ProTO: Proactive Topology Obfuscation Against Adversarial Network Topology Inference

Tao Hou[†], Zhe Qu[†], Tao Wang[‡], Zhuo Lu[†] and Yao Liu[†]
[†]University of South Florida, Tampa, FL, USA, {taohou@mail., zhequ@mail., zhuolu@, yliu@cse.}usf.edu
[‡]New Mexico State University, Las Cruces, NM, USA, taow@nmsu.edu

Abstract*—The topology of a network is fundamental for building network infrastructure functionalities. In many scenarios, enterprise networks may have no desire to disclose their topology information. In this paper, we aim at preventing attacks that use adversarial, active end-to-end topology inference to obtain the topology information of a target network. To this end, we propose a Proactive Topology Obfuscation (ProTO) system that adopts a detect-then-obfuscate framework: (i) a lightweight probing behavior identification mechanism based on machine learning is designed to detect any probing behavior, and then (ii) a topology obfuscation design is developed to proactively delay all identified probe packets in a way such that the attacker will obtain a structurally accurate yet fake network topology based on the measurements of these delayed probe packets, therefore deceiving the attacker and decreasing its appetency for future inference. We show that ProTO is very effective against active topology inference with minimum performance disruption. Experimental results under different evaluation scenarios show that ProTO is able to (i) achieve a detection rate of 99.9% with a false alarm of 3%, (ii) effectively disrupt adversarial topology inference and lead to the topology inferred by the attacker close to a fake topology, and (iii) result in an overall network delay performance degradation of 1.3% - 2.0%.

I. Introduction

The topology of a network is the fundamental information for building network infrastructure functionalities, such as path routing and packet forwarding. Many network applications require prior knowledge of the topology, especially for applications built on top of overlay network techniques [1], such as peer-to-peer (P2P) network, virtual personal networks (VPN), content delivery networks (CDN) and voice over IP (VoIP, e.g., Skype) [2]–[4]. In addition, network topology is the essential information required in network diagnosis and failure localization [5]–[8].

However, the knowledge of network topology can advance network attackers' malicious objectives, leading to more precise or effective attacks. For example, attackers can leverage topology information to craft advanced denial-of-service (DoS) attacks to concentrate on important nodes or links in a targeted network to maximize the attack impact [9] or even conceal malicious activities by confusing the global system failure monitoring algorithms [10]. Therefore, it may not always be desirable or even prohibitive to disclose the internal network topology to the outside, which is particularly important for organizational/enterprise systems to protect commercial interests and private information.

The undesirability or prohibition of disclosing network topology does not necessarily discourage attackers from acquiring such information by adversarial, active inference. There are mainly two types of topology inference techniques that can be used by attackers for the malicious purpose: internally cooperative topology inference [11] and external end-to-end topology inference (also called as tomography-based topology inference) [12]. The former technique usually utilizes tools (e.g., traceroute or ping) and cooperates with internal nodes to collect their corresponding response messages to infer topology (e.g., assuming internal nodes should respond to ping). As an alternative, external end-to-end topology inference shows the promise of discovering the topology through end-to-end path performance measurement (e.g., inferring through packet delays or loss rates on end-to-end paths) without internal nodes' cooperation. Studies [12]–[21] have shown that external end-to-end topology inference can achieve a high accuracy rate.

For the purpose of ensuring network security, it is necessary to develop countermeasures for defending against adversarial topology inference. To combat internally cooperative topology inference, network administrators can simply disable internal routers' response to traceroute or ping [22]. Advanced designs, such as NetHide [23], can prevent topology leaking through internally cooperative topology inference while keeping the functionalities of traceroute and ping. However, these existing techniques cannot defend against external end-to-end topology inference, which poses a real and crucial threat to networks considering that tomography based external measurement has been supported by a number of network products and manufacturers (e.g., Ericson [18], Cisco [19], Microsoft [20], Huawei [21]). Though these efforts aim to prompt the convenience of network management for meritorious inference, they can also be leveraged by malicious attackers.

In this paper, we focus on mitigating the risk of topology leakage due to adversarial external end-to-end topology inference. As an attacker can perform topology inference based on measuring the performance of probe packets going through a target network, an intuitive way to defense is to detect such probe packets then disable their forwarding. However, this way usually results in that the attacker draws attention to inference failure and then develops follow-on actions. Further, the network topology is relatively static. Once acquiring the topology information, the attacker does not need to frequently update such information. This indicates that the detection rate of a designed defense mechanism must be very high to prevent the attacker from easily obtaining such information even for once. As there always exists a tradeoff between detection rate and false alarm, a higher detection rate generally indicates a higher false alarm. Hence, simply denying forwarding any potential probe packet will lead to preventing a fair amount of legitimate traffic that is misidentified from going through the network. To solve these issues, we propose a Proactive Topology Obfuscation (ProTO) system that proactively defends against adversarial topology inference.

There exist two major modules in ProTO: (i) a probing behavior identification mechanism designed biased towards a very high detection rate while allowing for a slight false alarm and (ii) a topology obfuscation design proactively delaying all identified probe packets in a way that the attacker will obtain a structurally accurate yet fake network topology based on the measurements of these delayed packets. ProTO aims to deceive the attacker and decrease the possibility of further inference attempts. The system does not disrupt any packet forwarding inside the network, but only intentionally delays malicious probe packets identified by the identification mechanism. We implement and evaluate ProTO with various setups over realistic network topologies. To the best of our knowledge, ProTO is the first system designed against adversarial end-to-end topology inference. There are several key designs and contributions in ProTO to balance security and cost.

Identification of probing behavior: An attacker can disguise their probe packets as regular data packets going through the network, we propose a lightweight machine learning based classifier for ProTO to identify probe packets. Through combining offline self-training and online incremental updating, the classifier achieves a very high detection rate of 99.9% and also has a low false alarm rate of around 3% in our experiments. We also adopt a voting-based strategy to ensure improving the data representativeness in incremental updating, meanwhile maintaining a low computation overhead for performance-sensitive network devices.

Topology obfuscation: We first formulate the model for topology inference, and then adopt a min-max approach for topology obfuscation: as the maximum-likelihood estimation (MLE) in general minimizes the topology inference error, we aim to disrupt the topology inference of MLE used by a potential attacker. In particular, we propose the obfuscation method to delay probe packets such that a fake topology, which is structurally correct but independent of the real topology, will be obtained by the attacker. Experiments show that ProTO is able to effectively disrupt adversarial topology inference and lead to the topology inferred by the attacker close to the fake topology. We also prove that an attacker gains no information of real network topology from the fake topology.

Minimum disruption of packets: If a packet is identified as a probe packet, it will be delayed by ProTO. As the identification mechanism allows for false alarms, we must ensure that (i) the delay performance degradation of the packet is minimized such that a misidentified packet will have the minimum delay penalty, and at the same time (ii) topology obfuscation is achieved. We use an optimization framework to solve the objective. Experimental results show that ProTO leads to an overall network delay degradation of 1.3% - 2.0%.

II. Models and Research Scope

In this section, we present network and attack models, introduce the research problem, and present an overview of the ProTO system.

A. *Network and Attack Models*

Though topology inference techniques may be utilized for network management and have been supported by technology companies (e.g., Ericson [18], Cisco [19], Microsoft [20], Huawei [21]), they can be often leveraged by adversaries to obtain the topology of a network even when the network has no desire to disclose such information [10], [24], [25]. As the fundamental information for packet routing and forwarding, the knowledge of network topology can be utilized by attackers to advance their malicious purposes. Common attacks that can use topology information to advance or exacerbate impacts include Distributed Denial-of-Service (DDoS) attacks [9], [10], [23], Domain Name System (DNS) poisoning [26], [27], and Internet censorship [28]–[30].

In this research, we consider a network connected to a larger network system (e.g., the Internet). The nodes inside the network are cooperating with routing/forwarding of packets traveling through the network. There exists an attacker that has no prior knowledge of the network topology but aims to infer the topology information. To this end, the attacker can place or use nodes outside the target network to launch an external end-to-end topology inference.

External end-to-end topology inference follows a tree structure for packet probing and topology recovering, in which the attacker uses one source and a set of receivers $\mathcal{R} = \{1, 2, \cdots, R\}$ (R is the number of receivers) outside the network to infer the network topology. As an example shown on the left-hand side of Figure 1, let $\mathcal{T} = (\mathcal{V}, \mathcal{L})$ denote the topology tree of the target network with node set $\mathcal{V}$ and link set $\mathcal{L}$. The attacker's source s is connected to the root of the tree, and each receiver has a path to one leaf of the tree. A link with its endpoints, which is neither the root or a leaf node, is called an internal link in $\mathcal{T}$. Tree structure based topology inference is widely adopted to obtain a real network topology [31]. A more complicate topology (e.g., mesh networking) can be also obtained by constructing multiple trees with different root nodes [31].

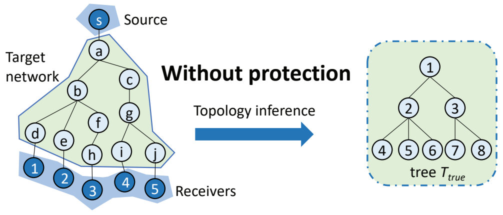


Fig. 1. An example for malicious topology inference.

In malicious topology inference, the attacker sends probe packets from source s, which pass through different paths inside the network to receivers $\mathcal{R}$ to obtain end-to-end path measurement results, such as packet loss rate or packet delay. In this paper, we use the delay metric as the measurement metric as it is the most widely-used one in topology inferences.

The design intuition of topology inference is that when packets are forwarded from the source to the receivers, they may go through shared links inside the network before they split and reach different receivers; therefore, the network topology fundamentally affects the correlations of delays observed at different receivers. In particular, Denote by $x_{i,j}$ the correlation delay for a pair of receivers i and j ($i, j \in \mathcal{R}$), and $x_{i,j}$is the sum of the delays on all shared links between the end-to-end path from the source to receivers i and the end-to-end path from the source to receiver j (e.g., the link between nodes a and b is only the shared link between source to receiver 1 and source to receiver 2 in Figure 1). Approaches [12]–[17] have been developed and used by the attacker to compute correlation delays $\{x_{i,j}\}_{i,j\in\mathcal{R}}$ and based on which the complete network topology can be recovered.

As Figure 1 shows, when there is no protection deployed, the attacker is able to recover the topology $\mathcal{T}_{true}$. Note that a non-branching node (i.e., node with less than two child nodes) is not identifiable in topology inference [17]. Thus, the recovered topology tree is a logical tree, which consists of branching nodes of the real topology and the logical links between them. For example, nodes c and g are merged as node 3 in the recovered logic tree $\mathcal{T}_{true}$ in Figure 1.

B. Design Objectives

It is essential to develop effective countermeasures to mitigate the risk of topology leakage due to external end-to-end topology inference. Traditionally, a potential way for designing countermeasures is to first identify possible probe packets then disable them (e.g., via banning the prober's IP address). However, the topology information of a network is relatively static information, and a network does not frequently change its network topology configuration. This means that an attacker can always try to send probe packets from time to time to obtain such information. The information is obtained as long as the attacker succeeds once. Moreover, disabling misidentified legitimate traffic may significantly degrade the network performance. Hence, the effectiveness of this traditional detect-then-disable approach solely relies on the complete accuracy of identifying malicious behavior, which can be quite challenging.

Our perspective is that instead of designing completely accurate identification and disabling probe packets from any identified source, we can relieve the burden of identification and proactively delay (potentially malicious) packets going through the target network. Therefore, we adopt a detect-then-obfuscate strategy. Specifically, we need a probing behavior identification algorithm that can be (even coarsely) designed biased towards a very high detection rate but allows for a slight or moderate false alarm rate: any malicious probing behavior can be identified and some legitimate traffic may also be misidentified. When a source is identified as a potential prober, we do not drop all of its packets, but proactively delay its packet forwarding with minimum disruption to prevent topology inference. In this way, a small amount of legitimate traffic under false alarm can also go through the network with minimum performance degradation.

Through proactively delaying malicious probe packets, network administrators do not need to suppress malicious topology inference by disabling the external end-to-end measurement, but can deliver a structurally accurate yet fake topology to the attacker. Thus, the design deceives the attacker and decreases the possibility of further inference attempts. We design a practical system ProTO that adopts this proactive topology obfuscation strategy to combat malicious inference and ensure the confidentiality of network topology.

C. Overview of ProTO System Design

We develop the ProTO system for a target network to achieve the design objectives. Figure 2 shows the system architecture of ProTO, which consists of two major modules: (i) identification and manipulation module and (ii) topology control module.

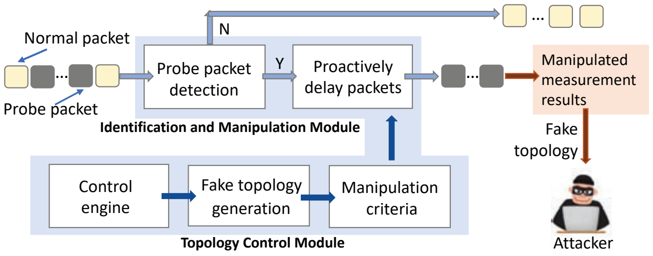


Fig. 2. The system architecture of ProTO.

Identification and Manipulation Module: this module first identifies the probe packets and then manipulates their forwarding delay inside the network according to the topology obfuscation specified by the topology control module. In Section III, we propose a lightweight machine learning based method to classify probe packets, which ensures a very high detection rate and also tolerates a low false alarm rate.

Topology Control Module: the module provides a uniform control interface for network administrators to manage the ProTO system. It generates obfuscated topologies and associated packet delay manipulation criteria as the outputs to the identification and manipulation module to delay identified probe packets. In Section IV, we formulate the topology obfuscation strategy that intentionally delays identified probe packets such that an attacker using topology inference only obtains a structurally correct yet fake topology, which is independent of the real network topology.

III. Probe Packet Identification

As discussed in Section II-C, the identification and manipulation module in ProTO aims to remove the burden of ensuring complete accuracy and design a probing identification mechanism biased towards a very high detection rate. This indicates that (i) a very high detection rate means any malicious probing behavior should be identified; (ii) it unavoidably leads to a number of false alarms (because there is always a trade-off between detections and false alarms). Although legitimate traffic that is misidentified may be intentionally delayed, ProTO ensures that the overall network performance loss is very limited according to the topology obfuscation mechanism offered in Section IV.

Identifying probe packets is essentially a data classification problem. In this section, we propose a machine learning based framework for efficient classification.

A. Identification Model

The identification model is designed as an incremental semi-supervised learning framework [32], [33], which is suitable for the scenarios with limited amount of labeled data. Before the deployment of ProTO, the system first performs a self-training phase to collect the initial training data, including the packets labeled as either probe or non-probe packets. This training dataset is then used to build the semi-supervised classifier. When ProTO is online, it continues to collect packets as non-labeled data. Then, feature data extracted from these collected packets becomes testing data that will be classified by the initial classifier and be added to the training set to incrementally improve the classification performance.

We develop a lightweight k-Nearest Neighbor (light-k-NN) approach to identify probe packets. Different to traditional k-NN, light-k-NN is more suitable for our use case by adopting two designs: 1) a multi-round dynamic method to adaptively train the weights for different features is designed. This method continuously tunes the weights with the increase of data size when the system is online. 2) a voting-based lazy-learning update strategy is implemented. Under this strategy, the incremental update of the data pool indeed increases the representativeness. At the same time, it maintains the data pool in a limited size, such that the performance and space overhead does not increase. We present the two design details in Sections III-B and III-C, respectively. In this way, light-k-NN is as easy as traditional k-NN to be used, but is more suitable for ProTO for real-time network devices.

Central to light-k-NN is the notion of distance between packets. In particular, the distance $D(P, P')$ between two packets P and P' is calculated by computing the distance between their numerical feature vectors, i.e.,

$$D(P, P') = \sum_{1 \leq n \leq F} w_n |f_n(P) - f_n(P')|, \tag{1}$$

where $f_n(P)$ and $f_n(P')$ denote the n-th entries of the feature vectors of packets P and P', respectively; F is the number of features used for packet classification; and w_n is the weight of the n-th feature. Under light-k-NN, if a packet is closer to the k packets identified as probe packets previously, it will be classified as a probe packet.

The key differences between probe packets and normal data packets are not only the characteristics of a single packet, but also the correlation relationships among different packets [12]–[21]. These correlation relationships reflect a holistic transmission patterns of probe packets. Unless not using external end-to-end inference, these holistic patterns cannot be evaded even if the attacker actively disguises its probe packets. In particular, we adopt the features listed in [34] for classification.

The vector $W = [w_1, w_2, ..., w_F]$ includes the weights for all features in computing the distances between packets. These weights are computed initially from the training dataset in the self-training phase and fine-tuned during the online operation.

B. Training the Weights

In comparison to traditional k-NN classifiers which assign either no weight or static weights to different features [35]–[37], light-k-NN adopts a multi-round dynamic method to adaptively tune the weights online. In particular, we first initialize all weights as $W_0 = \{1, 1, ..., 1\}$, then follow (1) to calculate their k-NN distances and classify all the packets in training dataset as either `probe` or `normal`. Denote by $\mathcal{S}_{probe}$ or $\mathcal{S}_{normal}$ as the sets that contain `probe` packets and `normal` packets, respectively. When ProTO is online and starts to monitor packets, we use a two-step training procedure to tune the weights for different features.

Step 1: In this step, we orient to examine the correctness and usefulness for each feature in the calculation of k-NN distance following the current weight set. The objective for this examination is to check if a feature is less-weighted or over-weighted in the calculation. Specifically, for each packet $P \in \mathcal{S}_{probe}$, we choose m packets in $\mathcal{S}_{probe}$ closest to P to form a set $\mathcal{S}_1$ and m packets in $\mathcal{S}_{normal}$ closest to P to form a set $\mathcal{S}_2$. Then, define the per-feature distance for feature $n \in [1, F]$ between P and $P' \in \mathcal{S}_1$ as $D_n(P, P') = |f_n(P) - f_n(P')|$, and compute the set of all per-feature distances $\{D_n(P, P')\}_{P' \in \mathcal{S}_1}$. Let d_{max} and d_{mean} be the maximum and mean values of the set, respectively. Then, for feature n, compute all per-feature distances between packet P and $P'' \in \mathcal{S}_2$ to obtain $\{D_n(P, P'')\}_{P'' \in \mathcal{S}_2}$, in which the number of per-feature distances larger than d_{max} is denoted by $C_1(P, n)$ and the number of per-feature distances smaller than d_{mean} is denoted by $C_2(P, n)$. Finally, we compute the sum $C_1(n) = \sum_{P \in \mathcal{S}_{probe}} C_1(P, n)$ and the sum $C_2(n) = \sum_{P \in \mathcal{S}_{probe}} C_2(P, n)$, respectively.

Step 2: This step aims to optimize the calculated k-NN distances by adjusting the weight for each feature based on the results of step 1. A large value of $C_1(n)$ indicates feature n is useful for identifying probe packets and should be given more weight. By contrast, a large value of $C_2(n)$ indicates that packet $P \in \mathcal{S}_{probe}$ has a small per-feature distance to normal packets, thus feature n is not an evident feature to differentiate the probe packet from normal packets and should be less-weighted. Accordingly, we adjust the weights for different features as an online tuning algorithm. For each packet arrival, we compare $C_1(n)$ with $C_2(n)$ for each feature n, and adjust the value of its weight w_n by $\Delta w_n = (C_1(n) - C_2(n))q/(m|\mathcal{S}_{probe}|)$, where the step q is set to be 0.01 by default.

C. Incrementally Updating Training

Since the initial training dataset collected from self-training before the system deployment is limited, incrementally updating the training dataset by adding the new classified data can improve the k-NN classifier when the system is online.

However, increasing the size of training dataset will also lead to a computational burden and space overhead, we develop a method that can ensure light-k-NN improves the accuracy incrementally, while also limiting the training size. In particular, we implement a voting system to maintain a data pool for the active training dataset, which contains two classes: `probe` or `normal`. The system maintains an upper bound of the number of packets for each class. When an incoming packet P is classified into a class $C \in \{\texttt{probe}, \texttt{normal}\}$, the vote count of each of the k nearest packets in the training dataset of class C closest to P will be incremented by 1 because they are considered useful for classification.

The new packet P will be then added into the training dataset of class C. If the number of packets in class C is greater than the upper bound, the packet with the least votes will be removed from the training dataset of class C. It can be expected that through this voting system, more packets close to the decision boundary will be kept and packets that are less important to the classification will be gradually removed to limit the training dataset size. For a newly added packet, its number of votes is initialized as the average number of votes of all other packets in its class.

IV. Topology Obfuscation

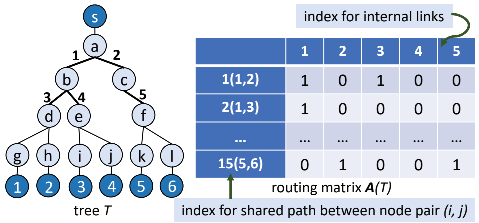
Once packets are identified as probe packets, they should be intentionally delayed for topology obfuscation. In this section, we design the topology obfuscation technique for the topology control module in ProTO to i) ensure the attacker only obtains a fake topology and ii) limit the cost of network efficiency. We first formulate the topology inference problem, then present our proactive topology obfuscation technique.

A. Inference Formulation

We first formulate the external end-to-end topology inference as a fundamental mathematical model. As discussed in Section II-A, the goal of the attacker is to obtain the topology tree of the target network by sending probe packets going through the network and measuring the correlation delays $\{x_{i,j}\}_{i,j\in\mathcal{R}}$. The set of all possible topology trees is denoted as $\mathcal{F}$, and we call $\mathcal{F}$ a *forest*. We denote the delay on link $l \in \mathcal{L}$ as μ_l. Then, the relationship between correlation delays $\{x_{i,j}\}_{i,j\in\mathcal{R}}$, the real topology $\mathcal{T}$, and the link delays $\{\mu_l\}_{l\in\mathcal{L}}$ can be formulated in a linear way as

$$\mathbf{x} = \mathbf{A}\boldsymbol{\mu}, \tag{2}$$

where $\mathbf{x} = [x_{1,2}, x_{1,3}, \cdots x_{1,R}, x_{2,3}, x_{2,4}, \cdots, x_{2,R}, \cdots, x_{R-1,R}]^T$ (the operator $\cdot^T$ denotes the matrix transpose) (i.e., $\mathbf{x}$ is obtained by stacking all elements in $\{x_{i,j}\}$ into a column vector); the vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \cdots, \mu_L]^T$ with L being the number of internal links in the network; and $\mathbf{A} = [a_{k,m}]$ is called the routing matrix, which depends on the topology $\mathcal{T}$. In particular, element $a_{k,m}$ in $\mathbf{A}$ has value 1 if the k-th link of the network is shared by the receiver pair corresponding to the j-th element in $\mathbf{x}$, and value 0 otherwise.



	1	2	3	4	5
1(1,2)	1	0	1	0	0
2(1,3)	1	0	0	0	0
…	…	…	…	…	…
15(5,6)	0	1	0	0	1

Fig. 3. An example: routing matrix $\mathbf{A}$ of tree $\mathcal{T}$.

We use a simple example to demonstrate how the routing matrix is determined. As shown in Figure 3, there are 6 receivers and thus the number of different receiver pairs is $\binom{6}{2} = 15$. In tree $\mathcal{T}$, the link set is $\{1, 2, 3, 4, 5\}$. The routing matrix $\mathbf{A} = [a_{k,m}]$ is therefore a 15-by-5 matrix with $1 \le k \le 15$ and $1 \le m \le 5$. In particular, $a_{k,m}$ is 1 if the m-th link is a shared link for the k-th receiver pairs. For example in Figure 3, the pair of receivers 1 and 2 (which share link 1 on their paths) corresponds to x_1, therefore $a_{1,1} = 1$.

Based on (2), we write the probability density of $\mathbf{x}$ as $p(\mathbf{x}|\mathbf{A}, \boldsymbol{\mu})$. The likelihood function of $\mathcal{T}$ can be written as

$$L(\mathbf{x}|\mathcal{T}) \equiv p(\mathbf{x}|\mathbf{A}, \widehat{\boldsymbol{\mu}}). \tag{3}$$

The MLE obtains the topology $\widehat{\mathcal{T}}$ that maximizes the likelihood by

$$\widehat{\mathcal{T}} \equiv \arg\max_{\mathcal{T}\in\mathcal{F}} L(\mathbf{x}|\mathcal{T}). \tag{4}$$

The calculated $\widehat{\mathcal{T}}$ is the desired topology of the target network in external end-to-end topology inference.

B. Topology Obfuscation

Topology obfuscation aims to camouflage the attackers by misleading them to recover a fake topology in the inference. From the formation of inference, it is obvious that the network can manipulate the path measurement results to achieve this goal. Intuitively, we should achieve the best obfuscation effort by maximizing the difference between the recovered topology and the real topology. However, if the attacker is aware of the defender's goal (e.g., maximizing the difference), the attacker may try to reverse the real topology from recovered topology via finding the topology that has the maximum difference.

A successful topology obfuscation strategy should deliver a non-reversible fake topology and avoid the aforementioned situation. From the mathematical perspective, a randomly generated $\mathbf{A}_m$ will maximize the difficulty to recover the real topology. We present a theoretical analysis for random generation of fake topology in Section IV-C. In particular, we randomly generate a fake topology $\mathbf{A}_m$ independent of the real topology $\mathbf{A}$ in (2). Then, we intentionally affect the probe packets going through the network to influence the path measurement results of an attacker such that it obtains $\mathbf{A}_m$ instead of $\mathbf{A}$ from topology inference. To this end, theoretically, based on the underlying formulation (2) for topology inference, we multiply by a manipulation matrix $\mathbf{F}$ both sides in (2) and obtain $\mathbf{Fx} = \mathbf{FA}\boldsymbol{\mu}$, where the left-hand side $\mathbf{Fx}$ represents the manipulated measurement results observed by the attacker. In order for the attacker to obtain the fake topology $\mathbf{A}_m$ based on the observation $\mathbf{Fx}$ using MLE. This linear equation must hold $\mathbf{Fx} = \mathbf{FA}\boldsymbol{\mu} = \mathbf{A_m}\boldsymbol{\mu}$. Therefore, our goal of topology obfuscation is to find the manipulation matrix $\mathbf{F}$ such that

$$\mathbf{FA} = \mathbf{A}_m, \tag{5}$$

given the real topology $\mathbf{A}$ and the fake one $\mathbf{A}_m$. Note that (5) is obtained based on the assumption that the attacker will use MLE to estimate the topology. The MLE generally minimizes the estimation error in statistical inference [38]. Thus, obfuscation based on (5) can be considered as a min-max strategy to disrupt the best performance that can be obtained by the attacker.

C. Fake Topology Generation and Security Analysis

To successfully deliver a non-reversible fake topology, the control module in ProTO randomly generates the fake matrix $\mathbf{A}_m$ once the system is deployed. We develop a random generation algorithm to get a structurally correct yet fake topology $\mathbf{A}_m$ that is independent of $\mathbf{A}$. Specifically, suppose $\mathbf{A}$ is an m-by-n matrix, ProTO keeps generating an m-by-n matrix with all elements randomly selected from $\{0, 1\}$ until the generated matrix represents a connected tree structure. Then, ProTO uses the generated matrix as $\mathbf{A}_m$ for topology obfuscation. In case the attacker may notice the obfuscation efforts, ProTO will keep the generated $\mathbf{A}_m$ as the target fake topology for a certain time so the attacker will always obtain the same topology even with multiple inferences.

We analyze the information security of the proposed random fake topology generation. The successful topology obfuscation strategy should ensure the attacker cannot derive the real routing matrix $\mathbf{A}$ even with the knowledge of the randomly generated matrix $\mathbf{A}_m$.

Mathematically, we analyze the security in the form of entropy, which denotes the average uncertainty of the topology to the attacker. For the attacker without the initial knowledge of the matrix $\mathbf{A}$, $\mathbf{A}$ can be treated as a random matrix and its entropy is defined as $H(\mathbf{A}) = -\sum_{\mathbf{a}\in\mathbf{A}} P_{\mathbf{A}}(\mathbf{a}) \log P_{\mathbf{A}}(\mathbf{a})$, where $P_{\mathbf{A}}(\mathbf{a})$ is the probability when $\mathbf{A} = \mathbf{a}$, $\mathbf{a}$ is a specific topology matrix. Similarly, the entropy of $\mathbf{A}$ conditioned on the attacker knowing $\mathbf{A}_m$ is defined as $H(\mathbf{A}|\mathbf{A}_m) = \sum_{\mathbf{a}_m\in\mathbf{A}_m} P_{\mathbf{A}_m}(\mathbf{a}_m) H(\mathbf{A}|\mathbf{A}_m = \mathbf{a}_m)$, where $P_{\mathbf{A}_m}(\mathbf{a}_m)$ is the probability when $\mathbf{A}_m = \mathbf{a}_m$. Due to the independence between $\mathbf{A}_m$ and $\mathbf{A}$, $H(\mathbf{A}|\mathbf{A}_m = \mathbf{a}_m) = H(\mathbf{A})$ and thus $H(\mathbf{A}|\mathbf{A}_m) = H(\mathbf{A})$, which indicates the knowledge of $\mathbf{A}_m$ gives the attacker no information of real routing matrix $\mathbf{A}$.

D. Optimization based Topology Obfuscation

After $\mathbf{A}_m$ is generated, the topology control modules computes the manipulation matrix $\mathbf{F}$ based on (5). It is worth noting that the topology matrix $\mathbf{A}$ is an m-by-n matrix with $m > n$. Thus, there will be infinite solutions for $\mathbf{F}$ in (5). As a result, we need to find the best solution to (5).

First, not all solutions can be practical in real-world systems. As $\mathbf{Fx}$ represents the path measurement delays observed by the attacker, $\mathbf{Fx}$ should have comparable values to the original measurement $\mathbf{x}$. Thus, we should impose a constraint in searching for $\mathbf{F}$ such that $\forall x \in \mathbf{Fx} - \mathbf{x}, x \in [0, \delta_{max}]$, where δ_{max} is called the maximum allowed deviation for the delay. Note that x should not be less than 0 because obfuscation efforts may not be able to decrease the packet delay due to the physical constraint of the network system, but it is feasible to intentionally increase the packet delay to manipulate the path measurements through the network.

Then, we transfer the problem of finding $\mathbf{F}$ such that $\mathbf{FA} = \mathbf{A_m}$ given aforementioned constraint into the following optimization problem

$$\begin{aligned} \underset{\mathbf{F}}{\text{minimize}} \quad & \|\mathbf{FA} - \mathbf{A_m}\|_2 , \\ \text{subject to} \quad & \forall x \in \mathbf{Fx} - \mathbf{x}, x \in [0, \delta_{max}]. \end{aligned} \tag{6}$$

There exists a tradeoff if choosing the value of δ_{max} for topology obfuscation: a large value can increase the network performance overhead; and a small value may decrease the effectiveness of the obfuscation efforts as the search range in the optimization (6) is limited. For the default setting, we choose the maximum value occurred for each path in normal measurements as default δ_{max}.

E. Obtaining Manipulation Matrix $\mathbf{F}$

In (6), the manipulation matrix $\mathbf{F}$ to be found is an m-by-m matrix, $\mathbf{A}$ and $\mathbf{A_m}$ are m-by-n matrices, representing the real topology and the fake topology, respectively. To solve (6), we first write $\mathbf{F} = \{f_{i,j}\}$, $\mathbf{A} = \{a_{p,q}\}$ and $\mathbf{A_m} = \{\widehat{a}_{p,q}\}$. Let $\mathbf{a_i} = [a_{1i},\ a_{2i},\ ...,\ a_{mi}]$ and $\mathbf{f} = [f_{11}, ...,\ f_{1m},\ ...,\ f_{m1}, ...,\ f_{mm}]^T$, then it holds that $\|\mathbf{FA} - \mathbf{A_m}\|_2 = \|\mathbf{Mf} - \mathbf{m}\|_2$, where $\mathbf{M}$ is an $m \times n$ by $m \times m$ matrix satisfying $\mathbf{M} = [\mathbf{B},\ ...,\ \mathbf{B}]^T$; $\mathbf{B} = \text{diag}(\mathbf{a}_1,\ \mathbf{a}_2,\ ...,\ \mathbf{a}_m)$; and $\mathbf{m}$ is a column vector with m^2 elements with the k-th element m_k in $\mathbf{m}$ being $\hat{a}_{p,q}$ in $\mathbf{A_m}$, where $p = \left\lfloor \frac{k}{n} \right\rfloor$ and $q = k - pn$. Let $\mathbf{H} = \text{diag}(\mathbf{x}^T,\ \mathbf{x}^T,\ ...,\ \mathbf{x}^T)$, we rewrite (6) as

$$\begin{aligned} \underset{\mathbf{f}}{\text{minimize}} \quad & \mathbf{f^T M^T M f} - \mathbf{2mMf} \\ \text{subject to} \quad & \mathbf{0} \leq \mathbf{Hf} \leq \mathbf{x}_{max}, \end{aligned} \tag{7}$$

where $\mathbf{x}_{max}$ denotes the maximum allowed delays for all links after obfuscation, each of whose elements is set to be the normal link delay in the network plus δ_{max}. Thus, solving (6) is equivalent to solving (7). In (7), $\mathbf{M^T M}$ is a semi-definite matrix and therefore (7) can be solved efficiently by quadratic programming in polynomial time.

F. Proactively Delaying Probe Packets

After probe packets are identified and the manipulation matrix $\mathbf{F}$ is calculated, the topology control module sends $\mathbf{F}$ to the identification and manipulation module for delay manipulation of probe packets. Specifically, ProTO first calculates the truth correlation delay for each pair of receivers based on the current link performance metrics in its network. Then, it computes the difference between the true correlation delay and the desired correlation delay based on the manipulation matrix $\mathbf{F}$. Finally, ProTO delays a probe packet by the time difference between the two correlation delays. As aforementioned, the light-k-NN identification framework in ProTO is designed biased towards a very high detection rate, while allowing false alarms. Such a design ensures obfuscating the topology by delaying all probe packets at the cost of a slight network performance degradation for wrongly identified normal data packets.

V. System Implementation and Evaluation

In this section, we use different network scenarios based on real world topologies to evaluate the effectiveness and efficiency of the proposed ProTO system. We first present the implementation of ProTO and setups of the evaluation testbed. Then, we evaluate the effectiveness of the proposed probe packet identification algorithm (i.e., light-k-NN) in the identification and manipulation module. Finally, we provide and discuss the overall performance against adversarial topology inference by successful topology obfuscation.

A. Implementation and Experimental Setups

ProTO is implemented in P4 [39] integrated with Python. P4 is a domain-specific language (DSL) for programming the data plane of network forwarding devices (e.g., switches and routers). In our design, the P4 program mainly focuses on packet processing related tasks, including packet capturing, feature extracting and packet manipulation (i.e., delaying the packets). While the algorithms used to generate the fake topology $\mathbf{A}_m$ and compute the manipulation matrix $\mathbf{F}$ are written in Python. The P4-based implementation for packet processing is hardware independent (i.e., requiring no knowledge of hardware during development) and can be compiled according to hardware specifications into realistic devices.

We use three real-world network topologies from Internet Topology Zoo [40] in the evaluation, including a small, a medium and a large network as shown in Figure 4. We create these three networks on two high-performance computing workstations. Each network node is created as an independent virtual machine that runs OpenWrt [41] as the operating system. OpenWrt is an open source Linux based operating system that can work as routing management system. The advantage of choosing OpenWrt is that it can compatibly execute P4-based code.

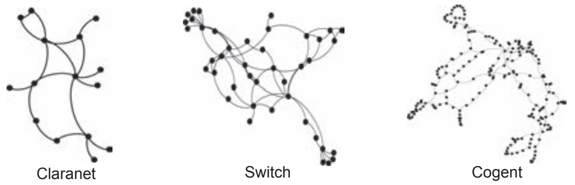


Fig. 4. Structures of three collected topologies.

We collect various types of data packets by running different network applications, including web browsing, file transfer, online chatting, and video streaming on a local-area network, and replay these data packets as the background network traffic in our experimental network. In this way, we simulate a realistic use scenario in which probe packets are mixed with regular data packets going through a target network. We also tune the amount of background traffic to measure the performance in different network traffic conditions. For the low utilization condition, the background loads for different links range from 5% to 50%, with an overall load of 30%; for the high utilization condition, the loads for different links range from 10% to 90%, with an overall of 45%.

B. Performance of Probe Packet Detection

To evaluate the probe packet detection performance of the proposed light-k-NN classifier, we first define two performance metrics (i.e., detection rate and false alarm rate); then we use these metrics to quantitatively examine if our light-k-NN design can achieve the design goal of guaranteeing the confidentiality of network topology by ensuring the enforcement of our detect-then-obfuscate strategy.

1) Performance Metrics: We evaluate the identification performance of the proposed light-k-NN classifier from two perspectives: 1) detection rate, which is defined as the number of probe packets correctly identified divided by the total number of probe packets, and 2) false alarm rate, defined as the number of normal packets misidentified as probe packets divided by the total number of all normal packets.

2) Detection Performance: The performance of the proposed light-k-NN classifier to detect probe packets is evaluated on the three networks. We find that the evaluation results in terms of detection and false alarm rates are nearly the same under low and high utilization conditions. Table I shows the evaluation results under the high network utilization condition. In the experiments, the size of the initial training dataset is chosen to be 500, 1000, 2000, and 3000. As discussed in Section III-C, ProTO incrementally increases the training size until an upper bound is reached. The incremental updating scale is defined as the ratio between the upper bound and the initial training size. In our experiments, the scale is selected from [1.0, 1.5, 2.0, 2.5] and we also evaluate the case of no online training updating as shown in Table I.

Impact of initial training dataset: We can observe in Table I that the size of initial training has a substantial impact on the detection performance of the light-k-NN classifier. Specifically, when the size increases from 500 to 2000, the detection rate increases substantially in all scenarios (i.e., in different networks with different updating scales). For example, the detection rate improves from 94.5% to 99.7% in Claranet with updating scale of 1.0. At the same time, the false alarm rate also decreases (e.g., the rate decreases from 11.4% to 3.4% in Claranet with updating scale of 1.0).

Impact of incremental updating: It is seen in Table I that the increase of the updating scale will lead to performance improvement of the classifier. For example, in Claranet, when

TABLE I
DETECTION PERFORMANCE FOR DIFFERENT SIZES OF INITIAL TRAINING DATASET AND UPDATING SCALES.

Topology	Size of Initial Training Dataset	Incremental Updating Scale During Online Training				
		No Updating	1.0	1.5	2.0	2.5
Claranet	500	D=0.922, F=0.143	D=0.945, F=0.114	D=0.961, F=0.096	D=0.978, F=0.087	D=0.982, F=0.079
	1000	D=0.960, F=0.095	D=0.976, F=0.076	D=0.985, F=0.053	D=0.992, F=0.042	D=0.993, F=0.035
	2000	D=0.985, F=0.040	D=0.993, F=0.035	D=0.997, F=0.034	D=0.999, F=0.030	D=0.999, F=0.028
	3000	D=0.991, F=0.038	D=0.997, F=0.034	D=0.998, F=0.030	D=0.999, F=0.028	D=0.999, F=0.026
Switch	500	D=0.922, F=0.143	D=0.945, F=0.115	D=0.960, F=0.086	D=0.977, F=0.078	D=0.981, F=0.070
	1000	D=0.960, F=0.095	D=0.975, F=0.077	D=0.984, F=0.054	D=0.991, F=0.043	D=0.992, F=0.036
	2000	D=0.985, F=0.040	D=0.993, F=0.035	D=0.996, F=0.034	D=0.999, F=0.030	D=0.999, F=0.028
	3000	D=0.990, F=0.039	D=0.997, F=0.034	D=0.997, F=0.031	D=0.998, F=0.028	D=0.999, F=0.026
Cogent	500	D=0.920, F=0.144	D=0.943, F=0.116	D=0.960, F=0.088	D=0.976, F=0.078	D=0.980, F=0.070
	1000	D=0.959, F=0.097	D=0.974, F=0.077	D=0.983, F=0.054	D=0.990, F=0.044	D=0.992, F=0.037
	2000	D=0.984, F=0.041	D=0.992, F=0.037	D=0.995, F=0.035	D=0.999, F=0.033	D=0.999, F=0.030
	3000	D=0.989, F=0.039	D=0.995, F=0.036	D=0.997, F=0.033	D=0.998, F=0.030	D=0.998, F=0.028

[1] D = detection rate, F = false alarm rate. [2] The Size of Initial Training Dataset indicates the number of data points for both `probe` packets and `normal` packets in the initial training dataset. [3] The Incremental Updating Scale indicates the scale of active training dataset (compared with the initial training dataset) during online training. For example, scale = 1.0 means the packets in training dataset will be updated during online training, but the total number of packets will keep the same with the initial dataset. While No Updating indicates the training dataset will not be updated during online training.

there is no online updating, the classifier with the initial training size of 500 has a detection rate of 92.2% and a false alarm rate of 14.4%; when the classifier incrementally updates its training and sets the updating scale to be 2, the detection rate is improved to 97.8% and the false alarm rate is reduced to 9.8%. Overall, when the updating scale becomes 2.5, the classifier achieves a detect rate of 98.0% – 99.9% and a false alarm rate of 2.6% – 7.9% in the three network scenarios.

Detection rate and false alarm rate over time: Based on the evaluation results in Table I, we set the size of the initial training dataset to 2000 and the updating scale to 2 for follow-on experiments. The setups achieve a good balance between the probing detection performance and the computational complexity incurred by the detection. Figure 5 shows the detection rate achieved by the light-k-NN classifier as the number of probe packets sent to ProTO. As we can see from the figure, when the ProTo is online and the attacker starts to send probe packets, the detection rate is gradually improved over time, reaching 99.9% in all three network scenarios. Figure 5 also shows the ratio of the online training size to the initial training size during incremental online updating in ProTO. The figure shows that the ratio increases linearly and eventually reaches the updating scale 2.0.

Figure 6 shows the false alarm rate and the ratio of the online training size to the initial training size, as functions of the number of normal packets sent to ProTO. In the figure, we can see that as ProTO gradually processes more incoming normal packets, the false alarm rate continues to drop and eventually remains stable at around 3% for all three network scenarios. Figures 5 and 6 show that ProTO achieves a detection rate of 99.9% and a false alarm rate of around 3% when it processes a sufficient number of packets.

C. Evaluation of Topology Obfuscation

After identification of probe packets, the objective of ProTO is to intentionally delay these packets in the network such that the attacker can only obtain a fake topology by using end-to-end topology inference. To evaluate the effectiveness of the topology obfuscation in ProTO, we conduct experiments on the three network scenarios for two cases: (i) no defense is used to combat topology inference and (ii) ProTO is activated to obfuscate probe packets. For each network scenario, we run the experiment 100 times with randomly generated topologies and average the results under cases (i) and (ii) to evaluate the effectiveness and cost of ProTO.

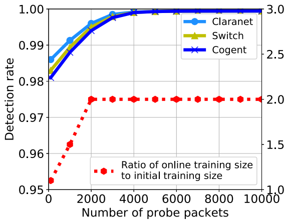


Fig. 5. Detection rate for probe packet identification.

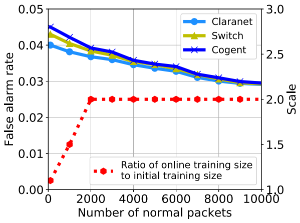


Fig. 6. False alarm rate for probe packet identification.

1) Effectiveness Metrics: As ProTO aims to mislead the attacker to obtain a fake topology, it is essential to measure the difference between the real topology and the fake one that the attack obtains. A popular metric to measure the difference between two trees is the Tree Edit Distance (TED) defined in [42]. TED calculates the difference between two trees $\mathcal{T}_1$ and tree $\mathcal{T}_2$ as a set of pre-defined editing operations by which tree $\mathcal{T}_1$ can be mapped/transformed to tree $\mathcal{T}_2$.

To make the evaluation more intuitive, we define a similarity score within [0, 1] based on TED. Given $\mathcal{T}_1$ and $\mathcal{T}_2$, we first compute their TED as TED_0, then calculate the TED between $\mathcal{T}_1$ and a zero-node tree, denoted by TED_1 (which can be considered as the cost needed to remove everything in $\mathcal{T}_1$) as well as the TED between $\mathcal{T}_2$ and a zero-node tree, denoted by TED_2 (which can be considered as the cost needed to construct $\mathcal{T}_2$ from scratch). Then, the similarity score is defined as $similarity\ score = 1 - \frac{\mathrm{TED}_0}{\mathrm{TED}_1 + \mathrm{TED}_2}$. The similarity score is 1 if $\mathcal{T}_1 = \mathcal{T}_2$, and has a smaller value if $\mathcal{T}_1$ is more evidently different from $\mathcal{T}_2$. We define the benchmark score as the average similarity score between the real network topology and a randomly generated topology, and obtain using simulations

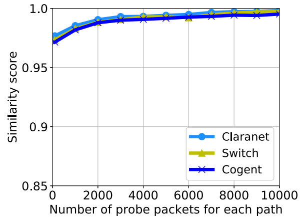


Fig. 7. Similarity score between the inferred topology and the real topology without protection.

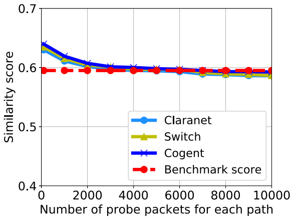


Fig. 8. Similarity score between the inferred topology and the real topology under protection.

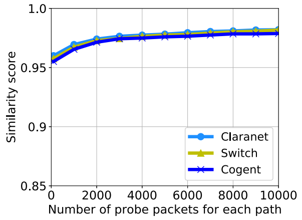


Fig. 9. Similarity score between the inferred topology and the intended topology.

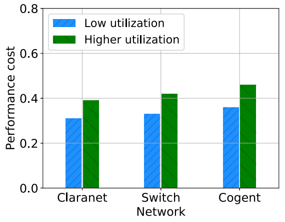


Fig. 10. The performance cost of normal packets misidentified by ProTO.

that the benchmark score is 0.6. Hence, we consider ProTO to be effective if the the similarity score between the real topology and the inferred topology is close to 0.6.

2) Evaluating Effectiveness of Topology Obfuscation: We first consider case (i) in which there is no defense for the networks. Figure 7 shows the similarity score between the real topology and the inferred topology, as a function of the number of probe packets (the minimum number is 100) sent along each path between the source and a receiver under topology inference. It is observed in Figure 7 that if no defense deployed, the attacker can easily obtain the real topology with high accuracy. For example, if the attacker send 10000 probe packets for each path between the source and a receiver, it is able to recover the topology with a similar score close to 1.

We then consider case (ii) in which ProTO is deployed. Figure 8 depicts the similarity score between the real topology (under ProTO's protection) and the inferred topology, as a function of the number of probe packets sent along each path between the source and a receiver. The figure shows that the similarity score between the real topology and the inferred topology is significantly reduced to around the benchmark score of 0.6. This indicates that the inferred topology under ProTO has little difference from a random topology and therefore ProTO is effective against topology inference.

We further analyze the similarity score between the inferred topology and the fake topology $\mathbf{A}_m$ that is generated according to Section IV-C in ProTO's control module to evaluate the difference between the intended topology by ProTO and the inferred topology by the attacker, which is shown in Figure 9. We can find in Figure 9 that overall, the optimization based topology obfuscation in (6) delivers an intended topology with high accuracy to the attacker.

3) Performance Cost of Topology Obfuscation: ProTO unavoidably introduces the network performance cost by delaying normal packets going through the network that are misidentified as probe packets. We aim to measure the performance degradation due to false alarm and intentional delaying. We define the performance cost as the ratio of the extra delay incurred by ProTO to the original delay for a normal packet going through the network. For each of the three network scenarios, we measure (i) the performance cost of normal packets that are misidentified as probe packets and delayed by ProTO and (ii) the average performance cost of all normal packets (that are either correctly identified or misidentified) going through the network.

TABLE II
AVERAGE PERFORMANCE COST FOR ALL NORMAL PACKETS.

	Low Utilization	High Utilization
Claranet	1.28%	1.93%
Switch	1.33%	1.95%
Cogent	1.35%	1.99%

Figure 10 shows the performance cost in case (i) in which only misidentified normal packets are measured. From Figure 10, we observe that ProTO increases the delay of a misidentified normal packet by 31%-37% under the low utilization condition, and by 39% - 45% under the high utilization condition. As only a limited number of normal packets can be misidentified by ProTO (e.g., 3% false alarm rate shown in Figure 6), the overall performance disruption is expected to be small for legitimate traffic. Table II shows the performance cost in case (ii) in which all normal packets going through the network are measured to compute the average performance cost. From Table II, we can find that the average cost due to the deployment of ProTO is around 1.3% and 2% for the low and high utilization conditions, respectively.

As result, it is concluded that ProTO is an effective system to combat adversarial topology inference with low performance cost.

VI. Conclusions

In this paper, we provide a systematic study on effectively defending against adversarial topology inference. We develop a practical system ProTO that adopts a detect-then-obfuscate framework to combat any potential attack. The ProTO system consists of two major modules: (i) a light-k-NN probing behavior identification mechanism is designed biased towards a very high detection rate and (ii) a topology obfuscation design that proactively delays all identified probe packets in a way such that the attacker will obtain a structurally accurate yet fake network topology based on the measurements of these delayed probe packets. Experimental results show that ProTO can effectively and efficiently combat adversarial topology inference.

Acknowledgement

This work at USF was supported in part by NSF CNS-1553304 and CNS-1717969.

References

[1] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *ACM SIGOPS Operating Systems Review*, 2002.

[2] Nabeel Farooq Butt, Mosharaf Chowdhury, and Raouf Boutaba. Topology-awareness and reoptimization mechanism for virtual network embedding. In *International Conference on Research in Networking*, 2010.

[3] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 2005.

[4] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *IEEE Communications Magazine*, 2009.

[5] Nick Duffield. Simple network performance tomography. In *Proc. of ACM IMC*, 2003.

[6] Paul Barford, Nick Duffield, Amos Ron, and Joel Sommers. Network performance anomaly detection and localization. In *Proc. of IEEE INFOCOM*, 2009.

[7] Liang Ma, Ting He, Ananthram Swami, Don Towsley, Kin K Leung, and Jessica Lowe. Node failure localization via network tomography. In *Proc. of ACM IMC*, 2014.

[8] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. Detection and localization of network black holes. In *Proc. of IEEE INFOCOM*, 2007.

[9] Susmit Panjwani, Stephanie Tan, Keith M Jarrin, and Michel Cukier. An experimental evaluation to determine if port scans are precursors to an attack. In *Proc. of IEEE DSN*, 2005.

[10] Shangqing Zhao, Zhuo Lu, and Cliff Wang. When seeing isn't believing: On feasibility and detectability of scapegoating in network tomography. In *Proc. of IEEE ICDCS*, 2017.

[11] RFC 792: Internet Control Message Protocol (ICMP). https://www.rfc-editor.org/rfc/rfc792.txt, 2018.

[12] Nick G Duffield, Joseph Horowitz, F Lo Presti, and Don Towsley. Multicast topology inference from measured end-to-end loss. *IEEE Transactions on Information Theory*, 2002.

[13] Nick G Duffield and Francesco Lo Presti. Network tomography from measured end-to-end delay covariance. *IEEE/ACM Transactions on Networking (TON)*, 2004.

[14] Hongyi Yao, Sidharth Jaggi, and Minghua Chen. Network coding tomography for network failures. In *Proc. of IEEE INFOCOM*, 2010.

[15] Pegah Sattari, Christina Fragouli, and Athina Markopoulou. Active topology inference using network coding. *Physical Communication*, 2013.

[16] Brian Eriksson, Gautam Dasarathy, Paul Barford, and Robert Nowak. Efficient network tomography for internet topology discovery. *IEEE/ACM Transactions on Networking (TON)*, 2012.

[17] Mark Coates, Rui Castro, Robert Nowak, Manik Gadhiok, Ryan King, and Yolanda Tsang. Maximum likelihood network topology identification from edge-based unicast measurements. In *Proc. of ACM SIGMETRICS*, 2002.

[18] Fabio Ubaldi, PEPE Teresa, and Marzio Puleri. Method and device for network tomography, November 9 2017. US Patent App. 15/529,819.

[19] Jean-Philippe Vasseur. Operations administration management for path computation element chains, August 28 2012. US Patent 8,254,272.

[20] Richard Black, Austin Donnelly, and Cedric Fournet. System and method for network topology discovery, May 13 2014. US Patent 8,724,512.

[21] Yulin Yuan, Zhiming Ye, and Xiaoji Fan. Method and device for discovering network topology, May 25 2017. US Patent App. 15/426,891.

[22] Mehmet H Gunes and Kamil Sarac. Analyzing router responsiveness to active measurement probes. In *International Conference on Passive and Active Network Measurement*, 2009.

[23] Roland Meier, Petar Tsankov, Vincent Lenders, Laurent Vanbever, and Martin Vechev. Nethide: Secure and practical network topology obfuscation. In *USENIX Security Symposium*, 2018.

[24] Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. Network tomography: Recent developments. *Statistical science*, 2004.

[25] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. Timing analysis for inferring the topology of the bitcoin peer-to-peer network. In *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE, 2016.

[26] Cristina L Abad and Rafael I Bonilla. An analysis on the schemes for detecting and preventing arp cache poisoning attacks. In *Proc. of ICDCSW*, 2007.

[27] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P Lee, and Wenke Lee. Recursive dns architectures and vulnerability implications. In *Proc. of NDSS*, 2009.

[28] Xueyang Xu, Z Morley Mao, and J Alex Halderman. Internet censorship in china: Where does the filtering occur? In *Proc. of Springer PAM*, 2011.

[29] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapé. Analysis of country-wide internet outages caused by censorship. In *Proc. of ACM IMC*, 2011.

[30] HB Acharya, Sambuddho Chakravarty, and Devashish Gosain. Few throats to choke: On the current structure of the internet. In *Proc. of IEEE LCN*, 2017.

[31] Xian Zhang and Chris Phillips. A survey on selective routing topology inference through active probing. *IEEE Communications Surveys & Tutorials*, 14(4):1129–1141, 2012.

[32] Xiaofei He. Incremental semi-supervised subspace learning for image retrieval. In *Proc. of ACM Multimedia*, 2004.

[33] Ahsanul Haque, Latifur Khan, and Michael Baron. Sand: Semi-supervised adaptive novel class detection and classification over data stream. In *Proc. of AAAI*, 2016.

[34] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical report, 2013.

[35] Eui-Hong Han, George Karypis, and Vipin Kumar. Text categorization using weight adjusted k-nearest neighbor classification. In *PAKDD*, 2001.

[36] Yihua Liao and V Rao Vemuri. Use of k-nearest neighbor classifier for intrusion detection. *Computers & security*, 2002.

[37] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 2011.

[38] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[39] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014.

[40] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 2011.

[41] OpenWrt Project: Welcome to the OpenWrt Project. https://openwrt.org, 2018.

[42] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 1989.