High-Speed Inter-domain Fault Localization

Cristina Basescu*, Yue-Hsun Lin†, Haoming Zhang‡, Adrian Perrig*
*Department of Computer Science, ETH Zurich, {cba,adrian.perrig}@inf.ethz.ch
†Samsung Research America, yuehhsun.lin@samsung.com
‡Carnegie Mellon University, haoming@cmu.edu

Abstract—**Data-plane fault localization enhances network availability and reliability by enabling localization and circumvention of malicious entities on a network path. Algorithms for data-plane fault localization exist for intra-domain settings, however, the per-flow or per-source state required at intermediate routers makes them prohibitively expensive in *inter-domain* settings. We present Faultprints, the first secure data-plane fault localization protocol that is practical for inter-domain settings. Faultprints enables a source to precisely localize malicious network links that drop, delay, or modify packets. We implemented an efficient version of Faultprints on a software router by taking advantage of the parallelism in the AES-NI module of Intel CPUs. Our evaluation on real-world traffic shows fast forwarding on a commodity server at 116.95 Gbps out of 120 Gbps capacity, and a goodput of 94 Gbps. Additionally, Faultprints achieves a high failure localization rate, while incurring a low communication overhead.**

Index Terms—**data plane security; secure fault localization; inter-domain communication; network reliability**

I. Introduction

Availability is an important property to achieve in the Internet: critical infrastructure and services require reliable data delivery. Yet *malicious and misconfigured entities in the network compromise availability through data-plane attacks: dropping, delaying, or modifying traffic*. Localization and circumvention of such malicious nodes on a network path enhances availability, and deters adversarial nodes through the threat of identifying them.

There are numerous attack examples that compromise network availability: i) First, a malicious Autonomous System (AS) may drop, delay, or modify packets, e.g., for censorship purposes. Even when the AS is not malicious, the AS may have compromised routers, in an attempt to disrupt communication; or the AS may experience configuration errors. Indeed, according to a 2014 worldwide survey [2], network infrastructure attacks continue to account for 17% of the attack target mix, for the second year in a row. Infrastructure outages caused by failures or misconfiguration rank third amongst most significant operational threats, experienced by 53% of the service providers. ii) ASes may engage in anti-competitive behavior. In May 2014, Level 3 Communications and Netflix accused five unnamed US ISPs of congesting their interconnection points to force content providers to reserve more ports, thus increasing the ISPs' revenue [1]. Congestion leads to packet drop, and to a decrease in service availability. iii) Another attack is traffic modification by government operations, as The Guardian reports about the NSA [5]. Such traffic modification can manifest through traffic delay along the intended path, through packet modification, or through traffic rerouting, thus absence of the packets along the intended path. For all these cases, secure fault localization enables localization of the problem, even when malicious entities attempt to hide and interfere with localization.

Related Work. The state of the art for localizing faulty network links is to store at each network node a summary of the observed packets, for a determined time period, and then have the nodes *securely* send their summaries for analysis, either on request or periodically. In one line of research, the summaries are sent to the packets' source [10, 11, 19, 38]. This empowers the source, but has the drawback of raising the storage requirements at nodes with an increasing number of sources [10, 19, 38]: namely, existing approaches requires shared keys between each source and each network node, which are stored at the network nodes. To relieve this storage strain, in a second line of research, summaries are sent to fewer entities: either to trusted direct neighbors [37], or to a trusted centralized location [36]. In this setting, fault localization runs collectively on a network of nodes.

Other fault localization approaches have different goals than Faultprints: WATCHERS [14, 20] explicitly targets single domain fault localization, while AudIt [7] and Network Confessional [8] localize packet drop and delay, but do not handle packet modification. Yet, packet modification can be equivalent to dropping, because the destination discards a packet with an invalid checksum. Secure Network Provenance [39] creates a provenance graph, which if performed on a per-packet basis would not scale to a core router.

Unfortunately, all secure fault localization approaches that can handle packet drop, delay, and modification are designed for *intra-domain environments* and do not scale to the size of the Internet for the following reasons. Keeping per-packet state on the router's *fast path*,[1] as well as per-flow, per-source, or per-path state, would render core routers prohibitively expensive, which can additionally be attacked through state exhaustion (AudIt, Network Confessional, and Secure Network Provenance also share this fate); relying on a centralized trusted entity would hinder deployment, as it is challenging for different countries to decide who should own the root of

Haoming Zhang's work was completed at ETH Zurich, and Yue-Hsun Lin's work was completed at Carnegie Mellon University.

[1]A router's *fast path* refers to packet forwarding in hardware, at line rate. In contrast, the *slow path* refers to software packet processing, for instance network management.

trust; and requiring all routers to be equipped with trusted hardware would delay deployment and complicate the router architecture. In addition, the communication overhead should be low, especially in benign cases, and the localization delay should be minimized.

Table I compares several fault localization systems with Faultprints, considering an attacker who drops, delays, and modifies packets. The first two rows present path-based approaches: Secure sketch FL [19], and ShortMAC [38]. They generally have a low communication overhead. ShortMAC achieves a good balance between the communication overhead and a small localization delay, even for 99% localization precision. But these solutions require *per-source storage* (routers store symmetric keys shared with each source), and also *per-flow storage* for traffic summaries, requiring up to 149GB of storage for a 10Gbps link in the case of Secure sketch FL. The storage and symmetric paths issues are addressed in TrueNet [37] and DynaFL [36], which only require small per-neighbor storage. But TrueNet requires trusted hardware, and DynaFL relies on a trusted central entity, both of which are unrealistic in an Internet-scale deployment.

Challenges and Contributions. The design decisions for practical fault localization protocols become apparent when we target a large-scale deployment in inter-domain settings. As Schuchard et al. [32] show, attacks on the control plane of the Internet can exhaust the memory of routers. In "N-square" Distributed Denial of Service (DDoS) attacks, such as Crossfire [22], N attack nodes establish $O(N^2)$ traffic flows between each other. Maintaining per-flow state during such attacks puts a significant strain on core Internet routers.

The main challenge in Faultprints is: *How can we achieve good precision to detect and localize misbehavior, while requiring only per-bandwidth storage on the routers' fast path?* Our insight is that each node on a network path can perform sampling of the packets it observes in a local Bloom filter, according to a function known to the source, but unknown to other nodes. Faultprints nodes require only constant storage because they consistently derive on the fly the key for the sampling function, and store sampled packets only for a limited amount of time. The storage requirement is only 46.8 MB for a router with 120 Gbps throughput. Faultprints can also support asymmetric paths, but achieving weaker security properties. We provide an efficient implementation based on parallelism offered by AES-NI support on commodity Intel CPUs. Our implementation forwards IPv4 Internet traffic (i.e., in the proportion determined by a CAIDA study [16]) on an off-the-shelf PC platform at a throughput of 116.20 Gbps out of 120 Gbps, providing a goodput of 94 Gbps. To the best of our knowledge, Faultprints is the first fault localization protocol that can operate in inter-domain settings.

II. Problem Setting

In this section we discuss the adversary model, describe the properties of our protocol and our assumptions.

A. Adversary model

We assume that an adversary can compromise any number of ASes on a path from source to destination. The ASes under the control of an adversary can drop, delay, and modify any packet they forward, as well as inject packets. However, the adversary cannot eavesdrop or influence traffic on links that are not adjacent to any of its routers, including their drop rate.

Malicious ASes can adapt their strategy in any way, including misbehaving only when they know they cannot be accurately localized – known as a *coward attack* [27]. Attackers can also launch *framing attacks*, where malicious ASes behave such that they incriminate other ASes of adversarial activity. Moreover, adversaries can collude by exchanging their cryptographic secrets, and by sharing information about the packets traversing them.

Malicious ASes and malicious end-hosts can launch denial-of-service (DoS) attacks by misusing Faultprints, e.g., through unnecessary control packets and an excessive number of flows to cause router state inflation. However, defending against traditional network congestion DoS attacks is out of scope for this paper.

B. Protocol properties

We consider an Internet-scale network where a routing protocol directs traffic from a source to a destination through intermediate ASes. Faultprints enables sources to localize the links adjacent to the adversarial nodes on the path to the destination, as localizing the adversarial nodes themselves was shown to be impossible by Barak et al. [11]. Faultprints is particularly suited for long-lived flows, or several short-lived flows sent by a source along the same path.

Under our attacker model, we seek the following goals for an Internet-scale deployment of fault localization:

- **Low, fixed storage at each node.** The storage requirements at each node must allow for Internet-scale deployment, where nodes may switch several Tbps of traffic. As we seek an upper bound on the storage required, each node's state must not depend on the number of transiting flows, nor the number of sources that originate the flows, nor the number of destinations, nor the path length.
- **Near line-rate forwarding speed.** The processing latency, both in the presence and absence of faults, must allow for line-rate forwarding of packets in the fast path.
- **Low communication overhead.** The communication overhead must be low, especially when there are no faults in the network.
- **High fault localization rate.** Using the definition of Zhang et al. [38], the fault localization rate is the number of data packets needed by the protocol to localize a malicious link, for fixed false positive and false negative localization rates.

C. Assumptions

We assume that the source knows an AS ingress/egress router-level symmetric path to the destination. (An AS ingress/egress router-level symmetric path indicates that the

TABLE I: Comparison of the practicality of existing Fault Localization (FL) protocols.

FL scheme	**Assumptions**		**Overhead**		**Practicality**	
	No trusted central entity	**No trusted hardware**	**Router storage per 10 Gbps link (FP: fast path, SP: slow path)**	**Comm. (extra %)**	**Max. eval. throughput**	**Localiz. delay for 99% accuracy (pkts)**
Secure sketch FL	✓	✓	FP: $149.87GB$[1]+$key*\#src$ SP: $timer*\#slowpath\_pkts$[2]	0.0002%	No eval	10^6
ShortMAC	✓	✓	FP: $21B*\#flows$+$key*\#src$ SP: $timer*\#slowpath\_pkts$	0.01%	0.9Gbps	$3.8*10^4$
TrueNet	✓	X	FP: 512KB[3]+ $40B*\#neighbors$	0.0001%[4]	∼1Gbps	10^4
DynaFL	X	✓	FP: $1.95MB$[5]$*\#neighbors$ + 1key	0.002% - 0.012%	No eval	$5*10^4$
Faultprints	✓	✓	FP: ∼46.8MB SP: $(timer$+$ctrl\_pkts)*\#slowpath\_pkts$	3.3%	119.7Gbps	$4*10^3$

[1] Node i stores a sketch of $i^2(10-\log_2 i)*100B$ for each source sending 100pps, e.g., node 3 stores 6.14MB. For a 10Gbps core link, and 512B packets, there are $24.41*10^5$ pps, and node 3 has to store $6.14*24.41*10^3 MB = 149.87GB$.

[2] $slowpath\_pkts$ denotes the protocol's control packets, processed on the router's slow path.

[3] For the less-expensive average fault localization, with the monitoring interval of 10^4 pkts (paper example), 512B packets require 512KB of storage.

[4] One counter report per monitoring interval (e.g., 10^4 packets).

[5] The storage per neighbor is $\lceil \frac{D}{L}+1 \rceil * 20 * \eta * L$, where D is the upper-bound on one-way latency (20ms in the paper), L is the epoch length (20ms), and η is the pps rate: for a 10Gbps link and 512B packets, $\eta = 24.41*10^5$.

sequence of AS ingress and egress routers on the path from the source to the destination is the exact inverse of the path from the destination to the source.) For simplicity of presentation, in the remainder of the paper we abstract the ingress/egress router operations and simply assume that an AS performs an operation. Many paths in today's Internet are asymmetric [21], thus, we assume proposed Internet architectures such as SCION [35] or Pathlet routing [18] which provide packet-carried forwarding state that can be reversed and offer router-level symmetric paths. For ease of presentation, we describe Faultprints in the context of SCION paths. Faultprints could also work in the current Internet, which we discuss in Sections X and XI.

Similar to previous protocols [7, 30, 36], Faultprints requires time synchronization to localize packet delay. We rely on nodes being *loosely* time-synchronized, within 100 milliseconds. In fact, today NTP allows for time synchronization along intercontinental paths within several tens of milliseconds, however, paths with large asymmetric delays can lower accuracy on the order of 100 milliseconds or more [29]. Since Faultprints sources have knowledge of network paths, they can detect such cases and lower their expectations of fine-grained localization of packet delay. We recommend clock synchronization using authenticated NTP version 4, as unauthenticated NTP is vulnerable to attacks [28].

The source and the destination share a symmetric key K_{SD}, e.g., set up using SSL / TLS, IPsec, or SSH. We also require that each AS has a public-private key pair. The ASes' public keys need to be known only to the source and the destination. The source and the destination can verify these public keys using a regular PKI, such as RPKI.

III. Faultprints Overview

We now present a high-level overview of Faultprints, to provide intuition for how we achieve a highly scalable, secure, accurate, and efficient inter-domain fault localization protocol. To catch malicious ASes, Faultprints relies on *deterministic packet sampling*: each AS samples observed packets in a way that is predictable by the source, but unpredictable by all other network entities. Packet sampling yields to the source a complete picture of the transit ASes that drop, delay, and modify packets.

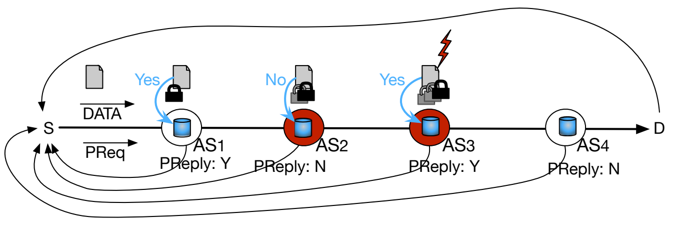

Fig. 1: An example of Faultprints operation with 4 ASes on the forward path. AS_3 is malicious and drops the depicted DATA packet, while the malicious AS_5 may drop AS_4's PREPLY.

In essence, at the beginning of the protocol, each AS$_i$ establishes with the source S a *secret packet sampling key* $K_{AS_i,S}$, enabling the prediction by the source of which packets will be sampled by each AS. An important aspect is that ASes **do not store** $K_{AS_i,S}$, as this would require per-source state; instead, Faultprints uses the DRKey protocol [24] to let AS$_i$ rapidly derive on-the-fly the sampling key $K_{AS_i,S}$ when a packet arrives. This approach offers several important properties: (1) only a single secret is stored for key derivation by AS$_i$, regardless of the number of sources, (2) since the ASes do not know each others' sampling keys, a malicious AS that wants to drop, delay, or modify a packet takes the risk of being detected in case its neighbor AS samples that packet, and (3) the source knows the secret packet sampling keys established with the ASes on the path, enabling the source to predict the sampling behavior of each AS for each packet. Although the actual protocol needs to handle numerous corner cases and attacks, we next present the main challenges and intuitions for Faultprints based on a concrete example.

Consider the setting of Figure 1. The source S communicates with the destination D over 4 intermediate ASes, of which AS_3 is malicious. Faultprints adds to the default data traffic (denoted by DATA) control packets of three types: (i) DACK: destination acknowledgement, (ii) PREQ: probe requests to investigate what happened to a lost DATA packet, and (iii) PREPLY: probe replies carrying reports back to S. Each AS$_i$ decides whether to sample the data packet shown in Figure 1 by computing a thresholding *packet sampling function* that uses $K_{AS_i,S}$. In this example, for the illustrated DATA packet, the sampling triggers at AS_1 and AS_3. The sampling would have triggered at AS_4 as well, had the data packet not been dropped by AS_3. Other DATA packets could be sampled by other ASes, possibly fewer, but overall each AS samples a fixed percentage of the packets it forwards. To achieve highly-efficient sampling while being robust against cheating ASes, Faultprints uses only symmetric cryptographic operations.

When an AS decides to sample a packet, it stores a packet fingerprint into a Bloom filter [13], a space-efficient probabilistic data structure. The fingerprint captures the *packet's presence, as well as the packet's content*, i.e., whether the packet was modified en route. To prevent Bloom filters from filling up, time is discretized into equal-sized units, called *epochs*, and the lifetime of each Bloom filter is constrained to an epoch.

Since the packet in Figure 1 is dropped en route, D does not send a DACK packet to S. Not having received a DACK packet, S decides to send a probe for the data packet. All the ASes reply to the probe, but their reply packets could also be dropped, delayed, as well as modified by malicious ASes. For instance, AS_2 could drop AS_4's probe reply, launching a framing attack against AS_4. To prevent framing attacks, Faultprints' probe replies are indistinguishable from each other based on their origin AS (Section 4.3).

The source receives information about sampled packets in probe replies, sent by each AS. The source checks the sampling information in the replies using the secret packet sampling keys $K_{AS_i,S}$. According to the replies the source receives, AS_4 behaved differently than dictated by its packet sampling function: AS_4 did not sample the packet, although it should have sampled it. Consequently, the source becomes suspicious of the link between AS_3, and AS_4. The source's suspicions sum into *corruption scores* for the ASes neighboring suspicious links, according to the probing results of different packets. It is also possible that malicious ASes try to hide misbehavior by refusing to send probe replies, or by dropping other ASes' probe replies, as it is the case for AS_2. Or they could drop acknowledgements to cause unnecessary probing. To localize such misbehavior, Faultprints uses a probabilistic model to let the source assign *misbehavior probabilities* to ASes on the paths taken by acknowledgements and replies.

Because the sampling is independent and unpredictable to other ASes, and the probe replies are indistinguishable from each other, the misbehaving ASes eventually either have a higher corruption score than their neighbors, or a higher misbehavior probability. The source identifies the link between such neighboring ASes as malicious.

IV. FAULTPRINTS DETAILED DESCRIPTION

We explain the three main phases of Faultprints, namely key setup, data sending, and probing, emphasizing the design decisions that make Faultprints operations fast. The scoring for fault localization is explained in Section V. Throughout the description of the protocol, we consider a source S that sends DATA packets towards a destination D through n intermediate ASes AS_i, and we use the notation listed in Table II.

TABLE II: Notation.

$AuthEnc_K(\cdot)$	Authenticated encryption using symmetric key K
$PRF_K(\cdot)$	Pseudo-random function keyed with K, output in $[0,1]$
$Sig_{PrivK_N}(\cdot)$	Signature using node's N private key
$H(\cdot)$	Cryptographic hash function
$MAC_K(\cdot)$	Message authentication code using key K
$Cst(P)$	Constant part of packet P
SESSIONID	DRKey session identifier
SV_{AS_i}	A local secret value of AS_i
$K_{AS_i,S}$	Secret key between AS_i and source S
K_{SD}	Symmetric key between S and D
$cTime_X$	Current time at entity X
Con_i	Precomputed authenticator for packet contents at AS_i
$Auth_{delay/modify}$	Nested authenticated value indicating packet delay/modification
P_{Sample}	Probability to store any packet in a node's Bloom filter
P_{Probe}	Probability to send a probe for any packet
DATA	Data packet
DACK	Acknowledgement packet (control packet)
PREQ	Probe request packet (control packet)
PREPLY	Probe reply packet (control packet)

A. Key Setup

To start using Faultprints, the source initiates the key setup phase, using the DRKey protocol [24, 34] to share a key with each AS on the path to the destination. These keys are valid for a predetermined period of time, and expired keys need to be updated if the communication session lasts longer. At the beginning of a session σ, the source picks a fresh random public-private key pair as the session key pair $(PK_\sigma, {PK_\sigma}^{-1})$, and then computes a unique session identifier SESSIONID based on the current time at the source, and the session public key (Equation 1). $cTime_S$ prevents replay attacks: ASes drop old packets, based on loose time synchronization. Rather than including a path in the SESSIONID computation, as in the original DRKey protocol, Faultprints extends the concept of a session to cover multiple paths, which may not all be known to the source when the session is created. Nevertheless, the source records which paths are used within the session. This design decision allows a session to persist when paths change. We discuss this aspect in detail in Section XI.

$$\text{SESSIONID} \leftarrow H(cTime_S, PK_\sigma) \tag{1}$$

The source inserts SESSIONID in a key setup packet, and locally stores the mapping between the forward paths and SESSIONID. In addition, $cTime_S$ and PK_σ are inserted by the source in the key setup packet. Each AS_i that receives the key setup packet derives the key shared with the source, using SESSIONID in the packet as the input to an efficient pseudo-random function (PRF), keyed with a secret SV_{AS_i} known only to the AS (Equation 2). Since the AS can efficiently re-derive the key on-the-fly based on the session identifier, the AS does not store any per-host keys, regardless of the number of sending sources.

$$K_{AS_i,S} \leftarrow PRF_{SV_{AS_i}}(\text{SESSIONID}) \tag{2}$$

To allow the source to learn the key, without disclosing it to other entities, AS_i encrypts the key with the session public key in the key setup packet (Equation 3). Since any node could encrypt an arbitrary key with the session public key, AS_i also signs the encrypted key, together with SESSIONID (Equation 4).

$$EncKEY_{AS_i,S} = Enc_{PK_\sigma}(K_{AS_i,S}) \quad (3)$$

$$SignKEY_{AS_i,S} = Sig_{AS_i}(EncKEY_{AS_i,S}, \text{SESSIONID}) \quad (4)$$

The encrypted keys and signatures generated by each AS are accumulated in the key setup packet towards the destination, which sends the key setup packet back to the source. The source learns the shared keys $K_{AS_i,S}$ by decrypting $EncKEY_{AS_i,S}$ using the session private key ${PK_\sigma}^{-1}$. However, if the SESSIONID was maliciously modified on the forward path, some ASes derived incorrect keys. The source detects both SESSIONID and $K_{AS_i,S}$ modification by verifying To check the signatures, the source uses the public key certificates of the ASes that were either added to the key setup message, or retrieved via RPKI.

Secret value rollover. We support nodes to change their secret value once a day, at midnight UTC. A challenge is to ensure a working protocol for flows that operate across a secret value rollover. We use the following approach to support between 24 and 48 hours of key validity without any additional protocol messages or data fields. Specifically, we use the Least Significant Bit (LSB) of the SESSIONID as a selector for which key a node should use to derive the shared secret key. By separating days into "odd" and "even" days, the LSB of the SESSIONID would indicate whether the "odd" day's key (LSB=1) or the "even" day's key (LSB=0) should be used for key derivation. Depending on whether the current day is "odd" or "even", today's or tomorrow's key would be selected. Upon generation of the SESSIONID, the sender can compute the validity period, and if it is shorter than 24 hours it can generate another SESSIONID until the LSB is different from the parity of the current day and thus extend the key lifetime beyond 24 hours. For longer session lifetimes, the SESSIONID could be changed every day and the key setup operation could be repeated. To determine the parity of a day, we can simply fix the parity of a certain day, e.g., 1 January 2016 is an "even" day, which determines the parity of all other days.

Key setup overhead. A source can send multiple flows within the same session, and has the incentive to do so to improve the fault localization speed and accuracy. Therefore, the latency introduced by the key setup on routers ($381\mu s$ per key setup in the DRKey evaluation) is amortized by the traffic forwarded during the lifetime of multiple flows.

B. Data Sending

We now describe the steps that source, intermediate nodes, and destination take for processing DATA packets. We assume the source already performed the DRKey setup, and has the keys $K_{AS_i,S}$.

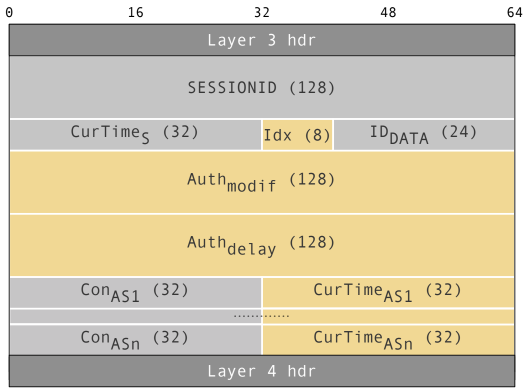

Fig. 2: The Faultprints header for DATA packets, between the network and transport layer headers dark gray). The light colors show the values updated by the ASes. All sized are indicated in bits.

When S sends a DATA packet, it inserts in the header the DRKey session identifier SESSIONID, used by each AS for key derivation, and the source's local timestamp $cTime_S$, used to detect packet delay. In addition, the source computes for each AS_i on the path a field Con_i to authenticate the packet, and binds the packet to the forward path through nested authenticators (Equation 5). $Cst(r)$epresents the constant part of the packet, i.e., excluding the variable IP header fields, such as TTL and checksum. These Con_i fields are used by ASes to efficiently expose packet modification and traffic rerouting attacks. To enable localization of packet modification and delay, the source reserves space in the Faultprints header for the packet modification authenticator, for the packet delay authenticator, and for a timestamp per AS on the path to indicate the receiving time of the packet. Reserving space prevents the packet from increasing in size while traversing the network, thus avoiding packet fragmentation or drop if the packet becomes larger than the Maximum Transfer Unit (MTU) size. The source also inserts $IndexAS$, an AS index initialized to 0 and incremented by each AS, used to access the correct per-AS fields in the packet header. Figure 2 depicts the Faultprints header for DATA packets.

$$\begin{aligned} Con_1 &\leftarrow MAC_{K_{AS_1,S}}(Cst(\text{DATA})), \\ Con_i &\leftarrow MAC_{K_{AS_i,S}}(Cst(\text{DATA})||Con_{i-1}) \end{aligned} \quad (5)$$

Before sending the packet, S computes a DATA packet identifier ID_{DATA}, used to match acknowledgments generated by D with the DATA packets they acknowledge. Since the acknowledgments are end-to-end, S computes ID_{DATA} as a packet authenticator between S and D (using the key K_{SD} shared with D) (Equation 6). ID_{DATA} can be as small as 24 bits, as we explain in the security argument (Section VI). ID_{DATA} is stored at S, as well as inserted in the packet header. S also stores the Con_i fields, which it uses for DATA packet probing.

$$ID_{\text{DATA}} = MAC_{K_{SD}}(Cst(\text{DATA})) \quad (6)$$

In Faultprints, each AS samples incoming DATA packets according to a deterministic sampling function. Sampling at each AS must be unpredictable to other ASes, otherwise an AS could launch coward attacks [27] and drop only the packets that would not increase the localization accuracy, i.e., drop packets that are not sampled by earlier and following ASes. At the same time, sampling performed at ASes must be verifiable by the source against a deterministic ground truth sampling behavior: when the source sends a probe request for an unacknowledged packet, the source must be able to identify sampling misbehavior. For these reasons, the sampling function behavior at AS_i is determined by the key $K_{AS_i,S}$ shared with the source.

Specifically, when receiving the DATA packet, AS_i first derives the key $K_{AS_i,S}$ shared with the source (Equation 2). Then, AS_i checks whether to sample the DATA packet by using a PRF computed over the constant part of the packet (i.e., excluding the variable IP header fields, such as TTL and checksum). Packets are sampled with a predefined sampling probability, P_{Sample}. If the packet is sampled, AS_i stores the packet fingerprint in a local Bloom filter.

Packets are sampled such that the source can later probe for missing packets. However, if ASes use the raw DATA packet contents for sampling, probe packets must include the entire contents of the probed DATA packet – an unnecessary overhead. Instead, nodes base their packet sampling and storage on a much smaller *packet fingerprint* $Auth_{modif,i}$, which reflects the packet contents observed by AS_i (Equation 8). $Auth_{modif,i}$ is a nested authenticator that AS_i computes over the previous packet fingerprint, and the corresponding Con_i field, namely the field precomputed by the source to describe the packet contents for AS_i (Equation 7). AS_i then updates the result in the $Auth_{modif}$ field in the packet header. Through nested authentication, once a packet is modified, the modification is reflected through the absence of expected fingerprints in the Bloom filters of subsequent sampling ASes. In this sense, packet modification has a similar effect as packet drop.

$$\begin{aligned} &Auth_{modif,0} \leftarrow ID_{\text{DATA}}||cTime_S, \\ &Auth_{modif,i} \leftarrow MAC_{K_{AS_i,S}}(Con_i||Auth_{modif,i-1}) \end{aligned} \tag{7}$$

$$PRF_{K_{AS_i,S}}(Auth_{modif,i}) \geq P_{Sample} \tag{8}$$

Basing the sampling decision and storage on the precomputed Con_i values is an important optimization, because computation of the MAC on the packet contents is necessary only when the packet is sampled. Since a malicious AS can tamper with the Con_i values in the packet header, AS_i checks *only for sampled packets* whether Con_i indeed equals the nested MAC of the constant part of the packet (Equation 9). In case of a mismatch, AS_i does not store the wrong fingerprint. It may seem the AS could simply drop the packet, but this action would let colluders located before and after AS_i know that the AS was supposed to sample the packet. Such colluders could simply replay packets that are sampled to avoid localization. Instead, the AS garbles $Auth_{modif}$ in the packet (that to colluders looks just like an update of $Auth_{modif}$), such that packet modification is visible through the chained $Auth_{modif}$ to all subsequent ASes, and forwards the packet. A bogus Con_i

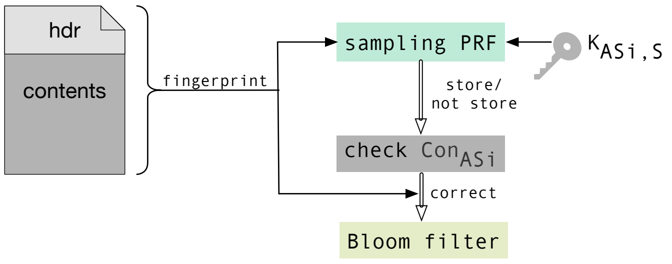

Fig. 3: Faultprints sampling of DATA packets.

can still cause an AS not to sample a packet that, in fact, the AS should have sampled. This case is revealed through Faultprints scoring, the mechanism to localize the malicious entity (Section V). Figure 3 illustrates packet sampling and fingerprint storage.

$$Con_i == MAC_{K_{AS_i,S}}(Cst(\text{DATA})||Con_{i-1}) \tag{9}$$

Alternatively, the source could summarize the packet contents in a hash value, instead of a MAC value for each AS on the forward path. However, this design decision deteriorates fast-path packet forwarding: in Equation 9, computing a hash for each sampled DATA packet is up to 6x more computationally expensive than computing a MAC (Section IX). Since fast-path forwarding speed is of primary importance on Internet core routers, we opted for MACs. It may seem a drawback that the source computes several MACs instead of a single hash value, but for the average path lengths in today's Internet (4.2 ASes [15]), both alternatives have a comparable computational overhead on the source (Section IX). Regarding the DATA packet overhead, we show in the security analysis (Section VI) that each Con_i value can be as small as 32 bits.

For delay localization, each AS also updates its timestamp $cTime_{AS_i}$ in the Faultprints packet header. The timestamp is a nested authenticator computed by the AS, to enable the source to detect timestamp alterations (Equation 10). Each AS updates the nested authenticator $Auth_{delay}$ in the packet header.

$$\begin{aligned} &Auth_{delay,0} \leftarrow ID_{\text{DATA}}||cTime_S, \\ &Auth_{delay,i} \leftarrow MAC_{K_{AS_i,S}}(cTime_{AS_i}||Auth_{delay,i-1}) \end{aligned} \tag{10}$$

DATA packet acknowledgments. When D receives a DATA packet from S, it first recomputes ID_{DATA}, and checks whether the value matches the one in the packet header. In case of a successful match, D creates a DACK packet to indicate successful data packet reception. The acknowledgment data Ack_{info} contains the identifier ID_{DATA}, recomputed by D, which allows the source to easily identify the acknowledged DATA packet (Equation 6)). D also adds to Ack_{info} the delay information $Auth_{delay}$ about the DATA packet accumulated along the forward path (Equation 11). Ack_{info} is authenticated with the key K_{SD} shared with S (Equation 12). Each AS on the return path forwards the acknowledgment.

$$Ack_{info} = ID_{\text{DATA}}||Auth_{delay}||cTime_{AS_1}||..||cTime_{AS_n} \tag{11}$$

$$\text{DACK}[\text{DATA}] = Ack_{info}||MAC_{K_{SD}}(Ack_{info}) \tag{12}$$

Yet attackers could drop, delay, or modify DACK packets, to prevent the information from reaching the source. Faultprints empowers the source to localize such attacks by requiring ASes on the reverse paths to sample DACK[DATA] packets in the Bloom filters, similarly to DATA sampling.

Bloom filter size. One of the goals of Faultprints is to bound the storage requirements at ASes. For this reason, the Bloom filter at each AS has a fixed size, depending on the link bandwidth. As the Bloom filter stores more packet fingerprints, the false positive probability for testing whether a packet is in the Bloom filter increases. To minimize the false positive probability, ASes use the Bloom filter to store packet fingerprints sampled during an epoch. Faultprints uses two alternating Bloom filters to allow sufficient time for packet probing, and to enable Bloom filter erasure. We show later in this section that two Bloom filters are sufficient to guarantee up to one epoch storage.

Even though epochs start at precise moments in time, ASes are not perfectly time synchronized. Therefore, different ASes may find themselves in different epochs, at the same wall clock time. To prevent this problem, ASes establish which Bloom filter to use based on each source's time, using the timestamp $cTime_S$ inserted by the source in the DATA packet. If $cTime_S$ is absent, then either a previous AS deleted the value – which is detected through Con_i and handled as a typical packet modification attack, or the source did not write the value in the packet. In the latter case, ASes simply use the current Bloom filter, which shrinks the probing time for a source with non-standard behavior.

C. Probing

If the source does not receive a valid acknowledgment within a certain timeout, e.g., a round-trip time (RTT), it means an adversary may have dropped, delayed, or modified either the DATA packet or the DACK packet, or in the common case, the packet was simply dropped. An acknowledgment is valid if it is authentic, and its identifier ID_{DATA} computed by the destination equals the identifier stored at the source. The source decides with probability P_{Probe} whether to probe an unacknowledged DATA packet, and the corresponding missing DACK packet. Since the source alone decides whether a certain packet would be probed, all DATA and DACK packet drop, delay, and modification attacks become daring attacks (i.e., an attacker has a non-zero risk of being localized).

DATA packet probing. The source probes a DATA packet to learn which ASes on the forward path observed the DATA packet with the correct content. To probe, the source retrieves ID_{DATA} and Con_i fields for the probed DATA packets, and uses them to assemble a PREQ packet, together with the $Auth_{modif}$ field initialized with $Auth_{modif,0}$, and SESSIONID, needed for key derivation. The source also inserts a counter Ctr unique per packet, which is later used for duplicate detection. A counter of 24 bits is sufficient for PREQ packets arriving at line rate ($7.81 * 10^6$ PREQ packets arrive at line rate, as explained in Section VII). The PREQ packet also contains the timestamp of the DATA packet, $cTime_S$, to identify the epoch corresponding to the DATA packet, as well as timing information for the replies, $ReplyTiming$. Later in this section we explain in detail why $ReplyTiming$ is required. Equation 13 depicts the PREQ packet. The source sends the PREQ packet along the forward path.

$$\begin{aligned}\text{PREQ[DATA]} = &\text{SESSIONID}||cTime_S||IndexAS||Ctr|| \\ &||Con_1||..||Con_n||Auth_{modif}||ReplyTiming\end{aligned} \tag{13}$$

To look up the probed packet in the Bloom filter, each AS_i first derives the key shared with the source (Equation 2), updates the value $Auth_{modif}$ (Equation 7), and checks whether the queried packet was sampled (Equation 8). If sampled, AS_i then queries the Bloom filter corresponding to the epoch of $cTime_S$ for the packet fingerprint $Auth_{modif,i}$. AS_i replies to the source with a bit indicating whether the queried packet fingerprint was stored ($bit_{Auth_{modif}} = 1$ if the packet fingerprint was stored).

If the bit is sent in clear, colluding attackers could gain insight into the location of nodes originating the reply. The attack exploits a corner case, based on the position of colluders on the forward and reverse paths; we explain the attack in detail below, in the paragraph on the indistinguishability of probe replies. Thus, the bit needs to be encrypted. In addition, to expose modifications of the request packet, which may cause an incorrect Bloom filter lookup, as well as reply modifications, AS_i inserts in the reply an authenticator over the request packet and over the encrypted bit (Equation 14).

$$\begin{aligned}\text{PREPLY}_{AS_i}\text{[DATA]} = &Enc_{K_{AS_i,S}}(bit_{Auth_{modif}})|| \\ &||MAC_{K_{AS_i,S}}(Enc_{K_{AS_i,S}}(bit_{Auth_{modif}})||\text{PREQ[DATA]})\end{aligned} \tag{14}$$

In Faultprints, each AS observing a PREQ packet sends a separate PREPLY packet to accommodate potentially asymmetric return paths. We discuss the additional cost of asymmetric paths in Section XI.

DACK packet probing. Similarly to DATA packet probing, the source sends probe requests for a portion of the DACK packets to every AS on the path. To probe acknowledgments, S stores the DACK packets that either contain an ID_{DATA} value the source does not recognize, or whose contents were not authentic according to the MAC computed by the destination (Equation 12). The request packet contains the acknowledgment DACK[DATA]. The replies are similar to PREPLY_{AS_i}[DATA], containing the bit of whether the packet fingerprint was found in the Bloom filter bit, and ID_{DATA} (instead of $cTime_S$) for fast matching between probes and replies.

PREPLY packet indistinguishability. PREPLY packets need to be indistinguishable amongst each other by their origin AS, to prevent malicious ASes to launch *framing attacks* by dropping the replies originating at specific ASes. The contents of probe replies is already indistinguishable regarding its origin AS to any entity that does not know the shared key,

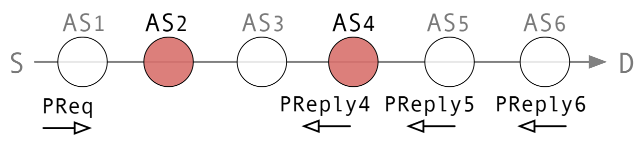

Fig. 4: Colluder nodes can track PREPLY packets, but targeted PREPLY damage is localized.

because the Bloom filter bits are encrypted. [2]

Still, the contents of each PREPLY packet does not change on its way to the source, letting colluders to track a packet along the path. Specifically, in Figure 4, AS_4 observes PREPLY packets from AS_5 and AS_6, and can inform AS_2 about them. But if AS_2 damages only those packets, its colluder AS_4 is localized by Faultprints. Thus, PREPLY indistinguishability leverages its problem setup: fault localization.

Attackers could also use the timing between PREQ and PREPLY packets to infer the number of hops from the AS that sent the PREPLY packet, through a *timing attack*. Faultprints prevents timing attacks on PREPLY packets by requiring each AS_i to return the PREPLY packet after a predetermined amount of time t_i, specified by the source in the PREQ packet: the source encrypts and authenticates in the PREQ packet a time delay for each AS to send the PREPLY packet (*ReplyTiming* in Equation 15). When an AS receives a PREQ packet, it decrypts and checks the authenticity of the timer at index *IndexAS*, then it waits for the indicated time delay before sending the PREPLY packet to the source. Since the source knows the values t_i and the RTT, it can estimate the amount of time to wait for the replies.

To determine the timers, the source must ensure that the delays each AS on the forward path adds to replies cause the RTT to tend towards the average RTT in today's Internet, lower bounded by a low RTT in today's Internet. Such a delay does not allow two colluding ASes to differentiate replies based on their origin: in Figure 4, malicious AS_2 knows the timing of the PREQ packet, but observes replies arriving after an RTT value averaging the RTT in the Internet. Thus, the AS originating the reply could be almost any AS in the Internet, without allowing AS_2 to guess reply origin AS_6, for instance.

For realistic timers, we use the RTT measurements performed by CAIDA for traffic sent from the West US coast to the rest of the world. The measurements exhibit two RTT peaks at 100 ms and 200 ms, with a long tail distribution towards 350 ms [17]. Based on these values, in Faultprints sources randomly choose timer values uniformly distributed from 100 ms to 350 ms, with an average delay of 225 ms.

The complete PREQ packet is depicted in Figure 5. Timers are authenticated and encrypted using AES in GCM mode. Each operation takes the respective Con_i value as additional authenticated data input, to detect attacks that copy *ReplyTiming* values and overwrite them in a different PREQ packet. The output size is 160 bits, corresponding to a 32 bit ciphertext, and a 128 bit authentication tag.

[2]To make PREPLY packets indistinguishable by the origin AS's IP address, PREPLY packets in a session use the IP of the source S.

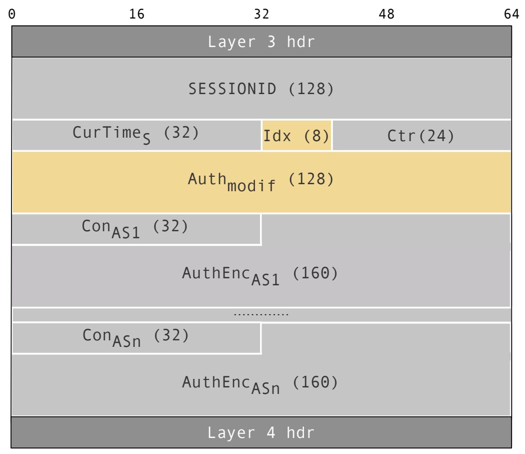

Fig. 5: The Faultprints header for PREQ packets, between layer 3 and layer 4 headers (dark gray). The values computed by the source are depicted in mid gray, and the light colors show the values updated by the ASes. The numbers are in bits.

$$ReplyTiming = AuthEnc_{K_{AS_1,S}}(t_1)||..||AuthEnc_{K_{AS_n,S}}(t_n) \quad (15)$$

D. Bloom filter handling, storage, and packet processing

Processing control packets should not hinder the fast path forwarding of data traffic, thus PREQ and PREPLY packets are processed on the router slow path (Figure 6). On the slow path, ASes store timers, as well as their own PREQ packets, before sending them. Other ASes simply forward the PREPLY packets as soon as they receive them. Moreover, to prevent replay of PREQ packets, which would inflate the number of stored PREPLY packets at ASes, each AS maintains a Bloom filter of PREQ packet counters, hashing the tuple SESSIONID$||Ctr$. When an AS receives a previously observed packet, it simply discards it, without storing any additional information. This Bloom filter is emptied after each epoch.

The PREPLY packets and timers stored at each AS for triggering replies imply per-PREQ packet state (there are as many replies as requests). However, since Faultprints routers keep this state for 225 ms on average, as explained above, and because the number of PREQ packets is naturally bounded by

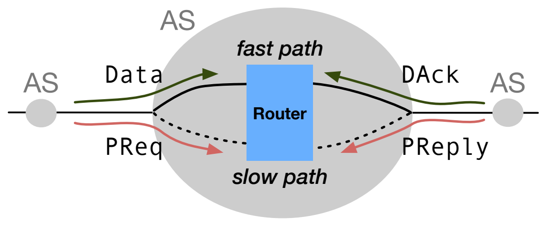

Fig. 6: While transiting an AS, DATA and DACK packets are processed on the fast path, while PREQ and PREPLY packets are processed on the slow path.

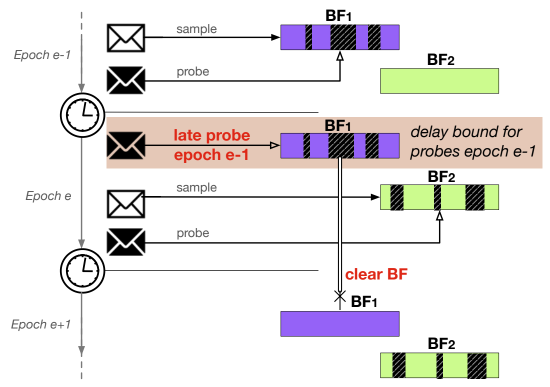

Fig. 7: Packet sampling and probing at an AS. Each AS requires two Bloom filters, to accommodate late probes, and to allow Bloom filter deletion.

the link bandwidth (thus the number of replies is bounded), the AS needs to keep *per-bandwidth state* on the slow path, at most 93.74 MB per 10 Gbps link. We give more details in the upper-bound analysis in Section VII.

Faultprints processes DATA packets on the *fast path*, requiring a *fixed* amount of state. However, actually setting the Bloom filter bits is not performance critical, because Bloom filters are only read on the slow path by control packet processing. Thus, Bloom filter operations are performed on the *slow path*. Each Bloom filter stores packet fingerprints until probing finishes for that epoch. DATA packets stored during an epoch can arrive later than the end of the epoch, due to propagation delay and clock drift between the source and the AS (recall that $cTime_S$ decides the epoch a DATA packet belongs to). Figure 7 shows that probe requests can arrive later than the end of epoch $e-1$, due to clock drift between the source and the ASes, and also because probes for the last packet in epoch $e-1$ incur a delay of at most RTT. To look up such late-probed packets, Bloom filters must be stored for at least an epoch plus an RTT plus the maximum clock drift. This period of time overlaps with the subsequent epoch, thus another Bloom filter is required at the AS, to store fingerprints of the packets arriving during epoch e. With epochs longer than an RTT plus the maximum clock drift (in total about 500 ms), two Bloom filters are sufficient at each AS. In fact, even for epochs as long as 1s, that accommodate a large time synchronization error, Bloom filters incur a relatively low storage overhead (Section VIII).

V. FAULT LOCALIZATION MECHANISM

The source proceeds with localizing an adversarial AS only after it detects packet loss, unusual delay, or modification. An adversary exists on the forward or reverse paths when the end-to-end corruption rate of the packets exceeds a threshold given by the natural drop rate of the links. We give details on how the source computes the end-to-end corruption rate in the theoretical analysis (Section VII). In this section we describe the fault localization mechanism.

Faultprints empowers the source to localize attackers based both on *correct* probe replies, and on the absence of authentic probe replies. We define a *correct probe reply* as a reply that is authentic, which means the reply corresponds to a PREQ packet previously sent by the source, and the response bit is authentic, and was confidentially sent (see Equation 14). The source maintains two values for each AS, to indicate a degree of misbehavior associated with the AS. One value, called **corruption score**, is obtained from correct probe replies: the source compares the real AS sampling behavior against their expected sampling, to infer malicious activity. A corruption score associated with an AS indicates the number of times the source suspects the AS of having misbehaved. A second value is required, because attackers and colluders may attempt to hide misbehavior by dropping, delaying, or modifying other entities' probe replies, and could even avoid sending a probe reply themselves, or send a garbled or delayed probe reply. The second value, called **misbehavior probability**, is a probability computed based on the number of incorrect probe replies (including absent replies) from each AS on the path.

The two score values essentially create a no-escape situation for attackers and colluders: once an attacker misbehaves with respect to DATA and DACK traffic, the attacker (or a colluder) either tries to cover misbehavior, and is detected through misbehavior probabilities, or the attacker sends back correct replies, and is detected through corruption scores. We present the two scoring methods below.

A. Corruption score computation

A high corruption score does not, by itself, necessarily imply misbehavior. Only by comparing corruption scores of AS neighbors can the source flag a link as malicious. Intuitively, a corruption score very different from the neighbors' score signals a different, possibly malicious behavior during Faultprints execution.

The source updates the corruption scores of ASes after it compares the Bloom filter bits in correct probe replies with the expected values. We consider AS_i to be the reply originator. The comparison yields the following possibilities:

RECEIVEDBIT and EXPECTEDBIT have the same value. In this case, AS_i appears honest, and its score remains unchanged.

RECEIVEDBIT $=$ FALSE and EXPECTEDBIT $=$ TRUE. In this case, AS_i did not sample the DATA packet, even though it should have sampled it. Therefore, either AS_i lies, or the packet has been dropped, delayed, or modified on the path before AS_i. The source aims to localize where the packet went missing, as depicted in Figure 8. For this, the source gathers all correct replies, identifies the last AS_x that observed the packet, i.e., replied $(\text{TRUE},\text{TRUE})$, and the first node AS_y that reported the packet as missing, i.e., replied $(\text{FALSE},\text{TRUE})$. AS_y may be the same as AS_i, or before AS_i on the path. Because the packet appears to have disappeared between AS_x and AS_y, the source increments the corruptions scores of these two ASes, and of all ASes in between.

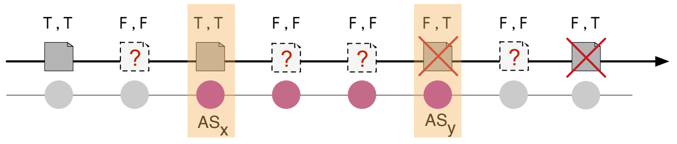

Fig. 8: Faultprints scoring. The letter pairs A,B denote: A if the packet was sampled, B if the packet should have been sampled.

ReceivedBit = True and ExpectedBit = False. In this case, AS_i should not have sampled the packet, but reported to have the packet in its Bloom filter. However, according to Faultprints, AS_i queries its Bloom filter only when the packet is sampled (section IV). Thus, if AS_i is honest, it queries its Bloom filter because the PReq packet was modified such that AS_i believes the probed packet is sampled. But a modified PReq packet causes an incorrect reply packet; this case is handled by misbehavior probabilities, explained later in this section. The only possibility left is that AS_i is malicious, thus the source increments AS_i's corruption score.

B. Misbehavior probability computation

We define as *damaged packets* the PReq and PReply packets that an adversary drops, delays, and modifies. To localize such adversaries, the source keeps per epoch counters dmg_i of the damaged PReply packets from each AS_i on the forward path. At the end of the epoch, the source localizes as malicious the AS_i which maximizes the probability $\mathscr{P}_i$ (Equation 16).

$$P(AS_i\ malicious|dmg_1, dmg_2, .., dmg_n) \overset{notation}{=} \mathscr{P}_i \qquad (16)$$

In the following, we first explain why PReq and PReply drops, delays, and modifications are equivalent for the source w.r.t. the amount of information they offer to localize the misbehavior. This equivalence allows the source to maintain a single counter dmg_i per AS_i for all these attacks. Then, we explain step by step how we compute $\mathscr{P}_i$.

PReq and PReply drop, delay, and modification provide equivalent information to the source. The source observes the effects of PReq packet drop, delay, and modification only indirectly, through PReply packets. In what concerns PReply packets, drop and delay events are indistinguishable by the source, because a delay longer than a large RTT (e.g., 400 ms) becomes a drop event for the source, and a delay shorter than RTT is regular packet arrival. PReply modification can be detected by the source by checking PReply authenticators, however, there is no additional information about the particular AS that may have modified the PReply. Thus, we consider these attacks equivalent, and refer to them as *packet damaging*.

Computation of $\mathscr{P}_i$. We first consider the **return path**. We characterize the state of a PReply packet as correct (Corr) or damaged (Dmg), and model the state of packets traversing ASes using Markov chains. The resulting Markov transition rate matrix denotes the packet's probability to switch from the current state (given by the row name) to the next state (given by the column name). For the transition rate computation, we assume the Markov property that the next state of the packet depends only on its current state, not on the sequence of states that preceded it. The Markov property holds when PReply packets are indistinguishable, and furthermore when each AS acts independently. In the following computation, we assume attackers do not collude. We relax this assumption in Section VI, where we explain that Faultprints scoring localizes colluding attackers one at a time: the first colluder to damage packets, which implicitly acts before other colluders could damage packets (thus independently w.r.t. the observable packet state), is localized first.

Equation 17 gives the transition rate matrices for a PReply packet transiting a benign AS (matrix B) and a malicious AS (matrix D). P_D is the probability of a malicious AS on the return path to damage a PReply packet. For equation simplicity, we assume all malicious ASes have the same P_D value. Nevertheless, the computation is very similar for different P_D values P_{Di}. A benign AS damages a PReply packet with a probability equal to the natural drop rate ρ.

$$B = \begin{array}{c c} & \begin{array}{cc} \text{Dmg} & \text{Corr} \end{array} \\ \begin{array}{c} \text{Dmg} \\ \text{Corr} \end{array} & \begin{pmatrix} 1 & 0 \\ \rho & 1-\rho \end{pmatrix} \end{array}, \quad D = \begin{array}{c c} & \begin{array}{cc} \text{Dmg} & \text{Corr} \end{array} \\ \begin{array}{c} \text{Dmg} \\ \text{Corr} \end{array} & \begin{pmatrix} 1 & 0 \\ P_D & 1-P_D \end{pmatrix} \end{array} \qquad (17)$$

From these matrices, the probability of a correct PReply packet to remain correct after a benign AS is $1-\rho$, and after a malicious AS it is $1-P_D$. Thus, the probability $P(t,r)$ of a correct PReply packet to be damaged after traversing t ASes on the return path, out of which r are malicious, is given by Equation 18. Note that the location of the malicious ASes does not influence the result, because we simply multiply these probabilities, and multiplication is commutative.

$$P(t,r) = 1-(1-\rho)^{t-r}(1-P_D)^r \qquad (18)$$

We now also consider the **forward path**, in addition to the reverse path. We denote by P_Q the probability of a malicious AS on the forward path to damage a PReq packet. The probability $P(t,r,f)$ of a PReply packet being damaged after the corresponding PReq packet traversed f malicious ASes on the forward path, and the PReply packet traversed r malicious ASes on the return path (both paths have length t, as they are symmetric), is given by Equation 19.

$$P(t,r,f) = 1-(1-\rho)^{t-r}(1-P_D)^r(1-P_Q)^f \qquad (19)$$

We now compute $\mathscr{P}_i$ as in Equation 20, where k denotes the number of ASes between the source and the destination. t_i is the number of ASes on AS_i's return path, and $r_{j,i}$ (respectively $f_{j,i}$) is 1 if and only if AS_j is on the return path (respectively the forward path) of AS_i. The fraction in Equation 20 is constant for each AS, therefore we only compute $P(dmg_1, .., dmg_k|AS_i\ mal)$, where *mal* denotes *malicious*. n represents the number of reply packets the source expects to receive. To localize the malicious link, the source finally computes all $\mathscr{P}_{i+1}/\mathscr{P}_i$ and identifies as malicious the link $AS_i \rightarrow AS_{i+1}$ with the highest $\mathscr{P}_i$ (Equation 21). We omit the intermediate steps of the computation.

$$\mathscr{P}_i = \frac{P(AS_i\ mal)}{P(dmg_1,...,dmg_k)} * P(dmg_1,...,dmg_k|AS_i\ mal) \quad (20)$$

$$P(dmg_1,...,dmg_k|AS_i\ mal) = \prod_{j=1}^{k}[\binom{n}{dmg_j} P(t_j,r_{j,i},f_{j,i})^{dmg_j} * \\ * (1-P(t_j,r_{j,i},f_{j,i})^{n-dmg_j})] \quad (21)$$

VI. Security Argument

We argue that Faultprints provides localization of links neighboring adversarial ASes that drop, delay, or modify traffic. We discuss why Faultprints is resilient to framing attacks, and also argue two or more colluding adversaries cannot gain an additional advantage compared to attackers operating independently. Finally, we explain why Faultprints does not create the opportunity for new DoS attacks.

A. DATA *and* DACK *packet damage*

Sampling of DATA and DACK packets is deterministic at each AS based on the shared key between the AS and the source, yet unpredictable to other ASes. Thus, an attacker that *drops and modifies* DATA *and* DACK *packets, and delays* DACK *packets* cannot choose to damage only packets that are not sampled by the next AS. In fact, the next AS samples packets uniformly at random in the space of all possible packets, including the damaged ones. If the attacker inflicts more damage than the natural packet damage, the source randomly picks which unacknowledged packets to probe, and collects Bloom filter bits revealed in PREPLY packets. If PREQ and PREPLY packets are not damaged, the source computes corruption scores and localizes a link adjacent to attacker. We analyze in the next subsection the case when PREQ and PREPLY packets are damaged.

To detect DATA *delay*, the source inspects the onion-authenticated timestamps added to the DATA packets by each AS, and included by the destination in the DACK packet. Onion authentication ensures these timestamps cannot be selectively modified by a malicious party. In case the timestamps in the DACK packet are damaged to hide DATA delay, the source first localizes a link adjacent to the attacker on the return path that damaged the DACK packet.

B. PREQ *and* PREPLY *packet damage*

PREQ packet damage is observed by the source only through PREPLY packet damage, as explained in section V. PREQ *and* PREPLY *modification* are detected using the authenticator in the PREPLY packet. If the PREPLY packet does not arrive at the source because of PREPLY *drop and delay attacks*, PREPLY packet absence increases the misbehavior probabilities of nodes. To escape localization, the attacker thus needs to thwart misbehavior probabilities by target-damaging PREPLY packets originating at chosen ASes. However, since PREPLY packets are indistinguishable by their originating AS, the attacker is unable to target-drop them with a probability better than random.

C. *Framing attacks by independent attackers*

To frame another AS of misbehavior, attackers could collude or act independently. We first analyze an independent attacker, and examine colluding attackers in the next subsection.

Assume the attacker attempts to cause wrong corruption scores, which are based on Bloom filter contents. An attacker acting independently can control traffic only on its outgoing links, with the contents of DATA and DACK traffic reflected in the Bloom filters of random subsequent ASes. However, the attacker cannot control traffic on non-adjacent links to selectively pollute Bloom filters of non-neighboring ASes, in an attempt to alter corruption scores. Nor can the attacker forward only packets that are sampled by non-adjacent ASes, because the attacker does not know the sampling keys. Moreover, these Bloom filters have the capacity to store line-rate traffic, therefore injection attacks cannot pollute them (Section VII-C). As a result, the attacker cannot leverage corruption scores to frame another AS of DATA and DACK packet drop and modification, and of DACK packet delay. In addition, since authenticators in DATA packets are onioned, the attacker cannot selectively change the information inserted by another AS, for instance to frame the AS of DATA packet delay.

Besides altering corruption score computation, the attacker can attempt to cause wrong misbehavior probabilities, which are based on PREQ and PREPLY packets. Similarly to the previous case, an attacker can drop, delay, and modify PREQ packets only on its outgoing links. Since each AS sends a PREPLY packet after observing a PREQ packet, and the PREPLY packet contents is linked to the PREQ packet, the attack on PREQ packets is visible through PREPLY packets. In addition, as explained previously, targeted drop, delay, and modification of PREPLY packets is not possible, because PREPLY packets are indistinguishable based on their origin AS.

D. *Colluding attackers*

Colluding attackers try to escape localization by framing another AS. The more packets of the same session the colluders observe that traverse the framed AS, the more sophisticated attacks they can launch. From this point of view, symmetric paths are the best case for colluders, because they can easily observe all the traffic in a session. The framed AS could be

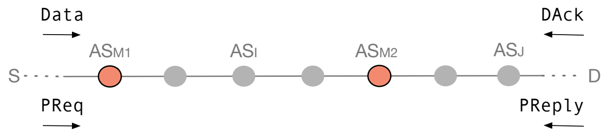

Fig. 9: Colluding attack: AS_{M1} and AS_{M2} try to frame AS_I and AS_J.

situated between colluding ASes, such as AS_I in Figure 9 situated on the forwarding path between malicious ASes AS_{M1} and AS_{M2}, or either before or after the colluders, such as AS_J.

In the case of AS_J, since attackers cannot influence traffic on links other than their outgoing links, their effect on the traffic AS_J and subsequent ASes observe is the same as the effect of an independent attacker. Thus the case reduces to the analysis in the previous subsection.

We now consider the case of AS_I. In a framing attack using *packet modification*, one attacker modifies a packet, and its colluder changes the packet back to its original contents. In case of DATA modification, ASes between AS_{M1} and AS_{M2} do not store in their Bloom filters a fingerprint of the modified packet. Moreover, even though AS_{M2} changes the packet back, it cannot guess the correct value of the nested $Auth_{modif}$, which is a 128-bit MAC. Since this value is used for fingerprinting, ASes following AS_{M2} do not store the packet in their Bloom filters either. Through corruption scores, the source localizes as malicious the link of AS_{M1} towards D. Similarly, PREQ modification would cause incorrect PREQ packets originating at ASes following AS_{M1} (or would cause PREPLY packets not to be sent at all, when the timers in the packet are changed). The attack is detectable through misbehavior probabilities, allowing the source to localize the same link neighboring AS_{M1}. If AS_{M2} modifies DACK packets and AS_{M1} changes them back, the source still receives valid DACK packets, thus the colluders do no harm. Similarly, PREPLY modification by AS_{M2}, followed by a change back by AS_{M1} does not do any harm.

Packet drop has a similar effect: the colluder can re-inject the dropped packets, but she cannot guess the correct value of the nested $Auth_{modif}$. Thus, the ASes between AS_{M1} and AS_{M2} do not store anything in their Bloom filter because they do not observe the packet, nor do the ASes following AS_{M2}, because they do not store the fingerprint of a wrong $Auth_{modif}$ value. PREQ drop and re-inject causes the ASes between AS_{M1} and AS_{M2} not to send PREPLY packets, detectable through misbehavior probabilities, as explained above. PREPLY and DACK drop and re-inject (without delay) do not have any effect.

To collaborate through packet *delay*, AS_{M1} delays DATA packets, and its colluder AS_{M2} changes the delay authenticators so as to frame AS_I. However, since the authenticators are onioned, and the colluders do not have the cryptographic keys of the other nodes, the colluder cannot craft delay authenticators. Also, an attacker could delay DACK, PREQ and PREPLY packets. Its colluder, however, cannot forward faster than line rate the delayed packet. Instead, the colluder can re-inject the delayed packets. For DACK packets, the attack either is harmless, or becomes a packet drop attack. For PREQ delay, the ASes between AS_{M1} and AS_{M2} may send delayed PREPLY packets, which either become incorrect packets when they are delayed for too long (explained above), or do not have any effect. Finally, for PREPLY packet delay, it is again either harmless, or causes incorrect packets from ASes between AS_{M2} and D, detected through misbehavior probabilities.

E. 32 bit Con_i values suffice

The MAC values Con_i computed by the source have only 32 bits. A malicious AS_M could try to change modify the packet contents and craft these authenticators accordingly, for instance only craft the Con_i field on the next AS, but not the other ones, so as to incriminate the outgoing link of the next AS. AS_M has a chance of only $\frac{1}{2^{32}}$ to guess the correct Con_i. For 4000 DATA packets sent by S, rate used in our simulation (Section VIII), even successfully modifying 5% of them (200 packets), which would minimally impact our high localization accuracy, has a very low probability ($\frac{1}{2^{6400}}$).

However, AS_M could inject several packets with different guesses for Con_i, to increase its chances of a correct guess. For instance, to increase the probability to 0.05 (as the above example), AS_M needs to inject about $215 * 10^6$ packets for each DATA packet. We assume the attacker has the computational power to compute these MACs without incurring a high delay.[3] However, even for a single smallest-size packet (64 bytes), this results in 13.74 Gbps additional traffic, and for 200 smallest-size packets – 2748 Gbps additional traffic. Clearly, such an overhead exceeds the capacity of most links.

A similar argument holds for the security of 24-bit ID_{DATA} values. We omit the details due to lack of space.

F. Faultprints does not introduce new DoS attacks

Malicious ASes, as well as malicious sources and destinations could aggressively send control packets (DACK, PREQ, PREPLY) to put a computational strain on routers to increase latency. Regardless of the number of DATA and control packets, the storage on Faultprints routers forwarding 120 Gbps is at most 561.66 MB on the fast path, and at most 1.23 GB on the slow path – both values representing worst case attack scenarios (Section VII). Also, Faultprints operations are very fast, achieving a router throughput 116.2 Gbps for Internet MIX traffic (Section IX).)

Defending against other DoS attacks, possible without a Faultprints deployment, e.g., link flooding, is out of scope.

VII. UPPER BOUND ANALYSIS

We analyze the maximum traffic corruption an adversary can inflict without risking detection and localization. We show in the simulation the chosen false positive rates are practical for an accurate fault localization. Then we compute the upper bound on storage overhead and number of control packets. In our analysis, we consider a forward path with f ASes, and a reverse path of r ASes. We use the parameters in Table III.

[3]Using our evaluation in Figure 15, the attacker spends 5.28 cycles per byte of input, considering the keys are already expanded. Thus the attacker spends about 338 cycles per MAC computation, and $726.7 * 10^8$ for all $215 * 10^6$ MAC values considered. For a delay of at most a second, the attacker requires computation capabilities at a total clock rate of at least 73 GHz.

TABLE III: Faultprints parameters.

ρ_i	Natural packet corruption rate of link L_i (unidirectional)
ρ	The maximum value among all ρ_i
ρ_i^*	*Average* corruption rate (natural and malicious) of link L_i
ψ	End-to-end corruption rate of a path
FP_{Bf}	False positive rate of Bloom filter

A. End-to-end maximum corruption rate

We measure the end-to-end fraction $\psi_{threshold}$ of packets that an adversary can corrupt (drop, delay, and modify) on a path without the source being able to detect the malicious activity. For the forward and reverse paths, the threshold on natural end-to-end traffic corruption is:

$$\psi_{threshold} = 1 - (1-\rho)^f * (1-\rho)^r \tag{22}$$

We denote by $\psi_{observed}$ the actual *observed* end-to-end corruption rate. While $\psi_{observed} \leq \psi_{threshold}$, the source decides there is no significant malicious activity on the path. Considering z ASes on the forward path and reverse paths are malicious, the source computes $\psi_{observed}$ as follows:

$$\psi_{observed} = 1 - \prod_{i=1}^{f+r-z} (1-\rho_i) * \prod_{k=1}^{z} (1-\rho_k^*) \tag{23}$$

To compute $\prod_{k=1}^{z}(1-\rho_k^*)$, the source records the observed fraction of corrupted DATA, DACK, PREQ, and PREPLY packets.

B. Control packets upper bound

Faultprints introduces control packets when sources decide to probe unacknowledged data packets. To probe one DATA packet, the source sends one PREQ packet along the forward path, and receives back at most f PREPLY packets. To probe a DACK packet on the reverse path, the source sends r PREQ packets, and receives at most r PREPLY packets. Assuming the source sends k DATA packets in total, the maximum number of probes is registered when attackers corrupt all k DATA packets or the corresponding DACK packets. In this case, the number of PREPLY packets is:

$$k \cdot P_{Probe} \cdot (f + 1 + 2 * r) \tag{24}$$

As an example, consider forward and reverse paths of 5 ASes (the average AS hop count is 4.2 in the Internet [9]), and P_{Probe} = 10%. The source sends 4000 DATA packets. Then, the upper bound on the number of probes and probe replies is 32000 packets. However, this bound is reached when an adversary corrupts either the DATA or the DACK packet, across *all packets sent by the source*. Faultprints localizes even weaker attackers (0.05% corruption rate) in the same settings with over 95% accuracy, as we show in our simulation (Section VIII). Thus such an attack ceases after 4000 DATA packets with high probability.

DACK packet information can be piggybacked on TCP acknowledgments, thus incurring no increase in the number of packets. For Layer 4 protocols other than TCP, multiple DACK packets can be bundled together in a Maximum Transmission unit (MTU)-sized packet, e.g., a DACK packet totals 100B for a path with 5 ASes, therefore 15 DACK packets can be bundled.

C. Storage upper bound

Fast path storage. We compute the storage upper bound on the AS fast path. On the fast path, Faultprints stores two Bloom filters, as explained in Section IV. The storage requirement depends on the total AS forwarding bandwidth (in Gbps, denoted by Bw), the epoch duration (in seconds, denoted by T), and the false positive rate of the Bloom filter (FP_{Bf}). Although in benign cases also the packet sampling rate (P_{sample}) influences the storage, in the worse case a malicious source may deliberately send packets that are all sampled by a target node. We consider smallest-size packets, 64 bytes in Ethernet, arriving at line rate. This scenario stores the largest number of packets – the worst-case scenario for Faultprints Bloom filters. The total number of packets Nr_{pkt} stored in an epoch is:

$$Nr_{pkt} = (Bw \cdot 10^9/8) bytes/64 bytes \cdot T, \tag{25}$$

and the number of bits of the two Bloom filters is:

$$-2 \cdot Nr_{pkt} \cdot \ln FP_{Bf}/(\ln 2)^2 \tag{26}$$

As an example, we choose a false positive rate of the Bloom filter of 0.01 (we show in the simulation (Section VIII) the chosen false positive rate is practical for an accurate fault localization). Epochs last 1 second, sufficient to allow sources to query Bloom filters within an RTT, as shown in a recent study on large-scale RTT measurements [26]. For an AS forwarding 120 Gbps of minimum-sized packets, the worst case Bloom filter storage, under attack, is 561.66 MB, practical for today's core routers. In case of a 10 Gbps link, the storage is 46.8 MB, as depicted in the introduction in Table I.

Slow path storage. On the slow path, Faultprints stores PREPLY packets, a Bloom filter to detect duplicate PREQ packets, and keeps timers per-PREQ packet, as explained in section IV. We consider a scenario where different sources' PREQ packets traverse the same AS_I. In the worst case, all the traffic AS_I receives consists of PREQ packets, and all these PREQ packets originate at different sources. As before, we consider forward paths of 5 ASes, which require PREQ packets of 192 bytes (160 bytes PREQ header, 28 bytes UDP/IP, 4 bytes padding). On an input link of 10 Gbps, AS_I receives per second at most $6.51 * 10^6$ PREQ packets. Since the AS stores PREPLY packets of 64 bytes each (32 bytes PREPLY header, 28 bytes UDP/IP, 4 bytes padding), for 225 ms on average, the AS stores 93.74 MB of PREPLY packets. To store $6.51 * 10^6$ packets in a Bloom filter, with a small false positive rate of 0.01, each AS requires 7.8 MB. To store a timer between 100 ms and 350 ms, each AS stores a value between 0 and 250, and adds the value 100. Thus, each timer is 1 byte. During 225 ms, AS_I stores 1.46 MB of timers. Thus, in total, AS_I stores 103 MB for a 10 Gbps input link, and 1.23 GB when receiving PREQ packets at a rate of 120 Gbps. Such a storage upper bound is sustainable on the router slow path, for instance in the router's DRAM.

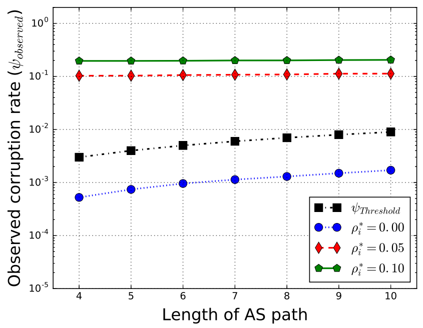

Fig. 10: Theoretical bound rate $\psi_{threshold}$ and observed rate $\psi_{observed}$ for varying malicious link corruption rates ρ_i^* and path lengths.

VIII. Simulation

We evaluate Faultprints through a simulation of the fault localization accuracy against various types of attacks. An adversary tries to maximize her inflicted damage by corrupting (drop, modify, and delay) any type of packet in the protocol. We first validate the maximum end-to-end corruption rate calculated in Section VII-A. We then evaluate the fault localization accuracy using corruption scores and misbehavior probabilities, as described in Section V.

Simulation Setup. Each simulation scenario uses a forward network path consisting of up to 10 ASes, including the source and the destination domains. According to a CAIDA study [15], the vast majority of Internet paths have at most 10 AS hops.

Our simulation scenarios consider one malicious node, at random locations on the forward or reverse path. We set the natural packet loss of each link L_i to $\rho_i = 0.001$. Each result represents an average over 1000 runs.

A. End-to-end maximum corruption rate

In this simulation, a source sends 4000 Data packets to a destination along AS-level paths of various lengths. The adversary corrupts packets with an adjustable rate ρ_i^*, ranging from 0 (no malicious activity in the network, only natural packet drop) to 0.1. Figure 10 depicts the measured end-to-end corruption rate $\psi_{observed}$. Based on this value, the source assesses whether adversaries on the path corrupt packets, as follows: The measured end-to-end corruption rate is always lower than $\psi_{threshold}$ when no adversary is present on the path (case $\rho_i^* = 0$), yet there is natural packet drop. Paths with adversaries where $\rho_i^* \ll \rho_i$ may evade detection, when the exhibited natural packet loss is very small. But these adversaries cause very little damage. By contrast, paths with adversaries with higher ρ_i^* always result in an end-to-end corruption rate $\psi_{observed}$ higher than $\psi_{threshold}$.

B. Localization accuracy

We now evaluate the effect on localization accuracy of the following parameters: the Bloom filter false positive rate, the number of data packets sent by the source, and the probing rate. Localization accuracy is defined as the probability for a source to successfully localize a malicious AS on its routing path, within an epoch, given an upper bound on the number of data packets sent by the source. The simulation shows how the adversary cannot escape detection once it corrupts Data and DAck packets: if the adversary also corrupts PReq and PReply packets, it is localized using misbehavior probabilities, and if the adversary behaves correctly w.r.t. PReq and PReply packets, it is localized using corruption scores. The source cannot know the adversary strategy, but it knows there is at least an adversary on the path, because of a higher end-to-end corruption rate than the threshold. Therefore the source computes both scores, and localizes according to the scoring method that yields the highest difference between an AS (the malicious one) and the others.

Our methodology is to find the minimum number of data packets for Faultprints to achieve localization with high accuracy. Therefore, in each experiment the source sends a fixed, sufficiently-large amount of data packets. The malicious corruption rate of links is $\rho_i^* = 0.05$.

Corruption scores. We first compute the corruption scores of each node, which considers only correct probe replies. Correct probe replies carry wrong information only when there are Bloom filter false positives. Therefore, we vary the false positive rate between 0.01 and 0.04. The probing rate is $P_{Probe} = 0.1$. Figure 12 shows the localization accuracy, denoted by δ, based on the corruption score calculation from collected probe replies. We observe that the Faultprints protocol reaches a high localization accuracy (e.g., $\delta \geq 0.95$) when the total number of data packet sent by source exceeds 3000. On the other hand, δ only slightly decreases when the false positive rate of the Bloom filters increases, which implies the AS storage can be significantly reduced, e.g., corresponding to a Bloom filter false positive rate of 0.04.

However, probe reply packets could be corrupted by natural packet loss. Even with such incomplete information, the corruption scores computed by the source should still allow it to distinguish benign cases from cases with malicious activity, given a known packet loss rate ρ_i. We simulate this scenario with a source sending 4000 packets along paths of varying length. We also vary the malicious link corruption rates. Figure 11 depicts the average of highest corruption score gap computed by the source in relation to the threshold upper bound.[4] Score gaps higher than the threshold indicate malicious activity. The threshold upper bound is given by the malicious activity rate threshold $\psi_{threshold}$ multiplied by the total number of data packets and by P_{Probe}, for various AS path length. The figure shows that corruption score gaps for benign cases are always lower than the threshold upper bound, and the opposite for cases with adversarial activity. Even more, although the deviation of corruption score gaps

[4]The corruption score gap is the difference between the corruption scores of neighboring ASes

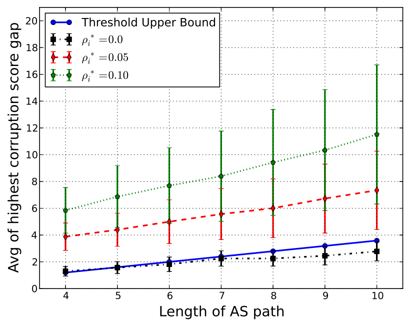

Fig. 11: Average and deviation of highest corruption score gaps computed by source, for varying malicious link corruption rates ρ_i^* and varying path lengths. AS parameters are P_{Probe} = 0.1 and FP_{Bf} = 0.02.

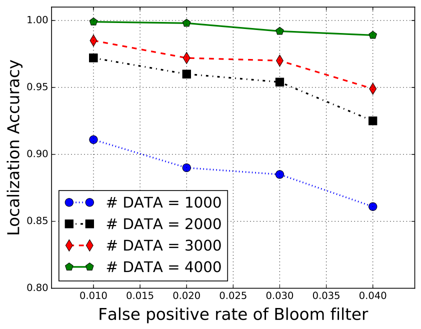

Fig. 12: Localization accuracy of corruption scores, with varying sending rates of DATA packets and false positive rate of Bloom filter.

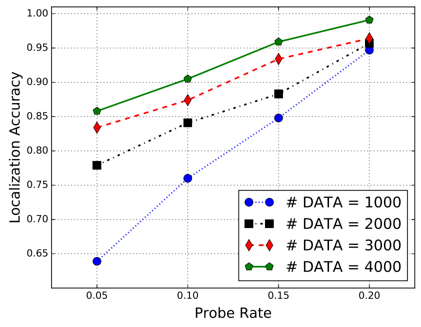

Fig. 13: Localization accuracy of misbehavior probabilities, with varying sending rates of DATA packets and probe rate P_{Probe}.

increases with path length, the source still correctly identifies adversarial activity.

Misbehavior probabilities. An adversary that corrupts PREQ and PREPLY packets tries to prevent the source from calculating corruption scores based on correct PREPLY packets. However, the probabilistic model localizes a malicious AS

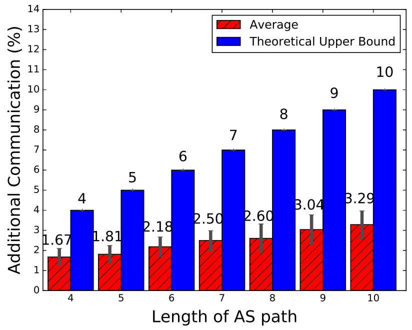

Fig. 14: Communication overhead along various path lengths: theoretical upper bound in plain colors, and average case in pattern colors.

based on absent or corrupt probe replies. Thus, in this scenario, we adjust the probing rate P_{Probe} to collect either more or fewer PREPLY packets. The simulation result is given in Figure 13, and shows the probabilistic model can achieve a high localization, i.e., $\delta \geq 0.95$ with a P_{Probe} of 0.15 and 4000 DATA packets.

We summarize that Faultprints can perform localization with high accuracy when the source either sends enough data packets, or performs more aggressive probing. In fact, even with only 1000 packets, and a P_{Probe} of 0.1, Faultprints achieves over 75% accuracy. This is especially useful for an enterprise sending many of its flows along the same path, for instance between two remote offices. An intermediate AS that tries to sabotage the communication is detected quickly by Faultprints.

C. *Probing overhead*

We evaluate the bandwidth required by PREQ and PREPLY packets (Figure 14). We set P_{Probe} to 0.1, to obtain a good localization accuracy. The result confirms that the total communication overhead increases with the number of transit ASes on the routing path. The average overhead incurred by probing is between 1.6% – 3.2%, with non-significant deviation.

IX. IMPLEMENTATION & EVALUATION

The software router is a commodity server in our testbed, equipped with two 8-core Intel Xeon E5-2680 2.7 GHz CPUs, four banks of 16 GB DDR3 RAM, and six dual-port Intel 82599EB X520-DA2 10GbE NICs. The software router is connected to a high-throughput traffic generator – a Spirent SPT-N4U-220 machine [6]. The traffic generator is both a source and a destination, sending 120 Gbps to the Faultprints software router, which the router forwards back to the generator.

A. *Software Router Implementation*

The software router performs all Faultprints operations when forwarding DATA packets: DRKey key computation, session authenticator computation, packet sampling, and fingerprint storing in the Bloom filter.

To evaluate the performance of Faultprints, we choose Intel DPDK [4] as a packet I/O engine to implement Faultprints data forwarding on the software router. DPDK is a highly-efficient packet I/O engine. Directly sending packets to the user application through a continuous polling mechanism, DPDK is suitable for high-bandwidth traffic settings.

To implement the Bloom filters, we use the open source library libbloom[5]. This library reduces the number of required hashes to a constant: only two hashes are required without loss in any asymptotic false positive rate [25]. The router probabilistically samples packets in its Bloom filter by evaluating the PRF on the $H(Cst(\text{DATA}))$, included in the Faultprints header.

To achieve high-speed cryptographic processing, we use the AES-NI instruction set [3] supported by Intel processors.

CBC-MAC vs. PMAC. Faultprints requires a MAC computation over the entire packet payload (Equation 7), resulting in poor performance for a conventional algorithm such as CBC-MAC, because the operation cannot be parallelized: each block encryption depends on the output of the previous block encryption. To increase the throughput, we implement the MAC functionality using PMAC [12], a parallelizable algorithm. We implement PMAC making full use of the parallelism in the hardware-accelerated AES-NI architecture: each core has its own 128-bit `xmm` registers and AES engine. Our implementation issues simultaneously four block encryptions without any chaining, leading to a speed up factor up to 4x.

We compare PMAC against the CBC-MAC implementation in the Intel AES-NI sample library. We measure the exact clock cycles using the `rdtsc` instruction for both algorithms. The result is summarized in Figure 15. For very small inputs, e.g., 16 or 32 bytes, PMAC performs worse than CBC-MAC. However, for larger inputs, PMAC is up to five times faster than CBC-MAC.

As an example, consider each core processes traffic on one 10Gbps port saturated with 512-byte packets. Using our PMAC implementation, a 2.7GHz Intel Xeon can process up to 1.698 GBytes per second ($\approx$13.56 Gbps), reaching line-rate processing on 10Gbps links. By contrast, CBC-MAC processing supports only up to 4.04 Gbps.

In our evaluation, the minimum packet size is 145 B, as we explain below. For such packet sizes, PMAC always outperforms CBC-MAC, thus we evaluate Faultprints throughput and goodput using only our PMAC implementation.

B. Router throughput and goodput

We generate 120Gbps of traffic of various packet sizes, send the packets to the Faultprints router. For each port connected to the software router, Spirent measures the sending bit rate (TX rate) and the receiving bit rate (RX rate). We define the forwarding efficiency by dividing the RX rate and the TX rate.

We set the sample rate $P_{Sample} = 0.1$, the false positive rate of the Bloom filter = 0.02, and the path length to 5 ASes. We compute the goodput as the amount of useful forwarded data,

[5]https://github.com/jvirkki/libbloom

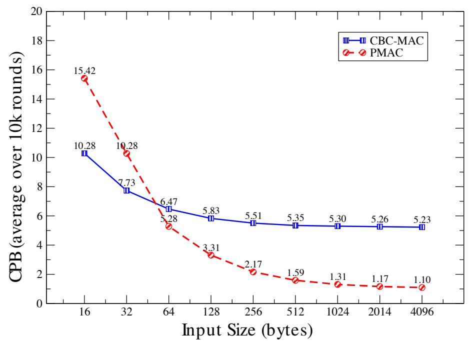

Fig. 15: CPB (Cycle Per Byte) for 1 key expansion and 1 MAC operation using CBC-MAC and PMAC, over various input sizes, averaged over 10^5 computations.

namely the forwarded data without Faultprints's overhead. Faultprints's overhead is composed of the Faultprints header of each DATA packet ($85B + 4 * nr_{AS} = 105B$), the Faultprints header of each DACK packet ($80B + 4 * nr_{AS} = 100B$), and the probing overhead (1.81% for 5 hops in Section VIII-C). Since links are full-duplex, and the DATA and DACK packets travel in opposite directions, measuring the goodput along the forward path accounts only for DATA packet overhead, but not for DACK packet overhead. To capture both overheads, we consider a scenario with two identical DATA flows traversing the link in opposite directions. In this scenario, the overhead of one flow's DACK packets reflects on the goodput of the other flow.

The smallest DATA packet in our evaluation consists of a TCP/IP header of 40 B, and the Faultprints header, in total 145 B. We vary the DATA packet size up to 1500 B. DACK packets always consist of the smallest TCP/IP header and the Faultprints header, in total 140 B.

First we measure the baseline case, when the router performs only DPDK forwarding. The sum of the RX rates on all ports is shown in Figure 16. In the analyzed cases, DPDK degrades for packets of up to 256 bytes: for 145-byte packets, DPDK achieves 97.87 Gbps, and for 256-byte packets, 115.42 Gbps. Packets of at least 512 bytes are forwarded at the line rate, 120 Gbps.

Figure 16 shows that a Faultprints router can process nearly 120Gbps of traffic for DATA packets of 1024 B or larger, and their corresponding DACK packets of 140 B. For smaller DATA packets, the throughput degrades to 94.49 Gbps, 79.44 Gbps, and 52.2 Gbps for packet sizes of 512 B, 256 B, and 145 B, respectively. Compared to throughput, the goodput results peak at 102.86 Gbps, when forwarding 1500 B DATA packets. This is a degradation of 14% of goodput compared to throughput. The goodput degrades more for smaller packets, because the Faultprints header constitutes a higher proportion of the DATA and DACK packets. For DATA packets of 145 B, 256 B, 512 B, and 1024 B, the goodput degradation compared to throughput is 72.45%, 52.65%, 33.01%, and 19.1%.

To understand Faultprints's performance in real-world conditions, we also evaluate the throughput for typical Internet

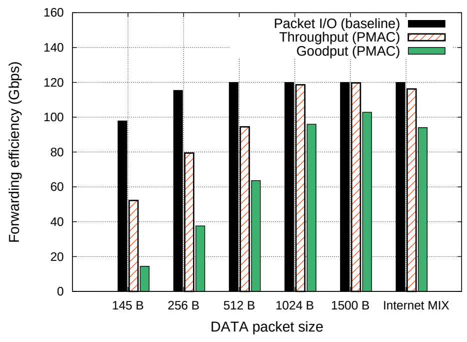

Fig. 16: Data plane throughput and goodput for IPv4 packets of 145 to 1500 bytes, and typical IPv4 Internet traffic [16].

traffic (Internet MIX), using CAIDA's packet size distribution for IPv4 traffic [16]. In this case, Faultprints achieves an overall throughput of 116.2 Gbps, and a goodput of 94 Gbps.

We thus conclude that Faultprints is suitable for high-speed routers.

X. Faultprints over Asymmetric Paths

Faultprints is an efficient and scalable approach for fault localization along Internet symmetric paths. Internet paths, however, are frequently asymmetric [21]. We explain the hurdles Faultprints overcomes towards fault localization along asymmetric paths, and the remaining challenges.

First, the source requires, besides the forward path, knowledge of all AS-level reverse paths to itself originating from each AS on the forward path. Reverse traceroute [23] aims to provide such paths, however, it is not widely deployed. Instead, as we only require AS-level paths, one can use the techniques of Nithyanand et al. [31]. They infer such paths by combining maps of the Internet topology with algorithmic simulations.

The Faultprints protocol described in Section IV requires changes to handle packets traversing return paths: DAck packets, and PReply packets. DAck packets are sampled by ASes on the return path, requiring a key shared between these ASes and the source. This requires minimal changes to the protocol: during key setup, the packet sent by the destination to the source must include SessionId, allowing ASes on the return path to derive a key, and append it to the packet encrypted and signed.

Faultprints already eases support of asymmetric paths, by requiring each AS to send a separate PReply packet. The alternative, as done by Zhang et al. [38], is to bundle replies into a single PReply packet: each AS on the path starts a timer when it observes the PReq packet, and either sends its own PReply packet when the timer expires, or appends its reply bits to a PReply packet received from upstream, onion-authenticates it, and deletes the timer. The advantage of this approach is a smaller number of PReply packets, namely one instead of N, where N is the path length, and a smaller total size of PReply packet contents, namely smaller by N-1

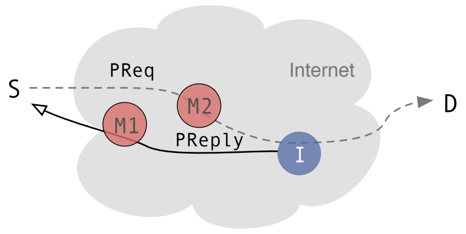

Fig. 17: PReply packets must be delayed by at least an RTT, to defeat a timing attack launched by colluders AS_{M1} and AS_{M2}.

UDP/IP headers ($(N-1)\cdot 28$ B). Also, the approach avoids by construction timing attacks on PReply packets. But their approach works only for symmetric paths. Besides, it still requires the same storage as Faultprints on the router slow path, namely for the timer and the data required to generate the PReq packet.

The most difficult challenge is to ensure PReply packet indistinguishability regarding their origin AS. Without this property, ASes could frame other ASes of misbehavior. PReply packets remain indistinguishable w.r.t. timing attacks, where an attacker on the forward path observes the PReq packet, and informs its colluder on the return path about the timing. Since PReply packets are delayed, the colluder cannot leverage time correlation (Figure 17). Moreover, it is essential for the reply bits to be encrypted. Otherwise, if bits were not encrypted, colluding nodes could launch an attack as follows: the first colluder drops Data packets, causing all subsequent nodes to reply with $bit_{Auth_{modif}} = 0$. When this colluder is not on the return paths taken by probe replies, but the second colluder is located on some or all return paths, the second colluder can simply drop replies containing $bit_{Auth_{modif}} = 0$. For instance, it drops all such replies originating only at non-neighboring nodes, otherwise it could be localized. In this case, a benign node is incriminated.

Yet, PReply indistinguishability along asymmetric return paths depends on the network topology. Indistinguishability requires cover traffic to prevent identification of the correct packet. However, in the extreme case of completely disjoint return paths, each such path carries a single PReply packet towards the source. Unless an AS observes sufficient additional Faultprints traffic *towards the same source*, the AS could identify PReply packets. Ensuring a sufficient number of packets (that are indistinguishable to the PReply packet) sent to the same source is challenging to accomplish.

Recent work of Internet path topology suggest that paths are rarely completely disjoint. For instance, de Vries et al. [33] found that ASes on a forward path have a probability of at least 0.6 to be situated also on the reverse path. These results are encouraging, but quantifying the effect of path asymmetry on PReply indistinguishability in Faultprints remains subject of future work.

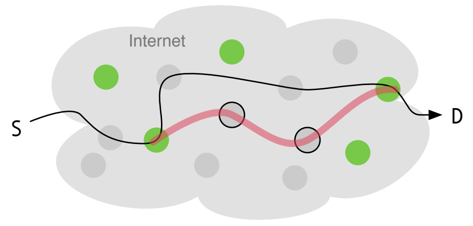

Fig. 18: Early-adopter ASes (dark colors) identify a faulty segment along the thicker line. The source chooses another path, giving incentive to the circled ASes to deploy Faultprints.

XI. Discussion

Incremental deployment. ASes have the incentive to deploy Faultprints to attract more customers, especially those that want high availability. In turn, these ASes would prefer paths through ASes that also deploy Faultprints. Deployment incentives are especially strong in source-controlled path architectures, because end hosts can select paths through ASes that deploy Faultprints.

Even if not all ASes on the path adopt Faultprints, early-adopter ASes allow sources to identify path segments (which may contain a single link or multiple ASes) where malicious activity occurs. Sources with such knowledge can then choose a path that avoids the troublesome segment. Of course, such a decision affects potential benign ASes on the suspicious segment. To maintain their traffic, such benign ASes can deploy Faultprints to keep themselves on the paths chosen by sources. The scenario is depicted in Figure 18, where the thick path segment is localized as malicious.

Path stability. Although paths may change, Faultprints is still effective as long as sufficiently many packets are sent along a path. However, Faultprints is used on paths that inherently exhibit errors, which tend to be unstable. For instance, a highly unreliable link may cause paths to fail often. In this case, DRKey can be performed several times, until every AS on the various paths shares a key with the source. As long as DRKey uses the same session identifier, up to a maximum time period determined by the session timestamp, ASes derive the same keys. Then, regardless of the path used, the source accumulates information about faults, and when enough information is collected, the source localizes the faulty link.

Faultprints's deployment over SCION allows Faultprints to control the path used. In fact, when BGP would issue a route change because of a link failure, leaving the source with no choice to choose a path, SCION allows the source to purposely forward traffic along the faulty path, so that the source itself localizes the faulty link. Path choice empowers a source to keep a session alive until a fault is localized, if the source desires so. Afterwards, the source can choose another path, and learns to avoid paths containing the unreliable link. In the case of BGP, the source still avoids for the moment the malicious AS due to the route change, but never learns whom to avoid in the future (e.g., if the source is an AS, it can use the knowledge to avoid peering with the malicious AS).

Faultprints works in the current Internet on stable paths. Although the paths used by Faultprints tend to exhibit errors, some types of errors do not cause a BGP route change. For instance, if a core AS selectively drops only the traffic of sources from a small country or organization, effectively an example of censorship, the drop rate may be small enough w.r.t. the total amount of traffic forwarded by the AS that its neighbor does not issue a route change. However, the discriminated sources continue to use the path and localize the error.

XII. Conclusion

Despite the importance of malicious AS localization, currently there is no fault localization protocol that is viable in inter-domain settings. Through probabilistic packet sampling at ASes, which is deterministic for the source but unpredictable for other ASes, Faultprints considerably reduces the storage requirements compared to other protocols, while retaining their accuracy. Faultprints is secure against malicious ASes that alone or together with colluders try to cover their misbehavior, or try to frame other ASes. Also, Faultprints can execute efficiently on a commodity PC: the evaluated throughput for Internet MIX traffic is 116.2 Gbps, and the goodput is 94 Gbps. We anticipate that Faultprints brings us closer to localizing and deterring malicious ASes in the Internet.

Acknowledgment

We would like to thank Jun Han for his help during the early stage of this project. We also thank Cristina Nita-Rotaru, Virgil Gligor, Samuel Hitz, Joel Reardon, and the anonymous reviewers for their valuable comments that helped improve the paper.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 617605.

We also gratefully acknowledge support by ETH Zurich, by NSF under award number CNS-1040801, and by Intel for their equipment donation that enabled the high-capacity router experiments.

References

[1] Accusation of ISPs abusing their market power in peering. http://gigaom.com/2014/05/05/level-3-accuses-five-unnamed-us-isps-of-abusing-their-market-power-in-peering.

[2] Arbor security report. https://www.arbornetworks.com/resources/infrastructure-security-report.

[3] Intel AES-NI white paper. http://www.intel.com/content/www/us/en/enterprise-security/enterprise-security-aes-ni-white-paper.html.

[4] Intel Data Plane Development Kit (DPDK). http://dpdk.org/.

[5] NSA tampers with US-made routers. http://www.theguardian.com/books/2014/may/12/glenn-greenwald-nsa-tampers-us-internet-routers-snowden.

[6] Spirent SPT-N4U-220 chassis. http://www.spirent.com/Ethernet_Testing/Platforms/N4U_Chassis.

[7] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and delay accountability for the Internet. In IEEE International Conference on Network Protocols (ICNP), 2007.

[8] K. Argyraki, P. Maniatis, and A. Singla. Verifiable Network-performance Measurements. In Proceedings of ACM CoNEXT, 2010.

[9] V. Asturiano. Update on AS path lengths over time. https://labs.ripe.net/Members/mirjam/update-on-as-path-lengths-over-time.

[10] 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 Transactions on Information and System Security, 2008.

[11] B. Barak, S. Goldberg, and D. Xiao. Protocols and lower bounds for failure localization in the Internet. In Advances in Cryptology (EUROCRYPT), 2008.

[12] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In Advances in Cryptology (EUROCRYPT), 2002.

[13] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970.

[14] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. EEE Network, 1998.

[15] CAIDA. AS path lengths. https://www.caida.org/research/traffic-analysis/fix-west-1998/aspathlengths/.

[16] CAIDA. Packet size distribution comparison between Internet links in 1998 and 2008. http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml.

[17] CAIDA. Round-Trip Time Internet Measurements. https://www.caida.org/research/performance/rtt/walrus0202/.

[18] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. Proceedings of the ACM SIGCOMM 2009 conference on Data communication, 2009.

[19] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-quality monitoring in the presence of adversaries: The secure sketch protocols. IEEE/ACM Transactions on Networking, 2014.

[20] 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 (S&P), 2000.

[21] W. John, M. Dusi, and K. Claffy. Estimating routing symmetry on single links by passive flow measurements. 2010.

[22] M. S. Kang, S. B. Lee, and V. D. Gligor. The Crossfire Attack. In Proceedings of the 2013 IEEE Symposium on Security and Privacy (S&P), 2013.

[23] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. Van Wesep, T. Anderson, and A. Krishnamurthy. Reverse traceroute. In Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI), 2010.

[24] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig. Lightweight source authentication and path validation. In Proceedings of the 2014 ACM Conference on SIGCOMM, 2014.

[25] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: Building a better Bloom filter. Random Structures & Algorithms, 2008.

[26] R. Landa, J. Araujo, R. Clegg, E. Mykoniati, D. Griffin, and M. Rio. The large-scale geography of internet round trip times. In IFIP Networking Conference, 2013.

[27] B. Liu, J. T. Chiang, J. J. Haas, and Y.-C. Hu. Coward attacks in vehicular networks. ACM SIGMOBILE Mobile Computing and Communications Review, 2010.

[28] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the network time protocol. In Proceedings of IS Network and Distributed System Security Symposium (NDSS), 2016.

[29] D. L. Mills. Executive summary: Computer network time synchronization. https://www.eecis.udel.edu/~mills/exec.html.

[30] 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.

[31] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira. Measuring and mitigating as-level adversaries against tor. In Proceedings of IS Network and Distributed System Security Symposium (NDSS), 2016.

[32] M. Schuchard, A. Mohaisen, D. Foo Kune, N. Hopper, Y. Kim, and E. Y. Vasserman. Losing control of the internet: using the data plane to attack the control plane. In Proceedings of the 17th ACM conference on Computer and communications security (CCS), 2010.

[33] W. Vries, J. J. Santanna, A. Sperotto, and A. Pras. ”how asymmetric is the internet?”. In Proceesings of AIMS, 2015.

[34] F. Zhang, L. Jia, C. Basescu, T. H.-J. Kim, Y.-C. Hu, and A. Perrig. Mechanized network origin and path authenticity proofs. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2014.

[35] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. Scion: Scalability, control, and isolation on next-generation networks. In IEEE Symposium on Security and Privacy, 2011.

[36] X. Zhang, C. Lan, and A. Perrig. Secure and scalable fault localization under dynamic traffic patterns. In IEEE Symposium on Security and Privacy (S&P), 2012.

[37] 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.

[38] X. Zhang, Z. Zhou, H.-C. Hsiao, T. H.-J. Kim, A. Perrig, and P. Tague. ShortMAC: Efficient Data-plane Fault Localization. In Proceedings of the Network and Distributed System Security Symposium (NDSS), 2012.

[39] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure Network Provenance. In Proceedings of ACM Symposium on Operating Systems Principles (SOSP), 2011.