Distributed system for cooperative deanonymization of Tor circuits

(extended abstract of the MSc dissertation)

Pedro Manuel Torres Pires de Medeiros
Departamento de Engenharia Informática, Instituto Superior Técnico
Advisor: Professor Nuno Miguel Carvalho dos Santos

***Abstract*—Current technology allows an interested party (for instance, an ISP) to closely monitor the communications of any user which, in turn, may allow to uncover data such as the user political views, consumer preferences, health condition, and other private information. Anonymity networks, such as Tor, have been designed to defeat network surveillance or traffic analysis activities, and preserve the anonymity of users when they use the Internet. Not only that Tor allows the construction of special services that also remain anonymous on the network making both the client and server anonymous. Unfortunately, these services can also be used to support illegal activities, which raises the need for tools that can help in de-anonymizing illegal Tor flows. In this work we study techniques that allow to de-anonymize illegal flows where both the client and the server are anonymous. Our work leverages on recent results that show that it is possible to perform traffic correlation based on deep neural networks. This paves the way for designing distributed, cooperative, de-anonymization tools, where multiple ISPs can contribute by sharing information regarding flow sets and then perform traffic-correlation. We perform an in depth evaluation of such a system where synthetic flows were generated based on real fully anonymous Tor onion services. We also present an extension to the system where the data provided by the ISPs is kept private and only when a correlation is found, the target flow can be de-anonymized while ensuring that the anonymity of the other flows involved in the query is preserved.**

I. Introduction

This work addresses the problem of traffic correlation on anonymity networks such as Tor. It studies the use of deep neural networks to perform traffic correlation. Although traffic correlation attacks are well known, they mostly target client only anonymity scenarios. We will focus on the scenario where both the client and server preserve their anonymity during communication. We present a thorough evaluation of traffic correlation attacks based on models combining both deep and regular neural networks for client and server anonymous communication.

Current technology allows an interested party (for instance, an ISP) to closely monitor the communications of any user over the Internet. Even if communication is encrypted, the destination of packets and the fingerprint of the flow may allow to uncover data such as the user political views, consumer preferences, health condition, and other private information[1], [2], [3], [4]. Anonymity networks, such as Tor[5], have been designed to defeat network surveillance or traffic analysis activities, and preserve the anonymity of users when they use the Internet. Tor works by using an overlay network of relay nodes that forward encrypted packets in a virtual circuit. Packets are encrypted using multiple, nested, layers of encryption, such that each relay can only extract information regarding the next hop in the virtual circuit but cannot extract information regarding the original origin or the final destination of the packet. In this way, the first node of the Tor circuit knows the user IP but does not know the final destination, and the last node knows the final destination but does not know the user IP. Tor, besides anonymyzing the sender, also provides the ability for the receiver to be anonymous as well. This fully anonymous circuits provide anonymity to both the sender and receiver performing twice the hops on the network.

The Tor network can have many legitimate users, such as individuals that seek to ensure their right to privacy, or journalists that need to report facts under adversarial conditions. Unfortunately, Tor can also be used to support illegal activities, such as drug dealing, illegal gun selling, or child pornography[6], [7], [8], [9], [10] just to name a few. This raises the need for tools that can help in de-anonymizing illegal Tor flows. Previous work as shown that by employing traffic analysis techniques it is possible to correlate flows entering the Tor network with flows exiting the Tor network to de-anonymize a flow[11], [12], [13], [14], [15]. Such attacks have been applied to the Tor network with promising results. First attempts focused on timing analysis and packet distribution in order to correlate two traffic flows[11], [16]. More recent work uses machine learning techniques with manually selected features in order to train a flow correlator [14]. The state-of-the-art leverages convolutional neural networks that learn to correlate flows and that perform flow correlation with significant success[15].

These techniques however only target traffic that is produced when accessing regular websites through sender anonymity only circuits. In this work we will apply such traffic correlation techniques to fully anonymous communication through Tor, where most illegal activities take place.

This project focuses on the design and implementation of a protocol and system that will allow one to perform Tor circuit de-anonimyzation when both the sender and receiver are anonymous. The system shall be able to correlate fully anonymous Tor circuits with reasonable results. We expect to achieve these goals while being able to process large amounts of data in reasonable time and without sacrificing the accuracy of traffic correlation. The system should also employ a protocol that allows ASes and authorities to work together. As an extension to the system, it shall also provide privacy guarantees on data provided by involving ASes.

In order to reach the proposed goal we started by the collection of a dataset of fully anonymous flow pairs which to the best of our knowledge as not been done yet. The next step to achieve our goal focused on the design of a system that can process fully anonymous circuits with high performance and precision. The final step to achieve our goal introduces preliminary work on extensions that make the system privacy preserving.

This work describes, implements and evaluates Torpedo, a system that allows for de-anonimyzation of fully anony-

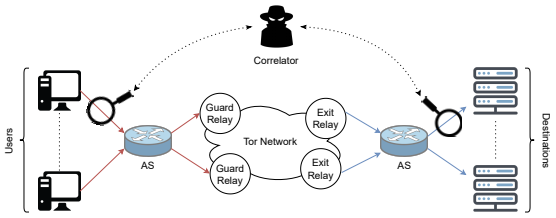


Figure 1. Traffic correlation attack model.

mous Tor circuits where both the sender and receiver are anonymous. We present the following contributions:

- Torpedo, the first system that applies traffic correlation attacks to Tor onion service traffic.
- Innovative correlator design that separates the correlation process into two distinct phases allowing for high throughput of correlations.
- Implementation and evaluation of preliminary privacy-preserving extensions to the system.
- Deliver a thorough evaluation of the system performance and precision showing that we can achieve high throughput while maintaining reasonable precision.
- Deliver a dataset of correlated flow pairs from fully anonymous circuits generated by accessing Tor onion services, both static and dynamic which can be used in future scientific work.

II. Background on Traffic Correlation

This section provides an overview of multiple correlation attack models. Traffic correlation consists in assessing whether two samples of network flows gathered in separate areas of the network correspond to the same connection. Correlation attacks have a prominent impact on the security of the Tor network, and enable the effective deanonymization of Tor circuits. A successful traffic correlation attack has the ability to deanonymize not only Tor clients but also servers running behind Tor onion services.

Figure 1 illustrates how such an attack can be performed when a user uses the Tor network. An attacker observes both end connections of the Tor circuit but does not need to observe the full path of the circuit traversing the Tor network. By applying these correlation attacks to the observed flows at each end, the attacker can to determine if the flows captured belong to the same connection. If that is indeed the case, since the attacker knows the IPs at each end of both flows, it has essentially deanonymized the user and it now knows it's IP and the IP it is accessing to.

Typically, traffic correlation attacks can be perpetrated by finding similarities between the packet sizes and inter-packet timing characteristics of packets pertaining to a given flow. For the specific case of Tor, the use of packet size features is largely useless for the purpose of traffic correlation as fixed sized cells are used to propagate data between endpoints. However, in order to maintain low-latency, Tor does not significantly obscure inter-packet timing, allowing an adversary with a privileged location in the network to succeed in performing traffic correlation attacks.

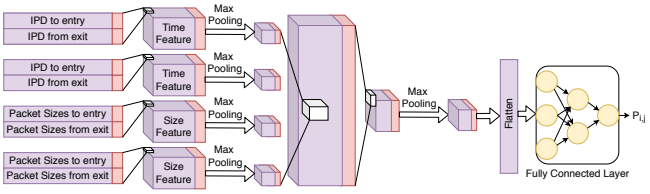


Figure 2. Network architecture of DeepCorr's CNN.

III. Related Work

A. Traffic Correlation using Neural Networks

The latest traffic correlation technique uses deep learning to learn features about the dynamics of the Tor network and achieve higher accuracy in flow correlation attacks. Deepcorr [15] is a state-of-the-art system that allows an attacker to train a convolution neural network on network flows. Importantly, DeepCorr offers a higher accuracy in flow correlation than previous approaches, while simply requiring the observation of the first 300 packets of data pertaining to a given flow. This represents a major improvement over the technique previously proposed by Sun et al. [14].

The compelling results achieved by DeepCorr [15] are attributed to the fact that the system trains a convolutional neural network (CNN) on real world data which allows the model to learn intrinsically how the Tor network operates. The architecture of DeepCorr's CNN is illustrated in Figure 2. The CNN takes as input four features for each flow – two vectors of inter-packet timings and two vectors of packet sizes – and is composed of two layers of convolution and pooling, and three layers of a fully connected neural network. The first convolution layer aims to capture the similarities between the input vectors that are expected to be correlated for associated Tor flows. The second convolution layer captures overall traffic features from the combination of timing and size information.

DeepCorr was tested by collecting a large dataset of flows using multiple virtual machines to connect to the 50,000 top Alexa websites via Tor network. This method ensured the real Tor network was used without collecting data pertaining to real users of the network.

B. Neural Networks with Encrypted Data

An open source project named TF Encrypted has leveraged some of the recent developments that allow neural networks to operate over encrypted data. It provides a framework for enabling encrypted deep learning readily available for the community of TensorFlow[1] and Keras[2] developers. Through the use of TF Encrypted, one can build a neural network through a programming model similar to Keras with the added value of being able to perform secure predictions. TF Encrypted poses the ideal candidate to implement a privacy preserving extension for our system, as it provides a usable framework very similar to Keras which is based on Tensorflow, used to develop DeepCorr.

[1]https://www.tensorflow.org/
[2]https://keras.io/

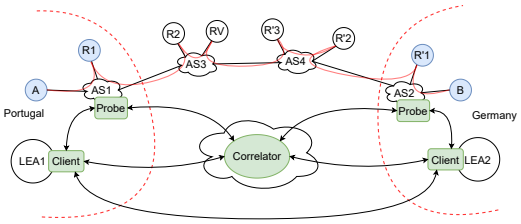


Figure 3. Torpedo system components: client, probe, and correlator.

IV. Torpedo

Torpedo consists of three distinct components: *client*, *probe*, and *correlator*. It also involves three types of participants: Law Enforcement Agencies (LEAs), ASes, and a Data Processing Provider (DPP). Figure 3 illustrates a system deployment on a concrete (and simplified) usage scenario. The client consists of a piece of software that offers a flow query interface to LEAs. It enables LEAs to submit queries to the system aimed at determining correlated flow pairs that belong to accesses from Tor users to onion services. The client sends these queries to the DPP which is responsible for running the correlator logic. The correlator implements a two-stage pipeline of neural networks (NNs) which is fed with flow pair information from participating ASes based on the criteria indicated in the query. This flow pair information is collected by Torpedo's probes that the participating ASes have deployed in their local networks. The correlator's NN pipeline outputs the results of the flow pair correlation and sends these results back to the originating client. The client displays a list of flow pair ids and respective IP addresses along with the probability that each flow pair belongs to a Tor user's access to a given onion service.

A. Correlator and Flow Pair Correlation

The correlator, runs on a trusted third party – the DPP – and it is responsible for performing the flow correlation operations. It takes as input a set of flows sent by the ASes involved in a given query and outputs a probability of correlation between pairs of these flows.

Internally, the correlator is composed of a two-stage pipeline where each stage is implemented by its own neural network with a particular function. We designate these stages by *ranking stage* and *correlation stage*. The former is the first stage of the pipeline and it is tailored to pre-process a high volume of flow pairs very efficiently and return an ordered list of the pairs which are more likely to be correlated. The second correlation stage is dedicated to deep traffic analysis as it performs the actual correlation of the flow pairs. The rational behind separating the correlator in two distinct phases rather than one only is its improved performance and the customizability. First, our two stage approach allows us to process large volumes of data in the ranking stage and use it as a filter for selecting the most probable correlated pairs to input to the correlation stage, which due to its nature takes orders of magnitude longer to process. Another benefit of this approach is allowing each stage to focus on a subset of the flow features leading to better over all performance of the model.

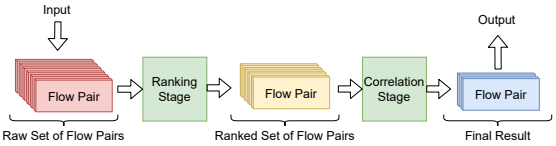


Figure 4. Correlator stages processing input flow pairs.

Figure 4 illustrates how both stages interact with one another to produce a final result of the possibly correlated flow pairs. The input to the ranking stage is the set of all pairs provided to the client by ASes for a given query. This stage processes all these pairs and sends its output of ranked pairs to the correlation stage. The ranking stage also filters out certain pairs that do not meet the minimum threshold to progress into the next and final stage. Finally, the correlation stage analyzes these pairs and outputs the final set of possibly correlated flows with their associated probability of correlation. The following two sections explain in detail how each of these stages work.

B. Ranking Stage

The ranking stage is composed of a simple neural network as described in Table I. This neural network takes three inputs which correspond to three features: *packets in forward direction*, *packets in reverse direction* and *time of capture*. These features are derived from the flows of each pair to be evaluated by the correlator:

- Packets in forward direction refers to the difference between the number of packets sent from the first flow and the number of packets received at the second flow;
- Packets in reverse direction refers to the same as in the forward direction but in the reverse order being packets sent from the second flow and received by the first flow;
- Time of capture refers to the difference of time between the first packet of each flow.

The neural network then uses three fully connected hidden layers only. Each hidden layer contains a specific number of nodes indicated in Table I as "Size". All the layers of this neural network use the ReLU activation function. The reduced number of layers of the neural network along with our feature selection allow for a highly efficient evaluation of flow pairs. This efficiency is desirable as the system may have to correlate large amounts of flow pairs due to the combinatorial nature of pairs. For instance if two ASes capture each 100 candidate flows for a given query, then we are looking to 100 x 100 pairs to evaluate. We can see how fast this total number of pairs grows in relation to the number of captured flows per AS.

The output of this stage consists of a number ranging from 0 to 1 which implies the probability of correlation for a given pair. This result can then be used to select the pairs that are most probably correlated and should be further evaluated by the correlation stage. We can perform this candidate selection by means of a configurable threshold or by amount of flows to evaluate in the correlation stage. For instance we can have a threshold T where pairs with

Layer	Details
Input Layer	Size: 3
Fully Connected 1	Size: 300, Activation: ReLU
Fully Connected 2	Size: 80, Activation: ReLU
Fully Connected 3	Size: 10, Activation: ReLU
Output Layer	Size: 1, Activation: Sigmoid

Table I
RANKING STAGE'S NEURAL NETWORK ARCHITECTURE.

Layer	Details
Input Layer	Size: Flow Length*8
Convolution Layer 1	Kernel num: 1000 Kernel size: (2,30) Stride: (2,1) Activation: Relu
Max Pool 1	Window Size: (1,5) Stride: (1,1)
Convolution Layer 2	Kernel num: 500 Kernel size: (4,10) Stride: (4,1) Activation: Relu
Max Pool 2	Window Size: (1,5) Stride: (1,1)
Fully Connected 1	Size: 1500, Activation: ReLU
Fully Connected 2	Size: 400, Activation: ReLU
Fully Connected 3	Size: 50, Activation: ReLU
Output Layer	Size: 1, Activation: Sigmoid

Table II
CORRELATION STAGE NETWORK ARCHITECTURE.

a result above T are selected as candidates. This is ideal when there are few constrains on the number of pairs that the correlation stage can process, but we can also order pairs by their result and select candidates by the most probable N when there is an imposed limit of a maximum of N pairs that can be processed by the correlation stage.

C. Correlation Stage

The correlation stage makes use of a highly more complex model than the ranking stage. This is because it focuses on traffic analysis by processing the inter-packet timings and packet sizes of the two flows in both directions for a given pair. It is implemented by a CNN which consists of a modified version of the Deepcorr's CNN (see Section III-A).

Table II describes the parameters of each layer of the CNN. The input is composed of 8 vectors of length N where N is the number of packets the model is trained on. From these 8 vectors 4 relate to the first flow and the other 4 to the second. These vectors contain the inter-packet timings for the incoming traffic, inter-packet timings for the outgoing traffic, packet sizes for the incoming traffic and packets sizes for the outgoing traffic. The network presents two convolutional layers and three fully connected layers. The convolutional layers are aimed at training filters that will recognise patterns in the traffic data inputted to the network. The output of these filters is then fed to the fully connected network to process and learn the relations between them.

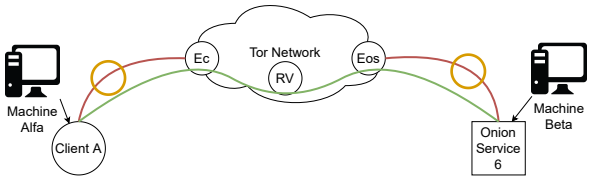


Figure 5. Sample capture model.

- The intuition behind the first convolutional layer is that it allows the network to learn features that correlate the same characteristics of both flows. To achieve this, we use a kernel size of (2,30) where we are processing 2 input vectors at a time, observing 30 packets from each at a given step i.e. the first 30 inter-packet timings of the outgoing traffic in the first flow and the first 30 inter-packet timings of the incoming traffic in the second flow. By using a stride of (2,1) we ensure this filters only match intended features and do not mix inter-packet timings with packet sizes nor incoming traffic with incoming traffic and vice versa.
- In the second convolutional layer this separation between inter-packet timings and size features is no longer present as the kernels are now processing data that came from both size and timing inputs. This layer serves for the network to learn more complex functions about the input flows that process both types of input (time and size related) together. We then proceed to the fully connected network which is composed of 3 layers before the final output, a number of 0 to 1, relating to the probability of that given pair being correlated.

The main differences between our network model and the original Deepcorr's is the size of each layer. We reduced the number of kernels and neurons in both the convolutional and fully connected layers, respectively, so that the network could be used with our available hardware.

V. DATASET

To collect onion service traffic we deploy a similar configuration to the one depicted in figure 5. We control both machines Alfa and Beta such that we can observe and collect traffic locally to their respective entry node in the Tor network. As a sample capture machine Alfa emulates a client, A, and machine Beta serves an onion service (OS), 6. A correlated flow pair is generated by accessing OS 6 through client A and capturing the corresponding traffic at machines Alfa and Beta. The flows captured at each machine form a correlated flow pair.

To collect multiple correlated flows with different characteristics we deployed multiple onion services and clients around the globe. To deploy both onion services and clients we used 8 different machines in the following locations: Amsterdam, Finland, London, Los Angeles, Sao Paulo, Singapore, Sydney and Tokyo. Each of these machines served both as a client and an onion service provider. The capturing process proceeds sequentially for each of the 8 clients in

Characteristic	Value
Onion services	8
Onion service pages	32
Pages per onion service	4
Clients	8
Requests per page	50
Pages per client	28
Emulated onion services	32
Emulated clients	224
Total number of flow pairs	11 200

Table III
FINAL DATASET CHARACTERISTICS.

each machine. Each client accesses all onion service pages hosted on the remaining machines 50 times per page.

The Tor Browser in conjunction with Selenium [17] was used to simulate a regular Tor user. This allows us to automate the requests to onion services programmatically using the Tor Browser which fully renders the page requesting both HTML and all necessary assets to do so.

Table III enumerates the characteristics of the final dataset. A total of 8 machines were serving a single instance of an onion service with 4 distinct pages. A total of 8 client machines were used however since we were refreshing the clients Tor circuit once all requests to a given page are complete we are effectively emulating 224 distinct clients. Each emulated client performed a total of 50 requests to the same onion service page giving a total of 11 200 accesses when considering all the 224 emulated clients. This represents 11 200 distinct correlated flow pairs.

VI. EVALUATION

Uncorrelated pair generation: The collected flow pairs only contain correlated pairs as expected. However, to train our classifiers uncorrelated pairs are essential as the network needs to learn how to distinguish between both classes. To generate an uncorrelated pair we can simply match the flow captured at the client with a flow from another connection. This scheme allows us to not only generate arbitrary numbers of uncorrelated pairs but also have some control over the distinguishability between pairs. This leads us to four distinct categories of uncorrelated pairs:

a) A client flow is matched with the onion service flow from another client to another onion service page; this maximizes the distinguishability as the full circuit is completely different as well as the generated traffic;
b) A client flow is matched with a onion service flow from that same client but to a different onion service page, where although the generated traffic is also completely different the guard nodes of the client is the same which may reduce the distinguishability;
c) A client flow is matched with an onion service flow to that same exact page, but the onion service flow was generated by another client, and therefore the circuit is totally different;
d) A client flow is matched with an onion service flow to that same exact page; the onion service flow was generated by the same client but from a different access which makes the circuit up until the rendezvous point exactly the same.

Precision metrics: For evaluating the precision, we use two metrics: *true positive rate* and *false positive rate*. True positive rate (TPR) refers to the number of correctly identified correlated pairs over the total number of correlated pairs in the dataset. False positive rate (FPR) refers to the number of uncorrelated pairs that erroneously have been marked as correlated over the total number of uncorrelated pairs in the dataset. The relation between TPR and FPR provides an effective method for comparing models, and it is inline with the goal of achieving the maximum possible TPR for the minimum possible FPR.

Performance metrics: For evaluating the performance of our system, we focus on the time the model needs to correlate a certain amount of flow pairs (*testing time*) as well as the time required for training (*training time*). This will allow the comparison of different strategies in terms of performance and compromises that can be made to improve performance at the cost of precision. To also take a more hardware independent approach we will provide relative speedup results in comparison to a base line since absolute times of both training and classification will vary greatly depending on the setup, one can however expect similar speedup rates between the experiments presented.

A. Ranking Stage Evaluation

This section presents a thorough evaluation of the ranking stage implemented by the Torpedo correlator's neural network presented in Section IV-B (see Table I). The ranking stage assigns each pair a score corresponding to the probability of correlation. By setting a configurable threshold, only the flow pairs that have a resulting score above that threshold will be selected to the correlation stage. Our evaluation aims to study if the ranking stage yields good results by consistently giving higher scores to correlated pairs as opposed to uncorrelated ones. To access the impact of the target flows characteristics we varied the dataset in two dimensions: *volume of traffic* and *time disparity*.

The following results show that Torpedo's ranking stage can indeed improve the system performance by pre-filtering most uncorrelated flows. It also improves the TPR to FPR ratio even when considering accesses to the same onion service which the correlation stage struggles to produce high accuracy results. Next we present our findings as we vary each of said dimensions individually and then combined.

Varying the volume of traffic: The results for this experiment are shown in Figure 6. In this chart, as in the following ones, we plot the CDF corresponding to the percentage of correlated flow pairs relative to the total number of pairs in various categories. Each category is consistently labeled as:

- "NO SKIP" refers to no restriction on how uncorrelated pairs are generated;
- "SKIP same size" forbids flows to onion services of the same size to form an uncorrelated pair;

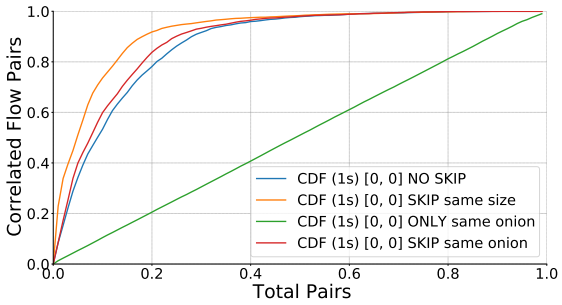


Figure 6. CDF concurrent pairs.

- "SKIP same onion" forbids flows accessing the same onion service to form an uncorrelated pair;
- "ONLY same onion" forces all uncorrelated pairs to accesses to the same onion service only;

The chart also provides a "[X,Y]" associated with each line. This range refers to the forced interval of time difference between flows in a pair. For instance, X=0 and Y=1 means that flows can deviate a maximum of 1 second (0 to 1 randomly distributed) from the original time difference of the correlated pair. In this first experiment all flow pairs have minimal time deviations. In total, there are 460 000 pairs with similar time difference for each instance of the dataset classes. By fixing all these flow pairs around the same point in time, the different categories will then explore what happens as a function of the volume of traffic returned by various onion services.

Unsurprisingly, Figure 6 shows an almost straight line for the "ONLY same onion" class. This result occurs because it is very difficult for the network to distinguish between accesses to the same onion service page (same overall size therefore very similar amounts of packets exchanged) at the same time. When no restrictions apply (i.e., class "NO SKIP"), we can expect to capture close to 80% of the correlated flow pairs, while only seeing 20% of the total number of pairs. When we forbid the same page or pages with similar size, the number of flow pairs needed to observe 80% of correlated pairs decreases as expected. This shows that even for flow pairs that are seemingly correlated, when only considering the time difference of first packet, the neural network is able to achieve fairly reasonable results. We can then see that the neural network of the ranking stage can greatly reduce the search space of flow pairs that the CNN of the correlation stage needs to process.

Varying the flow pairs' time disparity: By forcing all uncorrelated pairs to be a match between flows accessing the same onion service page we can effectively simulate 99/Y possible pairs per second targeting a given onion service, where Y is the maximum time deviation in an interval [0,Y]. For example, considering Y=1 second gives 99/1 = 99 pairs per second. Figure 7 plots the results of this experiment which are still very promising for real world data. For a time space of 16 seconds (blue line), we can capture 85% of correlated pairs by just observing 2% of the total pairs. In absolute terms, from the 9200 pairs selected at this stage, 3910 of those are indeed correlated. If we consider a maximum time difference of 1 second (red line), we are looking at over 20% for the same 2% of flow pairs, which means 920 of the total 9200 would be indeed correlated pairs.

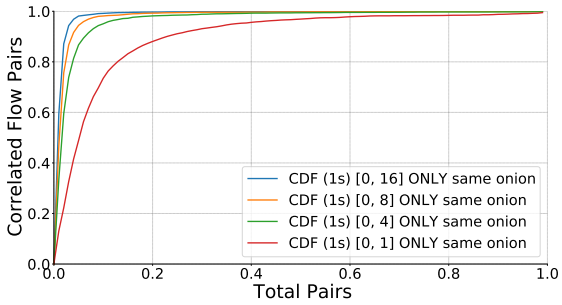


Figure 7. CDF uncorrelated only match the same onion.

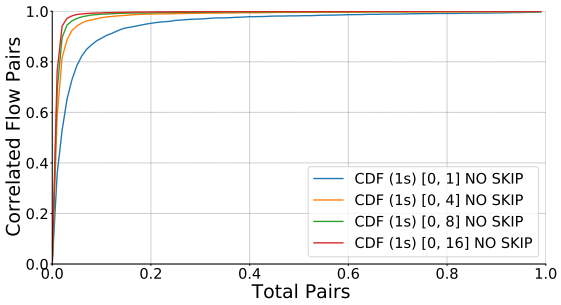


Figure 8. CDF up to 16 seconds time difference pairs.

Varying both dimensions: Figure 8 plots the results for our last experiment. If we consider 460K possible flow pairs occurring in a time space of 16 seconds (red line) we can capture 95% of correlated pairs by just observing 2% of the total pairs. In absolute terms it would mean that from the 9200 pairs selected at this stage, 4370 of those are indeed correlated. If we consider a maximum time difference of 1 second (blue line) we are looking at a little over 50% for the same 2% of flows which means 2300 of the total 9200 would be indeed correlated pairs. This however assumes no restrictions on the selection of uncorrelated pairs whereas the previous experiment reduced distinguishability by enforcing the "ONLY same onion" policy. Next, we focus on the evaluation of Torpedo's correlation stage.

B. Correlation Stage Evaluation

The correlation stage aims to correlate flow pairs through the inter packet timings and packet sizes of each flow. This operation is implemented in the second stage of the Torpedor correlator's pipeline. To evaluate the correlator CNN, we performed several experiments using our most realistic dataset, which uses real onion service web pages and respective assets, and where the Tor client requests were issued from the Tor browser itself. We present three different evaluations with different dataset holdouts.

Holdout 15% 85% experiment: Figure 9 shows the plot of true negative rates (TNR) for the uncorrelated categories and

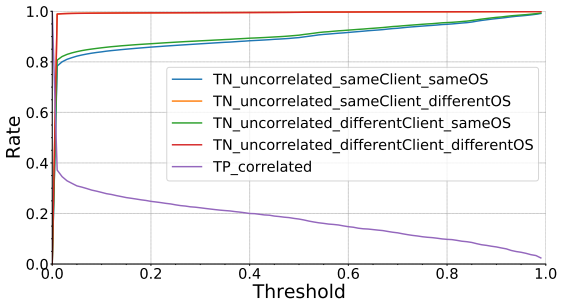


Figure 9. First results on realistic dataset.

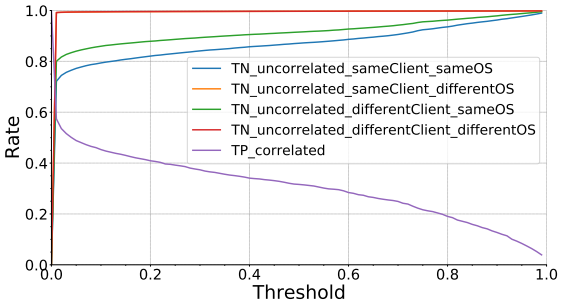


Figure 10. Results on realistic dataset for holdout 60% to 40%.

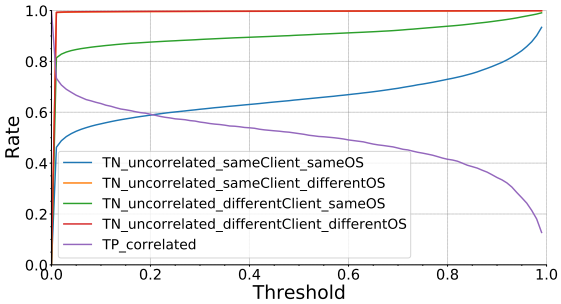


Figure 11. Results with the final dataset.

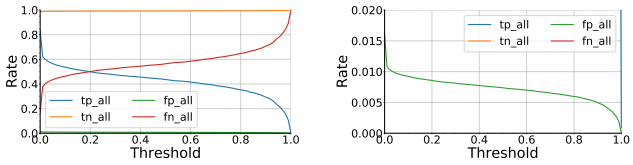
true positive rate (TPR) for the correlated category. This graph clearly shows where the network struggles to correctly classify pairs. We can see that for flow pairs where each flow was from a different onion service access, the network has a TNR of 99% while also achieving a TPR of 34% (red and orange lines are within 0.1% of each other). However, when uncorrelated pairs represent accesses to the same onion service the TNR drastically decreases, meaning we would be flagging many uncorrelated pairs as being correlated. The side-effect of this is that innocent users would probably be unjustly incriminated. Another conclusion we initially made from this particular experiment is that changing the client (therefore the circuit) did not significantly affect the classifier (lines green and blue) as they only differed by 2%. This conclusion, however, was based on a first iteration of this dataset where clients changed the circuit before every request. This change means that the same client was also using a different circuit just as a different client would, hence the proximity between these results.

Holdout 60% 40% experiment: We performed this same experiment, but now using the 60% training 40% testing dataset instance instead of the previous 15% training 85% testing. The reasoning for this change is that even though 15% training 85% testing may more accurately represent a real world case where a more limited amout of data could be captured for training purposes, the total number of pairs we collected may be too small to use such drastic training dataset reductions leaving the network very little data to learn from. This proved indeed to be somewhat the case as the following results show. Figure 10 plots the same rates as before, but now from a network trained using the 60% training 40% testing dataset. We can clearly see an increase in TPR from 34% to 57% for a slightly higher TNR for different onion service accesses of now 99.1%. We can also observe that the TNRs for uncorrelated same onion service accesses increase their distance from one another (green and blue lines). This result may come from the fact that during training there are intrinsically more examples of uncorrelated flow pairs where the access is originated from another client.

Final dataset experiment: Our last experiment represents the final iteration of our evaluation of Torpedo's correlator CNN. To overcome the shortcomings from the previous experiments where clients always changed circuit and we noticed a significant increase in TPR for higher volumes of training data, our final dataset now does not establish a new Tor circuit before each request to the same onion service (reuses the same browser instance as well without caching) and we doubled the accesses that clients made to each onion service from 25 to 50. We also kept the use of a 60% to 40% ratio for our train and test datasets respectively. Figure 11 plots the new results for this experiment.

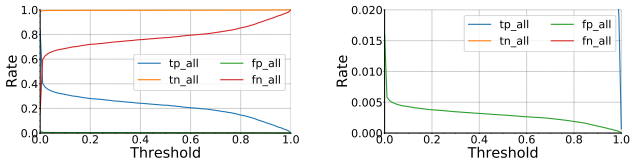
As shown in Figure 11, there are significant differences when compared to the plots from the two previous attempts. Starting with a significant increase in TPR, where we can now achieve 73% while maintaining a TNR for uncorrelated flow pairs from different onion services above 99.2%, this increase can be attributed to the higher data available for training as well as true examples of same client uncorrelated traffic. We can now clearly see the effects of changing clients (changing circuit) and using the same client (same circuit), namely that there is a considerable gap between these two lines, respectively green and blue. As expected, the TNR for the uncorrelated pairs where we match two flows that access the same onion service from the same client is significantly lower as there is little variation between the flows in these pairs. As the circuit and volume of traffic is the same, any existing traffic differences that allow the CNN to correctly classify these pairs are caused by the variations in the delays introduced by the Tor relays allocated to the circuits. These delays will be typically influenced by the Tor network workload experienced by each relay.

When we analyze the counterpart, where uncorrelated pairs match the traffic for the same onion service but from different clients, we gain 35% more TNR. This shows that the classifier has indeed learned to recognize patterns in the traffic. Had it not, we would expect a similar result



(a) Full plot. (b) Zoom on false positive.

Figure 12. Results for both stages combined.



(a) Full plot. (b) Zoom on false positive.

Figure 13. Results for both stages using combined training.

to the blue line. Interactions between client and server are intrinsically connected to the circuit that connects them since different circuits generate different delays in packets. This provides enough information for the network to make a more informed classification of the given pair.

As for final observation, in contrast to previous experiments, we can now achieve a TNR of 99.9% with a TPR of 24% for uncorrelated pairs with different onion service accesses, as well as 12.7% of TPR for a 99% TNR when considering uncorrelated pairs that match flows accessing the same onion service.

C. Full Pipeline Evaluation

This section focuses on the analysis of the results where both stages work in conjunction to correlate traffic. The results were taken by combining both stages where all flow pairs are first ranked using the ranking stage and then compared to a fixed optimised threshold. The pairs that meet the threshold are then passed onto the second stage where traffic analysis is performed to provide a final correlation probability score. All plots below depict TP/FP/TN/FN rates against the threshold of the second stage.

Full pipeline with separate training: Figure 12 (a) plots true positive, true negative, false positive and false negative rates for all instances of uncorrelated pairs. We can see that as expected false positive rates are low for all the threshold range. This is to be expected has there is no absolute time adulteration on this experiments and thus the ranking stage can accurately filter most of uncorrelated pairs. As a consequence we can also see that even for a threshold of 0 we only capture 83% of all correlated pairs as some are lost in the ranking stage as well. This setting provides a TPR of 57% for a FPR of 9.89e-3. In the other end of the spectrum we can achieve a FPR of just 1.44e-5 for a TPR of 9.21e-4. A zoomed in version of the plot can be seen in Figure 12 (b) where we can see the evolution of false positives while varying the threshold of the second stage.

Full pipeline with combined training: We replicated the previous experiment but on a second stage that had been trained using data already analyzed by the ranking stage. In this setting the same uncorrelated pair generation method is used, however, only pairs that meet the desired threshold of the ranking stage are used during training of the correlation stage. The rationale behind this method is to train the correlation stage on pairs that the ranking stage cannot accurately distinguish. In the previous experiment the correlation stage had been trained on both pairs that could accurately be classified by the ranking stage and pairs that could not, by focusing the training of the second stage on pairs that the ranking stage struggles to correctly classy we can expect to maximize their combination.

Figure 13 (a) and (b) plot our results. We can see that indeed the FPR is lower, but so it is the TPR. This feature can be used to tune the network to the user needs as we can now achieve 7.1% of TPR with a FPR of 9.6e-4. Comparing for a similar FPR of 5.8e-3 we now achieve 40.3% of TPR where before we only reached 34.5%. These results may be improved by segmenting the training dataset. Since we used the same dataset for training both stages, when training the correlation stage the ranking stage predictions were made on its training data which may disguise the network's difficulty in dealing with unseen data.

Performance analysis: To assess the total time of query response of the system we devised the following formula based on the time it takes to evaluate each of the raking and correlation stages in seconds per flow (s/f) and on the threshold of ranking stage propagation:

$$Tt = X \times Rp + X \times Th \times Cp$$

Tt is the total time in seconds for evaluating a given query. X is the number of flow pairs that have been given has input. Th is the threshold value of the ranking stage that controls the number of selected pairs which are passed over from the raking stage to the correlation stage. Rp and Cp correspond respectively to the time it takes to evaluate a given flow pair on the ranking stage and on the correlation stage, respectively. The first operand in the addition ($X \times Rp$) represents the time of execution for the ranking stage and is fixed for a given Rp and X. The second operand in the addition ($X \times Th \times Cp$) represents the time of execution for the correlation stage which for a fixed X and Cp can be customized by the Th parameter. According to our experiments, the ranking stage is able to achieve a performance of Rp=3e-5 s/f whereas the correlation stage can only reach Cp=3.5e-3 s/f. Both of these performances have been measured on a machine with an NVIDIA T4 16GB GPU, 2vCPU Intel, 32GB RAM. Considering both Rp and Cp presented above, for a Th value of 2% and a total of 1M pairs (X) we would have a total computation time of query of approximately 101 seconds.

Main findings: In summary, we present two distinct constructions of the same system with slightly different characteristics that can be used by the users to better suit their

Batch Size	Classification Time
1	479s
6	480s
12	479s
30	483s

Table IV
TF-ENCRYPTED SAME MACHINE MULTIPLE PROCESSES.

Batch Size	Classification Time
1	1177s
6	1229s
12	1089s

Table V
TF-ENCRYPTED 2VCPU WITHIN SAME LOCATION.

needs. The system is able to achieve reasonable true positive rates while maintaining very low false positive rates. Further optimizations that compromise precision for performance can be done to fit the correlator requirements.

VII. PRIVACY-PRESERVING EXTENSIONS EVALUATION

This section describes our preliminary results to implement the privacy-preserving mode for Torpedo. We used the multiparty computation framework, TF-Encrypted, to conduct our experiments. Given that in a realistic deployment of Torpedo, the privacy-preserving correlations would be performed by several participating parties connected over the Internet, we devised the following experiments.

Our main goal is to assess the impact of the network latency between the intervening parties to the overall performance of the system. We consider that the correlation process as supported by TF-Encrypted will involve 3 compute servers. We performed these experiments resorting to a reduced size convolutional neural network implemented using TF-Encrypted. All tests were performed classifying 300 random flow pairs with a length of 100 packets. The CNN used is similar in structure to the one described in DeepCorr github[3] code excepting that we used 1/8 of the original layers' size.

Testing on a single machine: To establish a baseline comparison point, we first tested the framework running on a single machine with 30GB of RAM and 8 vCPU cores based on Intel Skylake lineup of Xeon CPUs. We experimented with the following batch sizes: 1, 6, 12, 30. We found no apparent difference between the classification times for the 300 pairs while varying the batch size. Table IV presents the absolute results for this experiment.

Testing with multiple machines on a local cluster: These next tests were performed with the machine described above as master, input provider, and model provider. The remaining 3 compute servers were powered by similar machines with 10GB of RAM and 2 vCPU all in the same cluster (low network latency, around 2-3ms between hosts). Table V shows the results of this experiment. We found that the classification times tripled while in this setting compared to running in the same machine. The difference in classification time between different batch sizes seems to vary slightly with the best result (lowest classification time) being with the highest batch size of 12.

[3]https://github.com/SPIN-UMass/DeepCorr

Batch Size	Classification Time
1	629s
6	601s
12	590s
30	606s

Table VI
TF-ENCRYPTED4VCPU WITHIN SAME LOCATION.

Batch Size	Classification Time
1	13200s
6	7800s

Table VII
TF-ENCRYPTED DIFFERENT LOCATIONS.

Table VI shows the results for the same experiment but this time compute servers have 4 vCPU cores instead of 2 allowing us to access the scalability of the system and by increasing from 2 to 4 the classification times essentially were twice as fast also suggesting a linear scalability factor. This experiment shows even less variability between classification times in between batch sizes.

Testing with multiple servers connected over the Internet: As a final setting we approached a more realistic scenario (albeit a very pessimistic one) where the three compute servers were located around the globe (USA, London, and Australia). This led to a latency increase of two orders of magnitude now ranging from 180ms to 280ms between hosts. The same 4 core machines were being used in this setting along with the same master. As shown in Table VII, the increase in classification time followed a similar trend to the latency, increasing by an order of magnitude. However, it was now heavily affected by the batch size as expected. In this setting, we only tested for a batch size of 1 and 6 but the classification time between these two alternatives almost halved, representing a 41% decrease in classification time when increasing the batch size from 1 to 6. We can expect a similar improvement when increasing again from 6 to 12 as times where still very above the base line for a the same location setting. Continuing the improvement of Torpedo's privacy preserving mode is left for future work.

VIII. CONCLUSIONS

In this work, we have described the design and implementation of Torpedo, a distributed, cooperative, deanonymization system where ASes and law enforcement agencies cooperate to perform traffic correlation on Tor onion service flows. By adapting existing traffic correlation attacks based on neural networks to regular Tor traffic, Torpedo allows efficient correlation attacks to be performed on onion service traffic.

We present Torpedo, the first system to perform traffic correlation attacks on Tor onion service traffic. Torpedo uses an innovative system architecture based on a two-stage approach that allows for processing high volumes of data while maintaining reasonable precision on flow

correlation. A thorough evaluation of the Torpedo correlation component showed that the system is able to process large volumes of data in small periods of time, while maintaining a reasonable true positive rate and a low false positive rate. An extension providing privacy guarantees to the system was implemented and evaluated allowing Torpedo to perform traffic correlation on encrypted flow pairs while not requiring ASes to completely trust the correlator.

We identify multiple aspects of the presented system that demand further improvements. First, there is a need to fully implement the system with regard to the security protocols that assure a secure channel between all intervening parties. Second, our current evaluation assumes there is a clear way of intercepting individual flows communicating with a given onion service. If multiple users simultaneously access a given onion service it is not clear how client-specific flow capture and discrimination can be performed. Thus, a deeper study is need to access the system in such conditions. Third, the prototype implemented focused solely on the correlator component of Torpedo and does not implement the client and probe modules. One direction for future work comprises the implementation and evaluation of these modules. In particular, there is a pressing need to assess the probe module ability to flag and log Tor guard relay traffic in real time. This involves the design and implementation of all the security protocols related to the communication between parties involved in flow de-anonymization, as well as the support for the mentioned query types by the Torpedo components. Fourth, all modules should be implemented and fully integrated with TF-Encrypted to support the privacy-preserving correlation of flows. The thorough evaluation of this setup comprises another direction for future work. Finally, there is the need to assess the coverage of onion service Tor traffic that the cooperating ASes leveraging Torpedo can achieve. This involves the study of which ASes concentrate a larger number of guard nodes as well as which ASes concentrate large fractions of hosted onion services.

Acknowledgments

We would like to thank Professor Luís Rodrigues for the initial discussion regarding our system design. We are also grateful to Bernardo Ferreira, Bernardo Portela, and Diogo Barradas for their feedback and fruitful discussion provided during the preparation of this work.

References

[1] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi, "No nat'd user left behind: Fingerprinting users behind nat from netflow records alone," in *IEEE International Conference on Distributed Computing Systems*, 2014.

[2] S. Coull, C. Wright, F. Monrose, M. Collins, and M. Reiter, "Playing devil's advocate: Inferring sensitive information from anonymized network traces." in *Network and Distributed Systems Security Symposium*, 2007.

[3] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter, "Browser fingerprinting from coarse traffic summaries: Techniques and implications," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2009.

[4] J. Yuan, Z. Li, and R. Yuan, "Information entropy based clustering method for unsupervised internet traffic classification," in *IEEE International Conference on Communications*, 2008.

[5] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*, 2004.

[6] N. Christin, "Traveling the silk road: a measurement analysis of a large anonymous online marketplace," in *International Conference on World Wide Web*, 2013.

[7] K. Soska and N. Christin, "Measuring the longitudinal evolution of the online anonymous marketplace ecosystem," in *USENIX Security Symposium*, 2015.

[8] R. Hurley, S. Prusty, H. Soroush, R. J. Walls, J. Albrecht, E. Cecchet, B. N. Levine, M. Liberatore, B. Lynn, and J. Wolak, "Measurement and analysis of child pornography trafficking on p2p networks," in *International Conference on World Wide Web*, 2013.

[9] G. Weimann, "Going dark: Terrorism on the dark web," *Studies in Conflict & Terrorism*, 2016.

[10] M. Casenove and A. Miraglia, "Botnet over tor: The illusion of hiding," in *International Conference On Cyber Conflict*, 2014.

[11] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *International Conference on Financial Cryptography*, 2004.

[12] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *IEEE Symposium on Security and Privacy*, 2005.

[13] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *European Symposium on Research in Computer Security*, 2006.

[14] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: Routing attacks on privacy in tor," in *USENIX Security Symposium*, 2015.

[15] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: Strong flow correlation attacks on tor using deep learning," in *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[16] L. Overlier and P. Syverson, "Locating hidden servers," in *2006 IEEE Symposium on Security and Privacy*, 2006.

[17] S. S. Salunke, *Selenium Webdriver in Python: Learn with Examples*, 2014.