Combating Adversarial Network Topology Inference by Proactive Topology Obfuscation

Tao Hou, Tao Wang, Zhuo Lu, and Yao Liu

***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 evaluate ProTO under different evaluation scenarios. Experimental results 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%.**

***Index Terms*—Network Systems, Topology Inference, Network Security, Machine Learning.**

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 [2], such as peer-to-peer (P2P) networks, virtual personal networks (VPN), content delivery networks (CDN) and voice over IP (VoIP, e.g., Skype) [3]–[5]. In addition, network topology is the essential information required in network diagnosis and failure localization [6]–[9].

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 [10] or even conceal malicious activities by confusing the global system failure monitoring algorithms [11]. 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 [12] and external end-to-end topology inference (also called as tomography-based topology inference) [13]. 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 [13]–[22] 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 [23]. Advanced designs, such as NetHide [24], 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 [19], Cisco [20], Microsoft [21], Huawei [22]). 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 of defense is to detect such probe packets then disable their forwarding. However, this way may cause the attacker to draw attention to inference failure and then develop 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

Tao Hou, Zhuo Lu and Yao Liu are with University of South Florida, Tampa, FL 33620. E-mails: {taohou@mail., zhuolu@, yliu@cse.}usf.edu

Tao Wang is with New Mexico State University, Las Cruces, NM 88003. E-mail: taow@nmsu.edu

An earlier version of the work [1] was presented in IEEE INFOCOM 2020.

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 prevent 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 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 detection rate of 99.9% and has a 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%.

The remainder of this paper is organized as follows. In Section II, we introduce the preliminaries and related work. In Section III, we introduce the models and state the research problem. In Section IV, we design the probing behavior identification mechanism. In Section V, we present the strategy for network topology obfuscation. Then, we present and discuss the experimental setups and results in Section VI and finally conclude this paper in Section VII.

II. Preliminaries and Related Work

In this section, we introduce the background and related work of topology inference.

A. Topology Inference

Many topology inference techniques have been proposed to infer the topology of a network. These techniques can be classified as two categories: (i) internally cooperative topology inference and (ii) external end-to-end topology inference.

The topology discovery tools, which are developed based on internally cooperative inference, need cooperations from internal network nodes and collect their responses to probe packets, such as traceroute or ping packets, then utilize these responses to further infer the network topology. For example, Spring et al. developed a tool called Rocketfuel that can successfully recover the network topology by utilizing traceroute [25]; Archipelago (Ark) which is launched by CAIDA conducts large-scale traceroute-based topology measurements to obtain insights into Internet infrastructures [26]. There are also several topology discovery projects built on top of RIPE Atlas which provides Internet-wide traceroute measurement data for researchers to use [27].

External end-to-end inference is also called tomography-based inference [13]. In external end-to-end topology inference, a few probe nodes are placed outside a target network. Probe packets are sent between probe nodes to pass through the target network. Then, the network topology can be inferred based on the end-to-end measurements of probe packets. For example, [13], [14], [18] build different external end-to-end topology inference systems for topology recovery with the accuracy rate up to 95%. Recently, a lot of work has been focused on improving the effectiveness and efficiency of the inference techniques, and making them more practical [15]–[17] for even large-scale networks. Technology companies (e.g., Ericson [19], Cisco [20], Microsoft [21], Huawei [22]) have also developed specialized devices/techniques that can be used for external end-to-end topology inference. These efforts aim to assist network management by reducing network consumption and improving accuracy for topology discovery.

B. Topology Related Attacks

Though these inference techniques may be utilized for network management, 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 [11], [28], [29]. As the fundamental information for packet routing and forwarding, the knowledge of network topology can be utilized

by attackers to advance their malicious purposes, especially by geographical information related attacks. Common examples of using topology information to advance or exacerbate attacks include Distributed Denial-of-Service (DDoS), Domain Name System (DNS) poisoning, and Internet censorship.

DDoS: DDoS attack is a malicious attempt to disrupt the normal functionality of a network, it aims to overflow the capacity of a target network with overwhelming traffic [30]. With the target network's topology information, attackers are able to craft advanced attacks to concentrate on important nodes or links to maximize the attack impact [10] or even conceal malicious activities by confusing the global system failure monitoring algorithms [11]. It has been shown in [24] that the attack's efficiency can be significantly increased through precisely selecting target nodes or links based on topology information.

DNS poisoning: The DNS is a decentralized naming system designed for network address resolution [31]. DNS poisoning can divert network traffic away from legitimate targets and towards fake ones [32], [33]. Many attacks can be launched on top of DNS poisoning, such as man-in-the-middle (MITM) attack, blind packet injection and network phishing [34], [35]. With the knowledge of a victim network's topology, attackers can spoof the DNS address lookup database in a more accurate way. Furthermore, considering that DNS servers are organized as a hierarchy network, if attackers can infer the topology of the DNS server network, the poisoning's successful rate and impact can be maximized [36], [37].

Internet censorship: Censorship opposites the philosophy that the Internet is born free, it surveils and controls network communications which include content considered as sensitive or harmful [38]. The information of network topology can be used by censorship to facilitate restrictions on Internet access. A specific censorship policy may pursue targeting on particular points (e.g., sensitive users) in the network. With the assistance of topology information, the censorship system can accurately identify the target points in the network, therefore to emphatically monitor them [39]. The system can also improve the performance by selecting critical network locations as censor points based on topology information [40], [41].

C. Defenses against Malicious Inference

To defend against internally cooperative topology inference, the simplest way is to prohibit inference by disabling any internal node's response to traceroute or ping. However, traceroute or ping based network functionalities will also be disabled in the meantime. In addition, existing studies proposed many specified topology protection techniques [24], [42]–[44]. For example, [42] proposed a deception technique that can prevent adversaries from getting the true topology of a network. The technique mainly utilizes the inherent weaknesses of traceroute packets (i.e., lacking authenticity and integrity) to mislead the attackers to get a wrong topology. The work in [43] proposed a system called RDS that can deceive attackers to camouflage critical resources in the network. More recently, an obfuscation tool NetHide [24] is proposed to obfuscate topologies by modifying traceroute packets in the data plane.

However, to the best of our knowledge, there is no systematic tool developed to combat malicious external end-to-end topology inference. This type of inference does not rely on ping or traceroute and thus makes existing defense designs focused on internally cooperative topology inference ineffective to protect network topology information. Because attackers can perform external inference by sending probe packets going through the network in addition to other legitimate data traffic, it is non-trivial to develop a mechanism that can mitigate the risk of leaking topology information. Therefore, in this paper, we focus on developing a system to defend against external topology inference for a target network.

III. Models and Research Scope

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

A. Network and Attack Models

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 [45]. More complicated topologies (e.g., mesh networking) can be also obtained by constructing multiple trees with different root nodes [45], these include topologies only with acyclic parts and topologies with cyclic parts or multigraph parts [13]–[18], [45]–[48]. Though there are still limitations for external end-to-end topology inference, how to ameliorate the inference technique is not the primary research goal of this work.

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

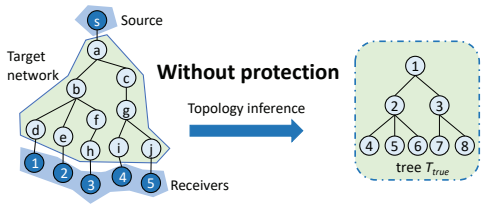

Fig. 1: An example for malicious topology inference.

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 [13]–[18] 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 [18]. 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 logical 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 [49]–[51] 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 for once inference. Moreover, disabling misidentified legitimate traffic may significantly degrade the network performance or even influence the network functionalities. Hence, the effectiveness of this detect-then-disable approach solely relies on the complete accuracy of identifying malicious behavior, which is 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.

There exist major challenges in developing ProTO: (i) how to create a biased identification towards the detection rate while still maintaining an allowable false alarm rate? (ii) how to make sure an attacker can indeed recover a fake, invertible topology? (iii) delaying packets inside the network inevitably leads to performance loss, how to balance the security and the network performance?

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.

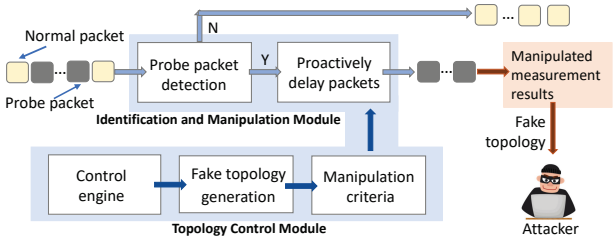

Fig. 2: The system architecture of ProTO.

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 IV, 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 V, 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.

IV. Probe Packet Identification

As discussed in Section III-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 V.

Identifying probe packets for topology inference is essentially a data classification problem. In this section, we propose a machine learning based framework for efficient classification. We first summarize the packet features we extract for classification, then present and discuss the proposed method.

A. Extracting Features

It is non-trivial to identify malicious probe packets from network traffic, as an attacker may try to disguise the probe packets as regular data (TCP/UDP) packets going through the network. In addition, if an attacker is aware of our detection, it can try to camouflage its packets to evade the packet characteristics based detection. By investigating existing external end-to-end inference methods [13]–[22] and analyzing patterns of probe packets, we notice characteristics associated with probe packets for topology inference that are different from normal data packets and can be potentially used for identification.

- In external end-to-end inference, the network topology is recovered by measuring the correlation delays among different pairs of receivers as discussed in Section III-A. Specifically, in calculating the correlation delay between a pair of receivers, the probe packets are grouped and sent to both receivers in a pair. All the packets for a pair of receivers have the same source address, while the destination addresses belong to one of the two receivers.
- The two groups of probe packets for a receiver pair always go along the same path inside the network from the source until they reach a branching node, from which they are directed to different receivers.
- In order to recover the topology consisting of all nodes in the network, the total number of receiver pairs should be large. Furthermore, a pair of receivers needs to be measured for hundreds or even thousands of times to obtain a mean delay value to remove random measurement noise [13], [14], [18]. Hence, the network should observe a high-volume of traffic with the same patterns. Normally, the interval of two probe packets usually are tens of milliseconds [13]–[18]. Therefore, for a general network with hundreds to thousands receiver pairs, the traffic with the same patterns will last for minutes to hours in external end-to-end inference.

These observations show that 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. These correlation relationships reflect the holistic transmission patterns of probe packets in external end-to-end inference. Unless not using external end-to-end inference, these holistic patterns cannot be evaded even if the attacker actively disguises its probe packets. As a result, it is still possible to identify a group of probe packets based on their characteristics. ProTO leverages a lightweight machine learning framework to identify probe packets for real-time operations.

Different from previous traffic identification studies [52]–[56] that usually extract hundreds of features for machined learning based identification, ProTO carefully shrinks the parameter space by selecting a limited number (36 by default and also customizable) of features which are related to the above observations. These 36 features are selected through pre-training. In particular, we first obtain weights for all features listed in [53] by adopting a weight training algorithm (see Section IV-C for details). Then, we select the top N weighted features from them to further train the labeled data and compare the evaluation results for different values of N. We define a metric Performance Gain to show the evaluation results for different selected N features.

$$\text{Performance Gain} = \frac{(A_c - A_p)/A_c - (P_c - P_p)/P_c}{(P_c - P_p)/P_c}.$$

where A_c and A_p represent the accurate rate for current and previous N values, respectively; and P_c and P_p are the computation overheads for current and previous values of N, respectively. Specifically, the computation overhead is defined as the training time on the initial training dataset. A positive Performance Gain indicates that the accuracy grows faster than the computation overhead with the increase of number of selected features. Another benefit of pre-training is to reduce the communication overhead caused by transmitting the collected features, because the feature space is shrank by this procedure. The evaluation results in Figure 3 show we can get perpetual positive Performance Gain with the increase of N until it reaches 36. Our subsequent experimental results also demonstrate that the identification scheme with 36 features can achieve a detection rate as high as 99.9%, at the same time maintain a low false alarm of 3%. ProTO is capable of capturing the differences between probe and normal packets, meanwhile ensures real-time processing of these features.

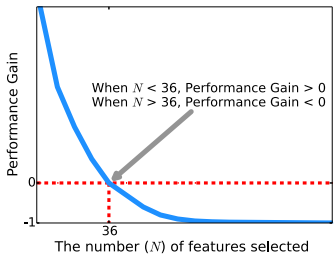

Fig. 3: Performance Gain for different N.

A combination of these features can represent holistic transmission patterns caused by topology inference. We list 6 features in Table I as examples to show the features selected for identification. For example, the source address, destination address, the transit path and several statistic features (e.g., count1 and count2 in Table I) for a series of packets together can form a basic pattern in our design observations. Next, we discuss how to continue to optimize different weights to

these features to improve the overall detection rate during the training and identification.

TABLE I: 6 examples in the features used in identification.

Feature Name	Description
Source address	IP address
Destination address	IP address
Transit path	Series of internal link indices
Time interval	Time interval to the previous packet
Count1	Number of packets with the same size of header bytes in a time window
Count2	Number of packets with the same size of control bytes in a time window

B. Identification Model

The identification model is designed as an incremental semi-supervised learning framework [57], [58], 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.

Though the network traffic may vary greatly over different time, the key features of the probe packets indeed remain consistent. As discussed in Section IV-A, the temporal and spatial correlations among probe packets are quite unique compared to common network traffic. Specifically, for each topology inference all probe packets will follow similar packet setting, travel through correlated router, and end up to different pairs of receivers. Also, probe packets are usually aggregated in groups (e.g., hundreds or thousands of packets) and transmitted in a constant time interval, to eliminate random measurement noise. In that way, the network should be able to capture a burst of traffic with the same patterns (e.g., consecutive packet interval, path correlation, consistency of start and end nodes, deviation of header and control message setting), when the attacker actively infers the network. As such holistic patterns are inherently existing in external end-to-end inference, they cannot be evaded even if the attacker intentionally disguises its probe packets. In addition, the identification model also adopts online incremental training mechanism that keeps learning the traffic pattern to further refine the decision boundary after the system is online. Particularly, the classifier will be incrementally updated and maintain a dynamic dataset pool keeping packets that can better characterize the probe and normal ones. In that way, the classifier is updated in a timely manner to accommodate varying network traffic over time.

Traditionally, k-NN is widely used in traffic classification, and it shows good performance in identifying specific traffic flow. The characters of k-NN are 1) It calculates the distances between each training sample and the target packet, and then selects the k-nearest samples to the target packet. These k samples jointly determine the class of the target packet. 2) More importantly, the distance of two packets, which is calculated based on multiple features, is the direct means for expressing the similarity of packets. While in probe packet identification, the key challenge is to capture the holistic transmission pattern, which is determined by multiple packets along with multiple features of each packet. Therefore, k-NN is well-suited for detecting probe packets.

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 IV-C and IV-D, 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.,

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

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 close to k packets which include more packets identified as probe packets previously, it will be classified as a probe packet.

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.

C. Training the Weights

In comparison to traditional k-NN classifiers which assign either no weight or static weights to different features [59]–[61], 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 examining 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

$$D_n(P, P') = |f_n(P) - f_n(P')|, \tag{2}$$

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. The details of this procedure are shown in Algorithm 1.

Algorithm 1: Quantifying the over-weighted or less-weighted degree for each feature per iteration.

Input : S_{probe}: the set of probe packets in the training dataset.
$\mathcal{S}_{normal}$: the set of normal packets in the training dataset.
Output: $C_1(n)$: the quantified less-weighted degree for feature n.
$C_2(n)$: the quantified over-weighted degree for feature n.

foreach *packet* $P \in \mathcal{S}_{probe}$ **do**
  /* Calculate the less-weighted and over-weighted degree of each feature for a packet P in $\mathcal{S}_{probe}$. */
  $\mathcal{S}_1 = P$'s m closest packets in $\mathcal{S}_{probe}$;
  $\mathcal{S}_2 = P$'s m closest packets in $\mathcal{S}_{normal}$;
  foreach *feature* $f \in [1, F]$ **do**
    foreach *packet* $P' \in \mathcal{S}_{probe}$ **do**
      $D_n(P, P') = |f_n(P) - f_n(P')|$;
    $d_{max} = \max\{D_n(P, P')\}_{P' \in \mathcal{S}_{probe}}$;
    /* d_{max} is the maximum per-feature distance */
    $d_{mean} = \text{mean}\{D_n(P, P')\}_{P' \in \mathcal{S}_{probe}}$;
    /* d_{mean} is the mean per-feature distance */
    foreach *packet* $P'' \in \mathcal{S}_{normal}$ **do**
      if $D_n(P, P'') > d_{max}$ **then**
        $C_1(P, n) = C_1(P, n) + 1$;
      if $D_n(P, P'') < d_{mean}$ **then**
        $C_2(P, n) = C_2(P, n) + 1$;
foreach *feature* $n \in [1, F]$ **do**
  /* Calculate the over-weighted and less-weighted degree of each feature overall all packets of $\mathcal{S}_{probe}$. */
  foreach *packet* $P \in \mathcal{S}_{probe}$ **do**
    $C_1(n) = C_1(n) + C_1(P, n)$;
    $C_2(n) = C_2(n) + C_2(P, n)$;

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 in each iteration following Algorithm 2 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. The value of q is the step/unit adopted in weight adjustment. A larger value of q will lead to a coarse-grained but faster adjustment in weight training, while a small value of q will result in a fine-grained but slower adjustment. We choose a medium number of 0.01 as the default value to achieve a balance between accuracy and performance in weight training. In ProTO, q can be customized by the network operator based on their deployment settings. Its value can be further tuned according to the training speed and model accuracy observed.

Algorithm 2: Adjusting the weights for different features per iteration.

Input : $C_1(n)$: indicates whether feature n should be more-weighted.
$C_2(n)$: indicates whether feature n should be less-weighted.

foreach *feature* $n \in [1, F]$ **do**
  /* Adjust the weight for each feature. */
  $\Delta w_n = (C_1(n) - C_2(n))q/(m * |\mathcal{S}_{probe}|)$;
  $w_n = w_n + \Delta w_n$;

D. Incrementally Updating Training

Since the initial training dataset collected from self-training before the deployment of the ProTO system is limited, incrementally updating the training dataset by adding the new classified data can improve the light-k-NN based 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 packet will be incremented by 1 if the packet belongs to the k nearest packets closest to P and is in the training dataset of class C as well. These packets are considered useful for classification.

The new packet P will be then added into the training dataset of class C and assigned a vote number with the least vote count in class C plus one. 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. The details of this strategy are shown in Algorithm 3. It can be expected that through this voting system, the decision boundary will be refined and become more precise, 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.

E. Discussions about Most Relevant Features

For features used in training and tuning, we adopt multiple approaches (e.g., Euclidean distance, Hamming distance, exclusive or) to quantify their distances between two packets. All the feature distances will be weighed and added together. As described before, the weight for each feature is tuned through training to indicate their significance such that all features are properly interleaved to represent the holistic pattern. We found in our evaluation that the eight most weighted features are transit path, numbers of packets with the same (header/control/data) bytes, time interval, source address,

Algorithm 3: Incrementally updating the training dataset following the voting-based strategy.

Input : P_{in}: the incoming packet.
S_{probe}: the set of probe packets in the training dataset.
S_{normal}: the set of normal packets in the training dataset.
S_k: the set of the k packets closest to P_{in}.

if P_{in} *is probe packet* **then**
  /* Adjust the set of probe packets. */
  foreach *packet* $P \in S_k$ *and* $P \in S_{probe}$ **do**
    $Vote(P) = Vote(P) + 1$;
  $V_{min} = min\{Vote(P)\}_{P \in S_{probe}}$;
  /* V_{min} is the minimum vote # in S_{probe}. */
  $V_{avg} = avg\{Vote(P)\}_{P \in S_{probe}}$;
  /* V_{avg} is the average vote # in S_{probe}. */
  $S_{probe} = S_{probe} + P_{in}$;
  $Vote(P_{in}) = V_{avg}$;
  if $|S_{probe}| > upper\ bound$ **then**
    $S_{probe} = S_{probe} - P_{V_{min}}$;
    /* $P_{V_{min}}$ is the packet with the minimum vote count in S_{probe}. */
else
  /* Adjust the set of normal packets. */
  foreach *packet* $P \in S_k$ *and* $P \in S_{normal}$ **do**
    $Vote(P) = Vote(P) + 1$;
  $V_{min} = min\{Vote(P)\}_{P \in S_{normal}}$;
  /* V_{min} is the minimum vote # in S_{normal}. */
  $V_{avg} = avg\{Vote(P)\}_{P \in S_{normal}}$;
  /* V_{avg} is the average vote # in S_{normal}. */
  $S_{normal} = S_{normal} + P_{in}$;
  $Vote(P_{in}) = V_{avg}$;
  if $|S_{normal}| > upper\ bound$ **then**
    $S_{normal} = S_{normal} - P_{V_{min}}$;
    /* $P_{V_{min}}$ is the packet with the minimum vote count in S_{normal}. */

destination address, and packet size. We set a time window to help characterize the inter-packet features (i.e., features reflecting correlations among different packets). Time window is empirically set as 1 min in our evaluation. In end-to-end topology inference, each pair of receivers will be measured for hundreds or even thousands of times in a short time period. The probe packets are highly correlated and have the similar features. Inspired by that, when computing the distance of an inter-packet feature between two packets, how frequently the packets with the same feature value appear is also considered. In particular, we count the number of packets of which the feature value is the same in the time window and use the multiplicative inverse of the count as the weight to multiply the feature distance. For probe packets, we should observe a larger count and get a smaller feature distance.

We use two examples to show how we map inter-packet features into numerical representations and how the time window helps identify probe packets. 1) Transit path: We denote the transit path of each packet as a link vector $L = [l_1, l_2, ..., l_L]$, where l_i is the i^{th} internal link of the network; l_i is set as 1 when the packet travels through the link, and 0 otherwise. The distance between two transit paths is measured as the Hamming distance. For example, the Hamming distance between paths $[1, 1, 1, 1, 0]$ and $[1, 1, 1, 0, 0]$ is 1. We further count the number of packets that have the same transit path within the time window. When computing the distance between two paths, the results will be weighed by the multiplicative inverse of the count. 2) Source (destination) address: It is represented as a 4-element vector and the distance is measured as the result of exclusive-or between two addresses. Given two address vectors $[192, 168, 10, 2]$ and $[192, 168, 10, 1]$, their distance vector is thus $[0, 0, 0, 3]$. In this example, the first three elements in distance vector are 0, indicating these two addresses are very close to each other, and may reside in the same subnet. Similarly, we count the packets that have the same source (destination) in the time window and weigh the address distance accordingly.

V. 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 topology inference, then present proactive topology obfuscation.

A. Inference Formulation

We first formulate the external end-to-end topology inference as a fundamental mathematical model. As discussed in Section III-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

$$\mathbf{x} = \mathbf{A}\boldsymbol{\mu}, \tag{3}$$

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 m-th link of the network is shared by the receiver pair corresponding to the k-th element in $\mathbf{x}$, and value 0 otherwise.

We use a simple example to demonstrate how the routing matrix is determined. As shown in Figure 4, 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 4, 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 (3), 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

$$L(\mathbf{x}|\mathcal{T}) \equiv p(\mathbf{x}|\mathbf{A}, \widehat{\boldsymbol{\mu}}), \tag{4}$$

where $\widehat{\boldsymbol{\mu}}$ is the maximum likelihood estimate of $\boldsymbol{\mu}$ given $\mathcal{T}$. The MLE obtains the topology $\widehat{\mathcal{T}}$ that maximizes the likelihood by

$$\widehat{\mathcal{T}} \equiv \arg\max_{\mathcal{T} \in \mathcal{F}} L(\mathbf{x}|\mathcal{T}). \tag{5}$$

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

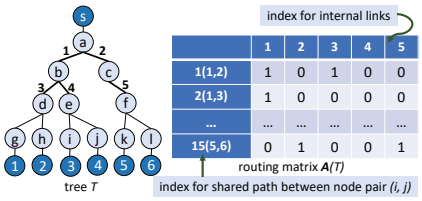

Fig. 4: An example: routing matrix $\mathbf{A}$ of tree $\mathcal{T}$.

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 V-C. In particular, we randomly generate a fake topology denoted by $\mathbf{A}_m$ independent of the real topology $\mathbf{A}$ in (3). 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 (3) for topology inference, we multiply by a manipulation matrix $\mathbf{F}$ both sides in (3) and obtain

$$\mathbf{F}\mathbf{x} = \mathbf{F}\mathbf{A}\boldsymbol{\mu}, \tag{6}$$

where the left-hand side $\mathbf{F}\mathbf{x}$ 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 $\mathbf{F}\mathbf{x}$ using MLE. The following linear equation must hold

$$\mathbf{F}\mathbf{x} = \mathbf{F}\mathbf{A}\boldsymbol{\mu} = \mathbf{A}_m\boldsymbol{\mu}. \tag{7}$$

Therefore, our goal of topology obfuscation is to find the manipulation matrix $\mathbf{F}$ such that

$$\mathbf{F}\mathbf{A} = \mathbf{A}_m, \tag{8}$$

given the real topology $\mathbf{A}$ and the fake one $\mathbf{A}_m$.

Note that (8) 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 [62]. Thus, obfuscation based on (8) 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 we denote it as $\mathbf{B}$ from the attacker's perspective. Its entropy is defined as $H(\mathbf{B}) = -\sum_{\mathbf{b}\in\mathbf{B}} P_{\mathbf{B}}(\mathbf{b}) \log P_{\mathbf{B}}(\mathbf{b})$, where $P_{\mathbf{B}}(\mathbf{b})$ is the probability when $\mathbf{B} = \mathbf{b}$, $\mathbf{b}$ is a specific topology matrix. Similarly, the entropy of $\mathbf{B}$ conditioned on the attacker knowing $\mathbf{A}_m$ is defined as $H(\mathbf{B}|\mathbf{A}_m) = \sum_{\mathbf{a}_m\in\mathbf{A}_m} P_{\mathbf{A}_m}(\mathbf{a}_m) H(\mathbf{B}|\mathbf{A}_m = \mathbf{a}_m)$, 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 $\mathbf{B}$, $H(\mathbf{B}|\mathbf{A}_m = \mathbf{a}_m) = H(\mathbf{B})$ and thus $H(\mathbf{B}|\mathbf{A}_m) = H(\mathbf{B})$, which indicates the knowledge of $\mathbf{A}_m$ gives the attacker no information of real routing matrix $\mathbf{B}$.

D. Optimization based Topology Obfuscation

After $\mathbf{A}_m$ is generated, the topology control module computes the manipulation matrix $\mathbf{F}$ based on (8). 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 (8). As a result, we need to find the best solution to (8).

First, not all solutions can be practical in real-world systems. As $\mathbf{F}\mathbf{x}$ represents the path measurement delays observed by the attacker, $\mathbf{F}\mathbf{x}$ should have comparable values to the original measurement $\mathbf{x}$. Thus, we should impose a constraint in searching for $\mathbf{F}$ such that

$$\text{any element in } \mathbf{F}\mathbf{x} - \mathbf{x} \text{ is within } [0, \delta_{max}], \tag{9}$$

where δ_{max} is called the maximum allowed deviation for the delay. Note that d 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 $\mathbf{F}\mathbf{A} = \mathbf{A}_m$ given (9) into the following optimization problem.

$$\begin{aligned} \underset{\mathbf{F}}{\text{minimize}} \quad & \|\mathbf{F}\mathbf{A} - \mathbf{A}_m\|_2, \\ \text{subject to} \quad & \text{any element in } \mathbf{F}\mathbf{x} - \mathbf{x} \text{ is within } [0, \delta_{max}]. \end{aligned} \tag{10}$$

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 (10) 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 (10), 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 (10), 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

$$\|\mathbf{FA} - \mathbf{A}_m\|_2 = \|\mathbf{Mf} - \mathbf{m}\|_2\,, \tag{11}$$

where $\mathbf{M}$ is an $m \times n$ by $m \times m$ matrix satisfying $\mathbf{M} = [\mathbf{C},\ ...,\ \mathbf{C}]^T$; $\mathbf{C} = \text{diag}(\mathbf{a}_1,\ \mathbf{a}_2,\ ...,\ \mathbf{a}_m)$; 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 (10) as

$$\begin{aligned} \underset{\mathbf{f}}{\text{minimize}} \quad & \mathbf{f^T M^T M f - 2 m M f} \\ \text{subject to} \quad & \mathbf{0 \le Hf \le x}_{max}, \end{aligned} \tag{12}$$

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 (10) is equivalent to solving (12). In our case of network obfuscation, $\mathbf{M^T M}$ in (12) is a semi-definite matrix.

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. In our implementation, ProTO intercepts identified probe packets at the exit nodes of the network, holds them in a special waiting queue after the intended delay, and then sends them out. 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.

VI. 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 [63] 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 [64] in the evaluation, including a small, a medium and a large network with information listed in Table II and topologies shown in Figure 5. We create these three networks on two high-performance computing workstations (each has dual Intel Xeon Gold 5122 3.70 GHz CPUs and 192 GB Memory) connected through 10 Gigabit Ethernet. Each network node is created as an independent virtual machine that runs OpenWrt [65] 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.

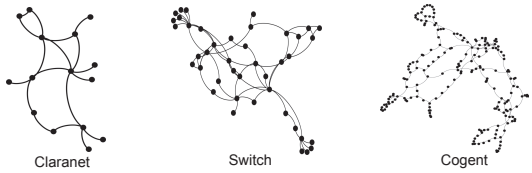

Fig. 5: Structures of three collected topologies.

As the most recognized external end-to-end topology inference techniques, we adopt the methods proposed in [13]–[18] to carry out the evaluation. Specifically, we follow the tree structure to perform the external end-to-end topology inference. For each topology, we first identify all the end nodes (i.e., nodes connected to the network with only one link) and the end circles (i.e., circles connected to the network with only one link). In the evaluation, each end circle will be treated as an end node. In inference, we use one of the end nodes as the source node and others as the receiver nodes. Internal circles are ignored in our evaluation, we simply consider these circles as branch paths from the source node to the receiver nodes. Meanwhile, 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

TABLE II: Statistics of three collected network topologies.

	Claranet	Switch	Cogent
Nodes	15	42	197
Links	18	63	243

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) the detection rate, which is defined as the number of probe packets correctly identified divided by the total number of probe packets, i.e., detection rate $= \frac{\text{\# of correctly identified probe packets}}{\text{total \# of probe packets}}$, and 2) the false alarm rate, defined as the number of normal packets misidentified as probe packets divided by the total number of all normal packets, i.e., false alarm rate $= \frac{\text{\# of misidentified normal packets}}{\text{total \# of normal packets}}$.

The detection rate measures the percentage of correctly identified probe packets among all probe packets. According to our detect-then-obfuscate (i.e., proactively delay) approach, the detection rate should be high to ensure the confidentiality of network topology. While the false alarm rate measures the amount of normal packets that are wrongly identified as probe packet. These wrongly identified packets will also be delayed by ProTO. The delay should not affect the network functionalities but allow for slight performance degradation. Hence, ProTO should reduce the false alarm rate as low as possible in probe packet detection.

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 III 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 IV-D, 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 III.

Impact of initial training dataset: We can observe in Table III that the size of initial training has a substantial impact

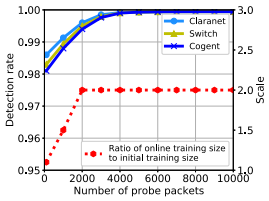

Fig. 6: Detection rate for probe packet identification.

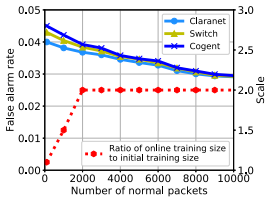

Fig. 7: False alarm rate for probe packet identification.

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 III that the increase of the updating scale will lead to performance improvement of the classifier. For example, in Claranet, when 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 detection 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 III, 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 6 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 6 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 7 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 6 and 7 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-

TABLE III: 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.

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.

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. In evaluation, the logical topologies of the three real-world networks are calculated and further comparison results are all obtained based on the logical topologies. A popular metric to measure the difference between two trees is the Tree Edit Distance (TED) defined in [66]. 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 $\mathcal{T}_2$. For instance, there are R *replacements*, I *insertions*, and D *deletions* for the mapping from tree $\mathcal{T}_1$ to tree $\mathcal{T}_2$, then the cost for the operations in this mapping can be calculated by TED=$c_r R$+$c_i I$+$c_d D$, where c_r, c_i, and c_d are the costs of a replacement, an insertion, and a deletion, respectively. In our evaluation, we adopt the unit cost for each operation, i.e., $c_r = c_i = c_d = 1$.

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 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

$$\text{similarity score} = 1 - \frac{\text{TED}_0}{\text{TED}_1 + \text{TED}_2}.$$

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 that the benchmark score is 0.6. Hence, we consider ProTO to be effective if 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 8 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 8 that if no defense deployed, the attacker can easily obtain the real topology with high accuracy. For example, if the attacker sends 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 9 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 V-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 10. We can find in Figure 10 that overall, the optimization based topology obfuscation in (10) 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. In particular, from the network perspective, delaying probe not normal packets has no substantial impact on the network performance, as they are of malicious intent and do not carry any data. However, when a normal packet has been falsely identified as a probe one, delaying it may potentially degrade the data delivery efficiency. 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. Specifically, for a misidentified packet,

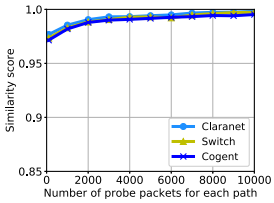

Fig. 8: Similarity score between the inferred topology and the real topology without protection.

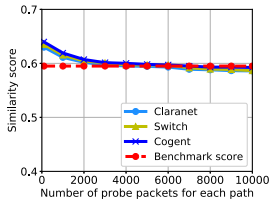

Fig. 9: Similarity score between the inferred topology and the real topology under protection.

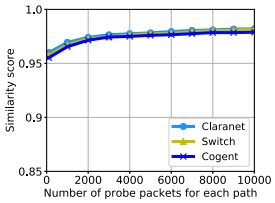

Fig. 10: Similarity score between the inferred topology and the intended topology.

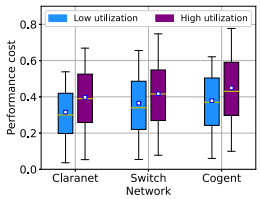

Fig. 11: The performance cost of normal packets misidentified by ProTO.

its original delay is measured as the traversal time of the packet going through the network without the intentional delay introduced by ProTO. In our experiment, we measure the original delay before enforcing additional delay at exit nodes. Therefore the original delay can be measured as the duration between the time instant that the packet enters the entry node and the time instant that the packet is processed at the exit node before intentionally delayed. 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 overall performance cost of all normal packets (that are either correctly identified or misidentified) going through the network.

Figure 11 shows the box plot of performance cost in case (i) in which only misidentified normal packets are measured. The box plot shows the distribution of performance cost of misidentified packets, including the minimum, the maximum, the median, the average, the first quartile and the third quartile. From Figure 11, we observe that ProTO increases the delay of a misidentified normal packet by 31% - 37% on average under the low utilization condition, and by 39% - 45% on average under the high utilization condition. Meanwhile, we note that there is no substantial change of packet drop rate, because ProTO just delays without actively dropping packets to obfuscate the measurement results. As only a limited number of normal packets can be misidentified by ProTO (e.g., 3% false alarm rate shown in Figure 7), the overall performance disruption is expected to be small for legitimate traffic. Table IV shows the performance cost in case (ii) in which all normal packets going through the network are measured to compute the overall performance cost. Specifically, the overall performance cost is defined as $\sum_{i\in\mathbf{C}_{mis}} d_i / \sum_{j\in\mathbf{C}} t_j$, where $\sum_{i\in\mathbf{C}_{mis}} d_i$ denotes the summation of additional delays of all misidentified normal packets ($\mathbf{C}_{mis}$ is the set of misidentified packets); and $\sum_{j\in\mathbf{C}} t_j$ is the summation of the original delays of all normal packets including non-misidentified and misidentified packets ($\mathbf{C}$ is the set of all normal packets and $\mathbf{C}_{mis} \subset \mathbf{C}$). The metric demonstrates the impact of additional delay introduced by misidentified packets over all normal packets. The results are shown in Table IV. Since only 3% normal packets have been misidentified, the overall performance cost due to the deployment of ProTO is around 1.3% and 2% for the low and high utilization conditions, respectively. We note that a number of exiting studies [67]–[71] produced related network security designs with overheads ranging from 5% to 12%. Comparing with them, our proposed technique can be considered efficient.

TABLE IV: Overall 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%

VII. 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.

We implement the ProTO system and evaluate its performance with various conditions. Experimental results show that ProTO can (i) achieve a detection rate of 99.9% with a false alarm of 3%, (ii) effectively disrupt adversarial topology inference by reducing the similarity score between the real and inferred topologies from nearly 1 to around the benchmark score of 0.6 (that represents the average similarity score between the real network topology and a randomly generated topology), and (iii) result in an overall network delay degradation of 1.3% - 2.0%.

Acknowledgement: The work at USF was supported in part by NSF-CNS 1717969.

References

[1] T. Hou, Z. Qu, T. Wang, Z. Lu, and Y. Liu, "ProTO: Proactive topology obfuscation against adversarial network topology inference," in *Proc. of IEEE INFOCOM*, 2020.

[2] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," *ACM SIGOPS OSR*, 2002.

[3] N. F. Butt, M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *International Conference on Research in Networking*, 2010.

[4] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, 2005.

[5] M. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communications Magazine*, 2009.

[6] N. Duffield, "Simple network performance tomography," in *Proc. of ACM IMC*, 2003.

[7] P. Barford, N. Duffield, A. Ron, and J. Sommers, "Network performance anomaly detection and localization," in *Proc. of IEEE INFOCOM*, 2009.

[8] L. Ma, T. He, A. Swami, K. K. Leung, and J. Lowe, "Node failure localization via network tomography," in *Proc. of ACM IMC*, 2014.

[9] R. Kompella, J. Yates, A. Greenberg, and A. Snoeren, "Detection and localization of network black holes," in *Proc. of IEEE INFOCOM*, 2007.

[10] S. Panjwani, S. Tan, K. M. Jarrin, and M. Cukier, "An experimental evaluation to determine if port scans are precursors to an attack," in *Proc. of IEEE DSN*, 2005.

[11] S. Zhao, Z. Lu, and C. Wang, "When seeing isn't believing: On feasibility and detectability of scapegoating in network tomography," in *Proc. of IEEE ICDCS*, 2017.

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

[13] N. G. Duffield, J. Horowitz, F. L. Presti, and D. Towsley, "Multicast topology inference from measured end-to-end loss," *IEEE Transactions on Information Theory*, 2002.

[14] N. G. Duffield and F. L. Presti, "Network tomography from measured end-to-end delay covariance," *IEEE/ACM Transactions on Networking (TON)*, 2004.

[15] H. Yao, S. Jaggi, and M. Chen, "Network coding tomography for network failures," in *Proc. of IEEE INFOCOM*, 2010.

[16] P. Sattari, C. Fragouli, and A. Markopoulou, "Active topology inference using network coding," *Physical Communication*, 2013.

[17] B. Eriksson, G. Dasarathy, P. Barford, and R. Nowak, "Efficient network tomography for internet topology discovery," *IEEE/ACM Transactions on Networking (TON)*, 2012.

[18] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, "Maximum likelihood network topology identification from edge-based unicast measurements," in *Proc. of ACM SIGMETRICS*, 2002.

[19] F. Ubaldi, P. Teresa, and M. Puleri, "Method and device for network tomography," Nov. 9 2017, uS Patent App. 15/529,819.

[20] J.-P. Vasseur, "Operations administration management for path computation element chains," Aug. 28 2012, uS Patent 8,254,272.

[21] R. Black, A. Donnelly, and C. Fournet, "System and method for network topology discovery," May 13 2014, uS Patent 8,724,512.

[22] Y. Yuan, Z. Ye, and X. Fan, "Method and device for discovering network topology," May 25 2017, uS Patent App. 15/426,891.

[23] M. H. Gunes and K. Sarac, "Analyzing router responsiveness to active measurement probes," in *International Conference on Passive and Active Network Measurement*, 2009.

[24] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "Nethide: Secure and practical network topology obfuscation," in *USENIX Security*, 2018.

[25] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *ACM SIGCOMM CCR*, 2002.

[26] Y. Hyun, "Archipelago measurement infrastructure," in *Proc. of the CAIDA-WIDE*, 2006.

[27] "RIPE Atlas, year=2018," https://atlas.ripe.net.

[28] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network tomography: Recent developments," *Statistical science*, 2004.

[29] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *IEEE UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*, 2016.

[30] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, 2004.

[31] "Domain Name System," https://en.wikipedia.org/wiki/Domain_Name_System, 2018.

[32] A. Barili and D. Lanterna, "On the effects of large-scale dns poisoning," in *Proc. of IEEE CNS*, 2015.

[33] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys & Tutorials*, 2016.

[34] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for dns." in *USENIX Security*, 2010.

[35] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *Proc. of IEEE S & P*, 2009.

[36] C. L. Abad and R. I. Bonilla, "An analysis on the schemes for detecting and preventing arp cache poisoning attacks," in *Proc. of ICDCSW*, 2007.

[37] D. Dagon, M. Antonakakis, K. Day, X. Luo, C. P. Lee, and W. Lee, "Recursive dns architectures and vulnerability implications." in *Proc. of NDSS*, 2009.

[38] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your state is not mine: a closer look at evading stateful internet censorship," in *Proc. of ACM IMC*, 2017.

[39] X. Xu, Z. M. Mao, and J. A. Halderman, "Internet censorship in china: Where does the filtering occur?" in *Proc. of Springer PAM*, 2011.

[40] A. Dainotti, C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, and A. Pescapé, "Analysis of country-wide internet outages caused by censorship," in *Proc. of ACM IMC*, 2011.

[41] H. Acharya, S. Chakravarty, and D. Gosain, "Few throats to choke: On the current structure of the internet," in *Proc. of IEEE LCN*, 2017.

[42] S. T. Trassare, R. Beverly, and D. Alderson, "A technique for network topology deception," in *Proc. of IEEE MILCOM*, 2013.

[43] S. Achleitner, T. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Cyber deception: Virtual networks to defend insider reconnaissance," in *In Proc. of ACM CCS MIST*, 2016.

[44] S. T. Trassare, "A technique for presenting a deceptive dynamic network topology," Naval Postgraduate School, Tech. Rep., 2013.

[45] X. Zhang and C. Phillips, "A survey on selective routing topology inference through active probing," *IEEE Communications Surveys & Tutorials*, 2012.

[46] O. Gurewitz and M. Sidi, "Estimating one-way delays from cyclic-path delay measurements," in *Proc. of IEEE INFOCOM*, 2001.

[47] A. Rai and E. Modiano, "Topology discovery using path interference," in *Proc. of IFIP Networking*, 2019.

[48] M. Ettehad, N. Duffield, and G. Berkolaiko, "Optimizing consistent merging and pruning of subgraphs in network tomography," *arXiv preprint arXiv:1908.03519*, 2019.

[49] E. Kruglick, "Preventing network tomography in software defined datacenter networks," May 31 2016, uS Patent 9,356,956.

[50] C.-C. Chiu and T. He, "Stealthy dgos attack: Degrading of service under the watch of network tomography."

[51] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed ip traffic using hop-count filtering," *IEEE/ACM Transactions on networking (ToN)*, 2007.

[52] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *ACM SIGMETRICS PER*, 2005.

[53] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," Tech. Rep., 2013.

[54] K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K. Hamlen, and B. Thuraisingham, "Adaptive encrypted traffic fingerprinting with bi-directional dependence," in *Proc. of ACM ACSAC*, 2016.

[55] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proc. of IEEE EuroS&P*, 2016.

[56] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," in *Proc. of ACM SIGCOMM CCR*, 2007.

[57] X. He, "Incremental semi-supervised subspace learning for image retrieval," in *Proc. of ACM Multimedia*, 2004.

[58] A. Haque, L. Khan, and M. Baron, "Sand: Semi-supervised adaptive novel class detection and classification over data stream." in *Proc. of AAAI*, 2016.

[59] E.-H. S. Han, G. Karypis, and V. Kumar, "Text categorization using weight adjusted k-nearest neighbor classification," in *Springer PAKDD*, 2001.

[60] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Computers & security*, 2002.

[61] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, 2011.

[62] G. Casella and R. L. Berger, *Statistical inference*. Duxbury Pacific Grove, CA, 2002, vol. 2.

[63] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, 2014.

[64] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, 2011.

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

[66] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM Journal on Computing*, 1989.

[67] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in SDN-based networks," in *Proc. of IEEE NetSoft*, 2018.

[68] L. Chaddad, A. Chehab, I. H. Elhajj, and A. Kayssi, "Network obfuscation for net worth security," in *Proc. of IEEE SDS*, 2020.

[69] X. Fang, N. Zhang, S. Zhang, D. Chen, X. Sha, and X. Shen, "On physical layer security: Weighted fractional fourier transform based user cooperation," *IEEE Transactions on Wireless Communications*, 2017.

[70] L. Li, C. Chigan, and S. Yuan, "Security-oriented DSA for network access control in cognitive radio networks," in *Proc. of IEEE HST*, 2018.

[71] M. Conti, M. Hassan, and C. Lal, "BlockAuth: BlockChain based distributed producer authentication in ICN," *Computer Networks*, 2019.