2012 IEEE Symposium on Security and Privacy

Secure and Scalable Fault Localization under Dynamic Traffic Patterns

Xin Zhang
CyLab / Carnegie Mellon University

Chang Lan
Tsinghua University

Adrian Perrig
CyLab, Carnegie Mellon University

***Abstract*—Compromised and misconfigured routers are a well-known problem in ISP and enterprise networks. Data-plane fault localization (FL) aims to identify faulty links of compromised and misconfigured routers during packet forwarding, and is recognized as an effective means of achieving high network availability. Existing *secure* FL protocols are *path-based*, which assume that the source node *knows the entire outgoing path* that delivers the source node's packets and that the path is *static* and *long-lived*. However, these assumptions are incompatible with the dynamic traffic patterns and agile load balancing commonly seen in modern networks. To cope with real-world routing dynamics, we propose the first secure *neighborhood-based* FL protocol, DynaFL, with *no* requirements on path durability or the source node knowing the outgoing paths. Through a core technique we named delayed key disclosure, DynaFL incurs little communication overhead and a small, constant router state independent of the network size or the number of flows traversing a router. In addition, each DynaFL router maintains only a single secret key, which based on our measurement results represents 2–4 orders of magnitude reduction over previous path-based FL protocols.**

I. Introduction

Modern ISP, enterprise, and datacenter networks demand reliable data delivery to support performance-critical services, thus requiring the data plane to correctly forward packets along the routing paths. However, *real-world* incidents [2], [3], [7], [21], [25], [32] reveal the existence of compromised routers in ISP and enterprise networks that sabotage network data delivery. Also, in a 2010 worldwide security survey [1], 61% of network operators ranked infrastructure outages due to misconfigured network equipment such as routers as the No. 2 security threat. Such misbehaving routers can easily drop, modify, delay or inject packets into data plane to mount Denial-of-Service, surveillance, man-in-the-middle attacks, and data exfiltration where a malicious router may replicate some packets and send them along other unexpected paths.

Unfortunately, current networks lack a *reliable* and *secure* way to identify misbehaving routers that jeopardize packet delivery. For example, a malicious or misconfigured router can "correctly" respond to `ping` or `traceroute` probes while corrupting other data packets, thus cloaking the attacks from `ping` or `traceroute`. Data-plane fault localization (FL) aims to localize *faulty links* that sabotage packet delivery in the data plane, thus providing *data-plane accountability*. By removing the identified faulty links from the routing tables or bypassing the faulty links in route selection, FL enables network communication to be carried only on non-faulty links, thus yielding high packet delivery guarantees [9], [14], [34].

Existing FL protocols that are secure against sophisticated packet modification and fabrication attacks [11], [14], [34] require that the sender *know the entire path* that delivers the source node's packets, and that the path be *long-lived* (e.g., stable over transmitting 10^8 packets [14]) to obtain a statistically accurate FL. However, recent measurement studies [10], [18], [20] show that a considerable fraction of current network flows are short-lived "mice" and routing paths are highly dynamic. Furthermore, emerging enterprise and datacenter networks call for more agile load balancing and dynamic routing paths. For example, a recently proposed datacenter routing architecture, VL2 [20], employs Valiant Load Balancing [24], [37] to spread traffic uniformly across network paths via random packet deflection. In this case, the actual routing path is determined *on the fly during forwarding* and thus cannot be predicted and known by the sender. Given the conflict between the "static-path" assumption and the "dynamic-path" reality, researchers have concluded that *existing* FL protocols are impractical for widespread deployment in large-scale networks with dynamic traffic patterns [14].

In addition, in existing secure FL protocols, a router must share some secret (e.g., cryptographic keys) with each source node sending traffic traversing that router, making the key storage overhead at an intermediate router linear in the number of end nodes. The proliferation of key copies shared by routers with all end nodes under non-uniform (and generally poor) administration also increases the risk of key compromise thereby enabling undetected attacks. In existing secure FL protocols, a router also needs to maintain per-path state for each path traversing that router, making the FL unscalable for large-scale networks.

We aim to bridge the current gap between the security of FL against strong adversaries and the ability to support dynamic traffic patterns in modern networks such as ISP, enterprise, and datacenter networks. More specifically, the desired FL protocol should be secure against sophisticated packet dropping, modification, fabrication, and delaying attacks by colluding routers, while retaining the following properties:

- **Path obliviousness:** A source node or a router does not

DOI 10.1109/SP.2012.27

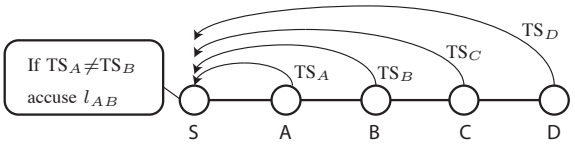


Figure 1. Path-based FL. TS$_r$ denotes the traffic summary generated by router r. For brevity, "TS$_A$≠TS$_B$" refers to "TS$_A$ deviates from TS$_B$ more than a certain threshold".

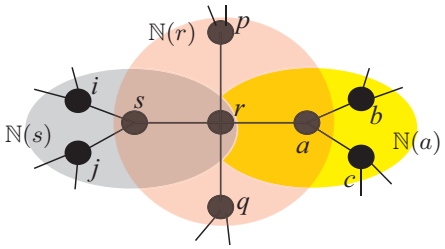


Figure 2. A neighborhood example.

need to know the outgoing/downstream path.

- **Volatile path support:** The FL protocol requires no maximum duration for a forwarding path.
- **Constant router state:** A router does not need to maintain per-path, per-flow, or per-source state.
- **O(1) key storage:** A router only manages a small number of keys regardless of the network size.

Path obliviousness and volatile path support together enable agile (e.g., *packet*-level) load balancing and dynamic routing paths (e.g., Valiant load-balanced paths). These two properties also *decouple* the data-plane FL from routing, thus enabling it to support a wide array of routing protocols. Finally, constant router state provides scalability in large-scale networks and O(1) key storage reduces the key setup overhead.

We observe that the "static-path" assumption in existing secure FL protocols stems from the fact that those FL protocols operate on entire end-to-end paths (*path-based*), to localize the fault to *one specific link*. As Figure 1 shows, each router maintains a certain "traffic summary" (e.g., a counter, packet hashes, etc.) for *each* path that traverses the router (thus requring per-path state), and sends the traffic summary to the source node S of each path. S can then detect a link l as malicious if the traffic summaries from l's two adjacent nodes deviate greatly, as Figure 1 illustrates. Hence, S needs to know the entire path topology to compare traffic summaries of adjacent nodes, and needs to send a large number of packets over the same path so that the deviation in traffic summaries can reflect a statistically accurate estimation of link quality. Finally, to authenticate the communication between the source and each router in the path, a router needs to share a secret key with each source that sends traffic through it.

In this paper, we explore *neighborhood-based* FL approaches, where a router r's data-plane faults (if any) can be detected by checking the consistency (or conservation) of the traffic summaries generated by the *1-hop neighbors* of r (denoted by $\mathbb{N}(r)$ in Figure 2). That is, in benign cases, the packets *sent to* r will be consistent with the packets *received from* r by all of r's neighbors as reflected in their traffic summaries. In this way, the FL is independent of routing paths and only depends on 1-hop neighborhoods, thus supporting arbitrary routing protocols and dynamic load balancing. Additionally, each router in a neigbhorhood-based approach only needs to maintain state for each neighbor. In summary, neighborhood-based FL localizes faults *to a specific 1-hop neighborhood* to reduce further investigation, to *trade localization precision for practicality* in modern networks with dynamic traffic patterns.

Though promising, neighborhood-based FL is susceptible to sophisticated packet modification and collusion attacks due to several security and scalability challenges. For example, for the sake of scalability, the traffic summary cannot be a copy of all the original packets (or even their hashes), but have to be a compact representation of the original packets via a certain **fingerprinting function** $\mathcal{F}$. On one hand, if $\mathcal{F}$ generates traffic summaries at different nodes *without* using different secret keys, a malicious router can predict the outputs of $\mathcal{F}$ at other nodes and tactically modify packets such that the outputs of $\mathcal{F}$ will stay the same as with the original packets. On the other hand, if $\mathcal{F}$ at different nodes uses *different* secret keys, we cannot compare and run consistency check over different nodes' traffic summaries. To address these challenges, we propose DynaFL, a protocol that employs a core technique called *delayed key disclosure*, which discloses the *same* key for computing $\mathcal{F}$ to different routers *after* they have forwarded the packets. To further minimize the protocol overhead, DynaFL employs a secure sampling mechanism also based on the delayed key disclosure, so that a malicious router cannot know if a packet is sampled or not at the time it forwards (corrupts) the packet. Finally, a router in DynaFL only shares a secret key with a centralized controller, thus achieving O(1) key storage.

Contributions. Our contributions are three-fold:

1. We raise the importance of pursuing a secure FL design to cope with *dynamic traffic patterns* in real-world networks with a small constant router state and key storage.

2. To the best of our knowledge, DynaFL is the first secure *neighborhood*-based FL protocol that achieves path obliviousness and volatile path support, *and* is secure against both packet loss and sophisticated packet modification/injection attacks.

3. In addition, a DynaFL router requires only about 4MB per-neighbor state based on our AMS sketch [6] implementation, whereas path-based FL protocols require per-path state.

We also show through measurements that the number of keys a router needs to manage in path-based FL protocols is 2 - 3 orders of magnitude higher than that in DynaFL (which is a single key shared with a centralized controller). Finally, our simulation results demonstrate DynaFL's small detection delay and negligible communication overhead.

II. Problem Statement

In this section, we formalize the notation, network setting, adversary model, and problem statement.

A. Notation

We use the terms *node* and *router* interchangeably to generally refer to devices that either perform layer-2 switching or layer-3 routing (so nodes do not include end servers). We denote the 1-hop neighborhood (or neighborhood, for brevity) of a node s as $\mathbb{N}(s)$, as Figure 2 illustrates. For a particular packet traversing a neighborhood $\mathbb{N}(s)$, the neighbor sending that packet to node s is called an **ingress node** in $\mathbb{N}(s)$ for that packet, and the node receiving that packet *from* s is called an **egress node**. We term a sequence of packets as a *packet stream* $\mathbb{S}$. Particularly, we denote the packet stream sent from node i to node j as $\mathbb{S}_{ij}$, and this packet stream is *seen* by nodes i and j as $\mathbb{S}_i^{\rightarrow j}$ and $\mathbb{S}_j^{\leftarrow i}$, respectively. The **difference** of two packet streams $\mathbb{S}$ and $\mathbb{S}^{'}$, denoted by $\Delta(\mathbb{S}^{'}, \mathbb{S}^{'})$, refers to the number of packets in one packet stream but not in the other, without considering the variable IP header fields such as the TTL and checksum fields.

B. Network Setting

We consider a network with dynamic traffic patterns and a relatively static network topology, which is best exemplified by today's ISP, enterprise, and datacenter networks. To provide maximum flexibility to support various routing protocols, and even packet-level load balancing, we pose *no* restriction on the routing protocols and load balancing mechanisms used in the network. We assume a *trusted* **administrative controller** (AC) in the network, which shares a pairwise secret key with each router in the network. As we will show later, the AC is mainly in charge of analyzing the traffic summaries gathered from different nodes and localizing any neighborhood with data-plane faults. Finally, we require nodes in the network be *loosely* time-synchronized, e.g., on the order of milliseconds. Loose time synchronization represents a common requirement for detecting packet delaying attacks [8], [9], [29] and nowadays even high-precision clock synchronization is available given the advent of GPS-enabled clocks and the adoption of IEEE 1588 [23].

C. Adversary Model

We consider a sophisticated adversary controlling multiple malicious nodes. Specifically, a **malicious node** corrupts data-plane packets by unexpectedly *dropping*, *modifying* and *delaying* legitimate packets sent by the source, and *fabricating* bogus packets that are not sent by the source. A malicious node can corrupt both the data packets and *control packets*, such as traffic summaries sent from a node to the AC and certain administrative messages sent from the AC to nodes. Furthermore, a sophisticated adversary has knowledge of and tries to disrupt the FL protocol to evade detection. Multiple **colluding nodes** can collectively perform the above data-plane attacks, conspiring to evade detection or frame benign nodes. The colluding nodes know each other's security credentials (e.g., secret keys used in the FL protocol).

Such a strong attacker model is not merely a theoretical conception, but has been widely witnessed in practice. For example, outsider attackers have leveraged social engineering, phishing [3], and exploration of router software vulnerabilities [2], [7] or weak passwords [21] to compromise ISP and enterprise routers [32]. In addition, a majority of network operators in a recent worldwide security survey [1] listed router misconfiguration, which also falls under our adversary model, as a primary cause of network outages [25]. As we will show in Section III-C, achieving FL security against surreptitious packet modification/fabrication attacks is challenging and dramatically complicates the protocol design.

D. Problem Formulation

Our goal is to design a practical and secure *neighborhood-based* FL protocol to identify a suspicious *neighborhood* (if any) that contains at least one malicious node. Recall that practicality translates to *path obliviousness*, *volatile path support* and *constant router state* as stated in Section I. We further adopt the (α, β, δ)**-accuracy** [19] to formalize the security requirements as below:

- If more than β fraction of the packets are corrupted by a malicious node m, the FL protocol will raise a neighborhood containing m *or* one of its colluding nodes as suspicious with probability at least $1 - \delta$.
- In benign cases, if no more than α fraction of the packets are spontaneously corrupted (e.g., dropped) in a neighborhood, the FL protocol will raise the neighborhood as suspicious with probability at most δ.

The thresholds α and β are introduced to tolerate spontaneous failures (e.g., natural packet loss) and are set by the network administrator based on her experience and expectation of network performance.

Neighborhood-based FL enables the network administrator to scope further investigation to a 1-hop neighborhood to find out which router is compromised. It is also possible to further employ dedicated monitoring protocols, which only need to monitor a small region (the identified neighborhood) of the network to find the specific misbehaving router.

III. Challenges and Overview

In this section, we first describe the high-level steps of general neighborhood-based FL and then explain the security challenges in the presence of strong adversaries. Finally, we present the key ideas in DynaFL that address these challenges.

A. High-Level Steps

The general steps a neighborhood-based FL takes are (i) *recording* local traffic summaries, (ii) *reporting* the traffic summaries to the AC, and (iii) *detecting* suspicious neighborhoods by the AC based on the received traffic summaries, as we sketch below. Though intuitive, these general steps face several potential security vulnerabilities and scalability challenges as Section III-C will show.

Recording. We divide the time in a network into consecutive **epochs**, which are synchronous among all the nodes including the AC in the network. For each neighbor r, a node s locally generates traffic summaries, denoted by $\text{TS}_s^{\rightarrow r}$ and $\text{TS}_s^{\leftarrow r}$, for the packet streams $\mathbb{S}_{sr}$ and $\mathbb{S}_{rs}$ in each epoch, respectively. Figure 3 depicts the router state in a toy example.

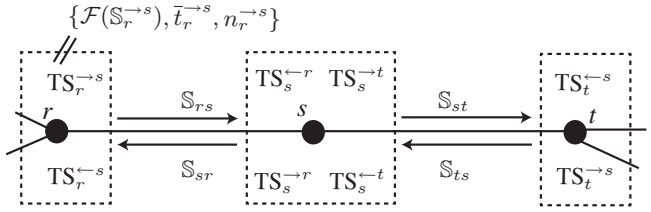


Figure 3. Router state for traffic summaries.

The traffic summary recorded by a node s should reflect both the packet contents and the arrival/departure time seen at node s to enable the detection of malicious packet corruption and delay. For the sake of scalability, the traffic summary can not simply be an entire copy of all the original packets (or their hashes using a *cryptographic* hash function such as SHA-1 which provides one-wayness and collusion resistance) and their timing information. Instead, we use a *fingerprinting function* $\mathcal{F}$ to reflect the *aggregates* of packet contents to reduce both router state and bandwidth consumption for reporting the traffic summaries to the AC. We denote the fingerprint for a packet stream $\mathbb{S}_{rs}$ generated by r as $\mathcal{F}(\mathbb{S}_r^{\rightarrow s})$, as Figure 3 depicts. In addition, as Figure 3 shows, for a packet stream $\mathbb{S}_{rs}$ (or $\mathbb{S}_{sr}$), the traffic summary of node r also contains the *average* departure time $\bar{t}_r^{\rightarrow s}$ (or arrival time $\bar{t}_r^{\leftarrow s}$) and the total number of packets $n_r^{\rightarrow s}$ (or $n_r^{\leftarrow s}$) in $\mathbb{S}_{rs}$ (or $\mathbb{S}_{sr}$) seen in the current epoch to enable the detection of packet delay attacks.

Reporting. At the end of each epoch, each node s sends its local traffic summaries to the AC.

Detection. After receiving the traffic summaries at the end of an epoch, the AC runs a consistency check over the traffic summaries in each neighborhood. A large inconsistency of the traffic summaries in a certain neighborhood $\mathbb{N}(s)$ indicates that $\mathbb{N}(s)$ is suspicious.

B. The Fingerprinting Function $\mathcal{F}$

Before we present the instantiation of $\mathcal{F}$, we first describe the general properties that $\mathcal{F}$ should satisfy. To enable the AC to detect suspicious neighborhoods, $\mathcal{F}$ should generate traffic summaries with the following two properties:

Property 1: Given any two packet streams $\mathbb{S}$ and $\mathbb{S}'$, the "difference" between $\mathcal{F}(\mathbb{S})$ and $\mathcal{F}(\mathbb{S}')$ can give an estimation of the difference between $\mathbb{S}$ and $\mathbb{S}'$, denoted by: $\Delta(\mathcal{F}(\mathbb{S}), \mathcal{F}(\mathbb{S})') \rightsquigarrow \Delta(\mathbb{S}, \mathbb{S}')$.
Defining the "difference" between $\mathcal{F}(\mathbb{S})$ and $\mathcal{F}(\mathbb{S}')$ is $\mathcal{F}$-specific, as we show shortly.

Property 2: Given any two packet streams $\mathbb{S}$ and $\mathbb{S}'$, $\mathcal{F}(\mathbb{S} \cup \mathbb{S}') = \mathcal{F}(\mathbb{S}) \cup \mathcal{F}(\mathbb{S}')$.
The $\cup$ operator on the left-hand side denotes a union operation of the two packet streams $\mathbb{S}$ and $\mathbb{S}'$. The $\cup$ operator on the right-hand side denotes a "combination" of $\mathcal{F}(\mathbb{S})$ and $\mathcal{F}(\mathbb{S}')$, which is $\mathcal{F}$-specific and defined shortly.

These two properties enable the conversion *from checking packet stream conservation to checking the conservation of traffic summaries in a neighborhood*. In other words, these two properties enable nodes to simply store the compact packet fingerprints instead of the original packet streams while still enabling the AC to detect the number of packets dropped, modified, and fabricated between two packet streams from their corresponding fingerprints.

Specifically, during the detection phase, the AC only needs to compare the difference between (i) the *combined* traffic summaries for packets *sent to* node s in $\mathbb{N}(s)$, i.e., $\cup_{i \in \mathbb{N}(s)} \mathcal{F}(\mathbb{S}_i^{\rightarrow s})$, and (ii) the *combined* traffic summaries for packets *received from* node s in $\mathbb{N}(s)$, i.e., $\cup_{i \in \mathbb{N}(s)} \mathcal{F}(\mathbb{S}_i^{\leftarrow s})$. By Properties 1 and 2:

$$\begin{aligned}
&\Delta\Big(\bigcup_{i\in\mathbb{N}(s)} \mathcal{F}(\mathbb{S}_i^{\rightarrow s}), \bigcup_{i\in\mathbb{N}(s)} \mathcal{F}(\mathbb{S}_i^{\leftarrow s})\Big) \\
&= \Delta\Big(\mathcal{F}\big(\bigcup_{i\in\mathbb{N}(s)} \mathbb{S}_i^{\rightarrow s}\big), \mathcal{F}\big(\bigcup_{i\in\mathbb{N}(s)} \mathbb{S}_i^{\leftarrow s}\big)\Big) \quad \text{based on Property 2} \\
&\rightsquigarrow \Delta\Big(\bigcup_{i\in\mathbb{N}(s)} \mathbb{S}_i^{\rightarrow s}, \bigcup_{i\in\mathbb{N}(s)} \mathbb{S}_i^{\leftarrow s}\Big) \quad \text{based on Property 1}
\end{aligned} \tag{1}$$

Note that $\Delta(\cup_{i \in \mathbb{N}(s)} \mathbb{S}_i^{\rightarrow s}, \cup_{i \in \mathbb{N}(s)} \mathbb{S}_i^{\leftarrow s})$ reflects the discrepancy between packets sent to and received from node s, and a large discrepancy indicates packet dropping, modification, and fabrication attacks in $\mathbb{N}(s)$.

Sketch for $\mathcal{F}$. The p^{th}moment estimation sketch [5], [17], [33] (as used by Goldberg et al. [19] for *path-based* FL) serves as a good candidate for $\mathcal{F}$. More specifically, p^{th}moment estimation schemes use a random linear map to transform a packet stream into a short vector, called the sketch, as the traffic summary. In *benign* cases, packets, if

viewed as 1.5KB (the Maximum Transmission Unit) bit-vectors, are "randomly" drawn from $\{0,1\}^{1536\times 8}$. Hence, different packet streams will result in different sketches *with a very high probability (w.h.p.)*. Goldberg et al. [19] also extensively studied how to estimate the number of packets dropped, injected, or modified between two packet streams from the "difference" of two corresponding sketch vectors, thus satisfying Property 1. Specifically, the difference $\Delta(\mathcal{F}(\mathbb{S}), \mathcal{F}(\mathbb{S})^{'})$ (used in Property 1) between two sketch vectors is defined as:

$$\Delta(\mathcal{F}(\mathbb{S}), \mathcal{F}(\mathbb{S})^{'}) = ||\mathcal{F}(\mathbb{S}) - \mathcal{F}(\mathbb{S})^{'}||_p^p \quad (2)$$

where $||x||_p^p$ denotes the p^{th}moment of the vector x. We can further prove that the sketch satisfies Property 2 and the combination of $\mathcal{F}(\mathbb{S})$ and $\mathcal{F}(\mathbb{S})^{'}$ used in Property 2 is defined as:

$$\mathcal{F}(\mathbb{S}) \cup \mathcal{F}(\mathbb{S})^{'} = \mathcal{F}(\mathbb{S}) + \mathcal{F}(\mathbb{S})^{'} \quad (3)$$

where + denotes the addition of two vectors. The proof is as follows.

Proof: A sketch function $\mathcal{F}$ over a set of elements $\mathbb{S} = \{p_1, p_2, \ldots, p_n\}$ can be implemented in a "streaming" mode using a hash function h [19], where:

$$h(p_i) \rightarrow \vec{v}_i \quad (4)$$

and $\vec{v}_i$ denotes a vector. More specifically:

$$\mathcal{F}(\mathbb{S}) = \mathcal{F}(\{p_1, p_2, \ldots, p_n\}) = h(p_1) + h(p_2) + \ldots + h(p_n) \quad (5)$$

Hence, given two packet streams $\mathbb{S} = \{p_1, p_2, \ldots, p_n\}$ and $\mathbb{S}^{'} = \{p_1^{'}, p_2^{'}, \ldots, p_{n'}^{'}\}$, we have:

$$\begin{aligned}\mathcal{F}(\mathbb{S} \cup \mathbb{S}^{'}) &= \mathcal{F}(\{p_1, \ldots, p_n, p_1^{'}, \ldots, p_{n'}^{'}\}) \\ &= h(p_1) + \ldots + h(p_n) + h(p_1^{'}) + \ldots + h(p_{n'}^{'})\end{aligned} \quad (6)$$

and:

$$\begin{aligned}\mathcal{F}(\mathbb{S}) + \mathcal{F}(\mathbb{S}^{'}) &= \mathcal{F}(\{p_1, \ldots, p_n\}) + \mathcal{F}(\{p_1^{'}, \ldots, p_{n'}^{'}\}) \\ &= h(p_1) + \ldots + h(p_n) + h(p_1^{'}) + \ldots + h(p_{n'}^{'})\end{aligned} \quad (7)$$

From Equations 6 and 7 we can see that: when $\mathcal{F}(\mathbb{S}) \cup \mathcal{F}(\mathbb{S}^{'})$ is defined as $\mathcal{F}(\mathbb{S}) + \mathcal{F}(\mathbb{S}^{'})$, we have $\mathcal{F}(\mathbb{S} \cup \mathbb{S}^{'}) = \mathcal{F}(\mathbb{S}) \cup \mathcal{F}(\mathbb{S}^{'})$, thus proving Property 2 for Sketch. ■

C. Challenges in a Neighborhood-based FL

From Property 1, we can further derive the following conditions on the fingerprinting function $\mathcal{F}$. Given any two packet streams $\mathbb{S}_r$ and $\mathbb{S}_t$ seen at nodes r and t, respectively, a fingerprinting function computed by r and t should satisfy:

$$\text{if } \mathbb{S}_r = \mathbb{S}_t, \mathcal{F}(\mathbb{S}_r) = \mathcal{F}(\mathbb{S}_t) \quad (8)$$

$$\text{if } \mathbb{S}_r \neq \mathbb{S}_t, \mathcal{F}(\mathbb{S}_r) \neq \mathcal{F}(\mathbb{S}_t) \text{ w.h.p.} \quad (9)$$

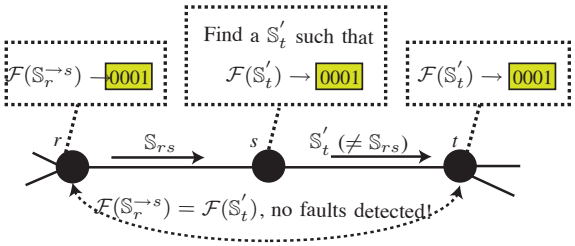


Figure 4. An example of stealthy packet modification attacks when nodes do not use different secret keys for computing $\mathcal{F}$. For simplicity, the sketch vector is represented as a '0-1' bit vector. The malicious node s modifies the packet stream in such a way that the modified packet stream $\mathbb{S}_t^{'}$ still results in the same sketch vector as $\mathbb{S}_{rs}$ at node t.

The first condition ensures the consistency of traffic summaries (more precisely, sketches in the traffic summaries) in the benign case when the packet streams are not corrupted between nodes r and t. The second condition ensures that if packet corruption happens between nodes r and t, inconsistency of the traffic summaries will be observed, which will then enable the estimation of packet difference in the corresponding packet streams (Property 1). However, these two conditions tend to be contradicting and lead to the following dilemma.

$\mathcal{F}$ without different secrets. If the random linear map in $\mathcal{F}$ (which can be implemented as a hash function [14]), is *not* computed with different secret keys by different nodes, a malicious node can predict the $\mathcal{F}$ output of *any other* node for *any* packet. Since $\mathcal{F}$ maps a set of packets (or their 160-bit cryptographic hashes) to a much smaller sketch, *hash collisions* will exist where two different packets produce the same $\mathcal{F}$ output (since sketch is not proven to perserve the collision resistance property of the cryptographic hash function). Hence, a malicious node can leverage such collisions to modify packets such that the modified/fabricated packets will produce the same $\mathcal{F}$ output at other nodes, violating the condition in (9). Figure 4 depicts such an example.

$\mathcal{F}$ with different secrets. If nodes compute $\mathcal{F}$ with different secret keys to satisfy the condition in (9), it is hard for the AC to perform a consistency check among the resulting sketches. For example, even the same packet stream would result in different sketches at different nodes, thus violating the condition in (8). Figure 5 depicts such an example. Since the sketch is only a *compact and approximate* representation of the original packet stream, the AC cannot revert the received sketches to the original packet streams to check packet stream conservation.

Scalability vs. sampling. Even with $\mathcal{F}$ for packet fingerprinting, a traffic summary over a huge number of packets can become too bandwidth-consuming to be sent frequently to the AC (e.g., every 20 milliseconds). For example, the number of packets for an OC-192 link (10Gbps) can be on the order of 10^7 per second in the worst case, which

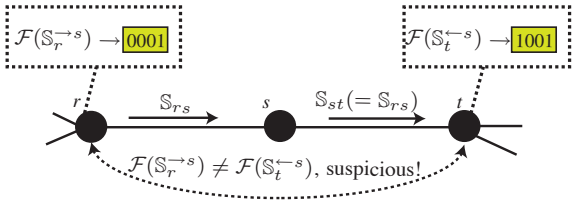


Figure 5. Illustration of the difficulty in using different secret keys when computing $\mathcal{F}$. The sketch vector is represented as a '0-1' bit vector for simplicity. In this example, nodes r, s and t use different secret keys when computing the Sketch to generate their traffic summaries.

swells the size of a sketch to hundreds of bytes to bound the false positive rate below 0.001 [19] and may require several KB/s bandwidth for the reporting by *each* node. Packet sampling represents a popular approach to reducing bandwidth consumption, where each node only samples a *subset* of packets to feed into $\mathcal{F}$ for generating the traffic summaries. To enable a consistency check of the traffic summaries in a neighborhood, all nodes in a neighborhood should sample the *same* subset of packets, and the challenge is how to efficiently decide which subset of packets all nodes should agree to sample. For security, the sampling scheme must ensure that a malicious node cannot predict whether a packet to be forwarded will be sampled or not. Otherwise, the malicious node can drop any non-sampled packets without being detected.

The problem is further complicated by the presence of *collusion attacks* in our strong adversary model as well as our *path obliviousness* requirement. Several existing sampling schemes are broken when applied to our setting. For example, in Symmetric Secure Sampling (SSS) [19], the packet sender and receiver use a shared Pseudo-Random Function (PRF) $\mathcal{P}$ to coordinate their sampling. Imported to our setting, e.g., using the neighborhood example in Figure 5, nodes r and t share a secret key K_{rt} and a PRF $\mathcal{P}$, compute $\mathcal{P}$ with K_{rt} for each packet, and sample the packet if the PRF output is within a certain range. In this way, node s *itself* cannot know whether a packet is sampled or not. However, this approach fails in our setting. We consider the topology in Figure 5 for example:

- If s and r collude, r can inform s of which packets are sampled, so that s can safely drop non-sampled packets and not be detected.
- Due to the dynamic traffic pattern, an ingress node r of a neighborhood $\mathbb{N}(s)$ does not know which egress node a packet will traverse in $\mathbb{N}(s)$ (if s has more neighbors besides r and t, there exist multiple possible egress nodes than t). Hence, r does not know which PRF or secret key to use for packet sampling, given that r shares a different secret key with each node in $\mathbb{N}(s)$.

D. DynaFL Key Ideas

In DynaFL, nodes temporarily store the cryptographic hashes (which are collision-resistant) for all packets received/sent *per neighbor* in an epoch. At the end of each epoch e, nodes use *epoch sampling* to decide if packets in the epoch are to be fingerprinted; if so, nodes generate the traffic summaries and report them to the AC. This reduces both the communication overhead for sending the traffic summaries to the AC and the computational overhead for generating and checking the traffic summaries. Specifically, nodes first use the *network-wide* identical per-epoch **sampling key** K_s^e (described shortly) for computing a PRF $\mathcal{P}$ to determine if the current epoch is "selected"; if and only if the current epoch is selected, nodes will use $\mathcal{F}$ with the *network-wide* identical per-epoch **fingerprinting key** K_f^e (described shortly) to map packets into per-neighbor traffic summaries. Using the same K_s^e and K_f^e enables consistency checking over the traffic summaries from different nodes.

To address the packet modification attacks and collusion attacks mentioned earlier, nodes do *not* know the per-epoch K_s^e and K_f^e until the *end* of each epoch e, after they *have forwarded* (or possibly corrupted) packets in epoch e. Thus, when a packet is to be forwarded (or corrupted), a malicious node does not know K_s^e and K_f^e, and thus cannot predict whether this epoch is selected for sending traffic summaries, and if selected, what the sketch output will be for this packet. To achieve this property, in DynaFL, the trusted AC periodically sends the per-epoch K_s^e and K_f^e via **key disclosure messages** to all nodes at the end of each epoch in a reliable way (described later) and nodes use the received K_s^e and K_f^e to select epochs and fingerprint packets that have already been forwarded or corrupted.

A malicious node may first attempt to locally hold all the packets in an epoch e, and only forward or corrupt packets at the end of e when the malicious node learns K_s^e and K_f^e, thus being able to launch the packet modification and selective packet corruption attacks as mentioned earlier. However, since the traffic summaries also include the average departure/arrival time of the sent/received packets, the malicious node will be detected with packet delay misbehavior in the detection phase.

Sections IV, V, and VI detail the recording, reporting, and detection phases in DynaFL, respectively. Section VII presents the security analysis and Section VIII evaluates DynaFL's performance through measurements and simulations.

IV. Recording Traffic Summaries

The main technical challenges in the recording phase are how to deal with *imperfect* time synchronization among nodes and packet transmission delay, and how to efficiently protect the key disclosure message from adversarial corruption. We explain how DynaFL solves these challenges in turn below.

A. Storing Packets

In the "ideal" case (with perfect time sychronization and no packet transmission delay), nodes simply need to store packets for the single "current" epoch and at the end of each epoch send the traffic summaries to the AC for that epoch. However, in practice, routers need to determine which epoch an incoming packet belongs to (or whether a received packet belongs to the current epoch or a previous, outdated epoch). One might attempt to let routers map received packets into epochs based on their local packet arrival time. However, this approach would introduce large errors for the following reasons:

- Though all the nodes in the network are *loosely* time-synchronized, e.g., ± 1 millisecond, the epoch intervals at different nodes may still be misaligned by up to a few milliseconds. This misalignment will result in a considerable number of packets being attributed to different epochs at different nodes, thus causing inconsistencies in the corresponding packet fingerprints.
- Due to the network transmission delay, a packet sent by a source at epoch e may arrive at another node at a different epoch $e+i$. In other words, a packet may have been received by an ingress node but not the egress node of a neighborhood at the end of an epoch when nodes need to generate their packet fingerprints, thus producing inconsistencies in the traffic summaries.

To deal with imperfect time synchronization, the source in DynaFL embeds a *local* timestamp when sending each packet. Such a timestamp can be added as an additional flow header, using the TCP timestamp, or in the IP option field, etc. Any router in the forwarding path will determine the corresponding epoch for each packet based on the embedded timestamp. In this way, we ensure that all routers put each packet in the same epoch for updating the traffic summaries. For example, if the timestamp embedded by the source is t_s and the epoch length is L, then all routers will map the packet into epoch $\lfloor \frac{t_s}{L} \rfloor$.

To eliminate traffic summary inconsistencies due to packet transmission delay, we also need to ensure that when generating traffic summaries for a certain epoch e, packets that are sent and not corrupted in epoch e are received by *all* the nodes in the forwarding paths. To this end, if the epoch length is set to L and the expected upper bound on the *one-way* packet transmission delay in the network is D, each router stores packets sent in the current epoch e *as well as* in previous $\lceil \frac{D}{L} \rceil$ epochs, denoted by $e-1, e-2, \ldots, e-\lceil \frac{D}{L} \rceil$. We call these epochs **live epochs**. Then at the end of an epoch e, nodes will generate and send to the AC the traffic summaries for the *oldest* live epoch $e-\lceil \frac{D}{L} \rceil$, in which the packets have either traversed all nodes in their forwarding paths or been corrupted. The periodic key disclosure messages that the AC broadcasts synchronize the current epoch ID and the oldest live epoch ID for which traffic summaries are needed for reporting.

Hence, a node s maintains the following data structures for each neighbor r for each epoch, as Figure 6 also shows.

- The packet cache $\mathbb{C}_s^{\leftrightarrow r}$ temporarily stores hashes for packets in both $\mathbb{S}_s^{\rightarrow r}$ and $\mathbb{S}_s^{\leftarrow r}$ that are seen in a live epoch (using a cryptographic hash function such as SHA-1). Each entry contains the packet hash and a bit indicating if the packet belongs to $\mathbb{S}_s^{\rightarrow r}$ or $\mathbb{S}_s^{\leftarrow r}$.
- The router stores the *sum* of packet departure timestamps $t_s^{\rightarrow r}$ seen in $\mathbb{S}_s^{\rightarrow r}$ and the sum of packet arrival timestamps $t_s^{\leftarrow r}$ seen in $\mathbb{S}_s^{\leftarrow r}$ in a live epoch with microsecond precision.
- Finally, the router stores the total number of packets $n_s^{\rightarrow r}$ seen in $\mathbb{S}_s^{\rightarrow r}$ and $n_s^{\leftarrow r}$ seen in $\mathbb{S}_s^{\leftarrow r}$ in a live epoch.

In DynaFL, a router s also needs to consider the case where its next-hop neighbor r is the destination for a certain packet, so that r will naturally not forward the packet. If it is the case for a certain packet, router s does not cache that packet for neighbor r.

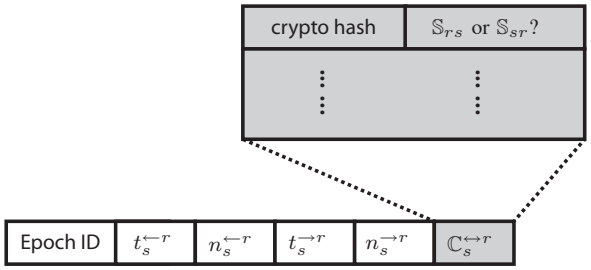


Figure 6. Router per-neighbor state details.

Among these data structures, $t_s^{\leftarrow r}$, $t_s^{\rightarrow r}$, $n_s^{\leftarrow r}$, and $n_s^{\rightarrow r}$ require small constant storage, around 8 or 4 bytes for each. $\mathbb{C}_s^{\leftrightarrow r}$ will be used for packet fingerprinting. The size of $\mathbb{C}_s^{\leftrightarrow r}$ depends only on the epoch length L and link bandwidth, but not the number of flows/paths traversing node s. As Section VIII-A shows, with an epoch length of 20 milliseconds and one-way network latency of 20 milliseconds, each router line-card requires only around 4MB of memory for an OC-192 link, which is practical today.

For simplicity's sake, we use $\mathbb{C}_s^{\rightarrow r}$ and $\mathbb{C}_s^{\leftarrow r}$ to denote the packets cached for $\mathbb{S}_s^{\rightarrow r}$ and $\mathbb{S}_s^{\leftarrow r}$ by node s, respectively.

B. Secure Key Disclosure

At the end of each epoch e, the AC discloses the sampling key $K_s^{e-\lceil \frac{D}{L} \rceil}$ and fingerprinting key $K_f^{e-\lceil \frac{D}{L} \rceil}$ to all nodes in the network via a *key disclosure message* $\mathtt{d_{AC}}$, and requests the traffic summaries for the most recently retired epoch $e-\lceil \frac{D}{L} \rceil$. Obviously, $\mathtt{d_{AC}}$ itself needs to be protected from data-plane attacks (dropping, modification, fabrication, or delaying) by a malicious node during end-of-epoch broadcasting. It might be tempting to let the AC use digital signatures to authenticate $\mathtt{d_{AC}}$ in order to address malicious modification and fabrication; however, frequently

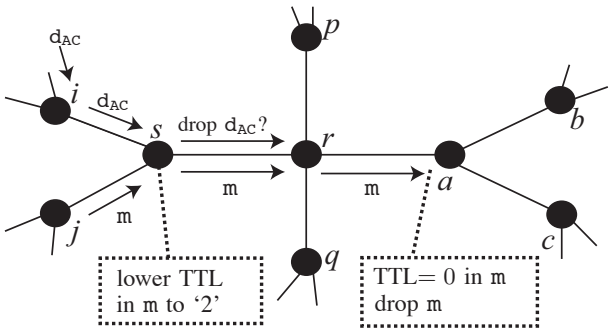


Figure 7. Possible attacks in the recording phase. A malicious node s may attempt to drop the key disclosure message $\mathtt{d_{AC}}$, or manipulate the TTL value to cause packets to be dropped at a remote place (node a in this example), thus framing a remote neighborhood ($\mathbb{N}(a)$ in this example).

generating and verifying the signatures on a per-epoch basis can be expensive (e.g., an epoch can be as short as 20 milliseconds and signature generation and verification time could be on the order of milliseconds).

Our key observation is that, the key disclosure message $\mathtt{d_{AC}}$ is transmitted at the end of each epoch synchronously among all the nodes. If a malicious node s drops $\mathtt{d_{AC}}$, the AC will fail to receive the traffic summaries of certain neighbors of s, thus detecting $\mathbb{N}(s)$ as suspicious. For example in Figure 7, if s drops $\mathtt{d_{AC}}$ instead of forwarding it to its neighbor r, node r cannot fingerprint the packets to generate traffic summaries, thus failing the consistency check of traffic summaries in $\mathbb{N}(s)$. As we show in Section V, the AC expects to receive traffic summaries within a short amount of time after each epoch ends; delaying $\mathtt{d_{AC}}$ more than that amount of time is effectively equivalent to dropping $\mathtt{d_{AC}}$ and causes the malicious node's neighborhood to be detected. Thus, the remaining problem is to prevent the modification and fabrication of $\mathtt{d_{AC}}$, which is equivalent to authenticating $\mathtt{d_{AC}}$ to all nodes in the network *without* the use of digital signatures. Section VII further elaborates why the authentication of $\mathtt{d_{AC}}$ is needed for security purposes.

The $\mathtt{d_{AC}}$ for epoch j includes an *epoch key* K^j, based on which the epoch sampling key K_s^j and the epoch fingerprinting key K_f^j can be derived using a Pseudo-Random Function (PRF), e.g.:

$$K_s^j \leftarrow PRF_{K^j}(1),\ \ K_f^j \leftarrow PRF_{K^j}(2) \tag{10}$$

Furthermore in DynaFL, time in the network is loosely time-synchronized and divided into consecutive epochs; the authentication of $\mathtt{d_{AC}}$ is required only once per epoch. Hence, we just need to authenticate K_s^j for each epoch, which can be efficiently achieved via a *one-way hash chain*. As Figure 8 shows, the AC applies a one-way function H (a cryptographic hash function) repeatedly on the root key K^r to derive a set of epoch keys. The AC publishes K_0 in a bootstraping broadcast message through the network so that nodes can verify if any given epoch key is indeed derived

Figure 8. One-way hash chain example.

from the genuine one-way hash chain and is thus authentic. We assume each node in the network has the correct public key of the AC, so that the AC can authenticate K_0 via digital signatures during the bootstraping phase. Along with K_0, an epoch number is included and authenticated in the bootstraping broadcast message to enable switching to a new key chain whenever needed.

Furthermore, DynaFL creates a spanning tree in the network rooted at the AC, along which $\mathtt{d_{AC}}$ is delivered to each node. Since DynaFL uses a *pre-generated, static* spanning tree for the broadcast messages, there is no need for dynamic path support when protecting $\mathtt{d_{AC}}$.

C. Sampling and Fingerprinting

Given the disclosed K_s^j and K_f^j at the end of an epoch e, each node t first uses the sampling PRF $\mathcal{P}$ with K_s^j, denoted by $\mathcal{P}_{K_s^j}$, to determine if the oldest live epoch j is selected. If so, node t then uses the fingerprinting function $\mathcal{F}$ to map the cached packet hashes in each per-neighbor stream into a sketch vector, i.e., $\mathcal{F}_{K_f^j}(\mathbb{C}_t^{\rightarrow r})$ or $\mathcal{F}_{K_f^j}(\mathbb{C}_t^{\leftarrow r})$, computed with the given K_f^j. Finally, node t generates *two* traffic summaries $\mathrm{T}_t^{\rightarrow r}$ and $\mathrm{T}_t^{\leftarrow r}$ for a neighbor r:

- $\mathrm{T}_t^{\rightarrow r}$ for packet stream $\mathbb{S}_t^{\rightarrow r}$ includes a fingerprint $\mathcal{F}_{K_f^j}(\mathbb{C}_t^{\rightarrow r})$, average packet departure time $\overline{t}_t^{\rightarrow r} = \frac{t_t^{\rightarrow r}}{n_t^{\rightarrow r}}$, and the total number $n_t^{\rightarrow r}$ of packets seen in $\mathbb{S}_t^{\rightarrow r}$ in epoch j;
- $\mathrm{T}_t^{\leftarrow r}$ for packet stream $\mathbb{S}_t^{\leftarrow r}$ includes a fingerprint $\mathcal{F}_{K_f^j}(\mathbb{C}_t^{\leftarrow r})$, average packet arrival time $\overline{t}_t^{\leftarrow r} = \frac{t_t^{\leftarrow r}}{n_t^{\leftarrow r}}$, and the total number $n_t^{\leftarrow r}$ of packets seen in $\mathbb{S}_t^{\leftarrow r}$ in epoch j.

Figure 9 summarizes the FL-related packet processing inside a DynaFL router. We detail $\mathcal{P}$ and $\mathcal{F}$ in the following.

Implementing $\mathcal{P}$. A n-bit epoch sampling key K_s^j is derived via a PRF (Equation 10) and is thus uniformly distributed in $[0, 2^n - 1]$. Given a *sampling rate* $\lambda \in (0, 1)$, an epoch j is selected iff:

$$K_s^j < \lambda \cdot 2^n \tag{11}$$

In this way, on average a fraction λ of the epochs will be selected. Since nodes use the same K_s^j for epoch sampling, benign nodes will select the same set of epochs, thus ensuring the consistency of the traffic summaries in a neighborhood.

Implementing $\mathcal{F}$. We use the second-moment sketch computed with K_f^j as a case study to implement $\mathcal{F}$, and analyze the size of the sketch vector to achieve Property 1 with (α, β, δ)-accuracy. We assume 10^7 packets per second in the worst case for an OC-192 link with an epoch length

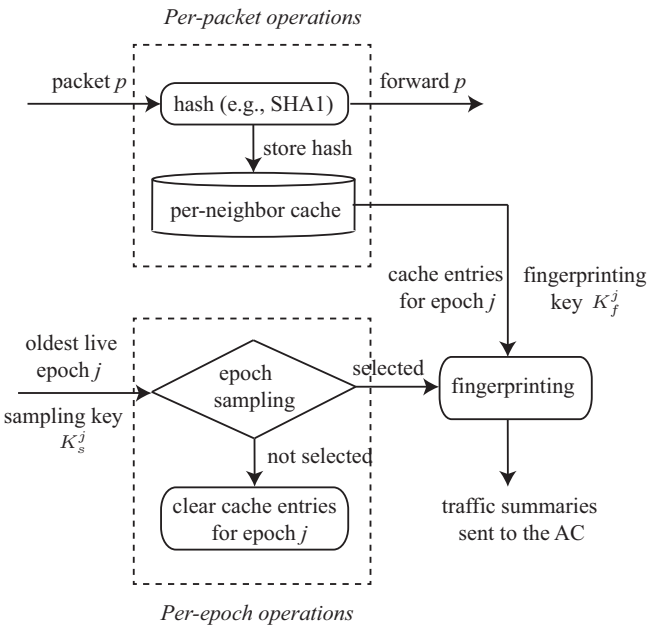


Figure 9. FL-related packet processing inside a DynaFL router.

of L (seconds). Then, the number of packets η in a sampled epoch is $\eta = L \cdot 10^7$. Using the classical Sketch due to Alon et al. [6] for example, the storage requirement for the sketch is given by:

$$M \times \log_2 \sqrt{2\eta \ln(\frac{200N}{\delta})} \tag{12}$$

$$\text{where } M > \frac{12}{\epsilon^2} \frac{1}{3-2\epsilon} \ln \frac{1}{\delta} \quad \text{and } \epsilon = \frac{\beta - \alpha}{\beta + \alpha}. \tag{13}$$

In Section VIII-A we derive numeric values for the size of the sketch vector based on the epoch length L.

Dealing with TTL attacks. Certain fields in the IP header, such as the TTL, checksum, and some IP option fields, will change at each hop. Both sampling and fingerprinting in DynaFL need to properly deal with these variant fields. Take the TTL field for instance hereinafter (though the arguments apply similarly to other variant fields). On the one hand, if $\mathcal{F}$ is computed over the entire packet including the TTL field, even in the benign case the same packet stream will leave different traffic summaries (or precisely, the sketch vectors) at ingress and egress nodes. On the other hand, if $\mathcal{F}$ is computed over the entire packets excluding the TTL field, a malicious node can modify the TTL field at liberty without affecting the traffic summaries. Figure 7 depicts an example TTL attack, where the malicious node s lowers the TTL value to 2 in the packets and causes the packets to be dropped at the 2-hop-away downstream node a, thus framing neighborhood $\mathbb{N}(a)$.

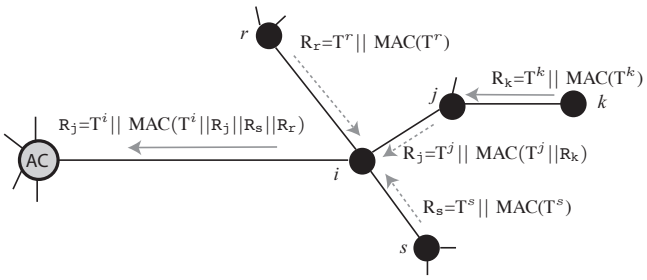


Figure 10. Example of secure transmission of traffic summary reports. For brevity, we denote the traffic summaries of a node i as T^i and omit the secret key for the MAC notation.

To address the TTL attacks, when computing $\mathcal{F}$, each node r performs either of the following:

- For a packet received from a neighbor, node r computes $\mathcal{F}$ over the entire packet *including* the TTL field.
- For a packet sent to a neighbor, node r computes and $\mathcal{F}$ over the packet, but with the TTL field additionally decreased by 2 (equal to the TTL value at the 2-hop-away egress node in $\mathbb{N}(r)$).

In this way, node r in Figure 7 simply uses the TTL value as contained in the packets received from s when computing $\mathcal{F}$, since the ingress nodes in $\mathbb{N}(s)$ (nodes i and j) must have computed $\mathcal{F}$ with an adjusted TTL value equal to that at node r.

The TTL value in a packet is also decremented by one for every second the packet is buffered at a router. Holding a packet longer than one second at a router is treated as a packet delaying attack and will be detected due to the use of the above construction.

V. Reporting Traffic Summaries

If an epoch is selected, after the fingerprinting procedure, a node t generates two traffic summaries $T_t^{\rightarrow r}$ and $T_t^{\leftarrow r}$ for each neighbor r, and sends them to the AC in a traffic summary report denoted by $\mathcal{R}_t$. The challenge in the recording phase is to protect the traffic summary reports from being corrupted.

In DynaFL, nodes form a static spanning tree rooted at the AC for sending the traffic summaries. Given the spanning tree, the goal is to protect the traffic summary reports $\mathcal{R}_t$s from different nodes destined to the AC. Although $\mathcal{R}_t$s are also subject to data-plane attacks, they are transmitted over *static* and *pre-generated* paths in the spanning tree. Hence, dynamic traffic is no longer a concern, thus substantially simplifying the problem. Specifically, DynaFL utilizes an *Onion Authentication* approach [34], [36] to protect the transmission of $\mathtt{d}_{\mathtt{AC}}$ along each path in the spanning tree. In a nutshell, within a short timer at the end of each epoch, each node t needs to send its traffic summary report $\mathcal{R}_t$ to the AC, and $\mathcal{R}_t$ is authenticated with a MAC computed using a pairwise secret key shared between node t and the

AC. The traffic summary reports from different nodes are sent in an *onion* fashion. For example in Figure 10, $\mathcal{R}_j$ includes the report $\mathcal{R}_k$ of node k. In this way, DynaFL efficiently protects the key disclosure message $\mathtt{d_{AC}}$ without the use of expensive asymmetric cryptography. Section VII gives a more detailed security analysis of such an Onion Authentication approach.

VI. Detection

The AC performs consistency checks for each neighborhood $\mathbb{N}(r)$ based on the received traffic summaries. However, since an epoch may only have a small number of packets, detecting a suspicious neighborhood based on the consistency checks for *individual* epochs can introduce a large error rate. Take an extreme case for example: if in a certain epoch a neighborhood $\mathbb{N}(r)$ only transmits a single packet and the packet was spontaneously lost, concluding that the packet loss rate is 100% and $\mathbb{N}(r)$ is suspicious would be inaccurate.

To deal with this problem, the AC still performs the consistency checks and estimates the discrepancy for individual epochs; but it makes the detection based on the *aggregated* discrepancies over a set of E epochs (called **accumulated epochs**), so that the total number of packets over the E epochs is more than a certain threshold N to give a high enough accuracy (e.g., $> 99.9\%$) on the detection results. Section VIII studies the value of N. Therefore, the AC stores the traffic summaries for each neighborhood and makes detection when the total number of packets N is reached. More specifically, let $n_x^{\leftarrow y}(e)$ and $n_x^{\rightarrow y}(e)$ denote the packets received from / sent to x ($n_x^{\leftarrow y}$ and $n_x^{\rightarrow y}$) in the traffic summary for epoch e, respectively; for a certain neighborhood $\mathbb{N}(r)$, whenever

$$max\{\sum_e \sum_i n_i^{\rightarrow r}(e), \sum_e \sum_i n_i^{\leftarrow r}(e)\} > N \tag{14}$$

(where $i \in \mathbb{N}(r)$ and e iterates over all the accumulated epochs), indicating N is reached, the AC performs the following checks to inspect if $\mathbb{N}(r)$ is suspicious:

1. Flow conservation. The AC first extracts $n_i^{\rightarrow r}(e)$ and $n_i^{\leftarrow r}(e)$ for each node i in $\mathbb{N}(r)$ for each epoch e, and calculates the difference between the number of packets sent to r and the number of packets received from r over all the E accumulated epochs. If the ratio of the difference to the total number of packets in all the E accumulated epochs is larger than a threshold β, i.e.:

$$\frac{|\sum_e \sum_i n_i^{\rightarrow r}(e) - \sum_e \sum_i n_i^{\leftarrow r}(e)|}{max\{\sum_e \sum_i n_i^{\rightarrow r}(e), \sum_e \sum_i n_i^{\leftarrow r}(e)\}} > \beta \tag{15}$$

then the AC detects $\mathbb{N}(r)$ as suspicious. The threshold β is set based on the administrator's expectation of the natural packet loss rate; e.g., in the simulations in Section VIII we set β to be four times of the natural packet loss rate in a neighborhood.

2. Content conservation. The AC then extracts the sketches in the traffic summaries in $\mathbb{N}(r)$, and estimates the discrepancy δ_f between the sketches for packets sent to r and the sketches for packets received from r. The AC detects $\mathbb{N}(r)$ as malicious if δ_f is larger than a certain threshold, i.e.,:

$$\delta_f > \frac{2\alpha\beta}{\alpha+\beta} \times max\{\sum_e \sum_i n_i^{\rightarrow r}(e), \sum_e \sum_i n_i^{\leftarrow r}(e)\}$$

where

$$\delta_f = ||\cup_{i\in\mathbb{N}(r)} \mathcal{F}_{K_f^j}(\mathbb{C}_i^{\leftarrow r}) - \cup_{i\in\mathbb{N}(r)} \mathcal{F}_{K_f^j}(\mathbb{C}_i^{\rightarrow r})||_2^2 \tag{16}$$

It has been proven [19] that the above threshold can satisfy the (α, β, δ)-accuracy defined in Section II-D.

3. Timing consistency. Finally, the AC extracts the difference between the average packet departure time and arrival time, and concludes that $\mathbb{N}(r)$ is suspicious if the difference is larger than the expected upper bound on the 2-hop link latency.

VII. Security Analysis

We show that DynaFL is secure against all attacks that are possible in the misbehavior space given our adversary model. By our definition, a malicious router can drop, modify, fabricate, and delay packets. In addition, a malicious router can attack data packets, key disclosure messages $\mathtt{d_{AC}}$, and reporting messages. We first show DynaFL's security against a single malicious node and then sketch DynaFL's security against colluding nodes.

Security against corrupting the data packets. Dropping, modifying, and fabricating data packets in a neighborhood $\mathbb{N}(m)$ will cause inconsistencies between sketches in $\mathbb{N}(m)$ as mentioned earlier. Delaying data packets in $\mathbb{N}(m)$ will cause abnormal deviation between average packet arrival/departure timestamps in $\mathbb{N}(m)$. If a malicious router changes the timestamps in data packets embedded by the source nodes, it is equivalent to modifying packets and packets may be mapped to different epochs, in which case such an attack will manifest itself by causing inconsistencies in the sketches of a neighborhood containing the malicious router.

Security against corrupting $\mathtt{d_{AC}}$. As we mentioned earlier, if a malicious node m drops the $\mathtt{d_{AC}}$, some nodes adjacent to m will fail to send the correct traffic summaries to the AC, thus causing a neighborhood containing m to be detected. We note that the authentication of $\mathtt{d_{AC}}$ is needed (through the one-way hash chain). Otherwise, a malicious node can replace the sampling and fingerprinting keys with its own fake keys, by which the malicious node can predict the output of other nodes's sketches and perform packet modification attacks. In addition, if the epoch IDs in $\mathtt{d_{AC}}$ were not authenticated, a malicious node can replace the oldest live epoch ID in $\mathtt{d_{AC}}$ for which the traffic summaries

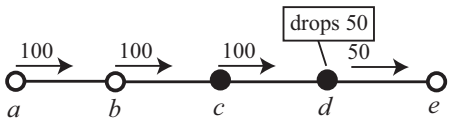


Figure 11. Example of DynaFL's security against colluding nodes. A number denotes the packet count each node sends.

are requested with the current epoch ID. In this way, inconsistencies of traffic summaries can be detected for some *benign* neighborhood due to the packet transmission delay as Section IV-A describes. With the (delayed) authentication of $\mathtt{d}_{\mathtt{AC}}$, any attempt to modify $\mathtt{d}_{\mathtt{AC}}$ will be detected (after $\lceil \frac{D}{L} \rceil$ epochs).

It is noteworthy that the $\mathtt{d}_{\mathtt{AC}}$ sent at the end of epoch e cannot simply disclose the MAC secret key K_{e-1} for the previous epoch $e-1$. This is because at the time K_{e-1} is disclosed, the $\mathtt{d}_{\mathtt{AC}}$ sent at the end of epoch $e-1$ may not have yet reached all nodes. Hence, a malicious node which has already received K_{e-1} might send K_{e-1} to a downstream colluding node via an out-of-band channel, so that the colluding node can break the authenticity of the $\mathtt{d}_{\mathtt{AC}}$ sent in epoch $e-1$. Hence, at the end of an epoch e, we disclose the MAC key for epoch $e - \lceil \frac{D}{L} \rceil$ to ensure the $\mathtt{d}_{\mathtt{AC}}$ sent in epoch $e - \lceil \frac{D}{L} \rceil$ has reached all the nodes in the network.

Security against corrupting reporting messages. First, due to the use of the Onion Authentication, a malicious node m cannot *selectively* drop the reporting messages of a *remote* (non-adjacent) node r, to frame a neighborhood containing node r. Since all the accumulated reporting messages are "combined" at each hop, m can only drop the reporting messages from its *immediate* neighbors, which will manifest a neighborhood containing m as suspicious.

Security against colluding attacks. We illustrate DynaFL's security against colluding attacks via a toy example shown in Figure 11. We show that for a malicious node m which actually corrupts packets, *as long as one benign node exists in* $\mathbb{N}(m)$*, a neighborhood containing either* m *or one of its colluding nodes will be detected*. The key observation is that since the traffic summaries are sent to the AC and the AC performs the detection, *each node can only claim one traffic summary per selected epoch*. To simplify the analysis while still unveiling the intuition, we only consider the number (but not the payload) of packets sent by each node, as shown in Figure 11. Suppose nodes c and d are colluding, and node d drops 50 packets. As long as node e is benign in $\mathbb{N}(d)$, to cover the misbehavior of d, the colluding node c has to send a traffic summary to the AC falsely claiming it sent "50" packets to d (and thus received "50" packets from node b). However, this claim will make the neighborhood $\mathbb{N}(b)$ suspicious since the benign node a will claim it sent 100 packets to b.

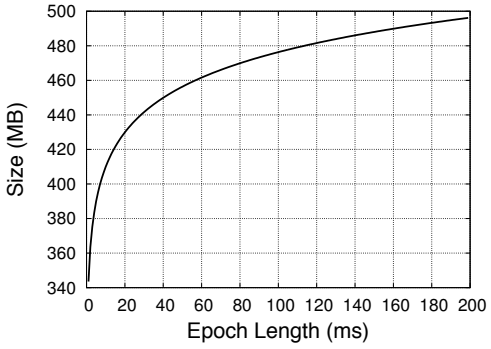


Figure 12. Sketch size for an OC-192 link with the average packet size of 300 bytes and $\delta = 0.001$.

VIII. Performance Evaluation

In this section, we analyze the protocol overhead and study the detection efficiency of DynaFL via measurements and simulations, with our implementation of the classic Sketch [6] in C++.

A. Storage Overhead

DynaFL incurs only per-neighbor state while existing secure path-based FL protocols require per-source and per-path state. In this section, we quantify the per-neighbor storage overhead of a DynaFL router, which primarily includes the packet cache and the sketch for each neighbor.

Sketch size. We derive numeric values of the sketch size based on Equations 12 and 13, using an example setting where the average packet size is 300 bytes and the link's capacity is 10 Gbps (an OC-192 link). Furthermore, we consider $\delta = 0.001$, $\alpha = 0.002$, and $\beta = 2\alpha$ for the (α, β, δ)-accuracy, i.e., the false positive rate and false negative rate of the sketch-based detection are limited under 0.001. Figure 12 plots the result, from which we can see that a sketch with fewer than 500 bytes can already yield a desirable accuracy.

Cache size and per-neighbor storage overhead. We now study the cache size for temporarily storing packet hashes in live epochs, which, together with the sketch size analyzed above, constitutes the per-neighbor storage overhead of a DynaFL router. We denote the upper bound of one-way network latency as D, epoch length as L, and the number of packets per second as η. Using 20-byte packet hashes, the cache size is given by:

$$\lceil \frac{D}{L} + 1 \rceil \cdot 20 \cdot \eta \cdot L \tag{17}$$

We omit the 1-bit indicator for each packet hash entry to indiciate which packet stream the packet belongs to (see Figure 6). Assuming the per-neighbor sketch size is 500 bytes, one-way latency $D = 20$ms, and the average packet size is 300 bytes for an OC-192 link, we derive

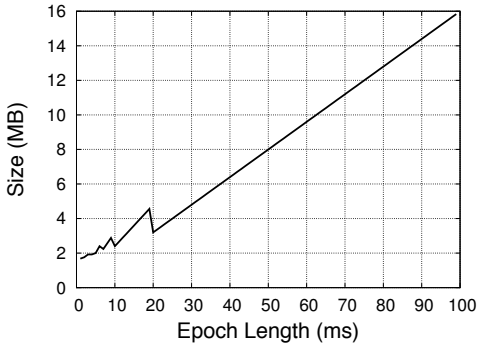


Figure 13. Router per-neighbor state for an OC-192 link with the average packet size of 300 bytes and one-way network latency as 20ms.

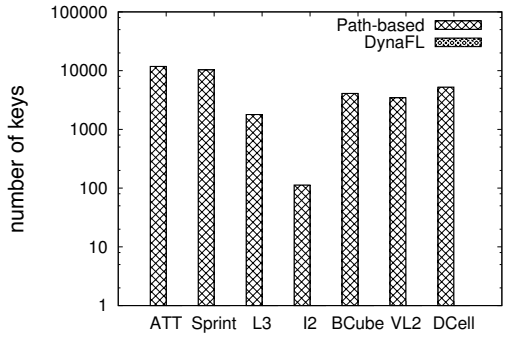


Figure 14. Key management overhead at each router. A router in DynaFL always requires just one key shared with the AC.

the per-neighbor storage overhead of a DynaFL router with different epoch lengths shown in Figure 13. We can observe that, with an epoch length of 20ms, only around 4MB is required per-neighbor. The "humps" exist in the curve due to the use of the ceiling function in Equation 17.

B. Key Management Overhead

One distinct advantage DynaFL presents is that each router in DynaFL shares only one secret key with the AC, whereas in path-based FL protocols it is *necessary* for each router to share a secret key with each source node in the network in the worst case [14], which dramatically complicates the key management and broadens the vulnerability surface. To quantify DynaFL's advantage over path-based FL protocols, we leverage the measured ISP topologies from the Rocketfuel dataset [31] and the topology from Internet2 [4]. Figure 14 shows the maximum number of keys each router needs to manage in path-based FL protocols; and a router in DynaFL always requires only one secret key shared with the AC (thus invisible in the figure). We can see that the number of keys a router needs to manage in path-based FL protocols is 100 to 10000 times higher than that in DynaFL.

C. Bandwidth Overhead

We analyze the bandwidth consumption on each link by the reporting traffic summaries based on the measured ISP topologies from the RocketFuel dataset [31]. Recall that the reporting messages are transmitted along a spanning tree rooted at the AC. Hence, the bandwidth consumption by the reporting messages on a link is determined by the number of children below that link and the degrees of the children.

For each ISP topology, we first select a "central" node as the AC, which is the node in the network that has the highest fraction of all shortest paths that pass through that node. Then, we create a minimum spanning tree rooted at the central node (or the AC) for transmitting reporting messages to the AC. We consider the epoch length L=20ms, a per-neighbor traffic summary as 500 bytes, and the epoch sampling rate is 1%. Hence, on average, each node only sends one reporting packet in every two seconds. Figure 15 plots the results for ISPs with AS numbers 1221, 1239, 1755, 3257, 3967, and 6461. From the results, we can see that the fraction of bandwidth used for reporting traffic summaries on a link is small for all topologies (e.g., between 0.002% and 0.012% for an OC-192 link).

D. Detection Delay

As Section VI states, the AC performs consistency checks and detects any anomalies only when the total number of packets over multiple epochs is accumulated more than a certain threshold N in order to give a low false positive and negative rate (e.g., <0.1%) on the detection results. Hence, the number of packets N characterizes the detection delay of the FL protocol. We fully implement the classic Sketch due to Alon et al. [6] in C++ with a four-way hash function, and perform simulations to study N.

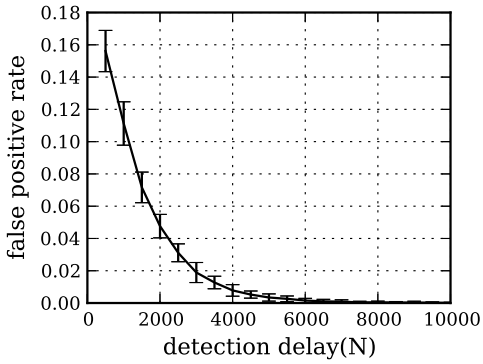
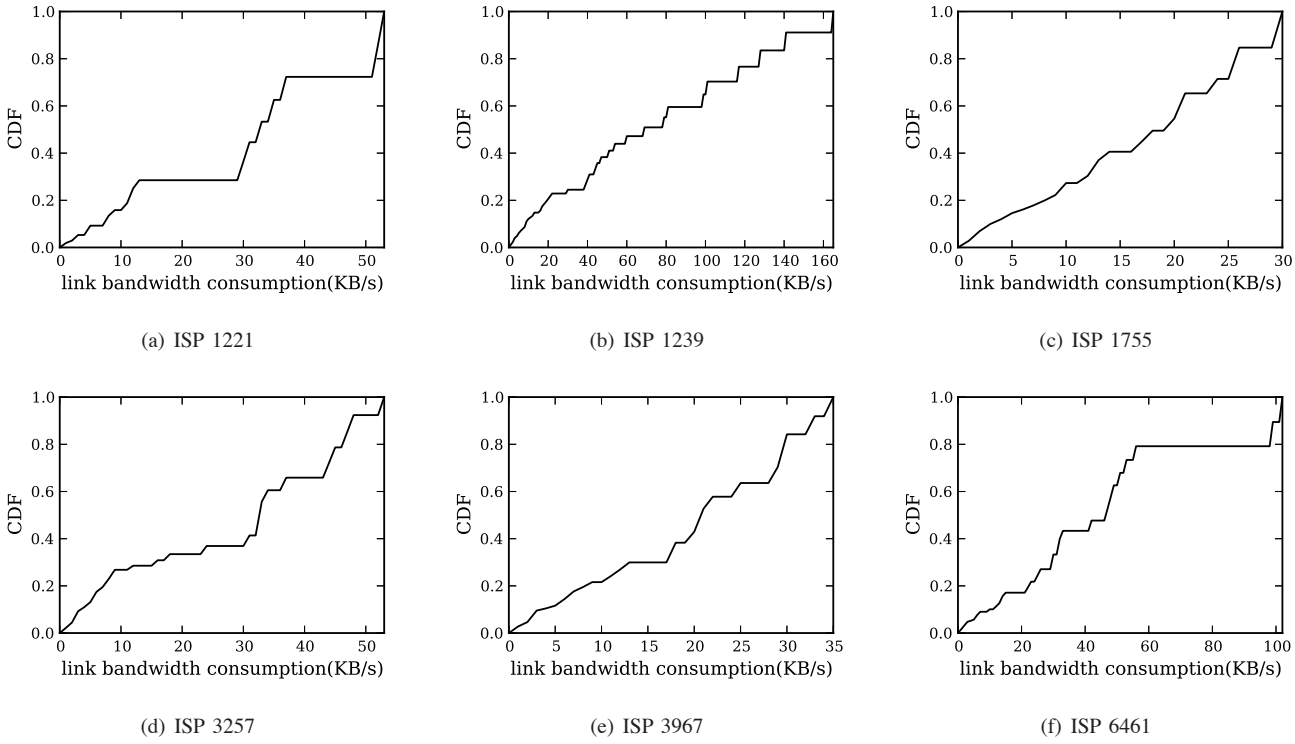


Figure 16. False positive rates with no malicious activity in a neighborhood with different numbers of nodes. The natural packet loss rate in a neighborhood is 0.001 and the detection thresholds for both flow conservation and content conservation are $T_d = \beta = 2\alpha = 0.004$.

Since in DynaFL, neighborhoods are inspected by the AC *independently*, we also perform simulations for independent



(a) ISP 1221 (b) ISP 1239 (c) ISP 1755

(d) ISP 3257 (e) ISP 3967 (f) ISP 6461

Figure 15. CDF of per-link bandwidth consumption for the reporting messages in DynaFL.

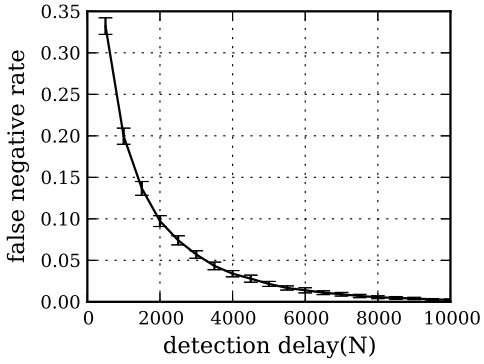


Figure 17. False negative rates in a malicious neighborhood with five nodes, where the malicious node *only drops* packets. The natural packet loss rate in a neighborhood is 0.001, the detection thresholds for both flow conservation and content conservation are $T_d = \beta = 2\alpha = 0.004$, and the malicious packet dropping rate is 0.005.

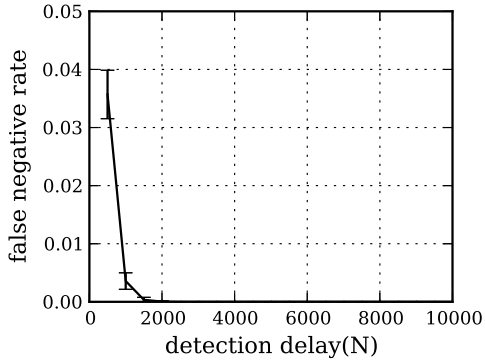


Figure 18. False negative rates in a malicious neighborhood with five nodes, where the malicious node both drops packets and modifies packets. The natural packet loss rate in a neighborhood is 0.001, the detection thresholds for both flow conservation and content conservation are $T_d = \beta = 2\alpha = 0.004$, the malicious packet dropping rate is 0.005, and the malicious packet modification rate is 0.005.

neighborhoods with different sizes. Since we showed DynaFL's security against colluding attacks in Section VII, we emulate a single malicious node in our simulations. Our setting is as follows. The natural packet loss rate in a neighborhood is 0.001 and the detection thresholds for both flow conservation and content conservation are $\beta = 2\alpha = 0.004$. Figure 16 depicts the false positive rates in benign cases where no malicious routers exist in the neighborhood. We can see that with $N > 5000$ packets, the false positive rate is under 1%.

Figure 17 shows the false negative rates with a malicious router which only drops packets with a probability of 0.005. Figure 18 plots the false negative rates with a malicious

router which both drops and modifies packets with a probability of 0.005, respectively. We can see that the sketch-based approach is effective in detecting packet modification attacks, since by modifying packets the malicious router is detected faster in Figure 18 than in Figure 17.

IX. Related Work and Discussion

Realizing its importance, researchers have recently proposed several approaches for network fault localization. As aforementioned, the known secure FL protocols are all path-based, failing to support dynamic routing paths, requiring per-path state at routers, and incurring per-source key sharing and management. Besides these fundamental limitations, we show that most FL protocols also suffer either from security vulnerabilities or high protocol overhead.

For example, WATCHERS [16], [22], AudIt [8] and Fatih [29] implement the traffic summaries using either counters or Bloom Filters [15] with no secret keys, thus remaining vulnerable to packet modification attacks as Section III-C shows.

Both ODSBR [12], [13] and Secure Traceroute [30] activate FL only when the end-to-end packet loss rate exceeds a certain threshold. However, a malicious node can safely drop packets when FL is not activated, and behave "normally" when FL is invoked. In addition, ODSBR does not consider natural packet loss, which can make the algorithm either not converge or incur high false positives by incriminating benign links.

Liu et al. propose enabling two-hop-away routers in the path to monitor each other [26] by using 2-hop acknowledgment packets. However, such a 2-hop-based detection scheme is vulnerable to colluding neighboring routers. Similarly, both Watchdog [28] and Catch [27] can identify and isolate malicious routers for wireless ad hoc networks, where a sender S verifies if the next-hop node f_i indeed forwards S's packets by *promiscuously* listening to f_i's transmission. Both Watchdog and Catch are vulnerable to collusion attacks, where a malicious node f_m drops the packets of a remote sender S (which is out of the promiscuous listening range of f_m) while the colluding neighbors in the promiscuous listening range of f_m intentionally do not report the packet dropping behavior of f_m.

Among the known secure proposals, the protocol due to Avramopoulos et al. [11] incurs high computational and communication overhead, because it requires acknowledgments from all routers in the path, and requires multiple digital signature generation and verification operations for *each* data packet. Recently proposed PAAI-1 [34], Statistical FL [14], and ShortMAC [36] all require stable routing paths and per-path state at routers. TrueNet [35] leverages trusted computing to achieve FL with constant small router state. However, TrueNet requires special hardware support such as a TPM.

X. Conclusion and Future Work

In this paper, we first raise the awareness of achieving a *practical* and *scalable* network fault localization protocol that can cope with dynamic traffic patterns and routing paths with constant, small router state. After identifying the fundamental limitations of previous FL protocols which are all path-based, we explore a neighborhood-based FL approach; we also propose DynaFL, which utilizes delayed key disclosure, a novel technique that enables secure yet efficient checking of packet content conservation.

While existing path-based FL protocols aim to identify a specific faulty *link* (if any), DynaFL localizes data-plane faults to a coarser-grained 1-hop neighborhood, to achieve four distinct advantages. First, DynaFL does not require any minimum duration time of paths or flows in order to detect data-plane faults as path-based FL protocols do. Thus, DynaFL can fully cope with short-lived flows which are popularly seen in modern networks. Second, in DynaFL, a source node does not need to know the exact outgoing path, unlike path-based FL protocols. Hence, DynaFL can support agile (e.g., packet-level) load balancing such as VL2 routing [20] for datacenter networks. Third, a DynaFL router only needs around 4MB per-neighbor state based on our classic Sketch implementation, while a router in a path-based FL protocol requires per-path state. Finally, a DynaFL router only maintains a single secret key shared with the AC, while a router in a path-based FL protocol needs to manage 100 to 10000 secret keys in measured ISP topologies.

DynaFL focuses mainly on unicast communication, while multicast and broadcast communication may cause the detection of "packet injection", since a packet may be "benignly" duplicated during the transmission. As future work, we plan to deal with multicast and broadcast scenarios.

XI. Acknowledgments

The authors gratefully thank Hsu-Chun Hsiao and Patrick Tague for constructive discussions and insightful suggestions, and the anonymous reviewers for their valuable feedback. This research was supported by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389, W911NF-09-1-0273, and MURI W 911 NF 0710287 from the Army Research Office, and by support from NSF under award CNS-1040801. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, NSF or the U.S. Government or any of its agencies.

References

[1] Arbor networks: Infrastructure security survey. http://www.arbornetworks.com/sp_security_report.php.

[2] Cisco security hole a whopper. http://www.wired.com/politics/security/news/2005/07/68328.

[3] Symantec warns of router compromise. http://www.routersusa.com/symantec-warns-of-router-compromise-2.html.

[4] This project has benefited from the use of measurement data collected on the internet2 network as part of the internet2 observatory project. http://netflow.internet2.edu/.

[5] D. Achlioptas. Database-friendly random projections. In *Proceedings of PODS*.

[6] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *STOC*, 1996.

[7] X. Ao. Report on dimacs workshop on large-scale internet attacks. http://dimacs.rutgers.edu/Workshops/Attacks/internet-attack-9-03.pdf.

[8] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and delay accountability for the Internet. In *IEEE ICNP*, 2007.

[9] K. Argyraki, P. Maniatis, and A. Singla. Verifiable network-performance measurements. In *ACM CoNext*, 2010.

[10] B. Augustin, T. Friedman, and R. Teixeira. Measuring load-balanced paths in the internet. In *ACM IMC*, 2007.

[11] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. In *IEEE Infocom*, 2004.

[12] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens. ODSBR: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks. *ACM Trans Inform. Syst. Secur*, 2008.

[13] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *ACM WiSe*, 2002.

[14] B. Barak, S. Goldberg, and D. Xiao. Protocols and lower bounds for failure localization in the Internet. In *EUROCRYPT*, 2008.

[15] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[16] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *IEEE Symposium on Security and Privacy*, May 1998.

[17] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 2004.

[18] I. Cunha, R. Teixeira, and C. Diot. Measuring and characterizing end-to-end route dynamics in the presence of load balancing. In *PAM*, 2011.

[19] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-quality monitoring in the presence of adversaries. In *Proceedings of SIGMETRICS*, 2008.

[20] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In *ACM SIGCOMM*, 2009.

[21] K. J. Houle, G. M. Weaver, N. Long, and R. Thomas. Trends in denial of service attack technology. Technical report, CERT Coordination Center.

[22] J. R. Hughes, T. Aura, and M. Bishop. Using conservation of flow as a security mechanism in network protocols. In *IEEE Symposium on Security and Privacy*, 2000.

[23] J.Eidson and K. Lee. IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In Sensors for Industry Conference, 2nd ISA/IEEE., 2002.

[24] M. Kodialam, T. V. Lakshman, and S. Sengupta. Efficient and robust routing of highly variable traffic. In *In Proceedings of ACM HotNets*, 2004.

[25] C. Labovitz, A. Ahuja, and M. Bailey. Shining light on dark address space. Technical report, Arbor Networks.

[26] K. Liu, J. Deng, P. K. Varshney, and K. Balakrishnan. An acknowledgement-based approach for the detection of routing misbehavior in MANETs. *IEEE Transactions on Mobile Computing*, 2007.

[27] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining cooperation in multi-hop wireless networks. In *Usenix NSDI*, 2005.

[28] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *ACM Mobicom*, 2000.

[29] A. T. Mizrak, Y. chung Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and isolating malicious routers. In *IEEE Transactions on Dependable and Secure Computing*, 2005.

[30] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Computer Communication Review (CCR)*, 33(1):77–82, 2003.

[31] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *ACM SIGCOMM*, 2002.

[32] R. Thomas. ISP security BOF, nanog 28. http://www.nanog.org/mtg-0306/pdf/thomas.pdf.

[33] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. *SODA*, 2004.

[34] X. Zhang, A. Jain, and A. Perrig. Packet-dropping adversary identification for data plane security. In *ACM CoNext*, 2008.

[35] X. Zhang, Z. Zhou, G. Hasker, A. Perrig, and V. Gligor. Network fault localization with small TCB. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, 2011.

[36] X. Zhang, Z. Zhou, H.-C. Hsiao, A. Perrig, and P. Tague. Shortmac: Efficient data plane fault localization. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2012.

[37] R. Zhang-shen and N. Mckeown. Designing a predictable internet backbone with valiant load-balancing. In *in IWQoS*, 2005.