flow correlation for batching mix strategies considering a single mix, four endpoints, and no cover traffic—a very simplistic setup compared to deployed systems such as Nym. Thus, while prior work on flow correlation against connection-based anonymity systems is plentiful, we currently lack methods to evaluate this threat in realistic mixnet setups that are packet-oriented.

3 FLOW MATCHING ON MIXNETS

3.1 System Model

We consider a set of *endpoints* (clients or network services) that communicate through the Nym mixnet via a *gateway* of their choice. We focus on pairwise communications between two endpoints, but note that our methods apply to closed communication groups that involve any number of endpoints (Section 3.3). As convention to distinguish the two endpoints, we denote the sender of the first packet in the exchange as *initiator* I, while the receiver of that first packet is *responder* R. The packets from I to R are considered to flow *upstream*, while the packets from R to I travel *downstream*. Figure 1 illustrates our abstract system model, where endpoints I and R communicate with each other via their respective gateways G_I and G_R. We call *ingress* the traffic sent from a gateway to the mixnet and *egress* the traffic received by a gateway from the mixnet. We assume that I and R are communicating exclusively with each other and that the same gateways (G_I and G_R) are used during their entire communication session.

In practice, endpoints in Nym connect to gateways via WebSocket, a bidirectional, ordered, message-oriented protocol. We call the bidirectional packet exchange of the WebSocket session between endpoint $E \in \{\mathrm{I},\mathrm{R}\}$ and its gateway G_E a *trace* $T_E = (T_E^{\mathrm{In}}, T_E^{\mathrm{Out}})$, where T_E^{In} and T_E^{Out} denote the sequences of ingress and egress Sphinx packets, respectively. For each packet in these directional sequences, the trace records its timestamp and size (normal packet or acknowledgment). The entire trace T_E observable by G_E as the result of acting as network interface for E is split into four flows. For endpoint E, packet type $M \in \{\mathrm{P},\mathrm{A}\}$, and packet direction $D \in \{\mathrm{In},\mathrm{Out}\}$, let flow $F_E^{D,M}$ denote the sequence of packet timings in T_E of type M and direction D.

3.2 Threat Model

Adversarial Capabilities. We consider an adversary that runs one or more Nym gateways. An adversarial gateway logs the observed traces T_E for all of the endpoints E it interacts with and makes these traces available for adversarial analysis. The attack is *passive*, i.e., malicious gateways do not delay, modify, inject, or drop packets. They follow the packet transmission protocols perfectly and only deviate from honest behavior by logging packet information. We assume the mixnet nodes and endpoints to be *honest*, i.e., they follow the protocols and do not collude with the adversary.

Depending on how endpoints select gateways, it may be easier or harder for the adversary to become the gateway of target endpoints of interest. The adversary may have to gain a target's trust if the gateway selection is manual, wait until it "gets lucky" if gateways are regularly reassigned in a randomized way, or optimize its parameters in case of automated selection that follows criteria such as proximity to the target, uptime, or server capacity. Regardless of the ease or difficulty of targeting specific endpoints, we consider that the adversarial gateways are used by at least two endpoints, allowing the adversary to assess whether any pair of observable endpoints are communicating with each other. Note that we require the adversary to simultaneously observe the traces of *both* candidate endpoints I and R to deploy the attack. There is, however, no requirement for the adversary to observe the traces of *other* endpoints that may also be communicating with each other through the mixnet at the same time as I and R.

Compared to an alternative adversary that records packet metadata at the communication link level, our gateway adversary obtains packet metadata at the Sphinx layer with much less noise and thus less necessary processing before analysis. This and the fact that becoming a Nym gateway is relatively accessible to a wide range of actors, leads us to focus on this type of adversary for this work.

Flow Correlation Versus Flow Matching. In Tor, a malicious guard and exit (first and last intermediaries) colluding to determine if they are part of the same circuit can check whether the observed flow of packets $F_{\mathrm{I}}^{\mathrm{In}}$ matches the received flow $F_{\mathrm{R}}^{\mathrm{Out}}$, as *every* packet in the input flow *must* appear in the output after a small delay. As circuits are bidirectional, the adversary similarly checks whether $F_{\mathrm{R}}^{\mathrm{In}}$ matches $F_{\mathrm{I}}^{\mathrm{Out}}$. The timing of circuit establishment and destruction at both ends is an additional feature to determine that the flows correspond to the same circuit. Furthermore, if an endpoint is simultaneously engaged in multiple connections routed through multiple circuits, the traffic sent or received via other circuits does not obfuscate the observed flows F_E^D with spurious packets that may hinder flow correlation. Thus, a pair of traces $T_{\mathrm{I}} = (F_{\mathrm{I}}^{\mathrm{In}}, F_{\mathrm{I}}^{\mathrm{Out}})$ and $T_{\mathrm{R}} = (F_{\mathrm{R}}^{\mathrm{In}}, F_{\mathrm{R}}^{\mathrm{Out}})$ are either a match or not a match, and there is no such thing as a "partial" match in this setting and threat model.

In mixnets, on the other hand, the traffic flows observed by gateways aggregate independently routed packets that may belong to different connections and it is not possible to distinguish which packet belongs to which end-to-end connection. In Nym, even if endpoints I and R are communicating exclusively with each other, the loops of cover traffic sent by clients ensure that some of the packets seen in input flow $F_{\mathrm{I}}^{\mathrm{In}}$ will appear in the same endpoint's output flow $F_{\mathrm{I}}^{\mathrm{Out}}$ and *not* be received at the other endpoint as part of $F_{\mathrm{R}}^{\mathrm{Out}}$. Attempting to match $F_{\mathrm{I}}^{\mathrm{In}}$ to $F_{\mathrm{R}}^{\mathrm{Out}}$ and $F_{\mathrm{R}}^{\mathrm{In}}$ to $F_{\mathrm{I}}^{\mathrm{Out}}$—the approach to evaluate this attack in Tor—is thus not adequate for the Nym mixnet.

When considering endpoints that are communicating exclusively with each other, all the packets in their ingress flows will appear in *one* of their egress flows (unless they are lost in transit), either to themselves in the case of loop cover packets or the counter-party in the case of data transmissions. We call this problem—where all ingress packets are represented as egress packets—*flow matching*. We leave the more complex problem of *partial flow correlation* for future work, where some fraction of packets in the observed ingress flows has no corresponding output in any of the observed egress flows and vice versa, as some packets received in egress flows were sent by endpoints that are not under observation by the adversary. With this more complex problem in mind, flow matching can also be understood as *complete flow correlation*.

3.3 Endpoints Closed Sets and Flow Merging

As shown in Figure 1, the adversary collects a bidirectional trace $T_E = \left\{T_E^{\mathrm{In}}, T_E^{\mathrm{Out}}\right\}$ for each endpoint $E \in \{\mathrm{I},\mathrm{R}\}$. Each directional trace (In, Out) is a sequence of packets characterized by their timestamp and size, which depends on whether it is a *payload* (P) or an *acknowledgment* (A) packet. The egress flows $F_E^{\mathrm{Out,P}}$ and $F_E^{\mathrm{Out,A}}$ are obtained by splitting T_E^{Out} into two vectors according to packet type $\{\mathrm{P},\mathrm{A}\}$. The adversarial gateway obtains the ingress payload flow $F_E^{\mathrm{In,P}}$ from T_E^{In}, which contains all the packets sent by E. Ingress acknowledgments are sent by the gateway itself immediately upon receiving and processing a payload packet, rather than received from endpoint E. Thus, the ingress acknowledgment flow is simply a copy of the egress payload flow, i.e., $F_E^{\mathrm{In,A}} = F_E^{\mathrm{Out,P}}$.

In flow matching, the goal is to determine whether a set of ingress flows matches a set of egress flows. Considering a single endpoint E, a match between $\{F_E^{\mathrm{In,P}}, F_E^{\mathrm{In,A}}\}$ and $\{F_E^{\mathrm{Out,P}}, F_E^{\mathrm{Out,A}}\}$ would reveal that E is not communicating with anyone, but just sending loop cover traffic to itself, which we call a *loner*. Given a pair of endpoints I and R, let $F_{\{\mathrm{I,R}\}}^{\mathrm{In}}$ be the combination of the ingress flows for both parties (separately for each type P and A, which we omit in the following for convenience) and $F_{\{\mathrm{I,R}\}}^{\mathrm{Out}}$ be the combination of the egress flows. A positive match between $F_{\{\mathrm{I,R}\}}^{\mathrm{In}}$ and $F_{\{\mathrm{I,R}\}}^{\mathrm{Out}}$ indicates that I and R are communicating with each other.

Generalizing to any number of parties, let $\mathcal{E} = \{E_1, \ldots, E_N\}$ be a set of endpoints communicating via the mixnet. Flow matching can be applied whenever the *endpoints closed set condition* is satisfied, i.e., all endpoints $E_i \in \mathcal{E}$ communicate with other endpoints in the set (including themselves), but do not exchange (send or receive)

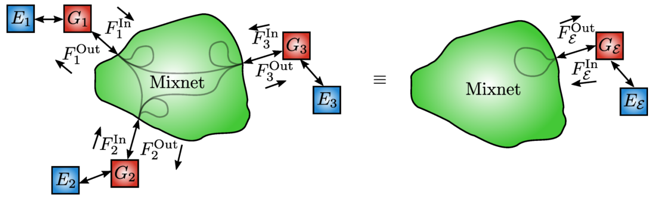


Figure 2: *Flow merging* in a closed set $\mathscr{E} = \{E_1, E_2, E_3\}$. Then, $F_{\mathscr{E}}^D = F_1^D \uplus F_2^D \uplus F_3^D$, with $D \in \{\text{In}, \text{Out}\}$.

packets with endpoints not in $\mathscr{E}$. If this is the case, then the combination of egress flows must contain the same packets included in the combination of ingress flows. We define the combined flows as:

$$F_{\mathscr{E}}^{\text{In}} = \uplus_{E_i \in \mathscr{E}} F_{E_i}^{\text{In}}; \quad \text{and} \quad F_{\mathscr{E}}^{\text{Out}} = \uplus_{E_i \in \mathscr{E}} F_{E_i}^{\text{Out}}, \tag{1}$$

where $\uplus$ stands for *multiset union and reordering*, which we call *merging* for convenience and illustrate in Figure 2. In order to test whether $F_{\mathscr{E}}^{\text{In}}$ and $F_{\mathscr{E}}^{\text{Out}}$ match, the adversary needs to build a detector that is able to distinguish matched from unmatched ingress and egress flows. Formally, this detector can be defined as follows:

$$d_{\text{Ideal}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = \begin{cases} 1 & \text{if} \quad F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}} \\ 0 & \text{if} \quad F_{\mathscr{E}}^{\text{Out}} \nparallel F_{\mathscr{E}}^{\text{In}} \end{cases}, \tag{2}$$

where $\parallel$ stands for *matches* and $\nparallel$ stands for *does not match*.

Focusing on the concrete case where two endpoints I and R communicate with each other, it only holds if:

$$d_{\text{Ideal}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = 1, \tag{3}$$

$$d_{\text{Ideal}}\left(F_{\text{I}}^{\text{Out}}, F_{\text{I}}^{\text{In}}\right) = 0, \text{ and} \tag{4}$$

$$d_{\text{Ideal}}\left(F_{\text{R}}^{\text{Out}}, F_{\text{R}}^{\text{In}}\right) = 0, \tag{5}$$

where $\mathscr{E} = \{\text{I}, \text{R}\}$. In other words, the adversary determines that endpoints I and R form a closed set and are not loners.

Given multiple packet types, the endpoints closed set condition must hold for all types. Thus, the detector is applied separately to both payload and acknowledgment packets. The ideal detector shown in Equation (2) provides the true answer considering that both flows are matching. Practical detectors built by the adversary thus must combine the evidence provided by the separate matching of both payload and acknowledgment flows. There are different methods an adversary may use to approximate the ideal detector of Equation (2). In this paper, we explore three relevant ones: a *statistical classifier* (Section 4), based on the adversary's a priori knowledge regarding the statistical nature of the delays introduced by mixnets in matched ingress and egress flows, and two more general *deep learning classifiers* (Section 5), called drift and shape, which learn this knowledge directly from collected data.

3.4 Packet Alignment

As a practical consideration, when the adversary analyzes ingress and egress flows, it is important to properly align them in order to ensure that the distribution of the drift exhibited by matched pairs is always the same. If both ingress and egress flow capture packets starting at the same global time, the first packets in the egress flow may correspond to packets sent before the capture process started, hence not included in the observed ingress flow. These packets should be discarded. In practice, the adversary can align both time series to ensure that the average difference between same-index egress and ingress timestamps approximates the average delay introduced by the mixes plus other processing and transmission delays. Once this is done, the non-overlapping parts of the aligned flows are cropped to ensure same length. In all our experiments, this alignment is done considering the full flows captured during the experiment duration.

4 STATISTICAL CLASSIFIER

The anonymity of continuous-time mixes has been modeled before by Danezis [7]. Assuming that the incoming traffic is a Poisson process, the anonymity set size of packets traversing a continuous-time mix is given by the differential entropy of its inverse delay characteristic function plus the logarithm of the incoming traffic average inter-packet delay λ^{In}, i.e.:

$$\mathscr{A} = \mathscr{dE}\left\{f'\left(t^{\text{In}} \mid t^{\text{Out}}\right)\right\} + \log \lambda^{\text{In}}, \tag{6}$$

where $\mathscr{dE}\{\cdot\}$ is the differential entropy operator, and $f'\left(t^{\text{In}} \mid t^{\text{Out}}\right)$ is the inverse delay characteristic function, which is the probability density describing the likelihood a message being ejected at time t^{Out} was injected into the mix at time t^{In}. In the case of a multi-hop mixnet, if we assume for simplicity that network propagation delays are negligible compared to mixing delays, then the inverse delay characteristic is the probability density function (pdf) of an Erlang random variable with shape r and rate ψ, denoted as $E(r, \psi)$, where r is the number of mix nodes in a packet's route through the mixnet and $\psi = \mu^{-1}$ is the inverse of the average of the exponential delay distribution introduced at each mix node. Rather than characterizing anonymity, our goal is to build effective statistical tests that distinguish matched from unmatched sets of ingress and egress flows corresponding to endpoints in $\mathscr{E}$, thus approximating the ideal detector defined in Equation (2) as much as possible.

4.1 Statistical Detector

Let us denote the pdf of $F_{\mathscr{E}}^{\text{Out}}$ given $F_{\mathscr{E}}^{\text{In}}$ and $F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}}$ as:

$$f^{+}\left(F_{\mathscr{E}}^{\text{Out}} \middle| F_{\mathscr{E}}^{\text{In}}\right) = f_{F_{\mathscr{E}}^{\text{Out}}}\left(F_{\mathscr{E}}^{\text{Out}} \middle| F_{\mathscr{E}}^{\text{In}}, F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}}\right),$$

and its corresponding likelihood as:

$$L\left(F_{\mathscr{E}}^{\text{Out}} \middle| F_{\mathscr{E}}^{\text{In}}, F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}}; \Theta\right) = L^{+}\left(F_{\mathscr{E}}^{\text{Out}} \middle| F_{\mathscr{E}}^{\text{In}}; \Theta\right),$$

where Θ is the set of parameters of f^{+} and packet type has been suppressed for convenience.

The different possible scenarios for the non-matched case are too many, so the adversary can focus on modeling $f_{F^{\text{Out}}}^{+}$ to build its statistical test and ignore the cases where $F_{\mathscr{E}}^{\text{Out}} \nparallel F_{\mathscr{E}}^{\text{In}}$. For matched flows, each timestamp in $F_{\mathscr{E}}^{\text{Out}}$ matches another timestamp in $F_{\mathscr{E}}^{\text{In}}$, as egress packets are just ingress packets delayed by the mixnet. Therefore, if we assume there are no propagation or processing delays and no packet losses, in mixnets with exponential delays, there exists a permutation of the ingress packet indexes $\mathscr{P}(n_{\mathscr{E}}^{\text{In}}) = \left(p_1, \ldots, p_{n_{\mathscr{E}}^{\text{In}}}\right)$ such that $t_{\mathscr{E},p_k}^{\text{Out}} - t_{\mathscr{E},k}^{\text{In}} \sim E(r, \psi)\ \forall k$.

Under these assumptions, it is possible to compute the exact likelihood of $F_{\mathscr{E}}^{\text{Out}}$ given $F_{\mathscr{E}}^{\text{In}}$ and $F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}}$ (i.e., the evaluation of $f^{+}_{F^{\text{Out}}}$ for some given parameters of the mixnet) by using a recursive procedure based on the following:

$$L^{+}\left(F_{\mathscr{E}}^{\text{Out}} \middle| F_{\mathscr{E}}^{\text{In}} ; r, \psi\right) = \sum_{l \mid t_{\mathscr{E},l}^{\text{In}} < t_{\mathscr{E},k}^{\text{Out}}} f_{E(r,\psi)}\left(t_{\mathscr{E},k}^{\text{Out}} - t_{\mathscr{E},l}^{\text{In}}\right) L^{+}\left(F_{\mathscr{E}}^{\text{Out}-k} \middle| F_{\mathscr{E}}^{\text{In}-l} ; r, \psi\right), \tag{7}$$

where $f_{E(r,\psi)}$ is the pdf of an $E(r, \psi)$ random variable with $r = 3$ and $\psi = 1/50$ms in the case of Nym, and $F_{\mathscr{E}}^{D-q}$ is the flow resulting from removing $t_{\mathscr{E},q}^{D}$ from $F_{\mathscr{E}}^{D}$ with $D \in \{\text{In}, \text{Out}\}$. Although this exact computation removes the impossible correspondences, it still requires impractical computational power for even modest flow lengths as it only evaluates permutations where $t_{\mathscr{E},p_k}^{\text{Out}} > t_{\mathscr{E},k}^{\text{In}}\ \forall k$. Therefore, the adversary is forced to rely on a practical approximation. Also, this computation would differ if we introduced network delays. Instead of evaluating all possible orders of packets, the adversary can use a strategy robust against packet reordering by paying attention to the drift of output and input flows. If ingress and egress flows are aligned, the difference of egress and ingress timestamps with the same index follows a stationary distribution for matched flows, but randomly drifts away from this distribution for unmatched flows. In an idealized case without reordering between the ingress and egress flows, i.e., $p_k = k$ in the index permutation, this strategy would be equivalent to the exact computation.

For convenience, let us define the logarithmic scoring function $f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right)$ as:

$$f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = \ell^{+}\left(F_{\mathscr{E}}^{\text{Out}} \middle| F_{\mathscr{E}}^{\text{In}} ; r, \psi\right),$$

where ℓ^{+} is the loglikelihood function of matched flows and stat is short for statistical. The adversary can use the matched flow stationary drift distribution to approximate this scoring function as:

$$f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) \approx \sum_{l} \log\left[f_{\Delta(\Gamma)}\left(t_{\mathscr{E},l}^{\text{Out}} - t_{\mathscr{E},l}^{\text{In}}\right)\right], \tag{8}$$

where $f_{\Delta(\Gamma)}$ is the pdf of the drift (delay of packets with the same index) in aligned matched flows and Γ is the set of parameters of this pdf. The distribution of the drift, given that reordering happens, differs from the $E(r, \psi)$ mixnet delay random variable regardless of the propagation delays. This distribution has a much lighter right tail than the Erlang delay introduced by the mixnet in individual packets, due to packet reordering that happens in mixnets. The adversary must approximate the actual $f_{\Delta(\Gamma)}$ from its own collection of matched flows. In this study, we use a Gaussian approximation, i.e., $\Delta(\Gamma) \sim \mathcal{N}(\eta_\Delta, \sigma_\Delta)$, where η_Δ and σ_Δ are the sample mean and standard deviation of the delays between packets with the same index in matched flows measured by the adversary.

As this score (which is also a loglikelihood) yields higher values for matched than for unmatched flows and the target value also depends on the adversary's objective and the flows' length, the adversary builds their Ingress/Egress Flow Matching hypothesis test as follows:

Egress flow $F_{\mathscr{E}}^{\text{Out}}$ matches ingress flow $F_{\mathscr{E}}^{\text{In}}$ if, and only if, the logarithmic score is above a threshold that depends on the length of the flows and the target working point Ω, i.e.:

$$F_{\mathscr{E}}^{\text{Out}} \parallel F_{\mathscr{E}}^{\text{In}} \iff f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) \geq \theta(n, \Omega),$$

or equivalently:

$$d_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = \begin{cases} 1 & \text{if} \quad f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) \geq \theta(n, \Omega) \\ 0 & \text{if} \quad f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) < \theta(n, \Omega) \end{cases} \tag{9}$$

4.2 Combination of Payload and Acknowledgment Flow Scores

As a fresh mixnet route is sampled for each acknowledgment packet attached to a payload packet (Section 2.2), the scores for payload and acknowledgment flows obtained by the adversary are statistically independent. Together with the fact that these scores are loglikelihoods, the adversary can compute the combined score $f_{\text{stat}}^{\text{comb}}$ as:

$$f_{\text{stat}}^{\text{comb}}\left(F_{\mathscr{E}}^{\text{Out}}, F_{\mathscr{E}}^{\text{In}}\right) = \sum_{M \in \{\text{P},\text{A}\}} f_{\text{stat}}\left(F_{\mathscr{E}}^{\text{Out},M}, F_{\mathscr{E}}^{\text{In},M}\right), \tag{10}$$

with each packet-type-specific score computed using Equation (9).

5 DEEP LEARNING CLASSIFIERS

Our machine learning approach to the mixnet flow matching problem is inspired by DeepCoFFEA, the state-of-the-art traffic correlation attack on Tor [23]. Although some of the choices behind DeepCoFFEA's design are also applicable to flow matching in mixnets, we make substantial modifications to the original design to effectively tackle the novel setting. In this section, we describe our approach highlighting the points where it diverges from DeepCoFFEA's.

5.1 Deep Learning

Similar to DeepCoFFEA and in contrast to the statistical classifier introduced above, this approach is based on a deep learning (DL) classifier. Instead of explicitly modeling the mixnet's delay characteristic and making decisions based on how well the traffic observations fit the hypothesis that the flows *match*, a DL algorithm aims to find a model f_{DL} with parameters W able to discriminate between *matched* and *unmatched* ingress and egress flows.

To obtain f_{DL}, DL algorithms search for the values of W that minimize a *loss function l*, which penalizes incorrect predictions of f_{DL} on a sample of correctly labeled data. This optimization process is referred to as *training* and hence the dataset used for training is known as the *training set*. The ability of DL algorithms to produce models that generalize well (i.e., perform well on samples that were not included in the training set) indicates good learning capability.

The resulting f_{DL} is a possibly non-linear model that, unlike the statistical classifier, does not assume a specific delay distribution for ingress and egress packets with the same index. Therefore, a DL classifier has the potential to overcome some of the limitations of the statistical classifier—albeit at a higher demand for training data and computational resources.

5.2 Input Representation

As discussed in previous sections, we tackle a flow matching problem that is slightly different from the one that DeepCoFFEA was designed to address. Rather than matching a pair of ingress and egress flows for each upstream and downstream direction, here the problem concerns the matching of pairs of merged ingress and egress flows for a type of packet and pair of endpoints communicating exclusively with each other. Following the notation introduced in previous sections, for an endpoint pair $\mathscr{E} = \{I, R\}$, we denote its associated merged ingress and egress flow pair as $F^{\text{In}}_{\mathscr{E}} = F^{\text{In}}_{I} \uplus F^{\text{In}}_{R}$ and $F^{\text{Out}}_{\mathscr{E}} = F^{\text{Out}}_{I} \uplus F^{\text{Out}}_{R}$, omitting packet type for convenience. Just like the statistical classifier incorporates different packet types into the test (Section 4.2), we consider acknowledgments and payload packets as independent samples to train the DL classifiers.

Note that our merged flow representation is especially well-suited to account for Nym's loop cover traffic, which does not degrade matching accuracy when the ingress and egress flows of the various endpoints are merged. As we demonstrate, DeepCoFFEA's performance is severely impaired when Nym endpoints send loop cover traffic (see Figure 15 in Appendix D). Our input representation is robust to loops, as merging ensures that loop cover packets appear in both the ingress and egress flows being matched by the classifier.

5.3 Window Partitioning

In line with DeepCoFFEA, we partition the flow pairs into windows to take advantage of the augmentation effect of multiple independent tests [23]. Our windows differ from DeepCoFFEA's in that they are packet-based rather than time-based. A packet-based windowing approach is better suited for flow matching, as for matched flows it is more likely that ingress and egress packets will fall in windows with the same index. To prevent a possible misalignment between the ingress and egress windows, it is crucial that we apply the technique described in Section 3.4 per each window index.

We consider windows that may overlap a constant number of packets. We denote by n_w the number of packets per window and by δ the fraction of packets by which each two consecutive windows overlap. For example, the window partitioning of the flow pair $F^{\text{In}}_{\mathscr{E}}$ and $F^{\text{Out}}_{\mathscr{E}}$ for $\delta = 0.5$ and $n_w = 10$, would return the packet indices within the intervals $[1, 10]$, $[6, 15]$, $[11, 20]$, etc., for each flow.

The final representation of a window comprises the inter-arrival times of the window's time sequence, and we treat it as an independent data point to train the DL-based models. Thus, the number of windows is a multiplicative factor of the size of the training set. The large training datasets collected for our experiments allow us to discard the last window of the flows, as it usually contains less than n_w packets. We do so for convenience, but, alternatively, we could have padded them with zeros [23]. To be able to pair every ingress window with an egress window, we also ensure the same number of windows across all training flows by discarding windows. Below, we describe the two types of DL classifiers that we have designed for flow matching: the *drift* network and the *shape* network.

5.4 Drift Network

The drift network exploits the same information as the statistical classifier. It takes in flow pair $F^{\text{In}}_{\mathscr{E}}, F^{\text{Out}}_{\mathscr{E}}$ and computes the sequence $t^{\text{Out}}_{\mathscr{E},k} - t^{\text{In}}_{\mathscr{E},k}$ for $k = 1, \ldots, n$. This sequence of time differences between egress and ingress flows captures their *temporal drift*, which correlates with whether the flows are matched or not.

Network Architecture. The architecture of the drift network differs significantly from the one of DeepCoFFEA: instead of learning intermediate representations of the flows through metric learning, it learns to classify them directly. Our tests show that such a simple network already performs remarkably well, indicating that metric learning and other dimensionality reduction techniques may only provide marginal improvements.

The drift network is similar to DeepCoFFEA in that it also has several hidden convolutional layers, but it is much smaller, as traffic matching is a simpler learning problem than traffic correlation. In addition, we use average pooling instead of max-pooling due to the importance of the localization of the drift: when the flows are unmatched, as time passes, the ingress and egress flows drift further apart, thus providing more signal to the classifier. We expand on the drift network's architecture and its parameters in Appendix E.1.

Testing. After training, the resulting drift model f_{DL} returns a score for each ingress–egress window pair that quantifies how confident the model is that the windows match. The adversary sets a threshold on these scores to make a final decision. To aggregate the decisions across all the windows in a flow, we average the window scores and then set the threshold on the average. This is in contrast to Oh et al.'s approach to set the threshold on each individual window score, count the number of windows that match, and then make a decision based on a majority-vote rule. We argue that the majority-vote rule loses useful information at window level, as it does not capture how far (or close) each vote was from the threshold value. Figure 14 in Appendix D shows that DeepCoFFEA performs better with our average aggregation rule. When acknowledgments are available, we aggregate the averaged scores for acknowledgments and payloads before applying the threshold, as two independent inputs for the same test data point.

5.5 Shape Network

Instead of exploiting drift information, the shape network attempts to match the ingress and egress inter-arrival timing sequences. Similar to DeepCoFFEA, the shape model is obtained by training a *triplet network* [15], a DL algorithm that we adapted to obtain new representations of the traffic flows that are more efficient for solving the flow matching problem.

Typically, triplet networks take three data points as input: a reference point known as *anchor*, a matching point to the anchor called *positive*, and a non-matching point called *negative*. We denote them as a, x^+, and x^-, respectively. Given a similarity metric S, the training objective of the triplet network is to find the parameters W of a model f_{DL} such that:

$$S(f_{\text{DL}}(a), f_{\text{DL}}(x^+)) > S(f_{\text{DL}}(a), f_{\text{DL}}(x^-)) + \alpha, \tag{11}$$

where the model f_{DL} is typically a convolutional neural network with parameters W and α is a slack variable called *margin*. The outputs of f_{DL} are low-dimensional vectors of fixed length, also known as *embeddings*. If f_{DL} satisfies Equation (11), its embeddings for matched flows are more similar to each other than those for unmatched flows by at least α, thus allowing to distinguish between

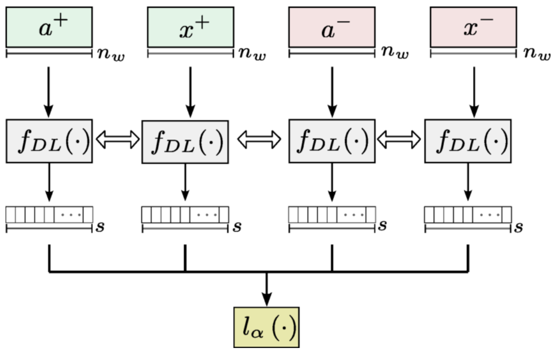


Figure 3: Adapted triplet network for the shape classifier. The inputs are the anchor-positive, positive, anchor-negative, and negative windows. The horizontal arrows between the f_{DL} boxes indicate the parameters W are shared among them. f_{DL}'s outputs are the embeddings which are passed to the modified l_α. The sizes of the windows and the embeddings are denoted as n_w and s, respectively.

them by just comparing their similarities. In practice, it might not be possible to always satisfy Equation (11), so we find W that minimizes the hinge loss function l_α, defined as:

$$l_\alpha := \max\left(S(f_{\mathrm{DL}}(a), f_{\mathrm{DL}}(x^-)) - S(f_{\mathrm{DL}}(a), f_{\mathrm{DL}}(x^+)) + \alpha, 0\right).$$

Our novel input representation requires us to modify the objective and the loss functions above. Flows are defined relative to $\mathscr{E}$, creating a dependency in the definition of an "anchor flow" to $\mathscr{E}$. For example, for an endpoint E_1, its matched pair is $\mathscr{E}^+ = \{E_1, E_2\}$, and one of its unmatched pairs may be $\mathscr{E}^- = \{E_1, E_3\}$, resulting in an anchor flow for each $\mathscr{E}^+$ and $\mathscr{E}^-$. We call the anchor flow for $\mathscr{E}^+$ the *anchor-positive* flow, defined as $a^+ := F^{\mathrm{In}}_{\mathscr{E}^+}$, and its corresponding positive flow is defined as $x^+ = F^{\mathrm{Out}}_{\mathscr{E}^+}$. Analogously, we define the *anchor-negative* flow as $a^- := F^{\mathrm{In}}_{\mathscr{E}^-}$ and its negative flow as $x^- = F^{\mathrm{Out}}_{\mathscr{E}^-}$. Therefore, the triplet network objective becomes:

$$S(f_{\mathrm{DL}}(a^+), f_{\mathrm{DL}}(x^+)) > S(f_{\mathrm{DL}}(a^-), f_{\mathrm{DL}}(x^-)) + \alpha, \qquad (12)$$

and l_α changes accordingly. This is in contrast to DeepCoFFEA's triplet network, which follows the usual single-anchor approach.

In Figure 3, we show a high-level view of our modified triplet network, abstracting the internal model f_{DL}. Recall that after window partitioning, the training data points are not flows but windows of size n_w. Note that although the input size of f_{DL} is n_w, the training results in one single f_{DL} model. We denote by s the embedding size, which is a parameter of the network. We follow DeepCoFFEA and use cosine similarity as S and embedding size $s = 64$.

Triplet Mining. Traditional triplet networks improve convergence by *triplet mining*, the search for negative instances that lie within the margin [37]. These negative instances are known as *semi-hard negatives*, as opposed to *hard* negatives, which already satisfy Equation (11). Semi-hard negatives are closer to positives and thus help the learning algorithm to focus on differences that are more relevant to satisfying the margin imposed by Equation (11).

For $\mathscr{E}^+$ in the previous example, we could could have picked any other non-communicating endpoint to define $\mathscr{E}^-$, indicating that there is room for triplet mining. Similarly to DeepCoFFEA, we compute the pair-wise similarities for all possible negative flows and select those that are within the margin of Equation (12). While DeepCoFFEA trains with one semi-hard negative per batch, we feed them all to the classifier. The computational complexity of triplet mining increases quadratically on the number of matched flows. Thus, in our implementation, we distribute the triplet mining computations across batches, calculating similarities for only a small subset of the negative flows at a time.

Network Architecture. The architecture of the network that fits the embeddings in the triplet network and produces the model f_{DL} is almost identical to DeepCoFFEA's. We depict the network architecture and its parameters in Figure 17a in Appendix E.1.

Testing. A shape model also returns a score for whether or not two windows match. More precisely, in the case of a shape model, this score is the cosine similarity between two window embeddings. Unlike DeepCoFFEA, we set a global threshold instead of a local one. For each anchor, Oh et al. take the maximum similarity score in the k-nearest negatives to the anchor as the threshold to discriminate instances relative to that anchor. The pool of negatives from which the final negative is picked includes all possible negatives, thus implicitly assuming that the adversary has visibility over all the egress links of the Tor network. In our threat model, the adversary does not necessarily control all the Nym gateways and therefore may not be able to observe all negative flows. Thus, we use a global threshold in all our evaluations, even though the results of our experiments indicate that a local threshold would provide superior performance (see Figure 13 in Appendix D). More details on the parameters of the network architectures and hyperparameter tuning for the learning algorithms can be found in Appendix E.

6 DATA COLLECTION

We consider a file download as application use case for our empirical evaluation of Nym in various setups. We implement a `curl` process (triggered by the *initiator* I) that downloads a file of 1 MiB size via HTTP over Nym from a (default) `python3` HTTP server (the *responder* R). By fixing variables such as file size and endpoint send rate (except responder send rate in the case of dataset `two-to-one`, see below) across experiments, we aim to assess the effect of isolated mixnet features. For this work, we assume the adversary knows when the HTTP request starts and ends within a trace.

We obtain six trace pair datasets to evaluate flow matching in Nym. We collect the first four datasets (called `baseline`, `no-cover`, `low-delay`, and `high-delay`), using a single-host experimental setup based on Docker, with Nym in version `nym-binaries-1.0.2`. We call this the *isolated setup* and depict it in Figure 9 in Appendix C. We simulate a fifth dataset (called `two-to-one`) on the isolated setup by crafting it from `baseline`. The final dataset (called `live-nym`), we collect from the live Nym mainnet using instrumented Nym components in version `nym-binaries-v1.1.13`. This is the *live-network setup* and we show it in Figure 10 in Appendix C.

Isolated Setup. We encapsulate the components of a minimal Nym deployment using Docker containers and orchestrate them via Docker Compose on a single instance running at a public cloud provider, based on a single-host Kubernetes setup shared with us by the Nym development team. Nym in version `nym-binaries-1.0.2` requires at least the following processes to be running for endpoints (*initiator*, *responder*) to exchange packets: a gateway (*gateway*), three mixes arranged in three layers with one mix per layer (*mixnode*), a process maintaining Nym's network state in a blockchain (*validator*), a process providing a network state interface to Nym processes (*validator-api*), and a process initializing Nym's network state (*busybox*). When building the Docker containers for an experiment, we include patches to enable data collection and possibly modify Nym in one aspect to evaluate this aspect's contribution to anonymity in isolation. In all experiments, we patch the gateway to track Sphinx packet metadata (timestamp, size) in both directions (ingress, egress) via the Nym address of the respective endpoint. This models our adversary's metadata gathering capabilities (Section 3.2). Endpoint Nym components are compiled with Rust in version `1.63.0` across all isolated setup experiments, while the remaining Nym-compiling Docker images use Rust in version `1.65.0` (`baseline`, `no-cover`) or `1.66.1` (`low-delay`, `high-delay`).

Next, we generate a random file that (appended by a fixed-width and download-specific identifier) is of 1 MiB size. Once all but the endpoint components have started up correctly and formed a functional Nym network, we collect our trace pair dataset. Until we reach a target number of successful HTTP-based downloads of the random file, we spawn a fresh initiator-responder pair (including fresh Nym identities for both Nym endpoints) and connect them to each other via Nym. Our modified gateway will track their packet metadata via their Nym addresses and we are able to establish the correct endpoint pairs via our experiment orchestration. Right before and right after the actual `curl` request on the initiator, we take the current time to delineate *start* and *end* of the download. We include HTTP header `'Accept-Encoding: identity'` to avoid compression distorting the size of the transferred file. We only mark a download as successful (and thus increment our counter towards reaching our target number) if the downloaded file at the initiator is byte-by-byte equal to the served file at the responder.

`baseline`. As our baseline dataset, we collect trace pairs over the isolated setup (i.e., no network effects like propagation delay or packet loss) without any changes to how Nym endpoints send messages. Thus, the default artificial delay introduced by the setup's mixes independently delaying the packets they process is the only influence on packet timings and reorderings. Also applying to all other isolated setup experiments, we backport a patch from the official Nym code base that fixes a buffer reordering bug in Nym endpoints. Additionally for these experiments, we patch the epoch time to be three minutes instead of one hour and apply the before-explained gateway patch to enable Sphinx-level metadata recording.

`no-cover`. In order to assess the contribution of cover traffic towards Nym's claimed privacy protections, we disable "gap-filling" and regular loop cover traffic for this experiment. The only packets in this dataset are thus "real", application-level packets. The remaining additional patches are functionally the same as for `baseline`.

`low-delay` (`high-delay`). Next to applying the same patches as in `baseline`, we reduce (increase) the endpoints' average per-mix delay parameter (μ) for any message sent out, from 50 ms to 20 ms (200 ms). As we keep the packet emission rates (λ_R, λ_C) the same as before, this reduces (increases) the set of other packets that a particular packet can be reordered with when being delayed at a mix. These two experiments thus change the ratio $\mu/\lambda_{T\in\{R,C\}}$, i.e., how long packets dwell at mixes on average in relation to how quickly endpoints send them.

`two-to-one`. In order to evaluate how our classifiers perform once positive and negative flows share endpoints and how flow merging works with larger endpoints sets, we model the setting of two initiators downloading simultaneously from a single (simulated) responder (`two-to-one`). We are interested in two cases: classifier performance when either both (positive) or exactly one (negative) of the two initiators are matched with the responder in the set (*semi-matched*) and either both (positive) or none (negative) of the two initiators are matched with the responder in the set (*unmatched*). Without collecting new data, we simulate these flow pairs by pairing always two responders from `baseline` to become one merged, logical responder and selecting initiators such that we arrive at above specified cases. Mind that we align each initiator-responder pair independently before merging both responders into a single logical responder. Finally, merging the two initiators and the single logical responder created this way, we obtain endpoints sets of size three, and are thus able to gauge classifier performance on larger endpoints sets that may overlap in endpoints. Note that the merged logical responders in `two-to-one` send messages at twice the default rate of Nym endpoints. This sketches a more realistic "Nym server" deployment where endpoints that only proxy between services and mixnet adjust their send rate linearly with the number of connected Nym endpoints (but this deviates from default Nym).

Live-Network Setup. In order to evaluate the attack in the presence of real-world network effects like network propagation delays and packet loss, we collect the dataset `live-nym` on the available Nym mainnet. This experimental setup only requires running a private gateway modified for capturing traces and an endpoint instance for each simultaneous initiator-responder pair (we run five endpoint pair instances in parallel). We are able to keep our gateway private by not bonding the gateway after initializing it. Bonding is required for any gateway intended for use by regular endpoints. With a small patch to our experiment Nym endpoints, we hard-code our private gateway as the only gateway in the network available to our experiment endpoints. By following this process, we obtain data for our own endpoints and avoid routing traffic for other Nym endpoints. As the traffic we generate in below experiment is indistinguishable from regular Nym traffic, we collect data in an ethical way and avoid negatively impacting the anonymity of unrelated Nym endpoints. As we collect packet timestamps at the gateway and thus exclude the propagation delay of the last hop between the endpoint and gateway, we run our gateway and endpoints instances in the same location. Due to collecting with only five simultaneous endpoint pairs, the additional load caused by our experiment on the Nym network is limited to ten additional Nym clients.

`live-nym`. Similarly to the `baseline` dataset, Nym endpoints behave as specified in the Nym system without any modifications. This dataset was collected on the deployed Nym network and thus corresponds to a more recent Nym version (`nym-binaries-v1.1.13`) than the one used in the isolated setup. A difference between both versions is the recently introduced adjustment of the packet emission rate based on backpressure from the gateway. We disable this feature to maintain consistency and better comparability with our isolated setup experiments. Since the Nym version used in our isolated setup, the previously separate `nym-client` functionality was added to the `nym-network-requester` process directly, eliminating the need to run it separately. All Nym components are compiled with Rust in version `1.68.2`.

Post-Processing. Once collected, we run three post-processing steps on each dataset. The first checks the integrity of an experiment's setup and execution. The second produces dataset versions from the initially collected "raw" datasets that are directly usable in either of our classifiers. In line with our assumption that the adversary knows when the HTTP request within a trace pair starts and ends, this step also restricts each trace pair to the duration of the `curl` request (stopwatch symbol in Figure 9 in Appendix C). We plot the duration of each successful file download (i.e., of all collected trace pairs) across datasets in Figure 11 in Appendix C. The third post-processing step partitions each collected dataset into trace pair subsets to be used exclusively for training, validation, and testing by the classifiers. Each collected dataset contains a little more than 35,000 trace pairs. For consistency across experiments, we first take the top-35,000 trace pairs from the sorted list of trace pair identifiers in lexicographic order. Then, we select the first 24,500 trace pairs for training, the next 5,250 for validation, and the final 5,250 for testing. We make all instrumentation code and all collected datasets publicly available [1].

7 EXPERIMENTAL RESULTS

Evaluation Details and Metrics. We make the source code of our statistical classifier (implemented in Octave) and our DL classifiers (implemented in Python using TensorFlow/Keras) publicly available [1]. We split the datasets into training, validation, and testing parts as explained in Section 6. The training set is used to obtain classifier parameters, i.e., estimation of the delay probability density function parameters as well as the actual training of our DL classifiers. We use the validation set to tune the hyperparameters of the DL algorithms (more details in Appendix E.2) and to track how well the models generalize on the validation set during training. Finally, the test set is used to assess the final performance of our classifiers on data excluded from the training and validation sets.

We primarily use the *receiver operating characteristic* (ROC) curves to represent a classifier's performance in terms of its *true positive rate* (TPR) in relation to its *false positive rate* (FPR). An adversarial *observation* refers to the metadata of one payload packet and its corresponding acknowledgment packet. Unless otherwise specified, this section's figures show results for the first 100 observations (i.e., the first 100 payload packets and their corresponding acknowledgments) per flow, after flow pairs have been aligned using all observations from the entire download. By plotting the x-axis in log scale, we focus on the operating points of the curve with low FPRs, as a low FPR is a requirement for the adversary to have high confidence in positive predictions. We also fix the window size of our deep learning classifiers to 100 observations. The classifiers are responsible for converting traces to flows and aligning flow

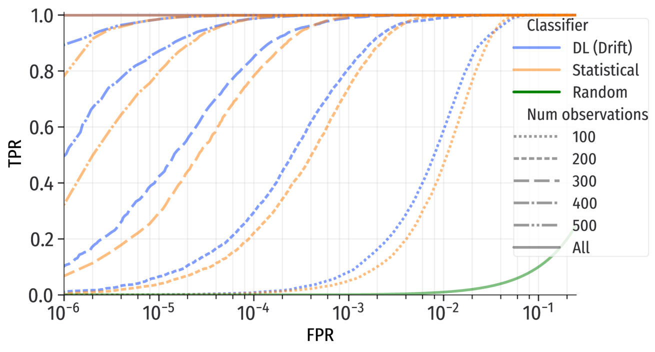


Figure 4: Flow matching performance (ROC curves) based on number of observations considered per candidate flow pair of our statistical and our drift classifier on dataset `baseline`.

pairs (Section 3.4). We report time and hardware requirements for training and evaluating each classifier in Appendix G.

Impact of Classifier Choice. For Figures 4 to 7 here and Figures 12 and 16 in the appendix, we only report the performance of our statistical and our drift classifier. While we include the shape network in this study as it is the most direct adaptation of DeepCoFFEA [23] to the setting of flow matching on mixnets, we do not evaluate it for the above mentioned settings, as its poor performance became clear early on (see Figure 8). At an FPR of 10^{-2}, our shape model performs on dataset `baseline` at the level that the other two classifiers reach on the much more challenging dataset `live-nym`. Thus, we focus exclusively on the results for our statistical and our drift classifier for most of the below discussed results.

Impact of Distinguishable Packet Type. We start by evaluating the adversarial advantage derived from having two types of packets (payload, acknowledgment) distinguishable by size. We show in Figure 12 in Appendix D that the best results are obtained when both packet types are taken into account, compared to using only either payload or acknowledgment packets. This is due to acknowledgments constituting independent observations of the drift, providing additional useful information to the classifiers. Thus, we always use both packet types going forward.

Impact of Number of Observations. Next, we consider how flow matching performance increases when the adversary has more observations available for matching decisions on dataset `baseline` (i.e., default Nym configuration and no real-world network effects). As Figure 4 clearly shows, each increment of 100 additional observations raises performance of either of our two classifiers significantly. Our drift model maintains a lead over the statistical classifier for most operating points on the curves, until they reach TPRs greater than circa 0.9, after which both classifiers either perform equally or the statistical classifier has a slight lead. Overall, flow matching performance on `baseline` is high, roughly 0.6 TPR (drift model) and 0.47 TPR (statistical classifier) at 10^{-2} FPR after 100 observations of flow pairs aligned using the entire download.

Impact of Base Rate (Simultaneous Communications). In Figure 16 in Appendix F, we show how the required number of observations to achieve high classifier performance (TPR $\geq$ 0.95) with high

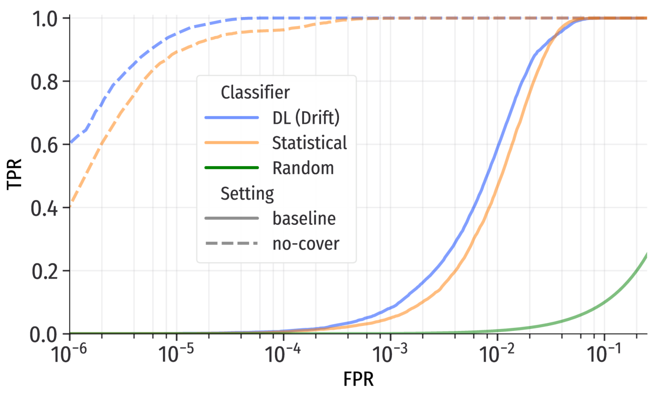


Figure 5: Flow matching ROC curves of our statistical and our drift classifiers on the `baseline` and `no-cover` settings.

confidence (precision ≥ 0.95) on `baseline` increases approximately logarithmically with the number of concurrent communication pairs. This indicates that our attacks scale remarkably well: we only need an additional 100 observations in order to maintain high classification performance at the next higher order of magnitude number of simultaneous one-to-one communications. Both classifiers need to process at least four windows of 100 observations each (circa 314.65 KB downloaded) to achieve the target performance level when faced with 1,000 simultaneous one-to-one packet exchanges.

Impact of Loop Cover Traffic. Nym endpoints send packets following a Poisson process. If no application-level packet is waiting to be sent at the scheduled timeout, a loop cover packet is sent instead to fill the gap. Additionally, endpoints send loop cover packets via a separate slow-rate background Poisson process, regardless of application-level traffic that may be ready. We assess the contribution of loop cover packets to Nym's anonymity by comparing classifier performance on datasets `baseline` and `no-cover`, which differ in the presence of loop cover packets, but are otherwise identical setups. The results are shown in Figure 5, where we observe that loop cover packets contribute significantly to thwarting flow matching attacks. As discussed above, `baseline` flow matching performance is high considering the heavy countermeasures Nym employs, but omitting cover traffic has devastating consequences for anonymity: both classifiers reach their `baseline`-level performance at FPRs close to four orders of magnitude lower (slightly higher than 10^{-6} instead of at 10^{-2}).

Impact of Average Per-Mix Delay. By changing the average per-mix delay (μ) that Nym endpoints use when building packets while keeping their inter-emission delays (λ_R, λ_C) fixed, we modify the number of other packets that a packet at a mix can be swapped with in datasets `low-delay` and `high-delay`. The curves in Figure 6 show that both classifiers' performances increase significantly when μ is dropped from 50 ms to 20 ms (reaching 1.0 TPR at 10^{-2} FPR), while their performance diminishes when we increase μ from 50 ms to 200 ms (dropping to circa 0.13 TPR at 10^{-2} FPR). This demonstrates the importance of μ for the threat of flow matching.

Impact of Larger Endpoints Sets. We aim to estimate how classifier performance changes when negative flows share endpoints

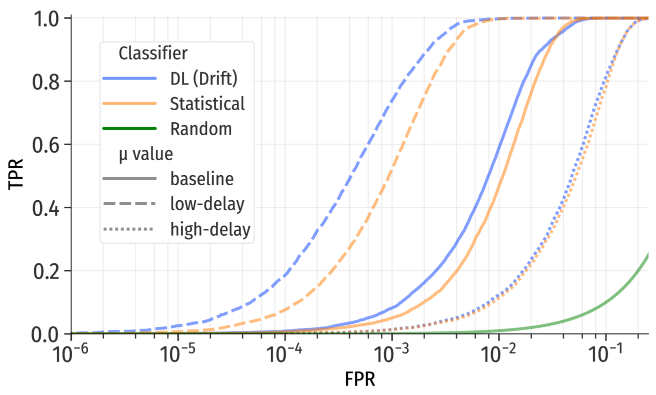


Figure 6: Flow matching ROC curves of statistical and drift classifier under μ = 50 ms (`baseline`), μ = 20 ms (`low-delay`), and μ = 200 ms (`high-delay`) average per-mix packet delay.

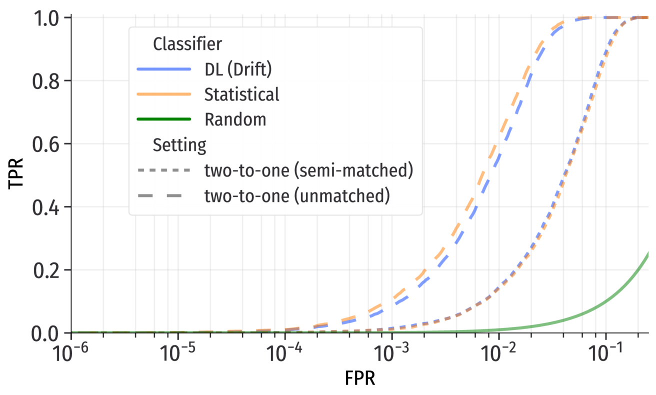


Figure 7: Flow matching ROC curves of the statistical classifier and drift model on `two-to-one`, i.e., on endpoints sets of size three simulated from `baseline`, with two initiators and one (faster) responder. Positive samples consist of two matched initiator-responder flow pairs for both cases, while negatives contain either one unmatched initiator (*semi-matched*) or two unmatched initiators (*unmatched*).

with positive flows. In Figure 7, we show the performance of both classifiers trained on `baseline` but evaluated on the simulated dataset `two-to-one`, i.e., performing flow matching on endpoints sets with three logical endpoints (two initiators, one merged double-rate responder). Flow matching performance remains high in the unmatched setting (without overlap, but larger sets), with circa 0.57 TPR (drift) and circa 0.62 TPR (statistical) at 10^{-2} FPR. In the semi-matched case (where roughly half the packets in merged negative flows overlap with a merged positive flow), both classifiers' performance drops to circa 0.15 TPR at 10^{-2} FPR. However, our classifiers have only analyzed 100 observations at this point and outperform random guessing by a wide margin. Thus, while reduced, utility in this more difficult setting remains significant. We also prove our flow merging technique to handle larger endpoints sets well.

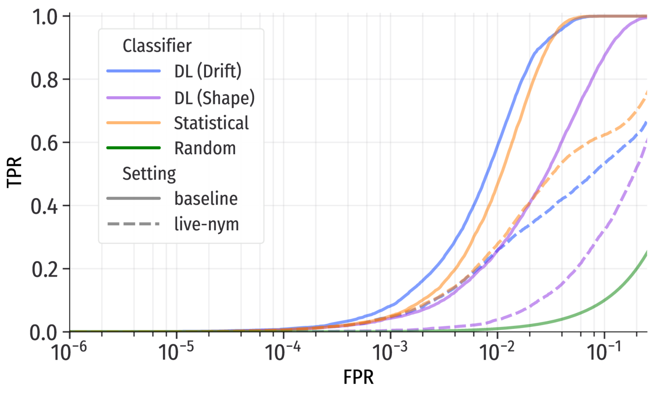


Figure 8: Flow matching ROC curves of our statistical classifier, and our drift and shape classifiers without (`baseline`) and with (`live-nym`) real-world network effects.

Impact of Internet-Level Network Effects. Finally, we evaluate our flow matching attacks in the presence of real-world network effects (e.g., propagation delays, packet losses). We do so by comparing results for the `live-nym` dataset collected on the deployed Nym mainnet to `baseline` collected on the isolated setup. Given that propagation time is not negligible anymore in `live-nym`, we increase the average propagation time in our packet alignment procedure to account for the average aggregate propagation time of four Internet-level links (from the gateway through three mixes back to the gateway). We estimate the mean propagation time of one link by analyzing a 1-hour RIPE Atlas Ping dataset [28] from January 20, 2023. We filter out incomplete measurements, exclude outliers greater than the 99th percentile assuming a well-connected server-to-server network, and use three *round-trip time* (RTT) measurements per probe and destination. We obtain 3,339,429 RTT values that we halve as an estimate for their logical link's propagation time and average to arrive at the final value of 51.103 ms.

As we can see in Figure 8, the performance of all our classifiers is significantly degraded under real-world network effects. While significant, this is also unsurprising, as packet losses severely interfere with our classifiers' assumption that all ingress packets must eventually appear in the egress flow. At an FPR of 10^{-2}, our drift model drops from about 0.6 to circa 0.26 TPR, our statistical classifier goes from roughly 0.47 down to circa 0.28 TPR, and our shape model diminishes from circa 0.26 to close to 0.03 TPR.

8 DISCUSSION

8.1 Flow Matching Effectiveness Against Nym

Under Nym's default configuration (`baseline`) and considering idealized network conditions that exclude propagation delays and packet losses, both our statistical and drift classifiers are able to determine accurately and quickly whether a flow pair is matched or unmatched. Both classifiers retain some utility even on larger endpoints sets with endpoints overlap in positive and negative samples (`two-to-one`). While these idealized network conditions are clearly unrealistic, they enable us to evaluate the role of the foundational anonymity techniques implemented in Nym's mixnet. By isolating the effects of the mixnet's Poisson sending and per-hop exponential mixing delays from random failures and network-related effects (present in `live-nym`), we can assess the distinct contribution that these techniques make to the overall anonymity that Nym offers to packet flows. We disclosed an intermediate as well as the final version of our results to the Nym team so that possible countermeasures (e.g., modifications to the cover traffic scheme) can begin to be considered as early as possible.

Our statistical and drift models outperform random guessing by wide margins. On average, Nym endpoints emit more than 50 packets per second, which means the adversary gathers the required number of observations after only a few seconds. With a few more seconds of eavesdropping on dataset `baseline`, at 500 observations, flow matching performance increases to circa 0.98 TPR at 10^{-5} FPR for both classifiers, while ensuring high confidence in their predictions even when there are 10,000 concurrent connections. After 14 seconds of observation (slightly more than the `baseline`'s median download duration, Figure 11 in Appendix C), our statistical and drift models essentially exhibit perfect classification.

One of Nym's strengths is its use of loop cover traffic. Without it, flow matching would be much easier and quicker. The choice of average per-mix packet delay parameter μ also impacts flow matching performance strongly: raising it to 200 ms reduces performance significantly. While such level of end-to-end latency increases download times only moderately (15.58 s median, Figure 11 in Appendix C), it might be prohibitive for some intended use cases of Nym. The largest drop in classifier performance, however, is due to real-world network effects that an adversary would encounter when deploying the attack in the wild, rather than an idealized lab setup. Propagation delays between nodes in Nym's overlay network are highly variable and of a magnitude comparable to the randomized mixing delays introduced for anonymity purposes. In practice, these delays introduce a large variability in end-to-end transmission times compared to the single-machine lab setting. Moreover, some packets are lost in transit in the real network. Packet loss is not considered in the statistical classifier's design, the DL classifiers' windowing strategy, or the packet alignment phase (which may be challenging to implement in realistic conditions).

Our statistical and drift classifiers track each other's performance quite closely across the evaluated configurations, with the drift model typically performing slightly better. Both designs rely on the same ingress to egress packet alignment and thus share the same weakness in real-world conditions, where packet losses occur. Our shape model—the classifier architecture closest to state-of-the-art Tor flow correlator DeepCoFFEA [23]—lags far behind in performance compared to our statistical and drift classifiers.

Finally, when contrasting our DL approaches with our statistical approach, we see that the right choice of DL classifier typically achieves slightly higher performance. Our DL approaches require more training data and computational resources than our statistical one, but in comparison are able to incorporate the drift distribution of unmatched flows and generalize better than the statistical classifier. Despite better generality of the drift classifier, the statistical classifier performs slightly better in the `two-to-one` semi-matched setting. This slight advantage of the statistical over the drift classifier may indicate the former has better transferability, i.e., robustness to a shift between the training and test distributions.

8.2 Limitations and Assumptions of our Attack

The endpoints closed set condition is the main assumption of our attacks. It allows the adversary to simplify the attack by merging ingress and egress flows, but it can only be applied by an adversary that observes *all* gateways used by *all* endpoints in the set. The attack may become infeasible if endpoints are concurrently communicating with many other endpoints, some of which may be outside the adversary's observation. Mind, however, that our classifiers do not rely on observing *other* simultaneous but unrelated packet exchanges for making matching decisions, but consider each candidate flow pair independently. Furthermore, the efficacy of the attack decreases if only a low number of packets are available in the exchange, becoming essentially ineffective when the packet exchange between endpoints is very short. From our results here, it is clear that even modest packet loss significantly degrades the performance of our attacks, as the classifiers are implicitly designed for scenarios where all the packets in the flows match. While non-trivial, it seems feasible to include these real-world network effects in classifiers and still rely on the endpoints closed set condition.

Packet alignment also plays an important role for our results. It improves discrimination capability by reducing the variance of the scores of matched flow pairs. We always align flows using all observations from their entire duration. This makes the results shown for lower numbers of packets slightly optimistic. Also, direct comparison between `baseline` and `two-to-one` is cumbersome, as the influence of alignment of both flow pairs accumulates in the latter, increasing the impact of this optimistic effect.

While some of our assumptions give an advantage to the adversary, others make flow matching more difficult than it could be in reality. Clearly, our base setting of each endpoint communicating exclusively with one other endpoint only captures some ways in which Nym may be used, with more sophisticated use cases posing a greater challenge to the adversary. The assumption that the adversary knows exactly when each file download starts and ends within a trace pair also reduces the attack's difficulty. At the same time, we consider a setting where equal-sized files are downloaded at the same time, making the experiment more challenging than in realistic use cases where endpoints transfer payloads of varying sizes and at different times.

8.3 Countermeasures

When the endpoints closed set condition holds, our classifiers are able to discriminate between pairs of endpoints communicating exclusively with each other and pairs of endpoints not communicating at all. As shown in Figure 15, loop cover traffic is effective in thwarting existing approaches such as DeepCoFFEA, which assume that all packets in a flow travel in either the upstream or the downstream direction—an assumption that no longer holds with loop cover packets. Loop cover packets, however, fail to provide effective protection against our approach, where they are accounted for by merging the sets of ingress and egress flows of the endpoints in the closed communication set.

An objective for an effective countermeasure is to invalidate the endpoints closed set condition. Our `live-nym` results demonstrate that even the minor deviations from this condition caused by a small fraction of packets being lost in transit significantly decrease flow matching accuracy. Thus, a logical countermeasure to flow matching is *non-loop* cover traffic, i.e., cover packets addressed to Nym entities other than the sender and present on the merged ingress flows but not the merged egress flows of an endpoints set. This can be achieved either by sending cover packets addressed to a large set of endpoints or by having them dropped at intermediate mixes on their way through the mixnet. Levine et al. proposed a countermeasure in this style called *defensive dropping* [17].

In Nym's current configuration, the adversary is able to tell apart payload and acknowledgment packets based on size. Removing this immediate traffic distinguisher by making acknowledgment packets the same size as payload packets would increase protection against flow matching attacks, at the cost of additional bandwidth. Also, our attacks rely on knowing the average per-mix delay of packets. Should this parameter become less certain (e.g., by adding multiple latency classes [13]) or be entirely unknown to the adversary, flow matching would become much harder.

9 CONCLUSIONS

In this work, we tackle the problem of *flow matching* in mixnets, where an adversary observing both communication ends tries to determine whether two endpoints are communicating via the anonymous network. We first show that existing techniques, developed for connection-oriented networks like Tor, are not well suited for packet-oriented mixnets that include packet reordering and cover traffic. We thus propose novel *flow matching* methods that are applicable to mixnet communications. We demonstrate the effectiveness of our techniques with an empirical analysis of the Nym mixnet.

As part of our solution, we propose and examine three classifiers that the adversary could use for the attack: a statistical classifier that relies on knowledge about the distribution of network delay and two deep learning classifiers that learn to discriminate matched from unmatched flow pairs directly by training on large amounts of network data. We collect six datasets of a file download scenario across different Nym configurations. In a lab setup without real-world network effects, our statistical and drift classifiers quickly deanonymize Nym communications. Under live network conditions, our classifiers' performances are significantly degraded due to packet losses and more variable routing delays, indicating the importance of flow gaps (intended or unintended) for designing countermeasures to this threat.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback that greatly improved this work. We thank Reyhane Attarian for her involvement in the early stages of this research project. Also thanks to Piyush Kumar and Devashish Gosain for discussions and help with understanding RIPE Atlas data.

Multiple people from Nym Technologies SA helped us out with this work. Thanks to Tommy Verrall for providing a version of Nym's Kubernetes-based deployment framework and for answering questions around it. Thanks to Simon Wicky and Ania Piotrowska for meeting with us to discuss results and provide feedback. And thanks to Mark Sinclair for meeting with us to discuss how private Nym gateways can be realized.

Lennart Oldenburg is funded by an FWO fellowship. This research is partially supported by the Research Council KU Leuven under the grant C24/18/049, by VLAIO through the CS ICON project "Cyber Security Artificial Intelligence" (CSAI), and by CyberSecurity Research Flanders with reference number VR20192203. Some of the resources and services used to conduct this work were provided by the Edinburgh Compute and Data Facility (ECDF) at the University of Edinburgh, and the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government.

REFERENCES

[1] Authors of this paper. 2023. Main repository listing this work's source code and datasets. Retrieved December 06, 2023 from https://github.com/mixnet-correlation/mixmatch-flow-matching-for-mixnet-traffic_popets-2024-2

[2] Adam Back, Ulf Möller, and Anton Stiglic. 2001. Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems. In *Information Hiding*. Springer Berlin Heidelberg, 245–257.

[3] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. 2007. Low-Resource Routing Attacks against Tor. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society (WPES '07)*. Association for Computing Machinery, New York, NY, USA, 11–20.

[4] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. 2000. Project "Anonymity and Unobservability in the Internet". In *Proceedings of the Tenth Conference on Computers, Freedom and Privacy: Challenging the Assumptions (CFP '00)*. Association for Computing Machinery, 57–65.

[5] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri De Ruiter, and Alan T. Sherman. 2017. cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations. In *Applied Cryptography and Network Security*. Springer International Publishing, 557–578.

[6] David L. Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (1981), 84–90.

[7] George Danezis. 2005. The Traffic Analysis of Continuous-Time Mixes. In *Privacy Enhancing Technologies*, David Martin and Andrei Serjantov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 35–50.

[8] G. Danezis, R. Dingledine, and N. Mathewson. 2003. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *2003 Symposium on Security and Privacy, 2003.* 2–15.

[9] George Danezis and Ian Goldberg. 2009. Sphinx: A Compact and Provably Secure Mix Format. In *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 269–282.

[10] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. 2021. The Nym Network. Retrieved February 08, 2023 from https://nymtech.net/nym-whitepaper.pdf

[11] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*. USENIX Association.

[12] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. 1996. Hiding Routing information. In *Information Hiding*. Springer Berlin Heidelberg, 137–150.

[13] Iness Ben Guirat, Debajyoti Das, and Claudia Diaz. 2024. Blending Different Latency Traffic With Beta Mixing. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2024 (2024), 15 pages. Issue 2.

[14] C. Gülcü and G. Tsudik. 1996. Mixing E-mail with Babel. In *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. 2–16.

[15] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer, 84–92.

[16] Dogan Kesdogan, Jan Egner, and Roland Büschkes. 1998. Stop- and- Go-MIXes Providing Probabilistic Anonymity in an Open System. In *Information Hiding*, David Aucsmith (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 83–98.

[17] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew Wright. 2004. Timing Attacks in Low-Latency Mix Systems. In *Financial Cryptography*. Springer Berlin Heidelberg, Berlin, Heidelberg, 251–265.

[18] U. Moeller, L. Cottrell, P. Palfrader, and L. Sassaman. 2004. *Mixmaster Protocol Version 2*. Internet-Draft draft-sassaman-mixmaster-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-sassaman-mixmaster/03/ Work in Progress.

[19] 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 (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1962–1976.

[20] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. 2017. Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, 2053–2069.

[21] Nym Technologies SA. n.d. GitHub - nymtech/nym. Retrieved May 27, 2023 from https://github.com/nymtech/nym

[22] Nym Technologies SA. n.d. Nym website. Retrieved February 08, 2023 from https://nymtech.net/

[23] Se Eun Oh, Taiji Yang, Nate Mathews, James K Holland, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2022. DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1915–1932.

[24] Andreas Pfitzmann and Marit Hansen. 2010. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. Retrieved February 15, 2023 from https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf

[25] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1199–1216.

[26] Jean-François Raymond. 2001. *Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems*. Springer Berlin Heidelberg, 10–29.

[27] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. 1998. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications* 16, 4 (1998), 482–494.

[28] RIPE NCC. 2023. RIPE Atlas Daily Dumps. Retrieved May 27, 2023 from https://data-store.ripe.net/datasets/atlas-daily-dumps

[29] Andrei Serjantov and Peter Sewell. 2003. Passive Attack Analysis for Connection-Based Anonymity Systems. In *Computer Security – ESORICS 2003*. Springer Berlin Heidelberg, 116–131.

[30] Vitaly Shmatikov and Ming-Hsiu Wang. 2006. Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses. In *Computer Security – ESORICS 2006*. Springer Berlin Heidelberg, 18–33.

[31] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. 2015. RAPTOR: Routing Attacks on Privacy in Tor. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 271–286.

[32] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. 2001. *Towards an Analysis of Onion Routing Security*. Springer Berlin Heidelberg, 96–114.

[33] The Tor Project. n.d. Tor Metrics. Retrieved May 16, 2023 from https://metrics.torproject.org/

[34] The Tor Project. n.d. Tor Project website. Retrieved February 06, 2023 from https://www.torproject.org/

[35] The Tor Project. n.d. What attacks remain against onion routing? Retrieved May 12, 2023 from https://support.torproject.org/about/attacks-on-onion-routing/

[36] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 137–152.

[37] Hong Xuan, Abby Stylianou, and Robert Pless. 2020. Improved embeddings with easy positive triplet mining. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2474–2482.

[38] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. 2005. On Flow Correlation Attacks and Countermeasures in Mix Networks. In *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 207–225.

[39] L. Øverlier and P. Syverson. 2006. Locating Hidden Servers. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 100–114.

A BRIEF INTRODUCTION TO TOR

With close to three million daily users and 7,000 servers (called *relays*) [33], Tor [11, 34] is the largest and most popular anonymity network currently available. Tor offers bidirectional low-latency *circuits* for clients to anonymously connect to Internet resources using Tor relays as intermediaries. While in principle any TCP-based traffic can be tunneled through Tor, web browsing is the network's primary application. Tor circuits rely on *onion routing* [12, 27] to achieve their privacy properties. They traverse multiple (typically three) Tor relays selected by the Tor client, such that each relay only knows its immediate predecessor and successor in the circuit, but not the full path. To complete the circuit setup, the Tor client shares keys with each of the selected relays. Application payloads exchanged through a Tor circuit are fragmented and padded into fixed-length *cells* and then encrypted multiple times using a key per intermediary relay. Each intermediary relay uses the shared key associated with the circuit to strip (or add) a layer of encryption before forwarding the resulting packet along the circuit. These per-hop cryptographic transformations ensure that cell headers and payloads observed at different points in the network cannot be identified as being the same packet.

B FLOW CORRELATION ATTACKS AGAINST CONNECTION-BASED ANONYMOUS COMMUNICATION SYSTEMS

Back, Möller, and Stiglic [2] describe generic flow correlation attacks (packet counting, latency measurements) against low-latency anonymity system Freedom. Focusing on link-level passive adversaries mainly attacking hop-by-hop, Serjantov and Sewell [29] develop two packet counting attacks against a connection-based anonymity system model that covers many proposed systems at that time. Levine et al. [17] partition observed flows at the two connection ends into non-overlapping time windows for binning packet arrivals. They design the effective countermeasure *defensive dropping*, where packets are lost in the mixnet on purpose to confuse such timing adversary. Shmatikov and Wang [30] also count packets in bins on a set of entry and exit links and compute correlations between bin count vectors for each possible pair. When no defenses are used, their attack achieves an equal error rate close to zero. When traffic is defended with their *adaptive padding* scheme, the equal error rate is close to 0.5, the optimal (defended) value.

Øverlier and Syverson [39] utilize flow correlation to demonstrate how quickly and cheaply specific Tor relays can be identified by an adversary operating a single relay (in 2006). In response to this threat, Tor adopted *entry guards*, i.e., a restricted, small set of eligible first-hop relays for each circuit of a user. Bauer et al. [3] reduce the observations required for flow correlation on Tor to only the cells from circuit establishment collected on compromised first and last circuit positions. Sun et al. [31] exploit specifics of the BGP protocol that is used for Autonomous-System-level routing for Tor flow correlation. They develop a suite of routing attacks called *RAPTOR*, among which their asymmetric traffic analysis achieves 95 % correlation accuracy (using Spearman's rank correlation coefficient) after five minutes of analysis. Nasr, Houmansadr, and Mazumdar [20] improve the scalability of previous flow correlation attacks using techniques from compressed sensing.

C DATA COLLECTION SETUPS

We visualize the isolated setup we use to collect datasets `baseline` (and thus, `two-to-one`), `no-cover`, `low-delay`, and `high-delay` in Figure 9 and the live-network setup we use to collect dataset `live-nym` in Figure 10. We depict each file download duration per dataset collected in Figure 11 (excluding for dataset `two-to-one`, as it is crafted from `baseline`).

D ADDITIONAL EXPERIMENTAL RESULTS

To develop our shape and drift classifiers, we conducted several experiments to evaluate the impact of our design decisions, particularly when they deviated from DeepCoFFEA's design. We measured the effect of incorporating acknowledgments to the test (Figure 12), setting a global threshold (Figure 13), and averaging the scores as an aggregation rule (Figure 14) in the evaluation. In addition, we compare the performance of the new input representation and the representation used by DeepCoFFEA (Figure 15) at coping with loop cover traffic.

Figure 12 shows the ROC curves of the drift classifier trained on dataset `baseline` for tests that include different combinations of packet types. As observed in the figure, including the acknowledgments into the test significantly improves the performance of the resulting model. This result is expected as—in contrast to Tor—acknowledgments in Nym do in fact provide useful information to the classifiers.

Before measuring the impact of any change in DeepCoFFEA's original design, we reproduced the results reported by the authors at a 0.002 loss on their dataset [23]. The ROC curve of our implementation of DeepCoFFEA, shown in Figure 13, is practically identical to the one reported in the DeepCoFFEA paper.

In Figure 13, we also plot the ROC curve when we change the threshold from local (default) to global. The gap between the curves indicates a decrease in performance, which we attribute to the fact that the local threshold is fitted to each individual decision by considering the relative distances to all potential negative examples. As discussed in Section 5, we opt for a global threshold because the adversary that we consider does not have the required visibility into the network.

Building upon these changes, in Figure 14, we again plot the ROC curve of our implementation of DeepCoFFEA with a global threshold that is applying a majority-vote rule. The other line in the graph represents the ROC curve from evaluating that version of the classifier with the average aggregate rule instead. In this case, the performance of the classifier improved after our change, as the average aggregation rule captures fine-grained information about the scores that is overlooked by the majority vote rule.

We also assess the effectiveness of our new representation in addressing the flow matching problem in the presence of loop cover traffic and compare it to DeepCoFFEA's representation. In Figure 15, we show the ROC curves for the two representations, Flows (ours) and Traces (DeepCoFFEA's). A caveat in this comparison is that the windows in the Traces representation are time-based, which result in a different number of observations per window, thus not allowing us to directly compare the two results. For a fairer comparison between the two we take the number of observations in Flows as the average number of observations in Traces. The average number of observations in the first window of Traces is 500, so we consider the first 500 observations to plot the Flows ROC curve.

E DETAILS ON THE DL CLASSIFIERS

For reproducibility, we fixed the PRNG seed and disabled all non-deterministic behavior as much as the GPU libraries allowed.

E.1 Network Architecture

In Figure 17, we depict the architecture of the shape (Figure 17a) and the drift (Figure 17b) networks, including the hyperparameter values that we used for our experiments, such as the number and size of the convolutional kernels and the activation function.

Drift Network. The architecture of the drift network is somewhat similar to the one of DeepCoFFEA. As shown in Figure 17b, the drift network also has multiple hidden convolutional layers with

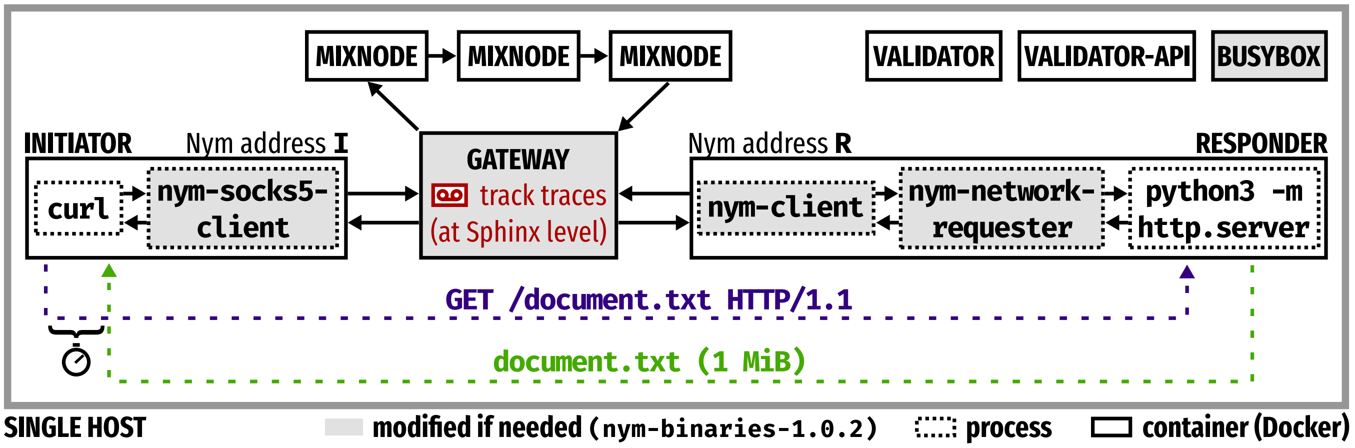


Figure 9: Single-host setup (called *isolated setup*) to collect our datasets over a minimal Nym deployment. Until a target number of succeeded downloads is reached, a freshly created initiator downloads a 1 MiB-sized file via HTTP over Nym from a freshly created responder. The remaining containers persist across all downloads. We track the packet exchange at the Sphinx packets level on the patched gateway via the endpoints' Nym addresses, and log start and end time of the download on the initiator.

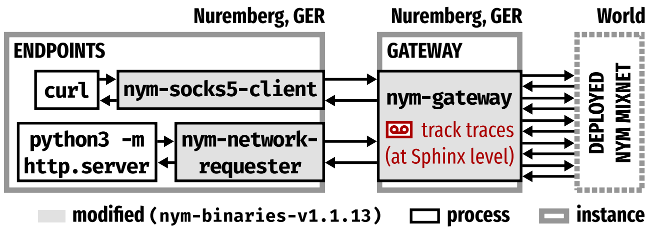


Figure 10: Deployed Nym mainnet experiment setup (called *live-network setup*). We inherit the entity spawning and file download behavior from the isolated setup (Figure 9).

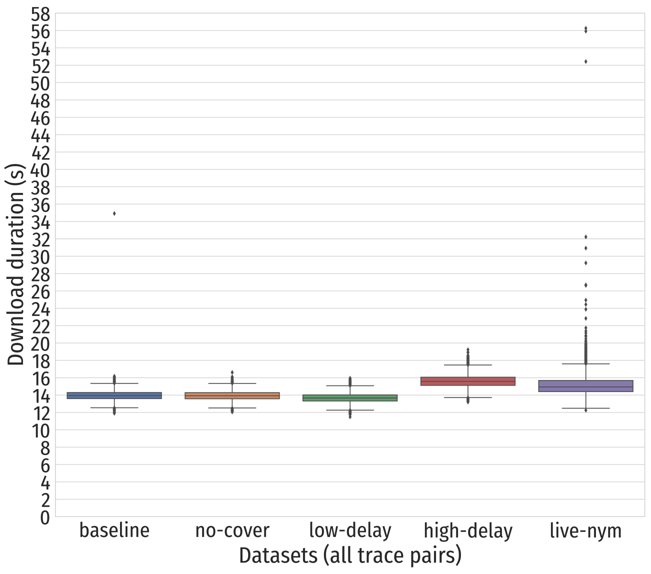


Figure 11: Duration of each successful 1 MiB file download as part of its respective dataset in this study. Mind that all collected trace pairs of each respective dataset are included, not only the first 35,000 we use for analysis in our classifiers.

a ReLU activation function followed by a pooling layer. However, the drift network is much smaller than DeepCoFFEA's, as detecting changes in drift is an easier problem than the problem DeepCoFFEA is designed to tackle. In addition, it uses average pooling instead of max pooling due to the importance of the localization of the drift.

Shape Network. In Figure 17a, we depict the architecture of the shape network. It is a deep convolutional neural network with four blocks. Each block has a double convolutional layer with the ReLU activation function, a max pooling layer, and a dropout layer, except for the last block that does not have a dropout layer. The output of the last block is then flattened and densely connected to the output of the network, which holds the embedding. The figure also details the number and sizes of the kernels.

In the design of the shape classifier, we did not tune for the number of kernels and kernel sizes and kept DeepCoFFEA's values, as the networks have similar architectures and input sizes, and hence require similar receptive fields.

The main difference with DeepCoFFEA's architecture is the input, as we represent our input as flows instead of traces. In addition, we use a ReLU activation function throughout the network, whereas DeepCoFFEA uses ELU in the first block and ReLU elsewhere. This change is because DeepCoFFEA may take negative inputs—direction is encoded in the sign—while our inputs are always positive. Thus, there is no clear justification for using an ELU only in the first block. Finally, for increased regularization, the dropout probability increases over the layers of our network, instead of remaining constant.

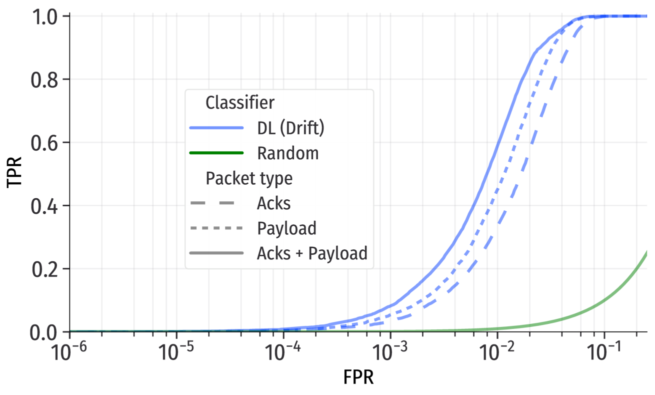


Figure 12: Flow matching performance (ROC curves) of our drift model on dataset `baseline` depending on which packet type is used: only payload packets, only acknowledgment packets, or both.

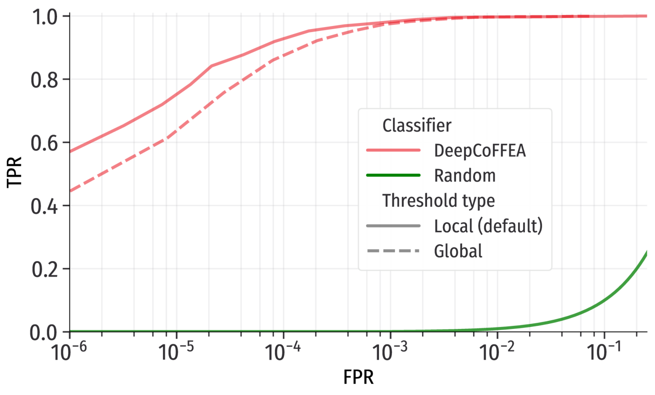


Figure 13: ROC curves of our implementation of DeepCoFFEA for a local and a global threshold. The model was trained and evaluated on DeepCoFFEA's dataset. To reproduce the results in DeepCoFFEA's paper, we trained the models until reaching a loss value of 0.002.

E.2 Hyperparameter Tuning

We use Bayesian optimization to search the space of hyperparameters and find the ones that provide the best results. Bayesian optimization is often used for tuning the hyperparameters of deep learning algorithms as they can find minimum points of the loss function faster than grid-search techniques. In Table 1, we list all the hyperparameters of the learning algorithms and the search spaces that we have considered during hyperparameter tuning.

By comparing the learning curves on the training and validation sets, we evaluated the convergence rates for a set of hyperparameters and detected cases of under- and overfitting. In this evaluation of the hyperparameters, we did not always consider all the hyperparameters together, but also conducted a more in-depth analysis for some important hyperparameters, such as window size and overlap.

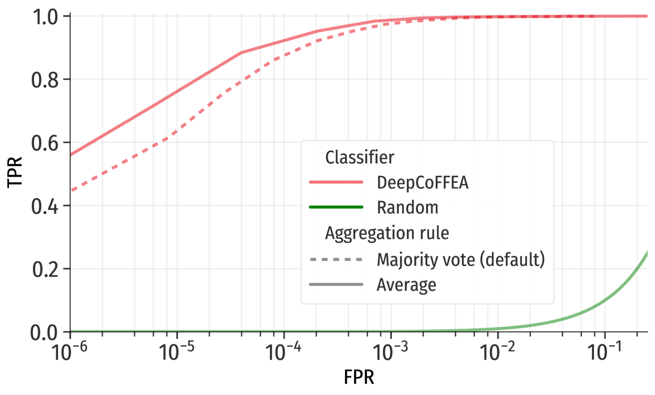


Figure 14: ROC curves of our implementation of DeepCoFFEA for different aggregation rules (average vs. maximum hit count) on DeepCoFFEA's dataset.

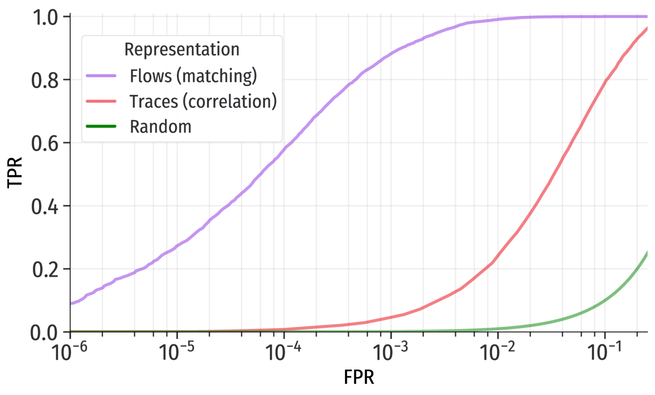


Figure 15: ROC curves of the baseline shape model for two different input representations: Traces and Flows. The Flows representation is the packet-based window representation of flows that we used in the rest of the paper. The Traces representation is the time-based window representation of traffic traces used by DeepCoFFEA. For a fair comparison, we take the first 500 packets for the Flows representation, as the first window of the Traces representation has approximately 500 packets on average.

The last column of Table 1 lists the best values of the hyperparameters in our evaluation. We found that many of the hyperparameters did not significantly contribute to improving the convergence of the learning algorithms. For example, for a fixed window size, a $\delta > 0$ overlap does not significantly improve convergence. For a fixed overlap value, smaller window sizes increase the computational demand, but do not seem to either improve the performance—at least for the window sizes that we tested.

The last two rows of the table show hyperparameters that are exclusive for the shape network. We observe that different margins have an impact on convergence: looser margins relax the condition in Equation (12) allowing to find more semi-hard negatives, but the semi-hard negatives provide less information and the convergence

Table 1: Deep learning algorithm hyperparameters, the search space for tuning, and the final value that we chose to present the results in Section 7.

Hyperparameter	Search space	Chosen value
Batch size	{8, 16, 32}	16
Training epochs	$\{10, 50, 100\}$	100
Learning rate	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}
Optimizer	SGD, NAdam	-
Window size (n_w)	$\{100, 200, 250, 300, 500\}$	100
Window overlap (δ)	$\{0.00, 0.25, 0.5, 0.75\}$	0.00
Margin (α)	{0.05, 0.10, 0.15}	0.10
Embedding size (s)	64	64

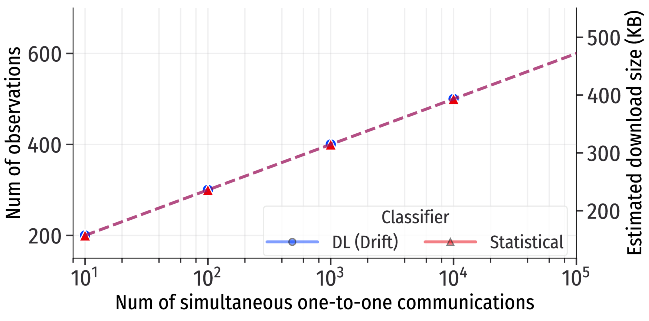


Figure 16: Number of `baseline` observations needed for a TPR of over 0.95 with at least 0.95 precision and a given number of simultaneous one-to-one communications (base rate).

is slower, and vice versa. Like with DeepCoFFEA, the best value in our tests was $\alpha = 0.1$.

Finally, we observed that the NAdam optimizer would converge faster than SGD in most experiments. However, for the same set of hyperparameters, NAdam would result in vanishing gradients in the experiments conducted on the Nym live network, so we used the SGD optimizer when training on the `live-nym` dataset.

F IMPACT OF BASE RATE (SIMULTANEOUS COMMUNICATIONS)

In Figure 16, we show how the required number of observations to achieve high classifier performance (TPR ≥ 0.95) with high confidence (precision ≥ 0.95) on `baseline` evolves when the number of simultaneous communications increases.

G HARDWARE SPECIFICATIONS

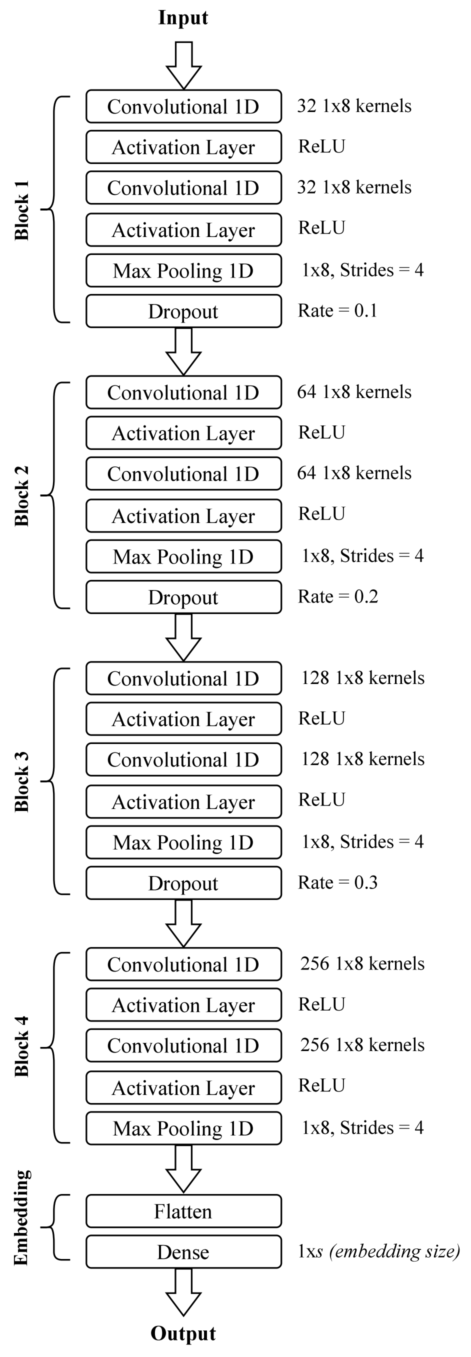
Computing time is proportional to the number of packets in the traces and inversely proportional to the number of employed CPUs. Thus, it is approximately the same for all experiments, though slightly shorter for the `no-cover` experiment due to the absence of loop cover packets in its traces.

We describe the hardware specifications that we used for evaluating each classifier below.

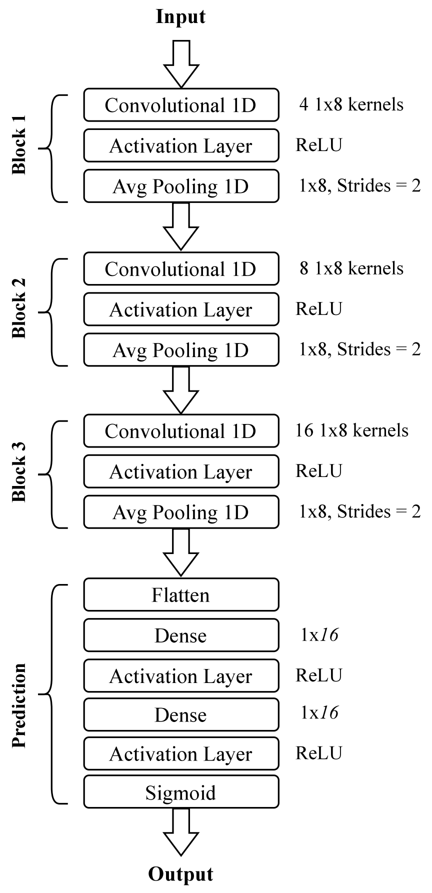
Statistical Classifier. We ran the evaluation of the statistical classifier in parallel by dividing the score matrix into a grid and processing each submatrix using `Octave` scripts. As an illustrative figure, the experiment on the `live-nym` dataset took approximately three days on a machine with 48 CPUs and 192GB of RAM.

DL Classifiers. For the training and evaluation of the neural-network-based classifiers we used a machine with 64 CPUs (two sockets with 16 cores per sockets and two threads per core) and 192GB of RAM. In particular, most of the training was performed on an NVIDIA GeForce RTX 3080 with 10GB of memory.

Training a model on the `baseline` dataset for 100 epochs takes approximately five days for the shape and two days for the drift classifier. The evaluation is divided into batches and takes approximately five hours for each model.



(a) Shape network.



(b) Drift network.

Figure 17: Network architectures of our DL classifiers.