DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning

Milad Nasr
University of Massachusetts Amherst
milad@cs.umass.edu

Alireza Bahramali
University of Massachusetts Amherst
abahramali@cs.umass.edu

Amir Houmansadr
University of Massachusetts Amherst
amir@cs.umass.edu

ABSTRACT

Flow correlation is the core technique used in a multitude of deanonymization attacks on Tor. Despite the importance of flow correlation attacks on Tor, existing flow correlation techniques are considered to be ineffective and unreliable in linking Tor flows when applied at a large scale, i.e., they impose high rates of false positive error rates or require impractically long flow observations to be able to make reliable correlations. In this paper, we show that, unfortunately, flow correlation attacks can be conducted on Tor traffic with drastically higher accuracies than before by leveraging emerging learning mechanisms. We particularly design a system, called DeepCorr, that outperforms the state-of-the-art by significant margins in correlating Tor connections. DeepCorr leverages an advanced deep learning architecture to *learn* a flow correlation function tailored to Tor's complex network—this is in contrast to previous works' use of generic statistical correlation metrics to correlate Tor flows. We show that with moderate learning, DeepCorr can correlate Tor connections (and therefore break its anonymity) with accuracies significantly higher than existing algorithms, and using substantially shorter lengths of flow observations. For instance, by collecting only about 900 packets of each target Tor flow (roughly 900KB of Tor data), DeepCorr provides a flow correlation accuracy of 96% compared to 4% by the state-of-the-art system of RAPTOR using the same exact setting.

We hope that our work demonstrates the escalating threat of flow correlation attacks on Tor given recent advances in learning algorithms, calling for the timely deployment of effective countermeasures by the Tor community.

CCS CONCEPTS

• **Information systems** → **Traffic analysis**; • **Security and privacy** → **Pseudonymity, anonymity and untraceability**; *Privacy-preserving protocols*; • **Networks** → **Network privacy and anonymity**;

KEYWORDS

Traffic Analysis; Tor; Flow Correlation Attacks; Anonymous Communications

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10.
https://doi.org/10.1145/3243734.3243824

ACM Reference Format:
Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. DeepCorr: Strong Flow Correlation Attacks on Tor, Using Deep Learning. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3243734.3243824

1 INTRODUCTION

Tor [16] is the most widely used anonymity system with more than 2 million daily users [74]. It provides anonymity by relaying clients' traffic through cascades of relays, known as onion-circuits, therefore concealing the association between the IP addresses of the communicating parties. Tor's network comprises around 7,000 public relays, carrying terabytes of traffic every day [74]. Tor is used widely not only by dissidents, journalists, whistleblowers, and businesses, but also by ordinary citizens to achieve anonymity and blocking resistance.

To be usable for everyday Internet activities like web browsing, Tor aims to provide *low-latency* communications. To make this possible, Tor relays refrain from obfuscating traffic features like packet timings as doing so will slow down the connections.[1] Consequently, Tor is known to be susceptible to *flow correlation* attacks [14, 51, 68] in which an adversary tries to link the egress and ingress segments of a Tor connection by comparing their traffic characteristics, in particular their packet timings and packet sizes.

This paper studies flow correlation attacks on Tor. Flow correlation is **the core technique** used in a wide spectrum of the attacks studied against Tor (and similar anonymity systems) [8, 20, 36, 38, 70, 72]. For instance, in the predecessor attack [83] an adversary who controls/eavesdrops multiple Tor relays attempts at deanonymizing Tor connections by applying flow correlation techniques. The Tor project adopted "guard" relays to limit such an adversary's chances of placing herself on the two ends of a target Tor connection. Borisov et al. [8] demonstrated an active denial-of-service attack that increases an adversary's chances of observing the two ends of a target user's Tor connections (who then performs flow correlation). Alternatively, various routing attacks have been presented on Tor [20, 38, 70, 72] that aim at increasing an adversary's odds of intercepting the flows to be correlated by manipulating the routing decisions.

Despite the critical role of flow correlation in a multitude of Tor attacks, flow correlating Tor connections has long been considered to be inefficient at scale [37, 55, 66]—but not anymore! Even though Tor relays do not actively manipulate packet timings and sizes to resist flow correlation, the Tor network naturally perturbs Tor packets by significant amounts, rendering flow correlation a

[1] Note that some Tor bridges (but not the public relays) obfuscate traffic characteristics of the Tor flows between themselves and censored clients by using various Tor pluggable transports [61].

difficult problem in Tor. Specifically, Tor connections experience large network jitters, significantly larger than normal Internet connections. Such large perturbations are resulted by congestion on Tor relays, which is due to the imbalance between Tor's capacity and the bandwidth demand from the clients. Consequently, existing flow correlation techniques [34, 45, 53, 72] suffer from high rates of false positives and low accuracies, unless they are applied on very long flow observations and/or impractically small sets of target flows. For instance, the state-of-the-art flow correlation of RAPTOR [72] achieves good correlation performance in distinguishing a small set of only 50 target connections, and even this requires the collection of 100 MB over 5 minutes of traffic for each of the intercepted flows.

In this work, we take flow correlation attacks on Tor to reality. We develop tools that are able to correlate Tor flows with accuracies *significantly higher* than the state-of-the-art—when applied to large anonymity sets and using very short observations of Tor connections. We argue that existing flow correlation techniques [13, 34, 45, 53, 68, 72] are inefficient in correlating Tor traffic as they make use of generic statistical correlation algorithms that are not able to capture the dynamic, complex nature of noise in Tor. As opposed to using such general-purpose statistical correlation algorithms, in this paper we use deep learning to *learn a correlation function that is tailored to Tor's ecosystem.* Our flow correlation system, called DeepCorr, then uses the learned correlation function to cross-correlate live Tor flows. Note that contrary to website fingerprinting attacks [10, 27, 58, 75, 76], DeepCorr does *not* need to learn any target destinations or target circuits; instead DeepCorr learns a correlation function that can be used to link flows on *arbitrary circuits*, and to *arbitrary destinations.* In other words, DeepCorr can correlate the two ends of a Tor connection even if the connection destination has not been part of the learning set. Also, DeepCorr can correlate flows even if they are sent over Tor circuits different than the circuits used during the learning process. This is possible as DeepCorr's neural network learns the *generic* features of noise in Tor, regardless of the specific circuits and end-hosts used during the training process.

We demonstrate DeepCorr's strong performance through large scale experiments on live Tor network. We browse the top 50,000 Alexa websites over Tor, and evaluate DeepCorr's true positive and false positive rates in correlating the ingress and egress segments of the recorded Tor connections. To the best of our knowledge, our dataset is the largest dataset of correlated Tor flows, which we have made available to the public.[2] Our experiments show that DeepCorr can correlate Tor flows with accuracies *significantly superior* to existing flow correlation techniques. For instance, compared to the state-of-the-art flow correlation algorithm of RAPTOR [72], DeepCorr offers a correlation accuracy[3] of 96% compared to RAPTOR's accuracy of 4% (when both collect 900 packets of traffic from each of the intercepted flows)! The following is a highlight of DeepCorr's performance:

- We use a total of 25,000 Tor flows collected by ourselves to train DeepCorr (we use 5,000 flows for training in most of our experiments). Training DeepCorr takes about a day on a single TITAN X GPU, however we show that an adversary needs to re-train DeepCorr roughly *once a month* to preserve its correlation performance.
- DeepCorr can be used as a generic correlation function: DeepCorr's performance is consistent for various test datasets with different sizes and containing flows routed over different circuits.
- DeepCorr outperforms prior flow correlation algorithms by very large margins. Importantly, DeepCorr enables the correlation of Tor flows with flow observations much shorter than what is needed by previous work. For instance, with only 300 packets, DeepCorr achieves a true positive rate of 0.8 compared to less than 0.05 by prior work (for a fixed false positive rate of 10^{-3}).
- DeepCorr's performance rapidly improves with longer flow observations and with larger training sets.
- DeepCorr's correlation time is significantly faster than previous work for the same target accuracy. For instance, each DeepCorr correlation takes 2ms compared to RAPTOR's more than 20ms, when both target a 95% accuracy on identical dataset.

We hope that our study raises concerns in the community on the escalating risks of large-scale traffic analysis on Tor communications in light of the emerging deep learning algorithms. A possible countermeasure to DeepCorr is deploying traffic obfuscation techniques, such as those employed by Tor pluggable transports [61], on *all* Tor traffic. We evaluate the performance of DeepCorr on each of Tor's currently-deployed pluggable transports, showing that meek and obfs4-iat0 provide little protection against DeepCorr's flow correlation, while obfs4-iat1 provides a better protection against DeepCorr (note that none of these obfuscation mechanisms are currently deployed by *public* Tor relays, and even obfs4-iat1 is deployed by a small fraction of Tor bridges [55]). This calls for designing effective traffic obfuscation mechanisms to be deployed by Tor relays that do not impose large bandwidth and performance overheads on Tor communications.

Finally, note that while we present DeepCorr as a flow correlation attack on Tor, it can be used to correlate flows in other flow correlation applications as well. To demonstrate this, we also apply DeepCorr to the problem of stepping stone detection [6, 26, 80] showing that DeepCorr significantly outperforms previous stepping stone detection algorithms in unreliable network settings.

Organization: The rest of this paper is organized as follows. In Section 2, we overview preliminaries of flow correlation and motivate our work. In Section 3, we introduce our flow correlation system, called DeepCorr. We describe our experimental setup in Section 4, and present and discuss our experimental results in Section 5. We discuss and evaluate possible countermeasures against DeepCorr in Section 6 and conclude the paper in Section 7.

2 PRELIMINARIES AND MOTIVATION

Flow correlation attacks, also referred to as *confirmation attacks*, are used to *link network flows* in the presence of encryption and

[2] https://people.cs.umass.edu/~amir/FlowCorrelation.html

[3] To be fair, in our comparison with RAPTOR we derive the accuracy metric similar to RAPTOR's paper [72]: each flow is paired with only one flow out of all evaluated flows. For the rest of our experiments, each flow can be declared as correlated with arbitrary number of intercepted flows, which is a more realistic (and more challenging) setting.

other content obfuscation mechanisms [14, 18, 26, 46, 53, 68, 81, 86]. In particular, flow correlation techniques can break anonymity in anonymous communication systems like Tor [16] and mix networks [15, 64, 65] by linking the egress and ingress segments of the anonymous connections through correlating traffic features [4, 14, 51, 63, 68, 78, 79, 87]. Alternatively, flow correlation techniques can be used to identify cybercriminals who use network proxies to obfuscate their identities, i.e., stepping stone attackers [69, 84, 86].

2.1 Threat Model

Figure 1 shows the main setting of a flow correlation scenario. The setting consists of a computer network (e.g., Tor's network) with M ingress flows and N egress flows. Some of the egress flows are the obfuscated versions of some of the ingress flows; however, the relation between such flows can not detected using packet contents due to the use of encryption and similar content obfuscation techniques like onion encryption. For instance, in the case of Tor, F_i and F_j are the entry and exit segments of one Tor connection (see Figure 1), however, such association can not be detected by inspecting the packet contents of F_i and F_j due to onion encryption. We call (F_i, F_j) a pair of *associated flows*.

The goal of an adversary in this setting is to identify (some or all of) the associated flow pairs, e.g., (F_i, F_j), by comparing traffic characteristics, e.g., packet timings and sizes, across all of the ingress and egress flows. Linking associated flow pairs using traffic characteristics is called flow correlation.

A flow correlation adversary can intercept network flows at various network locations. A Tor adversary, in particular, can intercept Tor flows either by running malicious Tor relays [8, 36, 83] or by controlling/wiretapping Internet ASes or IXPs [39, 70, 72]. We further elaborate on this in Section 2.3.

Note that in this paper we study *passive* flow correlation attacks only; therefore, *active* flow correlation techniques, also known as flow watermarks as introduced in Section 2.5, are out of the scope of this paper. Also, flow correlation is different from website fingerprinting attacks, as discussed in Section 2.5.

2.2 Existing Flow Correlation Techniques

As mentioned before, flow correlation techniques use traffic features, particularly, packet timings, packet sizes, and their variants (e.g., flow rates, inter-packet delays, etc.), to correlate and link network flows (recall that packet contents can *not* be used to link flows in this setting due to content obfuscation, e.g., onion encryption). For instance, the early work of Paxson and Zhang [86] models packet arrivals as a series of ON and OFF patterns, which they use to correlate network flows, and Blum et al. [7] correlate the aggregate sizes of network packets over time. Existing flow correlation techniques mainly use *standard statistical correlation metrics* to correlate the vectors of flow timings and sizes across flows. In the following, we overview the major types of statistical correlation metrics used by previous flow correlation algorithms.

Mutual Information The mutual information metric measures the dependency of two random variables. It, therefore, can be used to quantify the correlation of flow features across flows, e.g., the traffic features of an egress Tor flow depends on the features of its corresponding ingress flow. The mutual information technique has been used by Chothia et al. [13] and Zhu et al. [88] to link flows. This metric, however, requires a long vector of features (e.g., long flows) in order to make reliable decisions, as it needs to reconstruct and compare the empirical distributions of traffic features of target flows.

Pearson Correlation The Pearson Correlation coefficient is a classic statistical metric for linear correlation between random variables. Unlike the mutual information metric, the Pearson Correlation metric does not need to build the empirical distribution of the variables it is correlating, and therefore can be applied on a shorter length of data. The Pearson Correlation metric has been used by several flow correlation systems [45, 68].

Cosine Similarity The Cosine similarity metric measures the angular similarity of two random variables. Similar to the Pearson coefficient, it can be directly applied on the sample vectors of two random variables. This metric has been used by different timing and size correlation systems [34, 53] to link network flows.

Spearman Correlation The Spearman rank correlation metric measures the statistical dependence between the rankings of two variables. The metric can be defined as the Pearson correlation between ranked variables. The recent work of RAPTOR [72] uses this metric to correlate Tor flows.

2.3 Flow Correlation Attacks on Tor

Flow correlation is the *core technique* used in a broad range of attacks studied against Tor (and other anonymity systems). To be able to perform flow correlation, an adversary needs to observe (i.e., intercept) some fraction of flows entering and exiting the Tor network. The adversary can then deanonymize a specific Tor connection, if she is able to intercept both of the ingress and egress segments of that Tor connection (by performing a flow correlation algorithm on those flow segments). Therefore, an adversary can increase her chances of deanonymizing Tor connections by trying to intercept a larger fraction of Tor's ingress and egress flows.

There are two main approaches an attacker can take to increase the fraction of Tor connections she is intercepting. First, by running a large number of Tor relays and recording the traffic features of the Tor connections they relay. Various studies have shown that an adversary with access to such malicious relays can increase her chances of intercepting the both ends of a Tor connection in different ways [3, 8, 28, 49, 83]. For instance, Borisov et al. [8] demonstrate an active denial-service-attack to increase the chances of intercepting the ingress and egress segments of a target client's Tor traffic. The Tor project has adopted the concept of Tor guard relays [21] to reduce the chances of performing flow correlation by an adversary controlling malicious relays, an attack known as the predecessor attack [83].

Alternatively, an adversary can increase her opportunities of performing flow correlation by controlling/wiretapping autonomous systems (ASes) or Internet exchange points (IXPs), and recording the traffic features of the Tor connections that they transit. Several studies [22, 52, 72] demonstrate that specific ASes and IXPs intercept a significant fraction of Tor traffic, therefore are capable of performing flow correlation on Tor at large scale. Others [20, 38, 39, 70, 72] show that an AS-level adversary can further

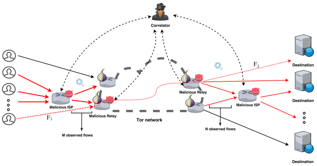

Figure 1: The main setting of a flow correlation attack on Tor. The adversary intercepts Tor flows either by running malicious Tor relays or wiretapping Internet ASes and IXPs.

increase her chances of flow correlation by performing various routing manipulations that reroute a larger fraction of Tor connections through her adversarial ASes and IXPs. For instance, Starov et al. [70] recently show that approximately 40% of Tor circuits are vulnerable to flow correlation attacks by a single malicious AS, and Sun et al. [72] show that churn in BGP as well as active manipulation of BGP updates can amplify an adversarial AS's visibility on Tor connections. This has lead to various proposals on deploying AS-aware path selection mechanisms for Tor [2, 20, 54].

2.4 This Paper's Contributions

While flow correlation is the core of a multitude of attacks on Tor [3, 8, 20, 22, 28, 38, 39, 49, 52, 54, 70, 72, 72, 83], existing flow correlation algorithms are assumed to be ineffective in linking Tor connections reliably and at scale [37, 55, 66]. This is due to Tor's extremely noisy network that applies large perturbations on Tor flows, therefore rendering traffic features across associated ingress and egress Tor flows hard to get reliably correlated. In particular, Tor's network applies large network jitters on Tor flows, which is due to congestion on Tor relays, and many Tor packets are fragmented and repacketized due to unreliable network conditions. Consequently, existing flow correlation techniques offer poor correlation performances—unless applied to very large flow observations as well as unrealistically small sets of target flows.[4] For instance, the state-of-the-art correlation technique of Sun et al. [72] needs to observe 100MB of traffic from each target flow for around 5 minutes to be able to perform reliable flow correlations. Such long flow observations not only are impractical due to the short-lived nature of typical Tor connections (e.g., web browsing sessions), but also impose unbearable storage requirements if applied at large scale (e.g., a malicious Tor relay will likely intercepte tens of thousands of concurrent flows). Moreover, existing techniques suffer from high rates of false positive correlations unless applied on an unrealistically small set of suspected flows, e.g., Sun et al. [72] correlate among a set of only 50 target flows.

Our Approach: We believe that the main reason for the ineffectiveness of existing flow correlation techniques is the intensity as well as the unpredictability of network perturbations in Tor. We argue that previous flow correlation techniques are inefficient in correlating Tor traffic since they make use of general-purpose statistical correlation algorithms that are not able to capture the dynamic, complex nature of noise in Tor. As opposed to using such generic statistical correlation metrics, in this paper we use deep learning to *learn a correlation function that is tailored to Tor's ecosystem*. We design a flow correlation system, called DeepCorr, that learns a flow correlation function for Tor, and uses the learned function to cross-correlate live Tor connections. Note that contrary to website fingerprinting attacks [10, 27, 58, 75, 76], DeepCorr does *not* need to learn any target destinations or target circuits; instead DeepCorr learns a correlation function that can be used to link flows on *arbitrary circuits*, and to *arbitrary destinations*. In other words, DeepCorr can correlate the two ends of a Tor connection even if the connection destination has not been part of the learning set. Also, DeepCorr can correlate flows even if they are sent over Tor circuits different than the circuits used during the training process.

We demonstrate DeepCorr's strong correlation performance through large scale experiments on live Tor network, which we compare to previous flow correlation techniques. We hope that our study raises concerns in the community on the increasing risks of large-scale traffic analysis on Tor in light of emerging learning algorithms. We discuss potential countermeasures, and evaluate DeepCorr's performance against existing countermeasures.

2.5 Related Topics Out of Our Scope

Active flow correlation (watermarking) Network flow watermarking is an *active* variant of the flow correlation techniques introduced above. Similar to passive flow correlation schemes, flow watermarking aims at linking network flows using traffic features that

[4] Note that *active* attacks like [68] are out of our scope, as discussed in Section 2.5, since such attacks are easily detectable, and therefore can *not* be deployed by an adversary at large scale for a long time period without being detected.

persist content obfuscation, i.e., packet sizes and timings. By contrast, flow watermarking systems need to *manipulate* the traffic features of the flows they intercept in order to be able to perform flow correlation. In particular, many flow watermarking systems [29–31, 33, 62, 79, 85] perturb packet timings of the intercepted flows by slightly delaying network packets to modulate an artificial pattern into the flows, called the watermark. For instance, RAINBOW [33] manipulates the inter-packet delays of network packets in order to embed a watermark signal. Several proposals [32, 44, 62, 79, 85], known as interval-based watermarks, work by delaying packets into secret time intervals.

While passive flow correlation attacks (studied in this paper) are information theoretically undetectable, a watermarking adversary may reveal herself by applying traffic perturbations that differ from that of normal traffic. Some active correlation techniques [12, 68] do not even aim for invisibility, therefore they can be trivially detected and disabled, making them unsuitable for large scale flow correlation. Additionally, while passive flow correlation algorithms can be computed offline, flow watermarks need to be performed by resourceful adversaries who are able to apply traffic manipulations on live Tor connections. In this paper, we only focus on passive flow correlation techniques.

Website Fingerprinting Website fingerprinting attacks [10, 24, 25, 27, 40, 47, 57, 58, 75–77] use a different threat model than flow correlation techniques. In website fingerprinting, an adversary intercepts a target client's ingress Tor traffic (e.g., by wiretapping the link between a Tor client and her guard relay), and compares the intercepted ingress Tor connection to the traffic fingerprints of a finite (usually small) set of target websites. This is unlike flow correlation attacks in which the adversary intercepts the *two ends* of an anonymous connection, enabling the attacker to deanonymize *arbitrary* senders and receivers. Existing website fingerprinting systems leverage standard machine learning algorithms such as SVM and kNN to classify and identify target websites, and recent work [67] has investigated the use of deep learning for website fingerprinting. In contrary, as overviewed in Section 2.2, prior passive flow correlation techniques use statistical correlation metrics to link traffic characteristics across network flows. We consider website fingerprinting orthogonal to our work as it is based on different threat model and techniques.

3 INTRODUCING DeepCorr

In this section, we introduce our flow correlation system, called DeepCorr, which uses deep learning algorithms to learn correlation functions.

3.1 Features and Their Representation

Similar to existing flow correlation techniques overviewed earlier, our flow correlation system uses the timings and sizes of network flows to cross-correlate them. A main advantage [23] of deep learning algorithms over conventional learning techniques is that a deep learning model can be provided with *raw* data features as opposed to engineered traffic features (like those used by SVM- and kNN-based website fingerprinting techniques [10, 24, 25, 27, 47, 57, 58, 75, 76]). This is because deep learning is able to extract complex, effective features from the raw input features [23] itself. Therefore, DeepCorr takes raw flow features as input, and uses them to derive complex features, which is used by its correlation function.

We represent a bidirectional network flow, i, with the following array:

$$F_i = [T_i^u; S_i^u; T_i^d; S_i^d]$$

where T is the vector of inter-packet delays (IPD) of the flow i, S is the vector of i'th packet sizes, and the u and d superscripts represent "upstream" and "downstream" sides of the bidirectional flow i (e.g., T_i^u is the vector of upstream IPDs of i). Also, note that we only use the first ℓ elements of each of the vectors, e.g., only the first ℓ upstream IPDs. If a vector has fewer than ℓ elements, we pad it to ℓ by appending zeros. We will use the flow representation F_i during our learning process.

Now suppose that we aim at correlating two flows i and j (say i was intercepted by a malicious Tor guard relay and j was intercepted by an accomplice exit relay). We represent this pair of flows with the following two-dimensional array composed of 8 rows:

$$F_{i,j} = [T_i^u; T_j^u; T_i^d; T_j^d; S_i^u; S_j^u; S_i^d; S_j^d]$$

where the lines of the array are taken from the flow representations F_i and F_j.

3.2 Network Architecture

We use a Convolutional Neural Network (CNN) [23] to learn a correlation function for Tor's noisy network. We use a CNN since network flow features can be modeled as time series, and the CNNs are known to have good performance on time series [23]. Also, the CNNs are invariant to the position of the patterns in the data stream [23], which makes them ideal to look for possibly shifted traffic patterns.[5]

Figure 2 shows the structure of DeepCorr's CNN network. The network takes a flow pair $F_{i,j}$ as the input (on the left side). DeepCorr's architecture is composed of two layers of convolution and three layers of a fully connected neural network. The first convolution layer has k_1 kernels each of size $(2, w_1)$, where k_1 and w_1 are the hyperparameters, and we use a stride of $(2, 1)$. The intuition behind using the first convolution layer is to capture correlation between the adjacent rows of the input matrix $F_{i,j}$, which are supposed to be correlated for associated Tor flows, e.g., between T_i^u and T_j^u.

DeepCorr's second convolution layer aims at capturing traffic features from the combination of all timing and size features. At this layer, DeepCorr uses k_2 kernels each of size $(4, w_2)$, where k_2 and w_2 are also our hyperparameters, and it uses a stride of $(4, 1)$.

The output of the second convolution layer is flattened and fed to a fully connected network with three layers. DeepCorr uses max pooling after each layer of convolution to ensure permutation invariance and to avoid overfitting [23]. Finally, the output of the network is:

$$p_{i,j} = \Psi(F_{i,j})$$

[5]Note that our work is the *first* to use a learning mechanism for flow correlation. In our search of effective learning mechanisms for flow correlation, we tried various algorithms including fully connected neural networks, recurrent neural network (RNN), and support vector machine (SVM). However, CNN provided the best flow correlation performance compared to all the other algorithms we investigated, which is intuitively because CNNs are known to work better for longer data lengths. For instance, we achieved an accuracy of only 0.4 using fulling-connected neural networks, which is significantly lower than our performance with CNNs.

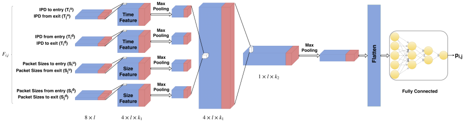

Figure 2: The network architecture of DeepCorr.

which is used to decide if the two input flows in $F_{i,j}$ are correlated or not. To normalize the output of the network, we apply a sigmoid function [23] that scales the output between zero and one. Therefore, $p_{i,j}$ shows the probability of the flows i and j being associated (correlated), e.g., being the entry and exit segments of the same Tor connection.

DeepCorr declares the flows i and j to be correlated if $p_{i,j} > \eta$, where η is our *detection threshold* discussed during the experiments.

The parameters (w_1, w_2, k_1, k_2) are the hyperparameters of our system; we will tune their values through experiments.

3.3 Training

To train our network, we use a large set of flow pairs that we created over Tor. This includes a large set of associated flow pairs, and a large set of non-associated flow pairs. An associated flow pair, $F_{i,j}$, consists of the two segments of a Tor connection (e.g., i and j are the ingress and egress segments of a Tor connection). We label an associated pair with $y_{i,j} = 1$. On the other hand, each non-associated flow pair (i.e., a negative sample) consists of two arbitrary Tor flows that do not belong to the same Tor connection. We label such non-associated pairs with $y_{i,j} = 0$. For each captured Tor entry flow, i, we create N_{neg} negative samples by forming $F_{i,j}$ pairs where j is the exit segment of an arbitrary Tor connection. N_{neg} is a hyperparameter whose value will be obtained through experiments.

Finally, we define DeepCorr's loss function using a cross-entropy function as follows:

$$\mathcal{L} = -\frac{1}{|\mathcal{F}|} \sum_{F_{i,j} \in \mathcal{F}} y_{i,j} \log \Psi(F_{i,j}) + (1 - y_{i,j}) \log(1 - \Psi(F_{i,j})) \quad (1)$$

where $\mathcal{F}$ is our training dataset, composed of all associated and non-associated flow pairs. We used the Adam optimizer [43] to minimize the loss function in our experiments. The learning rate of the Adam optimizer is another hyperparameter of our system.

4 EXPERIMENTAL SETUP

In this section, we discuss our data collection and its ethics, the choice of our hyperparameters, and our evaluation metrics.

4.1 Datasets and Collection

Figure 3 shows our experimental setup for our Tor experiments. We used several Tor clients that we ran inside separate VMs to generate and collect Tor traffic. We use each of our Tor clients to browse the top 50,000 Alexa websites over Tor, and captured the flows entering and exiting the Tor network for these connections (we use half of these flows for training in various experiments). Therefore, the entering flows are in Tor cell format, and the flows exiting Tor are in regular HTTP/HTTPS format. We used 1,000 arbitrary Tor circuits for browsing websites over Tor, i.e., each circuit was used to browse roughly 50 websites. We used different guard nodes in forming our Tor circuits; we were able to alternate our guard nodes by disabling Vanilla Tor's option that enforces guard relay reuse. We also used a regular Firefox browser, instead of Tor's browser, to be able to enforce circuit selection. We used Tor version 0.3.0.9, automated by a Python script.

Note that we did not set up our own Tor relays for the purpose of the experiments, and we merely used public Tor relays in all of our experiments. We captured the ingress Tor flows using `tcpdump` on our Tor clients. To capture the egress Tor traffic (i.e., traffic from exit relays to websites), we made our exit Tor traffic tunnel through our own SOCKS proxy server (as shown in Figure 3), and we collected the exit Tor traffic on our own SOCKS proxy server using `tcpdump`. Note that using this data collection proxy may add additional latency on the collected flows, so the performance of DeepCorr in practice is better than what we report through experiments. We also collected 500 websites through Tor pluggable transport to evaluate them as countermeasures against DeepCorr.

We collected our Tor traffic in two steps: first, we collected traffic over a two weeks period, and then with a three months gap we collected more Tor traffic for a one month period (in order to show the impact of time on training). We have made our dataset available publicly. To the best of our knowledge, this is largest dataset of

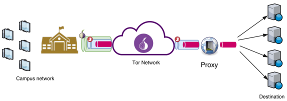

Figure 3: Our experimental setup on Tor

correlated Tor flows, and we hope it will be useful to the research community.

Note that while we only collect web traffic, this is not a constraint of DeepCorr, and it can be used to correlate arbitrary Tor traffic.

4.2 Ethics of Data Collection

To make sure we did not overload Tor's network, we ran up to 10 concurrent Tor connections during our data collection. Also, we alternated the guard nodes used in our circuits to evade overloading any specific circuits or relays. We did not browse any illegal content over Tor, and we used an idle time between connections of each of our clients. As explained above, we collected our ingress and egress Tor flows on our own Tor clients as well as our own SOCKS proxy server; therefore, we did *not* collect any traffic of other Tor users.

In our experiments with Tor pluggable transports, we collected a much smaller set of flows compared to our bare Tor experiments; we did so because Tor bridges are very scarce and expensive, and therefore we avoided overloading the bridges.

4.3 Choosing the Hyperparameters

We used Tensorflow [1] to implement the neural networks of DeepCorr. We tried various values for different hyperparameters of our system to optimize the flow correlation performance. To optimize each of the parameters, our network took about a day to converge (we used a single Nvidia TITAN X GPU).

For the learning rate, we tried $\{0.001, 0.0001, 0.0005, 0.00005\}$, and we got the best performance with a learning rate of 0.0001. As for the number of negative samples, N_{neg}, we tried $\{9, 49, 99, 199, 299\}$ and 199 gave us the best results. For the window sizes of the convolution layers, w_1 and w_2, we tried $\{5, 10, 20, 30\}$. Our best results occurred with $w_1 = 30$ and $w_2 = 10$. We also experimented with $\{2, 5, 10\}$ for the size of the max pooling, and a max pooling of 5 gave the best performance. Finally, for the number of the kernels, k_1, k_2, we tried $\{500, 1000, 2000, 3000\}$, and $k_1 = 2000$ and $k_2 = 1000$ resulted in the best performance. We present the values of these parameters and other parameters of the system in Table 1.

Table 1: DeepCorr's hyperparameters optimized to correlate Tor traffic.

Layer	Details
Convolution Layer 1	Kernel num: 2000 Kernel size: (2, 30) Stride: (2,1) Activation: Relu
Max Pool 1	Window size: (1,5) Stride: (1,1)
Convolution Layer 2	Kernel nume: 1000 Kernel size: (4, 10) Stride: (4,1) Activation: Relu
Max Pool 2	Window size: (1,5) Stride: (1,1)
Fully connected 1	Size: 3000, Activation: Relu
Fully connected 2	Size: 800, Activation: Relu
Fully connected 3	Size: 100, Activation: Relu

4.4 Evaluation Metrics

Similar to previous studies, we use the *true positive (TP)* and *false positive (FP)* error rates as the main metrics for evaluating the performance of flow correlation techniques. The TP rate measures the fraction of associated flow pairs that are correctly declared to be correlated by DeepCorr (i.e., a flow pair (i,j) where i and j are the segments of the *same* Tor connection, and we have $p_{i,j} > \eta$). On the other hand, the FP rate measures the fraction of non-associated flow pairs that are mistakenly identified as correlated by DeepCorr (e.g., when i and j are the segments of two unrelated Tor connections, yet $p_{i,j} > \eta$). To evaluate FP, DeepCorr correlates every collected entry flow to every collected exit flow, therefore, we perform about $N \times (N-1)$ false correlations for each of our experiments, where N is the number of test flow pairs in the underlying experiment (N is 5,000 in most of the experiments).

Note that the detection threshold η makes a trade off between the FP and TP rates; therefore we make use of *ROC curves* to compare DeepCorr to other algorithms.

Finally, in our comparisons with RAPTOR [72], we additionally use the *accuracy* metric (the sum of true positive and true negative correlations over all correlations), which is used in the RAPTOR paper. To have a fair comparison, we derive the accuracy metric similar to RAPTOR: each flow is declared to be associated with only *a single* flow out of all evaluated flows, e.g., the flow that results in the maximum correlation metric, $p_{i,j}$. For the rest of our experiments, each flow can be declared as correlated with arbitrary number of intercepted flows (i.e., any pairs that $p_{i,j} > \eta$), which is a more realistic (and more challenging) setting.

5 EXPERIMENT RESULTS

In this section we present and discuss our experimental results.

5.1 A First Look at the Performance

As described in the experimental setup section, we browse 50,000 top Alexa websites over Tor and collect their ingress and egress flow segments. For this experiment, we selected 5,000 connections to train DeepCorr, and we use another 5,000 connections for testing. Therefore, we feed DeepCorr about 5,000 pairs of associated flow pairs, and $5,000 \times 4,999 \approx 2.5 \times 10^7$ pairs of non-associated flow pairs for training. We only use the first $\ell = 300$ packets of each flow (for shorter flows, we pad them to 300 packets by adding zeros). Figure 4 presents the true positive and false positive error rates of DeepCorr for different values of the threshold η. As expected, η trades off the TP and FP error rates. The figure shows a promising performance for DeepCorr in correlating Tor flows—using only 300 packets of each flow. For instance, for a FP of 10^{-3}, DeepCorr achieves a TP close to 0.8. As shown in the following, this is *drastically better* than the performance of previous work.

On the practicality of false positive error rates Note that a 10^{-3} FP may seem too large for a real-world setting in which the malicious AS/IXP is intercepting several thousands of Tor connections at any time. First, the results presented here are for Tor flows with only $\ell = 300$ packets to demonstrate DeepCorr's unique performance on short flows (no previous work has done experiments with such short lengths of Tor flows with acceptable accuracies). As shown later, increasing flow length rapidly improves DeepCorr's correlation performance, e.g., from Figure 8 a flow length of 450 packets improves FP by close to two orders of magnitude compared to 300 packets (for a fixed TP of 0.8). This is also evident from Figures 11 and 12. Second, the correlation adversary can deploy a multi-stage attack to optimize accuracy and traffic collection. For instance, she can apply DeepCorr on the first 300 packets of *all* intercepted Tor flows, and then collect more packets for the flow pairs detected by the first stage of the attack. She then re-applies DeepCorr on the longer observations of those flow pairs. Third, the adversary can perform standard pre-filtering mechanisms to further reduce FPs, e.g., she can ignore all flow pairs with substantially different start times. In our experiments, all of the flows have the same starting times.

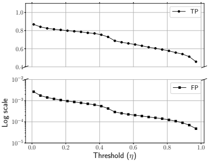

Figure 4: True positive and false positive error rates of DeepCorr in detecting correlated pairs of ingress and egress Tor flows for different detection thresholds (η). Each flow is only 300 packets.

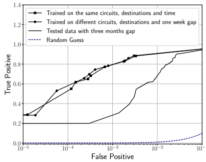

Figure 5: DeepCorr's performance does not depend on the circuits and destinations used during the training phase.

5.2 DeepCorr Can Correlate Arbitrary Circuits and Destinations

As discussed earlier, DeepCorr *learns* a correlation function for Tor that can be used to correlate Tor flows on—any circuits—and to—any destinations—regardless of the circuits and destinations used during the training process. To demonstrate this, we compare DeepCorr's performance in two experiments, each consisting 2,000 Tor connections, therefore 2,000 associated pairs and $2,000 \times 1,999$ non-associated flow pairs. In the first experiment, the flows tested for correlation by DeepCorr use the same circuits and destinations as the flows used during DeepCorr's training. In the second experiment, the flows tested for correlation by DeepCorr (1) use circuits that are totally different from the circuits used during training, (2) are targeted to web destinations different from those used during training, and (3) are collected one week after the learning flows.

Figure 5 compares DeepCorr's ROC curve for the two experiments. As can be seen, DeepCorr performs similarly in both of the experiments, demonstrating that DeepCorr's learned correlation function can be used to correlate Tor flows on arbitrary circuits and to arbitrary destinations. The third line on the figure shows the results when the training set is three months old, showing a degraded performance, as further discussed in the following.

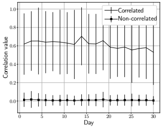

Figure 6: DeepCorr's correlation values for associated and non-associated flows for 30 consecutive days without re-training. The performance only starts to drop after about three weeks.

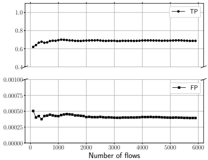

Figure 7: DeepCorr's performance is consistent regardless of the size of the testing dataset (we use a fixed, arbitrary η).

5.3 DeepCorr Does Not Need to Re-Train Frequently

Since the characteristics of Tor traffic change over time, any learning-based algorithm needs to be re-trained occasionally to preserve its correlation performance. We performed two experiments to evaluate how frequently DeepCorr needs to be retrained. In our first experiment, we evaluated our pre-trained model over Tor flows collected during 30 consecutive days. Figure 6 presents the output of the correlation function for each of the days for both associated and non-associated flow pairs. As we can see, the correlation values for non-associated flows do not change substantially, however, the correlation values for associated flows starts to slightly degrade after about three weeks. This suggests that an adversary will need to retrain her DeepCorr *only every three weeks*, or even once a month.

As an extreme case, we also evaluated DeepCorr's performance using a model that was trained three months earlier. Figure 5 compares the results in three cases: three months gap between training and test, one week gap between training and test, and no gap. We see that DeepCorr's accuracy significantly degrades with three months gap between training and test—interestingly, even this significantly degraded performance of DeepCorr due to lack of retraining is superior to all previous techniques compared in Figure 10.

5.4 DeepCorr's Performance Does Not Degrade with the Number of Test Flows

We also show that DeepCorr's correlation performance does not depend on the number of flows being correlated, i.e., the size of the test dataset. Figure 7 presents the TP and FP results (for a specific threshold) on datasets with different numbers of flows. As can be seen, the results are consistent for different numbers of flows being correlated. This suggests that DeepCorr's correlation performance will be similar to what derived through our experiments even if DeepCorr is applied on significantly larger datasets of intercepted flows, e.g., on the flows collected by a large malicious IXP.

5.5 DeepCorr's Performance Rapidly Improves with Flow Length

In all of the previous results, we used a flow length of $\ell = 300$ packets. As can be expected, increasing the length of the flows used for training and testing should improve the performance of DeepCorr. Figure 8 compares DeepCorr's performance for different lengths of flows, showing that DeepCorr's performance improves significantly for longer flow observations. For instance, for a target FP of 10^{-3}, DeepCorr achieves $TP = 0.62$ with $\ell = 100$ packets long flows, while it achieves $TP = 0.95$ with flows that contain $\ell = 450$ packets.

Note that the lengths of intercepted flows makes a tradeoff between DeepCorr's performance and the adversary's computation overhead. That is, while a larger flow length improves DeepCorr's correlation performance, longer flows impose higher storage and computation overheads on the traffic correlation adversary. A larger flow length also increase the adversary's waiting time in detecting correlated flows in real-time.

5.6 DeepCorr's Performance Improves with the Size of the Training Set

As intuitively expected, DeepCorr's performance improves when it uses a larger set of Tor flows during the training phase (i.e., DeepCorr learns a better correlation function for Tor with more training samples). Figure 9 compares DeepCorr's ROC curve when trained with different numbers of flows (for all of the experiments, we use a fixed number of 1,000 flows for testing). The figure confirms that increasing the size of the training set improves the performance of DeepCorr. For instance, for a target $FP = 10^{-3}$, using 1,000 training flows results in $TP = 0.56$, while using 5,000 flows for training gives DeepCorr a $TP = 0.8$. This shows that a resourceful adversary can improve the accuracy of her flow correlation classifier by collecting a larger number of Tor flows for training. Note that a

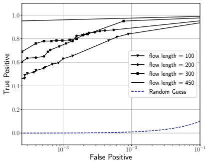

Figure 8: DeepCorr's performance rapidly improves when using longer flows for training and testing.

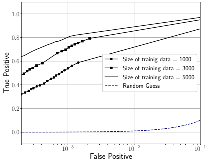

Figure 9: DeepCorr's correlation performance improves with more training data.

larger training set increases the training time, however the learning process does not need to repeat frequently as evaluated before.

5.7 DeepCorr Significantly Outperforms the State-Of-The-Art

In Section 2.2 we overviewed major flow correlation techniques introduced prior to our work. We perform experiments to compare DeepCorr's performance with such prior systems in correlating Tor flows. Figure 10 compares the ROC curve of DeepCorr to other systems, in which all of the systems are tested on the exact same set of Tor flows (each flow is at most 300 packets). As can be seen, *DeepCorr significantly outperforms the flow correlation algorithms used by prior work*, as we see a wide gap between the ROC curve of DeepCorr and other systems. For instance, for a target $FP = 10^{-3}$, while DeepCorr achieves a TP of 0.8, previous systems provide TP rates less than 0.05! This huge improvement comes from the fact that DeepCorr learns a correlation function tailored to Tor whereas previous systems use generic statistical correlation metrics (as introduced in Section 2.2) to link Tor connections.

Needless to say, any flow correlation algorithm will improve its performance by increasing the length of the flows it intercepts for correlation (equivalently, the traffic volume it collects from each flow); we showed this in Section 5.5 for DeepCorr. To offer reasonable accuracies, previous works have performed their experiments on flows that contain significantly more packets (and more data) than our experiments. For instance, Sun et al. evaluated the state-of-the-art RAPTOR [72] in a setting with *only 50 flows*, and each flow carries *100MB of data over 5 minutes*. This is while in our experiments presented so far, each flow has only 300 packets, which is equivalent to only ≈ 300 KB of Tor traffic (in contrast to RAPTOR's 100MB!). To ensure a fair comparison, we evaluate DeepCorr to RAPTOR in the exact same setup (e.g., 50 flows each 100MB, and we use the accuracy metric described in Section 4.4). The results shown in Figure 11 demonstrates DeepCorr's drastically superior performance (our results for RAPTOR comply with the numbers reported by Sun et al. [72]). On the other hand, we show that the performance gap between DeepCorr and RAPTOR is significantly wider for shorter flow observations. To show this, we compare DeepCorr and RAPTOR based on the volume of traffic they intercept from each flow. The results shown in Figure 12 demonstrate that DeepCorr outperforms significantly, especially for shorter flow observations. For instance, RAPTOR achieves a 0.95 accuracy after receiving 100MB from each flow, whereas DeepCorr achieves an accuracy of 1 with about 3MB of traffic. We see that *DeepCorr is particularly powerful on shorter flow observations*. We zoomed in by comparing RAPTOR and DeepCorr for small number of observed packets, which is shown in Figure 13. We see that DeepCorr achieves an accuracy of ≈ 0.96 with only 900 packets, in contrast to RAPTOR's 0.04 accuracy.

5.8 DeepCorr's Computational Complexity

Table 2: Correlation time comparison with previous techniques

Method	One correlation time
RAPTOR	0.8*ms*
Cosine	0.4*ms*
Mutual Information	1*ms*
Pearson	0.4*ms*
DeepCorr	2*ms*

In Table 2, we show the time to perform a single DeepCorr correlation in comparison to that of previous techniques (the correlated flows are 300 packets long for all the systems). We see that DeepCorr is noticeably slower than previous techniques, e.g., roughly two times slower than RAPTOR. However, note that since all the systems use the same length of flows, *DeepCorr offers drastically better correlation performance for the same time overhead*; for instance,

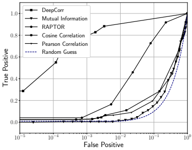

Figure 10: Comparing DeepCorr's ROC curve with previous systems shows an overwhelming improvement over the state-of-the-art (all the systems are tested on the same dataset of flows, and each flow is 300 packets).

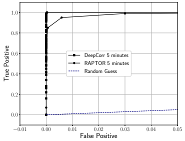

Figure 11: Comparing DeepCorr to RAPTOR [72] using the same flow lengths and flow number as the RAPTOR [72] paper.

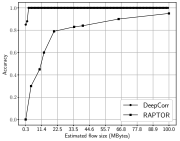

Figure 12: Comparing the accuracy of DeepCorr and RAPTOR [72] for various volumes of data intercepted from each flow. The RAPTOR values are comparable to Figure 6 of the RAPTOR paper [72].

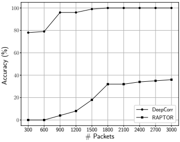

Figure 13: Comparing DeepCorr to RAPTOR in correlating short flows.

based on Figure 10, we see that DeepCorr offers a TP$\approx$ 0.9 when all previous systems offer a TP less than 0.2. Therefore, when all the systems offer similar accuracies (e.g., each using various lengths of input flows) DeepCorr will be *faster* than all the systems for the same accuracy. As an example, each RAPTOR correlation takes 20ms (on much longer flow observations) in order to achieve the same accuracy as DeepCorr which takes only 2ms—i.e., DeepCorr is 10 times faster for the same accuracy.

Compared to previous correlation techniques, DeepCorr is the only system that has a training phase. We trained DeepCorr using a standard Nvidia TITAN X GPU (with 1.5GHz clock speed and 12GB of memory) on about 5,000 pairs of associated flow pairs and 5000×4999 non-associated flow pairs, where each flow consists of 300 packets. In this setting, DeepCorr is trained in roughly one day. Recall that as demonstrated in Section 5.3, DeepCorr does not need to be re-trained frequently, e.g., only once every three weeks. Also, a resourceful adversary with better GPU resources than ours will be able to cut down on the training time.

5.9 DeepCorr Works in Non-Tor Applications as Well

While we presented DeepCorr as a flow correlation attack on Tor, it can be used to correlate flows in other flow correlation applications as well. We demonstrate this by applying DeepCorr to the problem of stepping stone attacks [6, 26, 80]. In this setting, a cybercriminal proxies her traffic through a compromised machine (e.g., the

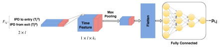

Figure 14: The network architecture of DeepCorr to detect stepping stone attacks

Table 3: DeepCorr's parameters optimized for the stepping stone attack application.

Layer	Details
Convolution Layer 1	Kernel num: 200 Kernel size: (2, 10) Stride: (1,1) Activation: Relu
Max Pool 1	Window size: (1,5) Stride: (1,1)
Fully connected 1	Size: 500, Activation: Relu
Fully connected 2	Size: 100, Activation: Relu

stepping stone) in order to hide her identity. Therefore, a network administrator can use flow correlation to match up the ingress and egress segments of the relayed connections, and therefore trace back to the cybercriminal. Previous work has devised various flow correlation techniques for this application [17, 33, 53, 59, 81].

For our stepping stone detection experiments, we used the 2016 CAIDA anonymized data traces [11]. Similar to the previous works [33, 34, 53] we simulated the network jitter using Laplace distribution, and modeled packet drops by a Bernoulli distribution with different rates. We apply DeepCorr to this problem by learning DeepCorr in a stepping stone setting. As the noise model is much simpler in this scenario than Tor, we use a simpler neural network model for DeepCorr for this application. Also, we only use one direction of a bidirectional connection to have a fair comparison with previous systems, which all only use one-sided flows. Figure 14 and Table 3 show our tailored neural network and our choices of parameters, respectively.

Our evaluations show that DeepCorr provides a performance comparable to "Optimal" flow correlation techniques of Houmansadr et al. [33, 34] when network conditions are stable. However, when the network conditions becomes noisy, DeepCorr offers a significantly stronger performance in detecting stepping stone attacks. This is shown in Figure 15, where the communication network has a network jitter with a 0.005s standard deviation, and the network randomly drops 1% of the packets.

6 COUNTERMEASURES

While previous work has studied different countermeasures against flow correlation and similar traffic analysis attacks [2, 9, 19, 35, 41, 42, 50, 56, 61, 82], they remain mostly non-deployed presumably due to the poor performance of existing flow correlation techniques at large scale [60, 66]. In the following, we discuss two possible countermeasures.

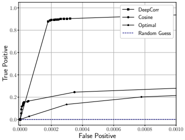

Figure 15: DeepCorr outperforms state-of-the-art stepping stone detectors in noisy networks (1% packet drop rate).

6.1 Obfuscate Traffic Patterns

An intuitive countermeasure against flow correlation (and similar traffic analysis attacks like website fingerprinting) is to *obfuscate traffic characteristics* that are used by such algorithms. Therefore, various countermeasures have been suggested that modify packet timings and packet sizes to defeat flow correlation, in particular by padding or splitting packets in order to modify packet sizes, or by delaying packets in order to perturb their timing characteristics. The Tor project, in particular, has deployed various pluggable transports [61] in order to defeat censorship by nation-states who block all Tor traffic. Some of these pluggable transports only obfuscate packet contents [56], some of them obfuscate the IP address of the Tor relays [48], and some obfuscate traffic patterns [50, 56]. Note that Tor's pluggable transports are designed merely for the purpose of censorship resistance, and they obfuscate traffic only from a censored client to her first Tor relay (i.e., a Tor bridge). Therefore, Tor's pluggable transports are *not* deployed by any of Tor's public relays.

As a possible countermeasure against DeepCorr, we suggest to deploy traffic obfuscation techniques by *all* Tor relays (including the guard and middle relays). We evaluated the impact of several Tor pluggable transports on DeepCorr's performance. Currently, the Tor project has three deployed plugs: meek, obfs3, and obs4. We evaluated DeepCorr on meek and obfs4 (obfs3 is an older version of obfs4). We also evaluated two modes of obfs4: one with IAT mode "on" [55], which obfuscates traffic features, and one with the IAT mode "off", which does not obfuscate traffic features. We used DeepCorr to learn and correlate traffic on these plugs. However, *due to ethical reasons, we collected a much smaller set of flows* for these experiments compared to our previous experiments; this is because Tor bridges are very scarce and expensive, and we therefore avoided overloading the bridges.[6] Consequently, our correlation results are

[6]Alternatively, we could set up our own Tor bridges for the experiments. We decided to use real-world bridges to incorporate the impact of actual traffic loads in our experiments.

Table 4: DeepCorr's performance if Tor's pluggable transports are deployed by the relays (results are very optimistic due to our small training set, which is for ethical reasons).

Plug name	TP	FP
obfs4 with IAT=0	≈ 0.50	0.0005
meek	≈ 0.45	0.0005
obfs4 with IAT=1	$\approx .10$	0.001

very optimistic due to their small training datasets (e.g., a real-world adversary will achieve much higher correlation accuracies with adequate training). We browsed 500 websites over obfs4 with and without the IAT mode on, as well as over meek. We trained DeepCorr on only 400 flows (300 packets each) for each transport (in contrast to 5,000 flows in our previous experiments), and tested on another 100 flows. Table 4 summarizes the results. We see that meek and obfs4 with IAT=0 provide no protection to DeepCorr; note that a 0.5 TP is comparable to what we get for bare Tor if trained on only 400 flows (see Figure 9), therefore we expect correlation results similar to bare Tor with a larger training set. The results are intuitive: meek merely obfuscates a bridge's IP and does not deploy traffic obfuscation (except for adding natural network noise). Also obfs4 with IAT=0 solely obfuscates packet contents, but not traffic features. On the other hand, we see that DeepCorr has a significantly lower performance in the presence of obfs4 with IAT=1 (again, DeepCorr's accuracy will be higher for a real-world adversary who collects more training flows).

Our results suggest that (public) Tor relays should deploy a traffic obfuscation mechanism like obfs4 with IAT=1 to resist advanced flow correlation techniques like DeepCorr. However, this is *not* a trivial solution due to the increased cost, increased overhead (bandwidth and CPU), and reduced QoS imposed by such obfuscation mechanisms. Even the majority [55] of Obfsproxy Tor bridges run obfs4 without traffic obfuscation (IAT=0). Therefore, designing an obfuscation mechanism tailored to Tor that makes the right balance between performance, cost, and anonymity remains a challenging problem for future work.

6.2 Reduce An Adversary's Chances of Performing Flow Correlation

Another countermeasure against flow correlation on Tor is reducing an adversary's chances of intercepting the two ends of many Tor connections (therefore, reducing her chances of performing flow correlation). As discussed earlier, recent studies [22, 52, 72] show that various ASes and IXPs intercept a significant fraction of Tor traffic, putting them in an ideal position to perform flow correlation attacks. To counter, several proposals suggest new relay selection mechanisms for Tor that reduce the interception chances of malicious ASes [2, 5, 41, 54, 71, 73]. None of such alternatives have been deployed by Tor due to their negative impacts on performance, costs, and privacy. We argue that designing practical AS-aware relay selection mechanisms for Tor is a promising avenue to defend against flow correlation attacks on Tor.

7 CONCLUSIONS

We design a flow correlation system, called DeepCorr, that drastically outperforms the state-of-the-art systems in correlating Tor connections. DeepCorr leverages an advanced deep learning architecture to *learn* a flow correlation function tailored to Tor's complex network (as opposed to previous works' use of general-purpose statistical correlation metrics). We show that with adequate learning, DeepCorr can correlate Tor connections (and therefore break its anonymity) with accuracies significantly stronger than existing algorithms, and using substantially shorter lengths of flow observations. We hope that our work demonstrates the escalating threat of flow correlation attacks on Tor in rise of advanced learning algorithms, and calls for the deployment of effective countermeasures by the Tor community.

ACKNOWLEDGMENTS

This work was supported by the NSF grants CNS-1525642, CNS-1553301, and CNS-1564067.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Masoud Akhoondi, Curtis Yu, and Harsha V Madhyastha. 2012. LASTor: A low-latency AS-aware Tor client. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 476–490.

[3] Axel Arnbak and Sharon Goldberg. 2014. Loopholes for Circumventing the Constitution: Unrestricted Bulk Surveillance on Americans by Collecting Network Traffic Abroad. *Mich. Telecomm. & Tech. L. Rev.* 21 (2014), 317.

[4] Adam Back, Ulf Möller, and Anton Stiglic. 2001. Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems. In *Information Hiding (Lecture Notes in Computer Science)*, Vol. 2137. Springer, 245–247.

[5] Armon Barton and Matthew Wright. 2016. DeNASA: Destination-naive as-awareness in anonymous communications. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (2016), 356–372.

[6] Avrim Blum, Dawn Song, and Shobha Venkataraman. 2004. Detection of interactive stepping stones: Algorithms and confidence bounds. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 258–277.

[7] A. Blum, D. Song, and S. Venkataraman. 2004. Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds. In *RAID*.

[8] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. 2007. Denial of service or denial of security?. In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 92–102.

[9] X. Cai, X. Zhang, B. Joshi, and R. Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *CCS*.

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

[11] caida trace [n. d.]. The CAIDA UCSD Anonymized Internet Traces 2016 - [2016]. http://www.caida.org/data/passive/passive_2016_dataset.xml.

[12] Sambuddho Chakravarty, Marco V Barbera, Georgios Portokalidis, Michalis Polychronakis, and Angelos D Keromytis. 2014. On the effectiveness of traffic analysis against anonymity networks using flow records. In *International conference on passive and active network measurement*. Springer, 247–257.

[13] Tom Chothia and Apratim Guha. 2011. A statistical test for information leaks using continuous mutual information. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*. IEEE, 177–190.

[14] George Danezis. 2004. The traffic analysis of continuous-time mixes. In *International Workshop on Privacy Enhancing Technologies*. Springer, 35–50.

[15] George Danezis, Roger Dingledine, and Nick Mathewson. 2003. Mixminion: Design of a type III anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*. IEEE, 2–15.

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

[17] D. Donoho, A. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford. 2002. Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. In *RAID*.

[18] David L Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit, and Stuart Staniford. 2002. Multiscale stepping-stone detection: Detecting

pairs of jittered interactive streams by exploiting maximum tolerable delay. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 17–35.
[19] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton. 2013. Protocol Misidentification Made Easy with Format-Transforming Encryption. In *CCS*.
[20] M. Edman and P. Syverson. 2009. AS-awareness in Tor path selection. In *CCS*.
[21] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg. 2012. Changing of the Guards: Framework for Understanding and Improving Entry Guard Selection in Tor. In *WPES*.
[22] Nick Feamster and Roger Dingledine. 2004. Location Diversity in Anonymity Networks. In *Workshop on Privacy in the Electronic Society*. Washington, DC, USA.
[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.
[24] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique.. In *USENIX Security Symposium*. 1187–1203.
[25] Gaofeng He, Ming Yang, Xiaodan Gu, Junzhou Luo, and Yuanyuan Ma. 2014. A novel active website fingerprinting attack against Tor anonymous system. In *Computer Supported Cooperative Work in Design (CSCWD), Proceedings of the 2014 IEEE 18th International Conference on*. IEEE, 112–117.
[26] Ting He and Lang Tong. 2007. Detecting encrypted stepping-stone connections. *IEEE Transactions on Signal Processing* 55, 5 (2007), 1612–1623.
[27] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 31–42.
[28] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. 2010. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)* 13, 2 (2010), 13.
[29] A. Houmansadr and N. Borisov. 2011. SWIRL: A Scalable Watermark to Detect Correlated Network Flows. In *NDSS*.
[30] Amir Houmansadr and Nikita Borisov. 2011. Towards Improving Network Flow Watermarks using the Repeat-accumulate Codes. In *ICASSP*.
[31] Amir Houmansadr and Nikita Borisov. 2013. The need for flow fingerprints to link correlated network flows. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 205–224.
[32] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. 2009. Multi-Flow Attack Resistant Watermarks for Network Flows. In *ICASSP*.
[33] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. 2009. RAINBOW: A Robust And Invisible Non-Blind Watermark for Network Flows. In *Network and Distributed System Security Symposium (NDSS)*.
[34] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. 2014. Non-blind watermarking of network flows. *IEEE/ACM Transactions on Networking (TON)* 22, 4 (2014), 1232–1244.
[35] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer. 2013. I Want My Voice to Be Heard: IP over Voice-over-IP for Unobservable Censorship Circumvention. In *NDSS*.
[36] Rob Jansen, Marc Juarez, Rafa Gálvez, Tariq Elahi, and Claudia Diaz. 2018. Inside Job: Applying Traffic Analysis to Measure Tor from Within. In *NDSS*.
[37] Filip Jelic. 2016. Tor's Biggest Threat – Correlation Attack. https://www.deepdotweb.com/2016/10/25/tors-biggest-threat-correlation-attack/.
[38] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. 2013. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *CCS*.
[39] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 337–348.
[40] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 263–274.
[41] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar. 2015. Defending tor from network adversaries: A case study of network path prediction. *Proceedings on Privacy Enhancing Technologies* 2015, 2 (2015), 171–187.
[42] G. Kadianakis. 2012. Packet Size Pluggable Transport and Traffic Morphing. Tor Tech Report 2012-03-004.
[43] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[44] Negar Kiyavash, Amir Houmansadr, and Nikita Borisov. 2008. Multi-Flow Attacks Against Network Flow Watermarking Schemes. In *USENIX Security Symposium*.
[45] Brian N Levine, Michael K Reiter, Chenxi Wang, and Matthew Wright. 2004. Timing attacks in low-latency mix systems. In *International Conference on Financial Cryptography*. Springer Berlin Heidelberg, 251–265.
[46] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. 2009. A new cell counter based attack against tor. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 578–589.
[47] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. 2010. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security*. Springer, 199–214.
[48] meek [n. d.]. meek Pluggable Transport. https://trac.torproject.org/projects/tor/wiki/doc/meek.
[49] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. 2011. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 215–226.
[50] H. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. 2012. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *CCS*.
[51] Steven J Murdoch and George Danezis. 2005. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy*. IEEE, 183–195.
[52] Steven J. Murdoch and Piotr Zieliński. 2007. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *Privacy Enhancing Technologies Symposium (Lecture Notes in Computer Science)*, Nikita Borisov and Philippe Golle (Eds.), Vol. 4776. Springer, Ottawa, Canada.
[53] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. 2017. Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2053–2069.
[54] Rishab Nithyanand, Oleksii Starov, Adva Zair, Phillipa Gill, and Michael Schapira. 2016. Measuring and mitigating AS-level adversaries against Tor. In *NDSS*.
[55] obfs4 2016. Turning on timing obfuscation (iat-mode=1) for some default bridges. https://lists.torproject.org/pipermail/tor-project/2016-November/000776.html.
[56] obfsproxy [n. d.]. A Simple Obfuscating Proxy. https://www.torproject.org/projects/obfsproxy.html.en.
[57] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2016. Website Fingerprinting at Internet Scale. In *Proceedings of the 23rd Internet Society (ISOC) Network and Distributed System Security Symposium (NDSS 2016)*.
[58] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. ACM, 103–114.
[59] V. Paxson and S. Floyd. 1995. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking* 3, 3 (June 1995), 226–244.
[60] Mike Perry. 2017. A Critique of Website Traffic Fingerprinting Attacks. https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks.
[61] PluggableTransports [n. d.]. Tor: Pluggable Transports. https://www.torproject.org/docs/pluggable-transports.html.en.
[62] Young June Pyun, Young Hee Park, Xinyuan Wang, Douglas S Reeves, and Peng Ning. 2007. Tracing traffic through intermediate hosts that repacketize flows. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 634–642.
[63] Daniel Ramsbrock, Xinyuan Wang, and Xuxian Jiang. 2008. A first step towards live botmaster traceback. In *Recent Advances in Intrusion Detection*. Springer, 59–77.
[64] Michael K Reiter and Aviel D Rubin. 1998. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)* 1, 1 (1998), 66–92.
[65] Marc Rennhard and Bernhard Plattner. 2002. Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*. ACM, 91–102.
[66] Fatemeh Rezaei and Amir Houmansadr. 2017. TagIt: Tagging Network Flows using Blind Fingerprints. In *Privacy Enhancing Technologies (PETS)*.
[67] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *NDSS*.
[68] Vitaly Shmatikov and Ming-Hsiu Wang. 2006. Timing analysis in low-latency mix networks: Attacks and defenses. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 18–33.
[69] Stuart Staniford-Chen and L Todd Heberlein. 1995. Holding intruders accountable on the Internet. In *Security and Privacy, 1995. Proceedings., 1995 IEEE Symposium on*. IEEE, 39–49.
[70] Oleksii Starov, Rishab Nithyanand, Adva Zair, Phillipa Gill, and Michael Schapira. 2016. Measuring and mitigating AS-level adversaries against Tor. In *NDSS*.
[71] Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. 2017. Counter-RAPTOR: Safeguarding Tor Against Active Routing Attacks. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 977–992.
[72] 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)*. 271–286.
[73] Henry Tan, Micah Sherr, and Wenchao Zhou. 2016. Data-plane defenses against routing attacks on Tor. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (2016), 276–293.
[74] tor metrics [n. d.]. Tor Metrics. https://metrics.torproject.org.
[75] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*. 143–157.

[76] Tao Wang and Ian Goldberg. 2013. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM, 201–212.
[77] Tao Wang and Ian Goldberg. 2016. On realistically attacking Tor with website fingerprinting. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (2016), 21–36.
[78] Xinyuan Wang, S. Chen, and S. Jajodia. 2005. Tracking Anonymous Peer-to-peer VoIP Calls on the Internet. In *CCS*.
[79] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. 2007. Network flow watermarking attack on low-latency anonymous communication systems. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 116–130.
[80] Xinyuan Wang and Douglas S Reeves. 2003. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 20–29.
[81] Xinyuan Wang, Douglas S Reeves, and S Felix Wu. 2002. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *Computer SecurityâĂŤESORICS 2002*. Springer, 244–263.
[82] C. Wright, S. Coull, and F. Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *NDSS*.
[83] Matthew K Wright, Micah Adler, Brian Neil Levine, and Clay Shields. 2002. An Analysis of the Degradation of Anonymous Protocols. In *NDSS*, Vol. 2. 39–50.
[84] Kunikazu Yoda and Hiroaki Etoh. 2000. Finding a connection chain for tracing intruders. In *Computer Security-ESORICS 2000*. Springer, 191–205.
[85] Wei Yu, Xinwen Fu, Steve Graham, Dong Xuan, and Wei Zhao. 2007. DSSS-based flow marking technique for invisible traceback. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 18–32.
[86] Yin Zhang and Vern Paxson. 2000. Detecting Stepping Stones.. In *USENIX Security Symposium*, Vol. 171. 184.
[87] Ye Zhu and Riccardo Bettati. 2005. Unmixing Mix Traffic. In *Privacy Enhancing Technologies Workshop*, David Martin and George Danezis (Eds.). 110–127.
[88] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. 2004. On flow correlation attacks and countermeasures in mix networks. In *International Workshop on Privacy Enhancing Technologies*. Springer, 207–225.