Network Transparency for Better Internet Security

Christos Pappas, Taeho Lee, Raphael M. Reischuk, Pawel Szalachowski, and Adrian Perrig

***Abstract*—The lack of transparency for Internet communication prevents effective mitigation of today's security threats: i) Source addresses cannot be trusted and enable untraceable reflection attacks. ii) Malicious communication is opaque to all network entities, except for the receiver; and although ISPs are control points that can stop such attacks, effective detection and mitigation requires information that is available only at the end hosts. We propose TRIS, an architecture that bootstraps transparency for Internet communication. TRIS enables the definition of misbehavior according to the unique requirements of hosts, and then it constructs verifiable evidence of misbehavior. First, hosts express desired traffic properties for incoming traffic; a deviation from these properties signifies misbehavior. Second, ISPs construct verifiable evidence of misbehavior for the traffic they forward. If misbehavior is detected, it can then be proven to the ISPs of the communicating hosts. We implement our architecture on commodity hardware and demonstrate that verifiable proof of misbehavior introduces little overhead with respect to bandwidth and packet processing in the network: our prototype achieves line-rate performance for common packet sizes, saturating a 10 Gbps link with a single CPU core. In addition, we tackle incremental deployment issues and describe interoperability with today's Internet architecture.**

***Index Terms*—Network transparency, accountability, verifiable misbehavior, host policies**

I. Introduction

Security threats and attacks have been increasing at an alarming rate in recent years. Akamai reports a 20% increase in network- and transport-layer attacks (e.g., reflection attacks, SYN flooding) over the recent months; web-application attacks (e.g., HTTP-request flooding) increased by 30% in the same period [1]. Businesses have responded by raising their cyber-security budgets by an average of 24% [2].

We identify three fundamental reasons that hamper the effective mitigation of most attacks in today's Internet. First, the lack of *source accountability* enables attackers to spoof addresses and evade detection mechanisms [3]. Second, the *network layer* allows any host to send packets to any other host without explicit consent from the receiver, and at a transmission rate that is typically determined by the sender [4], [5]. Third, *detection and defense mechanisms are not collocated*: malicious communication is detectable only by the receiving host since only end hosts have knowledge about their available resources; however, ISPs—not end hosts—are the critical control points to mitigate the attacks [6], [7].

We thus propose to enlist ISPs as essential components in dealing with cyber-security threats: i) ISPs are in a strategic position to cost-effectively stop attacks since they are on the path of the malicious traffic [6]. Furthermore, ISPs should be actively involved in mitigating cyber-security threats, according to the recent regulatory push for increased accountability and transparency [8]. ii) ISPs can gain a new source of revenue, build customer loyalty, and reduce customer turnover by selling new security services bundled with their connectivity services [9]; businesses already pay large fees to cloud-based traffic-scrubbing services, which require traffic to be redirected through the cloud infrastructure [10]. However, ISPs are in a weak position to effectively define and detect malicious activity against their customer hosts: the needs and resources of hosts are highly diverse and any automated misbehavior detection would be plagued by false-positives or false-negatives.

To bridge the gap between the knowledge of end hosts and the mitigation capabilities of ISPs, we attempt to answer the following question: Is it possible to account for the unique needs of hosts and prove misbehavior against them to ISPs? More so, can we do it without imposing unrealistic overhead to the communication performance?

The security community has provided multiple solutions to address the three above mentioned problems, but only in isolation: *Source accountability proposals* leverage cryptographic primitives to ensure that the identity of sources in the network can be trusted [3], [11], [12]. *Network capabilities* are authorization tokens that enable the receiver to distribute its downstream bandwidth according to its policies [4], [13], [14]. *Filtering proposals* use filters close to the attacker in order to minimize collateral damage [15], [16], [17]. Although we acknowledge the virtues of these proposals, we aim for an approach that jointly addresses the underlying problems. Our work combines existing primitives (source accountability) and novel ideas (granular host-specific policies and verifiable misbehavior) in an architecture that enhances accountability with respect to host misbehavior.

In this paper, we propose an architecture that generates verifiable proof of misbehavior. Communication is based on explicit consent from the two hosts, and consent is given through sending policies that define misbehavior if they are violated. ISPs construct verifiable evidence of such misbehavior, and a victim can prove to its ISP and the sender's ISP that it has been attacked.

Applications. Verifiable misbehavior serves as a building block to provide better security, rather than as a protection framework per se. For example, the ISP of a misbehaving host can reveal the host's identity in a legal recourse against the host. Also, ISPs can directly contact misbehaving customers and inform them of potentially infected software [18]. The 5-year "Cyber Clean Center" project in Japan—a collection of 76 security concerned ISPs—bore positive results in mitigating botnet activity [19].

The virtues of verifiable misbehavior go beyond the protection of single hosts. The clear line between misbehavior and benign traffic can complement AS-reputation systems [20], [21], [22] by providing input about misbehavior distributions, attack patterns, and attack strategies. For example, it makes

it easier to identify cybercrime-friendly ISPs that host high botnet activity without taking action [23], [24]. At the same time, TRIS incentivizes security-concerned ISPs to take action and enforce stricter security policies to protect their reputation. For example, GoDaddy (ASN 26496), although a legitimate AS, was repeatedly ranked at the bottom of the Hostexploit AS-reputation list [25].

Contributions. This paper proposes a cohesive architecture, TRIS, which leverages *TRansparency for better Internet Security*. Our architecture jointly tackles the three previously mentioned causes that hamper mitigation of today's security threats. It enables a host to provably demonstrate misbehavior and ensures that an innocent host cannot be falsely framed as malicious. Moreover, TRIS satisfies the following additional properties:

- fine-grained host-specific policies,
- efficient border routers (no per-flow state),
- compatibility with today's Internet practices and protocols

We provide a proof-of-concept implementation of a border router that can process typical Internet traffic at line rate. Furthermore, we design and implement a middlebox that performs the host-related functionalities, without requiring changes for the end-host network stack.

II. Overview

We describe TRIS starting with our goals and followed by our central ideas. We present our architecture in two steps. First, we describe a core protocol (Section III) that comes at a low deployment barrier. Then, we describe an extended protocol (Section V) that provides stronger security properties at the cost of a higher deployment barrier.

A. Goals

Our goal is to enable hosts to express policies with the desired properties for incoming traffic and to construct verifiable proof of misbehavior when a policy is violated. Consider two hosts H_1 and H_N that want to establish a communication session (Figure 1). Prior to any communication, hosts authenticate to their Autonomous Systems (ASes)[1] and receive required information to establish sessions with other hosts. The initiating host (H_1 in Figure 1) specifies a policy that the other host (H_N) should respect when sending traffic to H_1. Similarly, H_N specifies a policy that H_1 should respect. After the bidirectional policy specification, the hosts can exchange data.

We want to enable the host H_N to prove to the source and destination ASes (AS_1 and AS_N) that H_1 violated the sending policy. To this end, H_N provides the received packets and the policy as proof of misbehavior. The framework must enable a victim to provably protest, while protecting an innocent host from being unfairly framed by another host. For example, a malicious host can replicate the received packets and provide them as proof of a flooding attack. Thus, our architecture must ensure that malicious actions cannot be concealed and that benign hosts cannot be framed.

[1]In our descriptions we will use the term AS for protocol interactions.

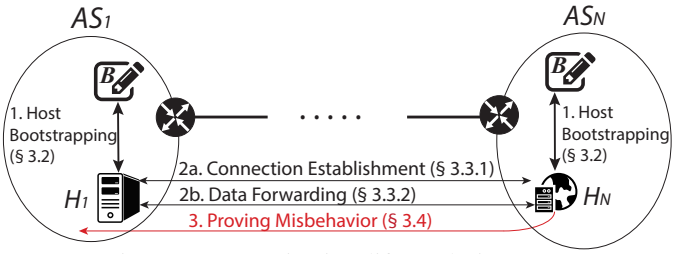

Fig. 1: Communication life cycle in TRIS.

Threat Model. We consider a threat model in which sender and recipient hosts can be malicious. Specifically, we are interested in i) attacks that conceal misbehavior by corrupting the corresponding evidence, and ii) framing attacks that blame an innocent host. We consider malicious ASes in the extended protocol (Section V).

B. Central Ideas

Our solution follows a layered approach, where each step builds on the previous step with the goal of providing verifiable proof of misbehavior.

1) Source Accountability

Source accountability enables the identification of the traffic source; it prevents impersonating other hosts and minting addresses that do not correspond to actual hosts.

In TRIS, addresses are linked with public / private keys, which are used by communicating hosts: i) to mutually authenticate, and ii) to construct communication policies that are publicly verifiable and cannot be repudiated.

ASes act as source accountability agents for their hosts and perform two functions. First, the source AS creates a strong notion of identity for its hosts. It ensures that subscribers use only authenticated addresses to send packets. Thus, every packet that leaves the source AS can be traced back to the host that originated it. Second, the source AS acts as a certificate authority for its hosts. It certifies the binding between the address and the public key of the host.

We argue that ASes are the ideal agents to perform these tasks. They know the identities and network attachment points of their hosts and can therefore enforce source accountability and certify their hosts' information to others.

2) Communication Policies

Communication policies govern the communication between two hosts and express desired properties for each direction of the exchanged traffic (e.g., sending rates, number of flows).

In TRIS, communication policies define misbehavior and enable its detection, in case a host deviates from the policy of its peer. Hosts can specify policies at three granularities: host-based, flow-based, and application-based policies. This enables hosts to achieve flexibility by specifying policies according to their resources and unique needs. Moreover, policies have specific validity periods. Each host notes two timestamps to indicate a starting time (the currently local time at the host) and an ending time. Validity periods allow hosts to update and/or re-negotiate their policies.

Host Policies. Host policies are coarse-grained policies that specify traffic properties for the aggregate traffic between two

hosts. Such policies are negotiated before the hosts exchange data and they can specify the following properties:

i. Maximum number of concurrent flows per source: an adversary may attempt to establish a large number of flows with a victim in order to exhaust the victim's resources and prevent legitimate hosts from establishing connections (e.g., as in SYN flooding attacks).
ii. List of acceptable (or unacceptable) ports: packets that are received at unused ports are typically dropped without consuming resources of the host. However, such packets consume bandwidth of a host's access link and thus, may affect established connections.
iii. Transmission properties: an adversary may launch a volumetric flooding attack by sending traffic at a high bit and/or packet rate. A host can specify any two out of the three Token Bucket (TB) parameters [26] that the sending host should use for shaping its traffic towards the receiving host. For example, a host can specify the committed information rate (CIR) and the maximum committed burst size (CBS) in order to maintain an average rate of CIR during a time interval of T_c seconds. Given two of the TB parameters, the third one can be determined from the equation $CIR = CBS/T_c$.

Flow Policies. Flow policies are more fine-grained than host policies, in that they specify the transmission properties for one specific flow between two hosts. They resemble end-to-end flow control, but enhanced with verifiability properties. Moreover, they have a higher priority than host policies in that a higher/lower sending rate of a flow policy does not count towards the aggregate sending rate defined in the host policy. However, the host policy still constrains the number of flows that can be generated.

We envision flow policies to be used mostly for TCP: the policy exchange requires one Round-Trip-Time (RTT) to let each host express its policy. The three-way handshake in TCP already requires this RTT, thus, the policy specification can be embedded in the TCP handshake to avoid additional latency. However, low-latency UDP services typically want to avoid this overhead; hence, they can fall back to the default host policy and negotiate a flow policy while traffic is already in transit. Hosts that exchange UDP flows have to negotiate a host policy once; this overhead is then amortized over the subsequent UDP flows.

Application Policies. Policies at the application layer provide the highest degree of expressiveness. We highlight two aspects for such policies:

i) Application layer request-rate metrics are needed, since policies that rely on traffic properties (e.g., bit rates or packet rates) are not sufficient. For example, a web server may execute computationally expensive queries for a certain request. Therefore, the server must restrict the request rate, not the packet and/or bit rate of the flow. ii) Attackers can leverage encryption (e.g., SSL/TLS) to conceal the application-layer commands/payload from security middleboxes.[2]

[2]Current defense practices employ TLS termination proxies or require the host to share the encryption keys with a security middlebox; both approaches sacrifice host privacy.

As an example, we will refer to HTTP(S) policies and the corresponding request (GET/POST) rates; it is the most widely used application and HTTP flooding attacks are frequent [27]. Furthermore, the use of end-to-end encryption makes provable misbehavior more challenging. In TRIS, an HTTP(S) server can specify an upper limit for the request rate and burstiness for the client to respect (using TB parameters, but for requests instead of bits). Then, the hosts exchange data by using multi-context encryption [28]: the HTTP request method is encrypted using a different key from the rest of the request. Thus, the key for the HTTP request can be revealed at a later stage, without sacrificing payload privacy (Section III-D).

Specifying Policies. Specifying meaningful policies for inbound traffic will be a challenging task: conservative policies may harm users' quality-of-experience, whereas lenient policies may open up attack vectors. We cannot provide concrete guidelines for specifying meaningful policies, since the needs and resources of receivers can be highly diverse. However, we have engineered our protocols to provide flexibility: first, we envision hosts to specify conservative sending rates for host policies to protect themselves. Then, if higher sending rates are needed, they can be specified through more detailed flow policies. Second, short-lived policy durations, specified through timestamps, enable the receiver to reconsider its policy for a sender, based on the sender's behavior and also based on the receiver's available resources. For example, the receiver can start with a conservative rate for a sender and then upgrade it later. Similarly, a receiver can downgrade the sending rate if demand for a service rises and computational resources are limited.

3) Verifiable Proof of Misbehavior

Verifiable proof of misbehavior enables the victim host to prove to its AS and to the sender's AS that the sender has violated a policy.

A fundamental challenge we address in TRIS is the following: Is it possible to prove host misbehavior with respect to a policy, without imposing impractical requirements for the infrastructure or unrealistic overhead to the communication performance?

Consider a straw-man approach in which the hosts exchange traffic and routers store statistics about the exchanged flows. Obviously, this approach comes with impractical storage requirements since routers forward traffic at several Gbps.

To eliminate excessive storage requirements, an alternate solution can leverage cryptography. Each host can use a public/private key pair and sign every packet with its private key. Thus, misbehavior becomes publicly provable since a host cannot deny having sent a packet that carries its signature. However, this solution comes with a prohibitive processing overhead and an excessive latency due to per-packet public-key operations.

We overcome the above limitations by combining concepts from both approaches: we involve ASes in constructing proof of misbehavior with the use of fast symmetric-key cryptography, but hosts—not ASes— store the proofs. More specifically, our approach works as follows: Hosts insert timestamps in packets. Then, the source and destination ASes verify the

TABLE I: Summary of symbols and notation.

k_{A_i}	Local symmetric secret key for AS_i.
$k_{H_iA_i}$	Shared symmetric key between host H_i and its AS (AS_i).
$k^0_{H_iH_j}$	Shared symmetric key between hosts H_i and H_j at the network layer.
$k^1_{H_iH_j}, k^2_{H_iH_j}$	Shared symmetric keys between hosts H_i and H_j at the application layer, for header and payload encryption, respectively.
$HID_{i\to j}$	Identifier that host H_i must use when sending packets to host H_j.
$K^+_{H_i}/K^-_{H_i}$	Public/private key pair of H_i.
$K^+_{A_i}/K^-_{A_i}$	Public/private key pair of AS_i.
α_i	Network address of H_i.
C_{H_i}	Certificate that binds α_i to $K^+_{H_i}$, signed by the AS of host H_i.
$\sigma_k(M)$	Message M together with a Message Authentication Code using key k.
$\{M\}_{K^-}$	Message M together with a signature using private key K^-.
$E_k(M)/E_k^{-1}(M)$	Symmetric encryption/decryption of message M using key k.
λ	Maximum one-way latency.
ϵ	Maximum clock-synchronization error.

validity of timestamps and insert symmetric cryptographic tokens in every packet. The tokens serve as stateless reminders that the AS forwarded the packets and—together with the timestamps—enable proof of misbehavior when a host complains about a policy violation. In Section IV-B, we show that our scheme achieves multi-Gbps forwarding rates on commodity hardware.

III. Protocol Description

We now present the details of our architecture. We build TRIS on well-established and mature technologies to keep our proposal as practical as possible. Specifically, some of our main design choices are the following:

- Asymmetric cryptography is used for infrequent operations (e.g., flow-policy creation), where non-repudiation is required.
- Symmetric-key cryptographic primitives are used for the rest of the data traffic.
- Storage requirements are shifted to the end hosts, keeping the border routers stateless.

Table I summarizes the notation that we use.

A. Assumptions

- Entities can retrieve and verify the public keys of all ASes. For instance, RPKI [29] enables entities to authenticate certificates that bind Autonomous System Numbers (ASNs) to public keys, based on the RPKI public root keys.
- Cryptographic primitives are secure: signatures and Message Authentication Codes (MACs) cannot be forged, and encryptions cannot be broken.
- Hosts and ASes in the protocol have synchronized clocks with an accuracy of a few milliseconds (e.g., using NTP).

B. Host Bootstrapping

The bootstrapping procedure is performed when a host connects to the network of its AS, prior to any communication session. The goal of this step is twofold: i) bootstrap source accountability by creating a strong notion of host identity, and ii) provide to the hosts all required information to establish communication sessions.

Initially, a host H_i authenticates to its AS (AS_i)[3] using the authentication credentials that were created by the AS during subscription for Internet service.[4] The AS operates a registry service (RS) that performs all required operations for host bootstrapping. Existing protocols that provide confidentiality and integrity [30], [31] can be used for the exchanged messages between the host and the RS.

Host bootstrapping proceeds over the secure channel between the host and the AS's RS as follows: Host H_i generates a public/private key pair $K^+_{H_i}/K^-_{H_i}$ that will be used to sign policies and to generate symmetric keys with other entities.[5] H_i sends its public key $K^+_{H_i}$ to AS_i, so that AS_i will generate a corresponding certificate for the host.

AS_i generates and sends the bootstrapping information to H_i. This information contains an address α_i for the host (e.g., IPv4 or IPv6) and a certificate C_{H_i}, which certifies that the host with the address α_i owns $K^+_{H_i}$. To this end, AS_i creates a certificate that contains α_i, $K^+_{H_i}$, and an expiration time to indicate the validity period; the certificate is signed with the private key of the host's AS (Equation 1). The Autonomous System Number (ASN) of the issuing AS is added, so that entities can fetch the corresponding public key and verify the signature. Moreover, AS_i generates a shared symmetric key $k_{H_iA_i}$ for H_i. The key is used to authenticate H_i's packets, proving to AS_i the ownership of the address α_i.

$$C_{H_i} = \{\alpha_i,\ K^+_{H_i},\ Exp\_time,\ ASN_i\}_{K^-_{A_i}} \tag{1}$$

The bootstrapping procedure is repeated whenever the AS issues a new address to the host (e.g., due to an expiration of the address lease), or whenever the host wants to update its public/private key pair. Furthermore, if a host has multiple public IPs (supported by IPv6), then a public key can be certified for each address or a single certificate can associate all the IP addresses with a single public key.

C. Communication Procedure

After a host has authenticated to its AS and has received the bootstrapping information, it can start communicating with other hosts. Three concepts are fundamental for the correct operation of TRIS: timestamp validation, policy identifiers, and replay detection. We provide a brief overview here and defer the details to Section III-C3.

- **Timestamp Validation:** The function *isTstmpValid()* is evaluated by the source and destination ASes on the

[3]For better readability, the index i indicates that H_i is a host of AS_i, since our descriptions are simplified with one host per source/destination AS.

[4]The credentials can be pre-configured into the host's access device, e.g., DSL modem.

[5]In practice, public/private key pairs used to sign policies and to derive shared keys are different. For ease of exposition, we present them as a single key pair.

forwarded packets to ensure that the reported timestamps are recent, i.e., timestamps do not deviate from the current time beyond a certain threshold.

- **Policy Identifiers:** Hosts may exchange multiple policies (e.g., per-flow or per-application). Therefore, packets must be associated with their corresponding policies to avoid ambiguities when complaining about misbehavior: if two different applications exchange packets, these packets must be associated with either one or the other application to detect policy violations. For now, the function *sendingPol()* abstracts the policy details (e.g., sending rates) and the policy-identifying information.
- **Replay Detection:** Detecting replay attacks is required to prevent innocent hosts from being framed (Section VI). We use a combination of sequence numbers and timestamps to reveal replay attacks when a victim complains; replay detection is not performed during forwarding, thus, it does not degrade forwarding performance.

1) Connection Establishment

Connection establishment triggers a policy exchange in which each host specifies its preferences for the incoming traffic.

Figure 2 depicts the operations for connection establishment and policy exchange.[6] The following steps are necessary only for host policies, but not for flow and application policies: i) Host identifiers (HIDs) are generated only during a host-policy exchange to demultiplex hosts behind NATs. ii) The shared symmetric key $k^0_{H_1 H_N}$, which is used to authenticate messages between two communicating hosts, is computed during a host-policy exchange.

Host identifiers (HIDs) are generated during a host-policy exchange and their role is to demultiplex multiple hosts that share a single IP address (e.g., with NAT devices). Host H_1 generates an identifier $HID_{N\rightarrow 1}$ (Line 1) that host H_N should put into *all* packets destined to H_1; the same step takes places for the reverse direction (Line 14). Since HIDs are host generated they are not globally unique, but are used in conjunction with public addresses to identify hosts. Once the details of policy P_1 have been specified (Lines 2-4), H_1 signs P_1 with its private key $K^-_{H_1}$ (Line 5) so that the policy is publicly verifiable and non-repudiable. Then, H_1 sends the signed policy (msg_1) along with its certificate (C_{H_1}) to its peer host H_N. The sent message (msg_2) is protected with a MAC that is computed with the shared key $k_{H_1 A_1}$ between the host and the AS (Line 6). The MAC ensures that the host's address is valid by proving to its AS that the host owns the correct shared key for the MAC computation.

The border router of the host's AS (AS_1), receives the packet and processes it as follows. First, it verifies that the MAC of the host is valid (Line 7); an invalid MAC indicates a spoofing attempt and the packet is dropped. Then, it checks if H_1 has indicated a valid timestamp for the start of the policy P_1 (Line 8). This check is also performed by the destination AS, AS_N, to ensure that H_1's timestamp is valid and that H_1 does not create a malicious policy that will implicate the peer host H_N. AS_1 inscribes a sequence number in the packet, which is used together with the timestamp to detect replay attacks during connection establishments. The sequence number can be implemented as a simple packet counter (see Section III-C3). Then, AS_1 computes a MAC over the message and the inscribed information with the local secret key k_{A_1} (Line 9); this information is used during the complaint phase and the MAC ensures that modifications will be detected. We highlight that the MAC that was in the incoming packet, computed with the shared key $k_{H_1 A_1}$ with the host, is dropped: AS_1 has verified that the host's address is legitimate and the newly inscribed MAC protects the integrity of the address in the packet.

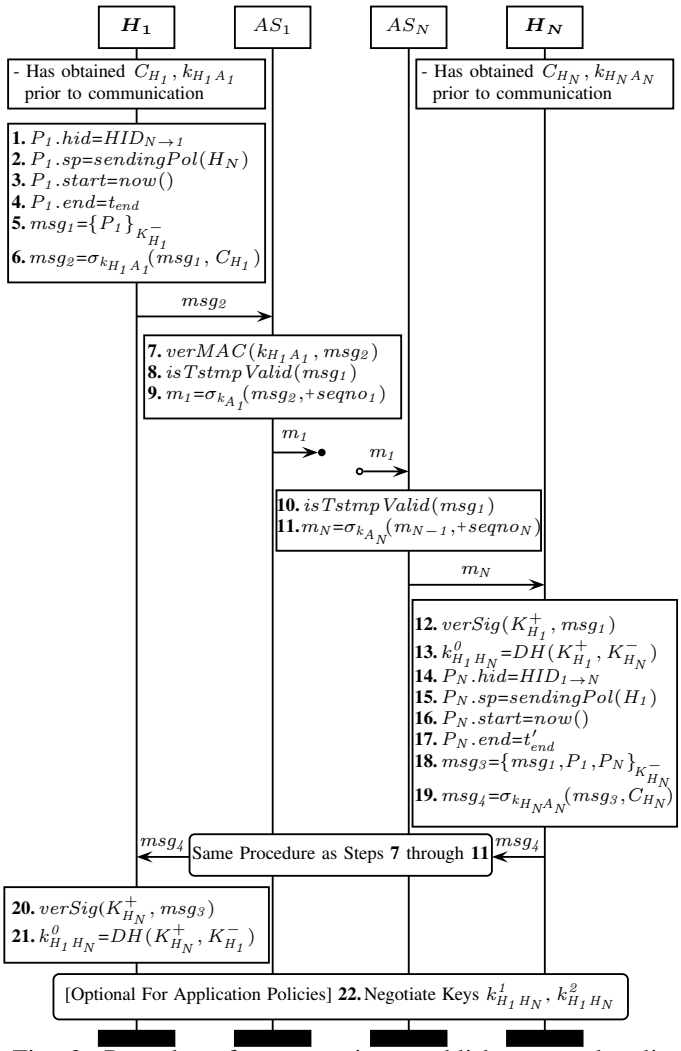

Fig. 2: Procedure for connection establishment and policy exchange.

Upon reception, AS_N, checks the validity of the timestamp (Line 10), inserts a sequence number and a MAC (Line 11), and then forwards the packet to H_N. The sequence number and MAC is required also from AS_N, in case H_N tries to frame H_1 to AS_N by replicating packets and providing them as evidence of a flooding attack.

The receiving host H_N completes the policy exchange. First, it verifies the signature of H_1 (Line 12) using the public key $K^+_{H_1}$ (obtained from C_{H_1}). Then, it generates a shared symmetric key $k^0_{H_1 H_N}$ with H_1 (Line 13); it uses an authenticated Diffie-Hellman key exchange with H_1's public

[6] An application policy can be exchanged at the same time as a flow policy, during connection establishment.

key $K^+_{H_1}$ and its own private key $K^-_{H_N}$. It generates $HID_{1\to N}$ for H_1 to use (Line 14), it provides its own desired properties for the receiving traffic (Line 15), and a validity period (Lines 16-17). H_N signs the received policy P_1 and its policy P_N with its private key (Line 18) and sends it back to H_1. The sent message is protected with a MAC, computed with the shared key between H_N and AS_N (Line 19), as in the forward direction (Line 6). On the reverse path, the same steps take place. In case H_N does not want to receive traffic from H_1, it does not return a policy. H_1 is considered malicious if it sends further traffic without having received a signed policy from H_N. Furthermore, a malicious host that floods a victim with policy-request packets can be detected: the packets have inscribed MACs from the source and destination ASes and indicate misbehavior since only one policy-exchange packet is needed (or a few in case of packet loss).

Upon reception, H_1 verifies the signature of H_N (Line 20) and generates the same symmetric key $k_{H_1H_N}$ (Line 21). Application-layer policies follow the same procedure, with one additional step in case of application-layer encryption. Hosts generate two more shared symmetric keys: one for application-header encryption ($k^1_{H_1H_N}$) and one for application-payload encryption ($k^2_{H_1H_N}$). These keys are generated based on TLS PKI certificates, not based on AS-issued certificates. This procedure is a typical TLS handshake that follows after the TCP handshake.

2) Data Forwarding

After the policy exchange (whether host, flow, or application policy), hosts can send traffic. Figure 3 describes the required operations when two hosts use application-layer encryption.

The sender H_1 encrypts the application-layer header and data (Lines 1-2) using the two separate application-layer keys $k^1_{H_1H_N}, k^2_{H_1H_N}$ that were generated during connection establishment. H_1 creates a packet and adds $HID_{1\to N}$, received from H_N, and the current time as the sending time of the packet (Line 5). Then, it computes a MAC (Line 6) over the entire packet. The MAC is computed with the key $k^0_{H_1H_N}$ that is shared with the host H_N; it enables detection of packet modification en route. H_1 computes another MAC (Line 7) over the entire packet, including the MAC computed with the key $k^0_{H_1H_N}$. The surrounding MAC computed with $k_{H_1A_1}$, proves to AS_1 that the address of H_1 is legitimate since H_1 has generated a valid MAC. The packet is then sent out.

The border router of AS_1 verifies that the MAC with the shared key $k_{H_1A_1}$ is correct (Line 8), and then it verifies the validity of the timestamp (Line 9). Furthermore, it inscribes a sequence number and a MAC over all information using its own local secret k_{A_1} (Line 10). The border router of AS_N verifies the validity of the timestamp (Line 11), inscribes a sequence number and a MAC computed with the AS's local secret k_{A_N} (Line 12), and forwards the packet to H_N. We highlight two points about border routers: i) They perform the same operations on packets, whether they are policy packets or data packets (compare Figure 2 and Figure 3). ii) Processing involves only efficient symmetric-key operations. This design leads to simple and efficient border routers.

H_N receives the packet and verifies that the end-to-end

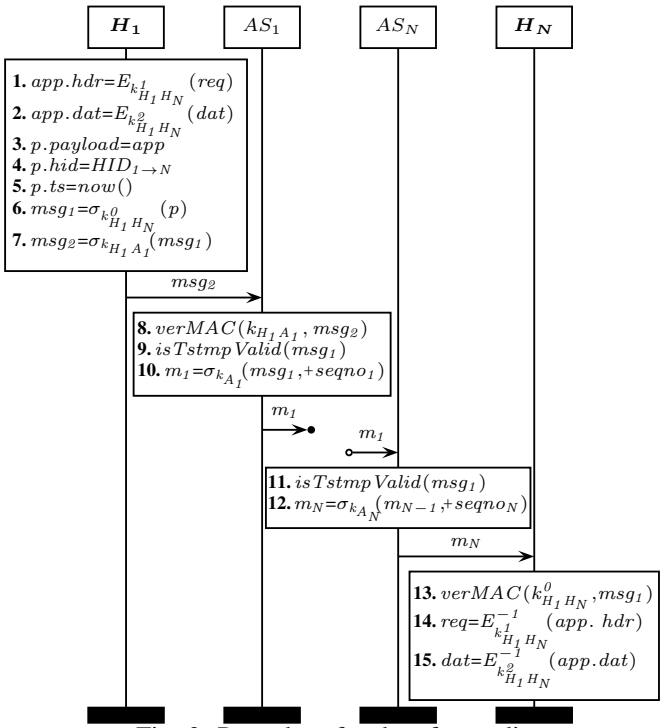

Fig. 3: Procedure for data forwarding.

MAC is correct; it uses the public IP address and $HID_{1\to N}$ in the packet to look up the symmetric key. A malicious host H_1 cannot use an arbitrary $HID_{1\to N}$ to avoid detection: i) Exchanged HIDs have been signed during connection establishment, and ii) the end-to-end MAC verification (Line 13) would fail since HIDs are used to look up the symmetric keys. Then, H_N uses $k^1_{H_1H_N}$ and $k^2_{H_1H_N}$ to decrypt the application layer request and data, respectively (Lines 14-15).

3) Details

We provide now the details of timestamp validation, policy identifiers, and replay detection.

Validation of Timestamps. In TRIS, the host inserts a timestamp in the packet, and source/destination ASes on the path check if the timestamp is recent, i.e., it does not deviate from the current time beyond a certain threshold. Timestamps in packets are expressed with millisecond granularity. Thus, timestamps can be used to identify misbehavior of sending policies, which are expressed at the granularity of seconds.

We now describe the implementation of the function *isTstmpValid(p)*. When an AS receives a packet, it verifies that the time difference between its local view of the time and the indicated time in the packet does not exceed a certain threshold $\lambda + \epsilon$. The factor λ refers to the maximum one-way latency, and ϵ refers to the maximum clock-synchronization error between the clocks of two entities.

We suggest values for these parameters as follows. On one hand, λ must be small so that it prevents a host from reporting delayed timestamps that conceal abnormal sending rates. On the other hand, λ must be large enough to account for the one-way latency and prevent dropping of legitimate packets. Thus, we suggest using twice the value of a maximum one-way latency estimate; according to a recent latency-measurement study [32], we set $\lambda = 200ms$. This value is small enough

to prove misbehavior (since policies are expressed at the granularity of seconds) and large enough to prevent dropping of legitimate packets. The value of ϵ is determined by the accuracy of clock synchronization, and we conservatively set $\epsilon = 100ms$; synchronization over the Internet has a typical accuracy of a few tens of milliseconds [33].

Policy Identifiers. Each policy has an identifier, which must be present in each sent packet. This requirement is important since two hosts may exchange multiple policies. We specify identifiers for each type of policy.

Host policies are the least granular policies; two hosts exchange such a policy for each direction of communication. The policy-identifying information is the (source, destination) public-address tuple together with the host identifiers $HID_{1\rightarrow N}$ and $HID_{N\rightarrow 1}$; this information is present in every data packet. Flow policies are additionally identified by source and destination ports.

For application policies, we use the flow information, but without the source port (referring to the H_1 to H_N direction). This is due to the following constraints: i) We have to couple the application identifier to information that is present in the data packets such as the flow information, and ii) data of one application can be carried by multiple flows, i.e., multiple (source, destination)-port tuples. Therefore, including only the destination port, i.e., the listening port for the application, satisfies both constraints.

The task of specifying policy identifiers is further complicated by practices such as multihoming. In today's Internet, addresses serve both as identifiers and as locators [34]; hence, a host with two addresses is recognized as two different hosts. Consequently, a multihomed host that uses two addresses to communicate with a peer must specify two host policies. Similarly, two application policies are also needed if the underlying flows that carry application data are distributed over both addresses (e.g., as in multipath TCP [35]).

Replay Detection. The goal of our replay-detection mechanism is to detect replay attacks, both for connection establishment packets and for regular data packets. A malicious destination host may frame a source host by providing the same packets multiple times as evidence of misbehavior; similarly, a malicious source host may send packets with the same content multiple times and deny misbehavior by accusing the destination of replaying the received traffic. To detect such attacks, source and destination ASes use sequence numbers. We do not consider packet duplication by the network as a notable concern since it happens only occasionally.

Replayed traffic is only *retroactively* identified when a host complains; it is not dropped in the network at the time of packet forwarding. Dropping replayed packets in the network would introduce an excessive storage overhead for keeping a history of forwarded packets at routers.

Our mechanism builds on the combination of timestamps and sequence numbers. The source host inserts the timestamp, which can be trusted by source/destination ASes in the complaint phase, because a packet is forwarded only if the timestamp is recent. The sequence number of each AS is protected by a MAC, ensuring that modification of sequence numbers is detected. The sequence-number mechanism can be implemented as a simple packet counter. We highlight that this is not a per-flow packet counter and there is no synchronization needed among the border routers of an AS.

D. Proving Misbehavior

Proving misbehavior is an offline procedure that enables a victim host to provably complain about a policy violation. The victim host provides verifiable evidence of the misbehavior to its own AS, which brokers for its customer by sending the evidence to the AS of the misbehaving host. The evidence consists of the signed policy, the received traffic, and optionally the application-layer encryption key $k^1_{H_1H_N}$.

Initially, the victim host provides the evidence to its AS. The AS examines the validity of the evidence, and if valid sends the evidence to the source AS. Similarly, the source AS examines the evidence and acknowledges or rejects the complaint; we describe the steps to examine the evidence at the end of the section.

An approved complaint signifies that the reporting host has indeed been attacked by a malicious or compromised host. The source/destination ASes can then act according to their security policies: destination ASes can install blocking filters to protect their customers and source ASes can notify or disconnect malicious customers.

Evidence Examination. Policy packets that were sent during connection establishment contain information that is used to prove misbehavior (e.g., maximum burst size or number of concurrent flows). The data packets contain information for the actual traffic that was sent. Source and destination ASes examine this evidence as follows.

i. ASes verify the signatures of the two hosts in the policy. The corresponding public keys of the hosts are obtained from the certificates in the policy. The certificates are signed by the host's ASes, whose public keys can be obtained from RPKI.
ii. ASes verify the MAC that they inscribed in the header and examine the sequence numbers to detect replay attacks. Under our threat model, replayed packets indicate that the destination host tries to falsely blame an innocent host. For performance reasons, replay detection is performed only retro-actively, after a victim complains, and not during packet forwarding on the fast path.
iii. ASes check the traffic properties against the policy details. For example, they count the number of concurrent flows and compare with the policy specification. To detect sending-rate violations, they run Token-Bucket based on the packet lengths and the timestamps in the packet headers. For application-layer policies, they use the symmetric key $k^1_{H_1H_N}$ and decrypt the application-layer request. Then, they check for violations, without compromising the privacy of the payload (encrypted with $k^2_{H_1H_N}$).

IV. Implementation & Evaluation

A. Packet Header

Our proposal requires additional information in every sent packet. We describe the length of the additional fields and the

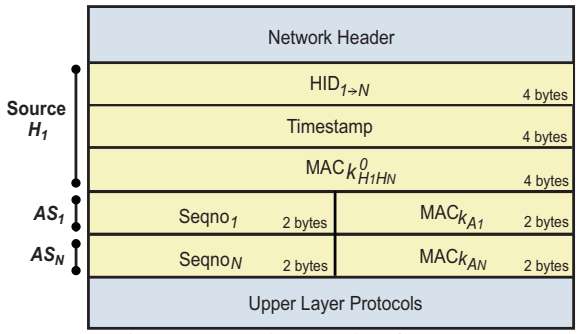

Fig. 4: TRIS data packet format.

format of the packet header; then, we quantify the introduced bandwidth overhead.

In order to moderate the bandwidth overhead, we use the concept of short MACs [36], [37]: the MACs that are inscribed in the packets are considerably shorter than the length of a typical MAC (e.g., 16 bytes for an AES-based MAC). However, this approach does not compromise security for two reasons: i) we care about the collective proof, which is derived from an aggregate of packets (rather than from a single packet), and ii) a malicious host H_N can at best generate random MACs without feedback about their validity since the keys (k_{A_1}, k_{A_N}) that are used to generate MACs are only known to the ASes. While the short MAC length allows an adversary to generate few valid MACs, it is statistically difficult to generate a sufficient number of valid MACs to influence the collective proof; the invalid MACs that are generated as a byproduct will reveal the misbehavior.

Data Packets. Figure 4 shows the format of a data packet. The host identifier is 4 bytes long, which is sufficient to demultiplex hosts of large networks that share a single public address. We allocate 4 bytes for the timestamp that is inscribed by the source host; it can encode a time period of 49 days at the granularity of one millisecond, which is sufficiently large even for long-lived host policies. The end-to-end MAC between the hosts, computed with $k^0_{H_1 H_N}$, is 4 bytes long and protects the whole packet (except for the mutable fields such as the TTL). Note that this MAC is independent from any MAC generated by protocols in higher layers (e.g., TLS). Then, the source and destination ASes inscribe each a 2-byte sequence number and a 2-byte MAC computed over the packet content (including the sequence number). The 2-byte sequence number is long enough to detect replay attacks since it can uniquely identify 65K packets within one millisecond; this renders multiple occurrences of a sequence number per timestamp value suspicious.

To quantify the bandwidth overhead, we look at the traffic statistics of three backbone-link packet traces obtained from CAIDA.[7] In total, each packet carries additional 20 bytes of data in the TRIS header. Furthermore, we note that this space is pre-allocated in the packet by the source host; this ensures that the packet length does not increase en route and packets do not get dropped because of the MTU length. Table II shows the bandwidth overhead for the mean and median packet sizes observed in the three traces.

[7]https://www.caida.org/data/passive/trace_stats/

	Trace 1		Trace 2		Trace 3	
pkt. size	747 B	463 B	906 B	1420 B	691 B	262 B
overhead	2.14%	3.46%	1.77%	1.13%	2.32%	6.11%

TABLE II: Bandwidth overhead for the mean and median packet sizes for 3 CAIDA packet traces.

The introduced bandwidth overhead is considerably low given the provided benefits. For example, the IPv6 header would introduce a 2.65% bandwidth overhead over IPv4, assuming a packet size of 747 bytes.

TRIS incurs a storage overhead for end hosts since they have to store incoming packets as proof of misbehavior in case of policy violations. To get a pessimistic idea of the storage overhead, we have considered a middlebox that stores information for approx. 1500 users [38] and processes all the traffic observed in Trace 1 (Table II). For the peak packet rate of the trace, we find that the middlebox has to store 390 MB every second. We assume a flow duration of 15 minutes after which packets are deleted; 98% of flows are less than 15 minutes long [39]. This yields an overall peak storage requirement of 343 GB that the middlebox must have available, which is well within practical limits for today's hardware; for the actual traffic of an access network and for more realistic flow durations the overhead would be significantly lower.

Policy-Exchange Packets. We describe the format of a policy-exchange packet. For the start and end time of the policy, we allocate 4 bytes. Similarly, 4 bytes are needed for the HID. The signature of the host over the policy requires 64 bytes: we use the Ed25519 SUPERCOP REF10 signature implementation[8], which uses 64-byte signatures (and 32-byte public keys). The certificate of the host C_{H_1} is 108 bytes: a 4-byte address α_1, a 32-byte public key $K^+_{H_i}$, the 4-byte ASN of the issuing AS, and a 4-byte expiration time for the certificate; finally, there is a 64-byte signature over the certificate's information, computed with the source AS's public key. Furthermore, the source adds a 4-byte MAC, computed with $k_{H_1 A_1}$.

The source and destination ASes insert a 2-byte sequence number and a 2-byte MAC, which are computed with the local secret keys k_{A_1} and k_{A_N}, respectively. We have not considered the bandwidth overhead of policy-exchange packets since the overall bandwidth overhead is dominated by data packets.

B. Border Router

Border routers perform different tasks depending on their position on the path (source vs. destination AS). Border routers verify the validity of the timestamp, inscribe a sequence number, and compute a MAC over the packet's content using their local secret key. The source AS must additionally verify the MAC that is inscribed by its host.

We implement all described procedures in software, using the Data Plane Development Kit [40] on a commodity server. The server is equipped with an Intel Xeon E5-2680 CPU and a 10 GbE Network Interface Card (NIC). We dedicate only one CPU core to perform all required processing to show a pessimistic lower bound on the throughput. We connect the NIC to a traffic generator, which generates load on the software router. For our evaluation, we use IPv4 packets; we modify the Forwarding Information Base (FIB) in order to accommodate

[8]http://bench.cr.yp.to/supercop.html

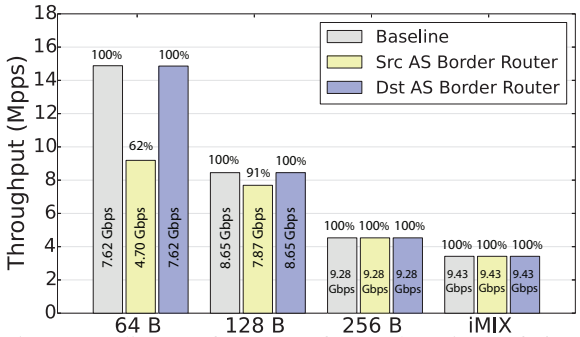

Fig. 5: Forwarding performance for packet sizes of 64, 128, and 256 bytes and for iMIX (340 bytes avg.)

the shared keys with other ASes; the FIB contains entries for 55k different ASes.[9]

Figure 5 shows the forwarding performance of data packets for three packet sizes (64, 128, and 256 bytes) and a representative mixture of Internet packet sizes (iMIX) [41]. The minimum data-packet length is 64 bytes, which is sufficiently long to accommodate the additional TRIS-header; the minimum packet length translates to the highest packet rate and is the worst case for packet processing. The baseline for the experiments is the forwarding performance without additional processing. Each bar shows the achieved throughput in terms of packet rate, with the corresponding bit rate indicated inside the bar; on top of the bar we annotate the achieved throughput as a percentage of the baseline throughput. We show the performance for border routers of source and destination ASes.

For 64-byte packets, our results show that for the source AS, the throughput degrades by 38%; the destination AS achieves the baseline performance. This difference is due to the one additional MAC computation at the source AS. For 128-byte packets, the source AS's performance degrades by 9% and the destination AS performs optimally. As the packet size increases, i.e., the packet rates decreases, the performance degrade diminishes. For 256-byte packets and a representative mixture of Internet packet sizes (iMIX) [41], both routers perform optimally. Furthermore, i) the computational complexity of data packets does not depend on the complexity of the policy, since all operations on data-packet fields (Figure 4) are policy agnostic; ii) the computational complexity for policy-packets is the same as for data packets for border routers since asymetric cryptography is used only by end hosts.

V. Extending TRIS

We describe an extension that builds on the core ideas of TRIS and provides additional security properties. Our extended protocol involves the transit ASes, so that host misbehavior can be proven to all deploying ASes on the communication path—not only the source/destination ASes. This enhancement raises awareness of attacks and misbehavior to more entities in the network and is particularly useful in case of rogue source ASes: if a source AS does not take action against its misbehaving hosts, then other ASes can deprioritize or even block traffic from such ASes, given verifiable evidence of misbehavior.

[9]http://www.cidr-report.org/as2.0/

On the downside, the extension has a higher deployment barrier since it requires additional actions from all the ASes on the communication path and also requires stronger assumptions from the network, compared to the core protocol: source ASes need to know the interdomain path that their traffic will follow to the destination AS.

Extended Threat Model. We consider a stronger threat model in which source, transit, and destination ASes can be malicious and conceal misbehavior by destroying the corresponding evidence or try to frame an innocent host. In our security analysis (Section VI), we describe how we can detect and constrain the location of such actions on the communication path.

A. Protocol Modifications

Next, we describe the additional actions that source, transit, and destination ASes have to perform.

ASes leverage RPKI and their public/private key pairs to derive long-term pairwise symmetric keys. More precisely, a source AS AS_1 shares a symmetric key $k_{A_1A_N}$ with every destination AS AS_N, which is established through pairwise DH key exchanges.

1) Connection Establishment

During connection establishment (Figure 2), AS_1 additionally adds the AS path to the destination that the traffic will follow, so that the proof of misbehavior can be sent back to all the corresponding ASes on the path. The AS path information, together with the sequence number are protected with a MAC computed with the symmetric key $k_{A_1A_N}$ (Line 9), which is shared with the destination's AS (AS_N). Note that the MAC is computed with the key $k_{A_1A_N}$—not k_{A_1}—so that AS_N can detect modification from malicious transit ASes.

The transit ASes (AS_t) on the path perform similar steps as AS_1. Specifically, they insert a sequence number and a MAC that is computed over their inserted information with the AS's local secret key k_{A_t}. Sequence numbers by the transit ASes are used to constrain the location of a replaying adversary, based on the patterns of repeating sequence numbers (Section VI-B). The MAC protects the integrity of the information inserted and serves as a stateless reminder to the AS that it forwarded the traffic. The destination AS, similar to the source AS, inserts the AS path that traffic will follow towards the source AS. Note that the return path to the source AS can be different, since AS paths are not necessarily symmetric [42].

2) Data Forwarding

For data forwarding, AS_1 creates a new packet header and adds the same information as in the core protocol (Figure 3). However, the packet header is created with sufficient length to accommodate the information of the transit ASes on the path. Transit ASes (AS_t) perform similar operations as AS_1 and insert a sequence number and a MAC computed with their local secret k_{A_t}. The role of the MAC and the sequence number is to provide integrity and detect replay attacks. AS_N does not perform any additional actions.

3) Proving Misbehavior

The procedure of proving misbehavior is extended to two rounds because transit ASes can be malicious as well. Similar

to the core protocol, the evidence consists of the signed policy and the received traffic with the embedded proofs, but the evidence must be sent to all the ASes on the path.

In the first round, the victim host provides the evidence to its AS. The AS examines the validity of the evidence, and if valid sends the evidence to all ASes on the communication path (transit and source). Similarly, these ASes examine the evidence and acknowledge or reject the complaint. An approved complaint by an AS signifies that it forwarded traffic that violates the policy properties. However, this does not mean that the source host has misbehaved; if a transit AS has replayed traffic, the source is not responsible for the violation. Therefore, the victim's AS collects approved and rejected complaints and proceeds to the second round.

In the second round, the victim's AS sends the collected information back to the ASes. Based on the collective information of approvals and rejections, ASes conclude whether the source host has misbehaved or if it has been falsely blamed. For example, if all ASes acknowledge the complaint, then the source host has misbehaved. We discuss framing of innocent hosts in Section VI-B.

B. Overhead Evaluation

Involving transit ASes incurs additional overhead compared to the core protocol. Specifically, packet headers carry more information and border routers of transit ASes also have to perform additional operations.

For *data packets*, the source AS creates the TRIS header sufficiently long so that it accommodates the information from each transit AS on the path. This information includes the sequence number and the MAC inserted by each AS; each field is 2 bytes long. Furthermore, the packet header includes a pointer field (1 byte) that points to the correct location in the packet header, where the next AS should add its information.

To quantify the total bandwidth overhead, we analyze the same packet traces from CAIDA (Section IV-A). The source AS inserts 9 bytes in the packet, and every other AS inserts 4 bytes; we assume an average AS-path length of 4 hops.[10] Table III shows the bandwidth overhead for the mean and median packet sizes in the three traces. We observe that the bandwidth overhead is higher compared to the core protocol (Table II), but we believe it is well within reach of today's overprovisioned network capacities.

For *policy-exchange packets*, the source AS must additionally indicate the list of ASes on the path; 4 bytes are used for each AS. Again, we do not evaluate the bandwidth overhead since such packets are sent infrequently.

The *border routers* of transit ASes perform additional actions as described in Section V-A2. The processing load of these operations is the same with that of border routers in destination ASes. Thus, we obtain the same performance (Figure 5) for transit border routers, i.e., achieving the baseline performance.

[10]RIPE reports an average AS-path length of 3.9 hops for IPv4 and 3.5 hops for IPv6 [43].

	Trace 1		Trace 2		Trace 3	
pkt. size	747 B	463 B	906 B	1420 B	691 B	262 B
overhead	4.95%	7.99%	4.08%	2.61%	5.35%	14.12%

TABLE III: Bandwidth overhead for the extended protocol and the mean and median packet sizes for 3 CAIDA packet traces.

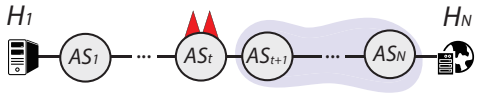

Fig. 6: Scope of security properties, when AS_t is malicious and i) corrupts evidence of misbehavior for H_1 or ii) replays traffic and frames H_1.

VI. Security Analysis

We describe the security properties of TRIS, according to the threat model presented in Section II-A. We analyze two (overlapping) attack classes: i) attacks that conceal misbehavior by destroying the corresponding evidence, and ii) framing attacks that blame an innocent host. We explicitly indicate which attacks apply only to the extended threat model. Furthermore, we note the flooding the proof-verification process is not a notable concern, since it is an offline process that does not affect forwarding performance.

A. Evading Misbehavior Detection

In this class of attacks, we consider a malicious host (H_1 in Figure 6) that violates a policy of its communication peer H_N and attempts to evade detection. We start by describing attacks that H_1 can launch on its own; then we describe collusion attacks.

H_1 may attempt to hide misbehavior by reporting a different network-layer source address in sent data packets; e.g., an address that belongs to another host or an address that does not correspond to a host.[11] Such attacks are prevented by the first building block of TRIS, i.e., source accountability: every outgoing packet contains a MAC that is computed with the shared key $k_{H_iA_i}$ between the host and the host's AS. Without this key, the adversary cannot create valid MACs and the spoofed packets will be dropped by the source AS. Furthermore, a host cannot generate new addresses on its own. Each host is assigned exactly one address, which is generated by the host's AS during the bootstrapping procedure (Section III-B). In a similar fashion, H_1 may report a false HID in sent packets. Specifically, it can generate a random HID or use one from sniffed traffic. In both cases, H_1 cannot generate a valid end-to-end MAC since it does not have a valid corresponding key. Furthermore, the policy exchange during connection setup explicitly states the valid HID for data traffic.

H_1 can attempt to hide misbehavior by tampering with the policy of H_N. The policy of H_N is protected with a signature that is computed with the private key $K^-_{H_N}$ of H_N. Thus, H_1 cannot modify the policy properties without invalidating H_N's signature. Furthermore, H_1 cannot claim that it did not receive the policy of H_N, since receiving the policy acts as an admission to send packets; H_1 would be deemed malicious if it sent traffic without receiving H_N's policy.

Under the extended threat model, H_1 can collude with

[11]This attack can also be considered a framing attack: H_1 frames another host in the eyes of the other network entities.

an on-path AS (e.g., AS_t in Figure 6) in order to corrupt the evidence of misbehavior in data packets. More precisely, H_1 sends traffic that violates H_N's policy; AS_t forwards packets, but corrupts *all* the MACs of the previous ASes in the packet. Thus, in the first complaint round only the ASes after AS_t will acknowledge the misbehavior. This degrades the security guarantees of TRIS, however, complaints to the ASes after AS_t will still be successful. More precisely, *proof of misbehavior is successful to all benign ASes adjacent to the victim's AS*, as shown by previous work [36].

B. Framing Attacks

In this class of attacks, we consider a malicious entity (host or AS) that tries to frame an innocent host (H_1 in Figure 6).

A communication peer H_N can attempt to frame H_1 by tampering with its own policy. More precisely, H_N sends its signed policy to H_1, but presents a different policy when it complains. It presents a policy with a lower sending rate than the original policy, thus accusing H_1 of a violation. However, H_1 has the original policy, which is signed by H_N; policies cannot be repudiated. Since the correct signature can be computed only by H_N, it is clear to the source and destination ASes that H_N has misbehaved.

H_N can modify packet contents in an attempt to forge evidence of misbehavior; e.g., modify timestamps to craft a high sending rate, or even substitute the source addresses in packets to frame another host. Such modifications are caught in the complaint phase, since every bit in the packet is protected by the MACs of the source and destination ASes.

Another family of framing attacks are packet replays, which come in two variations: i) a malicious host replicates the received packets when it complains to the source/destination ASes, and ii) a malicious transit AS (AS_t) replays packets (under the extended threat model). Both variations frame the source host. Such attacks are detected from the built-in replay-detection mechanism that is based on timestamps and sequence numbers: if AS_t replays packets, then the combinations of the timestamp and the sequence numbers of the first $t-1$ ASes appear multiple times. This constrains the location of the attack to either AS_t or AS_{t+1}, since AS_t may increment its sequence number normally. The approach does not identify the replaying AS, but informs ASes of an in-network replay attack. The same mechanism applies when a host replicates received packets: the combination of the timestamp and the sequence numbers of the source/destination ASes appear multiple times.

VII. Practical Considerations

A. NAT Devices

Multiple hosts that share a connection through a NAT are represented as a single host to the AS. To attribute misbehavior to a certain host, the NAT acts a small AS for its hosts, while it still acts as one host for the AS. The NAT ensures that internal hosts are held accountable and cannot frame another internal host by address spoofing. Since only a public address can be held accountable for misbehavior by the public Internet, there are a few protocol modifications when NATs are involved.

For connection establishment (Figure 2), the NAT is also involved in the policy exchange. The host initiates the policy by specifying the desired traffic properties, but not the policy identifier, since the source address and port will be translated. The NAT fills in the policy identifier and signs the policy with its AS-provided key. In addition, the shared symmetric key with the peer host is generated by the NAT. For data packets, the NAT uses the symmetric key $k_{H_1 H_N}$ to compute the end-to-end MAC around the packet. Similarly, this is performed by the NAT because the network and transport-layer header gets rewritten by the NAT.

B. Host Modifications

An important obstacle in deploying new architectures is the requirement to update the network stack of the end hosts [44]. Security-concerned users have an incentive to update their network stack and specify fine-granular policies, but this is not in the interest of all users. We outline two deployment strategies that leave the host's network stack intact, while providing the security properties of TRIS: a middlebox deployment (Section VII-B1) and a gateway deployment (Section VII-B2). The main idea behind both methods is that the additional functionality required by the hosts is delegated to another device. The main difference is that the middlebox is owned by the AS, whereas the gateway by the host. This difference raises trust and performance implications, which we describe in the following.

1) Middlebox Deployment

We design and implement a middlebox that can be deployed by ASes in order to offer TRIS services to its customer hosts. We envision the middlebox to be collocated with the first-hop router, serving a few hundreds or thousands hosts in the access network; a study for CDN deployment identified a total of 1478 distinct users over a span of 42 days [38].

The middlebox deployment introduces the following three differences to the protocol operations of Section III:

- The middlebox generates and stores the public/private key pair $K^+_{H_i}/K^-_{H_i}$ (and the corresponding certificate C_{H_i}) for each host H_i; it uses this key pair for the required connection-establishment operations. The key pair is not used to generate encryption keys and thus it can be delegated to the middlebox.
- Source accountability is no longer achieved through the host-generated MAC in data packets. Therefore, the middlebox must perform ingress filtering [45], ensuring that a malicious host does not spoof its address.
- Application policies are no longer possible since the use of two separate keys for end-to-end encryption requires host modification. In theory, it is possible to construct application policies if the middlebox is allowed to access the encryption key by acting as a man-in-the-middle. However, disclosing encryption keys to the AS is beyond any realistic threat model. In Section VII-B2, we describe how this approach can work in a trusted environment.

More specifically, the middlebox acts as a transparent proxy for end-to-end communication and performs the following steps for connection establishment and data-packet sending.

For outgoing connection establishments (Figure 2), it specifies a policy for the incoming traffic of the peer host H_N and signs the policy with the host's private key (Lines 2-5).

When the policy reply for the connection establishment arrives, the middlebox verifies the signature of H_N and generates the shared symmetric key between the two hosts (Lines 20, 21).

For incoming connection establishments (Figure 2), it verifies the signature of the initiating host H_1 (Line 12), generates the shared symmetric key between the two hosts, specifies the sending policy for the incoming traffic, and signs the policy packet with the host's private key (Lines 12-18).

For outgoing data packets (Figure 3), it indicates the sending time of the packet and inserts a MAC over the packet, computed with the shared key between the hosts (Lines 5, 6). Furthermore, the middlebox rate-limits the traffic sent from its hosts, so that the sending rates conform to the policy that the middlebox has negotiated with the communication peer. For incoming data packets (Figure 3), it verifies the MAC with the shared key between the hosts (Line 11) and stores the packet as proof of potential misbehavior.

We implement the described host procedures and all required data structures of the middlebox in software. We use cryptographic primitives based on Curve25519 [46] for public-key operations; key exchange is based on the elliptic curve variant of Diffie-Hellman (ECDH). Our implementation uses DPDK on the same commodity setup as described in Section IV-B.

To evaluate our middlebox, we need information about the load and typical traffic patterns of an access network. Due to the lack of extensive data sets, we base our evaluation by analyzing a one-hour packet-level trace of a tier-1 ISP, released by CAIDA in 2016.[12] Note that the obtained statistics are an overestimate for our purpose, since the traffic load of an access network is considerably lower than the load of a backbone link of a tier-1 ISP.

Forwarding Performance. First, we focus on the efficiency of connection establishments, which require public-key cryptographic operations. From the trace, we compute a peak flow-generation rate of 8443 flows/sec. We generate the same load on the middlebox and report that the middlebox can handle it without a performance degradation. We further increase the load to find the maximum throughput, which peaks at 9100 connection establishments per second. We thus conclude that the middlebox can definitely handle the processing load of connection establishments in an access network.

Second, we focus on the forwarding performance of data packets, which is mainly influenced by symmetric-key cryptographic operations. From the trace, we compute a peak packet rate of 533 Kpps. We impose the same load on the middlebox and observe that it can handle it without packet loss. We further increase the load to reach the throughput saturation point, which is at 10.6 Mpps—a twenty-fold higher capacity than the maximum imposed load.

Storage Overhead. The use of a middlebox introduces a more considerable storage overhead since it serves and stores information for multiple users. It has to keep per-user state (a public/private key and a certificate), which is common for middlebox devices. However, it has to keep also per-flow state (a policy, a shared key, and packet counters) and especially it stores packets in order to prove policy violations.

First, the amount of fast-path memory (SRAM) that is required is minimal since only per user and per flow state is needed at the fast path: the flow generation and data-packet processing procedures require looking up the user's keys and tracking the per flow policies. However, the received packets that serve as proof of misbehavior can be stored in the slow path, i.e., stored in DRAM and transferred to stable storage.

More specifically, storing packets is implemented as a ring buffer with a head and a tail pointer. Every received packet is stored at the location instructed by the head pointer; the head pointer is then incremented to store the next packet. A consumer thread transfers the stored packets to stable storage, starting from the tail pointer's location. Only a minimal amount of fast memory is required, which is the buffer's area that will store the incoming packet; this area is then evicted to slow memory and copied over to stable storage. This approach minimizes the amount of fast memory and bridges the speed gap between fast and slow memory.

Second, the overall storage overhead is well within practical boundaries. Note that the middlebox can store the packets of a flow only for the flow's duration; after flow expiration it discards the stored packets in the common case where no misbehavior is detected. In case of longer-lived host policies that specify a threshold for the total number of flows, the middlebox can store a single packet from the flow that proves its existence.

To put the storage overhead into context, we look at the same CAIDA trace. For the peak packet rate of 533 Kpps, we find that the middlebox has to store 394 MB every second; the original packet and 28 bytes/packet additionally for the TRIS header (for an interdomain path of 4 hops). To obtain a conservative estimate, we assume a flow duration of 15 minutes after which packets are deleted; 98% of flows are less than 15 minutes long [39]. This yields an overall maximum storage requirement of 346 GB for the middlebox, which is well within practical limits for today's hardware.

2) Gateway Deployment

The second deployment strategy is using a gateway, which performs the additional host functionalities and resides in the trust zone of the host. For example, it can be a device owned by an enterprise network or the functionality can be implemented by the home router of a residential ISP customer. We highlight the following points for the gateway operation:

- The gateway generates and stores the public/private key pair $K^+_{H_i}/K^-_{H_i}$ for its hosts. Unlike to the middlebox, it performs the host-bootstrapping phase in order to obtain the corresponding certificates C_{H_i} from the AS.
- Application policies are possible if the gateway is configured as a TLS termination proxy, which is common practice in enterprise environments. Hosts have to install an enterprise-generated certificate, trusting the enterprise as a certificate authority. Then, the gateway operates as man-in-the-middle by replacing the certificate of the communication peer with a self-generated certificate that is accepted by the host. Note that the communication

[12] Equinix-Chicago, direction A, April 2016

between the gateway and the host is assumed to be secure.

Delegating the host-related functionality to a gateway contradicts our motivation of collocating intelligence and defense mechanisms: the gateway cannot tell what constitutes an attack against a host. Yet, delegating host's functionality is a necessary step for incremental deployment. However, hosts can manually configure the gateway with some predefined policies in the same way that home routers are configured through web interfaces.

C. Network-Layer Deployment

We have presented our ideas without considering a particular underlying Internet architecture. Deployment of TRIS in today's Internet is not straightforward due to extensibility limitations of the existing protocols, and especially of IPv4.

IPv6 enables an elegant implementation using Extension Headers (EHs) [47]. We define a new EH that is processed only by border routers of TRIS-enabled ASes; TRIS-agnostic ASes follow the typical forwarding procedure without considering the TRIS header.

More precisely, we define a new hop-by-hop EH with a corresponding IP protocol number for TRIS; this would be assigned by the Internet Assigned Numbers Authority (IANA). The TRIS EH is placed after the IPv6 header or after other hop-by-hop EHs, if present. The `Next Header` field in the IPv6 header indicates the presence of a TRIS header, which contains all the necessary information, as described in Section IV-A. Furthermore, two additional fields are necessary: a `Next Header` field that points to the Transport Layer protocol, and a `Header Length` field that indicates the length of the EH to enable routers to correctly parse the EH. Using EHs in IPv6 provides a straightforward and backwards-compatible deployment path, which is not the case for IPv4.

Interoperability with IPv4 is more complex due to the protocol's inherent limitations with regard to extensibility: IPv4 lacks EHs and routers are typically configured to drop IPv4 packets with non-supported IP protocol numbers. Therefore, we describe a deployment path that leverages the IP-in-IP protocol for packet encapsulation [48]; it enables virtual point-to-point links that can encapsulate other protocols. We use IP-in-IP to interconnect TRIS-enabled routers over the IPv4 network. More precisely, for the core protocol, source and destination ASes set up a single tunnel. For the extended protocol, an end-to-end path consists of multiple tunnels, with the tunnel end points supporting TRIS. The original packet of the source—together with the TRIS header—is encapsulated in an outer IPv4 packet that is used to transfer the original packet from the tunnel's entry point to the exit point, even over non-supporting ASes.

Furthermore, ASes can disseminate the addresses of their TRIS tunnel end points by piggybacking this information in BGP messages; alternately, they can advertise tunnel end points in their RPKI certificates [29].

VIII. Related Work

Accountability architectures are commonly used as building blocks for better security. Source accountability proposals ensure that source addresses in packet headers can be trusted. AIP [3], a major proposal in this area, provides source accountability at a high deployment cost: transitioning to a self-certifying host-address space (the hash of a public-key) leads to large routing and forwarding tables; also, the requirement for asymmetric cryptographic operations on the data plane would harm forwarding performance substantially. APIP [49] is an accountability architecture that introduces accountability delegates, which vouch for their customers' traffic. However, in APIP a malicious sender can omit reporting its traffic to its accountability delegate. Our approach fixes this shortcoming by using the host's ISP as the accountability delegate, which by default is on the path of the traffic. Both AIP and APIP are designed to offer only source accountability, but a clear guideline of what constitutes misbehavior is missing. In contrast, TRIS leverages receivers to define which traffic profiles are acceptable. In addition, APIP is designed to balance accountability and privacy as it provides sender-flow unlinkability. While an orthogonal issue, this can be achieved in TRIS if an ISP allocates multiple addresses (with corresponding certificates) to its customers, so that they can decide which address to use for each flow.

Active defense mechanisms against DDoS attacks can be divided into two main categories – capabilities and filtering. In capability proposals [4], [13], [14], sources obtain short-term cryptographic authorization tokens from the destinations; the tokens are put into the packets and are then verified by routers on the path. Filtering proposals stop malicious traffic in the network before reaching the victim [15], [16], [17]. Typically, filters are installed in upstream routers and as close to the source as possible. Capabilities and filtering rely on the destination to identify the misbehavior; then, the network protects the host by dropping misbehaving traffic. Our architecture does not provide active protection; rather, it makes misbehavior provable to the network, based on flexible, host-specific policies. The knowledge of malicious activity can then be used to provide better security.

Middlepolice [50] is a hybrid approach that combines the deployability of cloud-based solutions with the destination-based control of capability systems. Victim hosts can specify policies that will be enforced by the cloud service. However, the scope of the policies is more limited compared to our proposal, as in TRIS a destination can specify a different policy for each communicating host; this does not scale when outsourcing policies to the cloud. Furthermore, Middlepolice requires rerouting all traffic through the cloud, causing an average AS-path inflation of 59%. On the contrary, Middlepolice comes with a lower deployment barrier than TRIS, as it builds on cloud deployments without requiring significant network upgrades from ISPs.

IX. Conclusion

This paper proposes increased accountability as a building block towards better security. Hosts can express their preferences and provide verifiable proof of misbehavior to all ASes on the communication path in case of misbehavior. We have implemented TRIS and demonstrated that it comes with a modest bandwidth overhead. In addition, forwarding does not require per-policy state at border routers; our software-router

prototype achieves line-rate performance for common packet sizes, saturating a 10 Gbps link with a single CPU core.

We believe that a more accountable network layer will be used in conjunction with active defense mechanisms; and that verifiable proof of misbehavior can aid the regulatory discussions with respect to Internet security.

X. Acknowledgements

We would like to thank the reviewers for their insightful feedback and suggestions. 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 gratefully acknowledge support from ETH Zürich and from the Zürich Information Security and Privacy Center (ZISC).

References

[1] "Q3 2017 State of the Internet Security Report," "http://bit.ly/2neaUat", 2017.
[2] "The Global State of Information Security Survey," "http://www.pwc.com/gsiss", 2016.
[3] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," in *Proceedings of ACM SIGCOMM*, 2008.
[4] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet Denial-of-Service with Capabilities," *SIGCOMM Computer Communication Review*, 2004.
[5] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker, "Off by Default!" in *Proceedings of ACM Hotnets*, 2005.
[6] Y. Huang, X. Geng, and A. B. Whinston, "Defeating DDoS Attacks by Fixing the Incentive Chain," *ACM Trans. Internet Technol.*, 2007.
[7] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," *SIGCOMM Comput. Commun. Rev.*, 2004.
[8] Federal Communications Commission, "Open Internet Order," "http://www.fcc.gov/openinternet", 2015.
[9] B. Rowe, D. Wood, D. Reeves, and F. Braun, "The Role of Internet Service Providers in Cyber Security," http://bit.ly/1YaTvsZ, 2011.
[10] "Cloudflare Features and Pricing," "https://www.cloudflare.com/plans", 2016.
[11] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and Adoptable Source Authentication," in *Proceedings of USENIX NSDI*, 2008.
[12] A. Li, X. Liu, and X. Yang, "Bootstrapping Accountability in the Internet We Have," in *Proceedings of USENIX NSDI*, 2011.
[13] A. Yaar, A. Perrig, and D. Song, "SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks," in *Proceedings of IEEE Security and Privacy*, 2004.
[14] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting Network Architecture," in *Proceedings of ACM SIGCOMM*, 2005.
[15] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling High Bandwidth Aggregates in the Network," *SIGCOMM Computer Communication Review*, 2002.
[16] K. Argyraki and D. R. Cheriton, "Active Internet Traffic Filtering: Real-time Response to Denial-of-service Attacks," in *Proceedings of USENIX ATC*, 2005.
[17] X. Liu, X. Yang, and Y. Lu, "To Filter or to Authorize: Network-layer DoS Defense Against Multimillion-node Botnets," in *Proceedings of ACM SIGCOMM*, 2008.
[18] "FCC looks to ISPs for Cybersecurity Assistance," "http://bit.ly/1r9IJWr", 2012.
[19] "Talking Bots with Japan's "Cyber Clean Center"," "http://bit.ly/1PehLD2", 2010.
[20] "Hostexploit," "http://hostexploit.com".
[21] "BGP Ranking," "http://bgpranking.circl.lu/".
[22] "Internet Storm Center," "http://dshield.org".
[23] K. Jeremy, "ISP Cut off From Internet After Security Concerns," "http://bit.ly/1ZrhMcH", 2008.
[24] K. Jeremy and M. Robert, "After weeklong fight, rogue ISP Troyak struggles for life," "http://bit.ly/1Zdsbsb", 2010.
[25] Hostexploit, "World Host Report," "http://bit.ly/1stlKGZ", 2014.
[26] Cisco, "Cisco Policing and Shaping Overview," "http://bit.ly/1HOHr9V", May 2015.
[27] "Q2 2016 State of the Internet Security Report," "http://akamai.me/2fGSURh", 2016.
[28] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste, "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS," in *Proceedings of ACM SIGCOMM*, 2015.
[29] ARIN, "Resource Public Key Infrastructure," "http://bit.ly/1EJCQoT".
[30] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865, 2000.
[31] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, "Diameter Base Protocol," RFC 6733, 2012.
[32] R. Durairajan, S. K. Mani, J. Sommers, and P. Barford, "Time's Forgotten: Using NTP to Understand Internet Latency," in *Proceedings of ACM HotNets*, 2015.
[33] "Understanding and Using the Network Time Protocol," "http://bit.ly/1oOCp4a".
[34] J. Saltzer, "On the Naming and Binding of Network Destinations," RFC 1498, 1993.
[35] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *Proceedings of USENIX NSDI*, 2012.
[36] C. Pappas, R. M. Reischuk, and A. Perrig, "FAIR: Forwarding Accountability for Internet Reputability," in *Proceedings of IEEE ICNP*, 2015.
[37] X. Zhang, Z. Zhou, H.-C. Hsiao, A. Perrig, and P. Tague, "ShortMAC: Efficient Data Plane Fault Localization," in *Proceedings of NDSS*, 2012.
[38] C. Imbrenda, L. Muscariello, and D. Rossi, "Analyzing Cacheable Traffic in ISP Access Networks for Micro CDN Applications via Content-centric Networking," in *Proceedings of ACM ICN*, 2014.
[39] L. Quan and J. Heidemann, "On the Characteristics and Reasons of Long-lived Internet Flows," in *Proceedings of ACM IMC*, 2010.
[40] "Data Plane Development Kit," "http://dpdk.org".
[41] A. Morton, "IMIX Genome: Specification of Variable Packet Sizes for Additional Testing," RFC 6985, 2013.
[42] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, "On routing asymmetry in the internet," in *Proceedings of IEEE GLOBECOM*, Nov. 2005.
[43] RIPE Labs, "Update on AS Path Lengths Over Time," "http://bit.ly/1tbTeKF".
[44] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Intelligent Design Enables Architectural Evolution," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011.
[45] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827 (Best Current Practice), Internet Engineering Task Force, May 2000.
[46] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *Public Key Cryptography (PKC)*, 2006.
[47] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, 1998.
[48] C. Perkins, "IP Encapsulation within IP," RFC 2003, 1996.
[49] D. Naylor, M. K. Mukerjee, and P. Steenkiste, "Balancing Accountability and Privacy in the Network," in *Proceedings of the ACM SIGCOMM Conference*, 2014.
[50] Z. Liu, H. Jin, Y.-C. Hu, and M. Bailey, "Middlepolice: Toward enforcing destination-defined policies in the middle of the internet," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

Christos Pappas received his PhD degree in Computer Science from ETH Zürich in 2018, working in the Network Security group. He then worked as a senior researcher at ETH Zürich until 2019. Before joining ETH Zürich, he worked as a researcher on search algorithms for 3D audiovisual content in the National Technical University of Athens. Currently, he works as an information security engineer at UBS AG. His interests include network transparency, privacy/accountability, and identity and access management.

Taeho Lee received his PhD degree in Computer Science from ETH Zürich in 2018. He was with the Network Security group where his main research area was designing a secure future Internet architecture that aims to support both privacy and accountability. Before joining ETH Zürich, he worked as a researcher at Electronics and Telecommunications Research Institute (ETRI) in South Korea where he was part of the future Internet research team, focusing on a mobility-oriented network architecture. Currently, he is a software engineer at Google, Inc.

Raphael M. Reischuk is head of cyber security and principal consultant at the Swiss technology and innovation company Zühlke. He is a member of several international program committees for information security, and vice president of the security commission of ICT Switzerland; he is a frequent and passionate speaker at international conferences and appears regularly on topics of network, web, and cyber security. Before joining Zühlke, he worked as a senior information security researcher at ETH Zürich, where he has done research and teaching on secure Internet architectures. Raphael Reischuk received his PhD with distinction in web and cloud security at the Information Security and Cryptography Group at CISPA, Saarland University, and Cornell University.

Pawel Szalachowski is an Assistant Professor at Singapore University of Technology and Design (SUTD). Prior to joining SUTD, he was a senior researcher at ETH Zürich. He received his PhD degree in Computer Science from Warsaw University of Technology in 2012. He is interested in building and analyzing secure networked systems.

Adrian Perrig is a Professor at the Department of Computer Science at ETH Zürich, Switzerland, where he leads the Network Security group. He is also a Distinguished Fellow at CyLab, and an Adjunct Professor of Electrical and Computer Engineering at Carnegie Mellon University. From 2002 to 2012, he was a Professor of Electrical and Computer Engineering, Engineering and Public Policy, and Computer Science (courtesy) at Carnegie Mellon University, becoming Full Professor in 2009. From 2007 to 2012, he served as the technical director for Carnegie Mellon's Cybersecurity Laboratory (CyLab). He earned his MS and PhD degrees in Computer Science from Carnegie Mellon University, and spent three years during his PhD at the University of California at Berkeley.