Network Fault Localization with Small TCB

Xin Zhang, Zongwei Zhou, Geoff Hasker, Adrian Perrig and Virgil Gligor
{xzhang1, zongweiz, hasker, perrig, gligor}@cmu.edu
Carnegie Mellon University

***Abstract*—Clear evidence indicates the existence of compromised routers in ISP and enterprise networks. Fault localization (FL) protocols enable a network to localize specific links of compromised routers sabotaging network data delivery and are recognized as an essential means to enhancing network availability in the face of targeted attacks. However, theoretically proven lower bounds have shown that secure FL protocols in the *current* network infrastructure *inevitably* incur prohibitive overhead. We observe the current limits are due to a *lack of trust* relationships among network nodes. We demonstrate that we can achieve much higher FL efficiency by leveraging trusted computing technology to design a *trusted* network-layer architecture, TrueNet, with a small Trusted Computing Base (TCB). We intend TrueNet to serve as a case study that demonstrates trusted computing's ability in yielding tangible and measurable benefits for secure network protocol designs.**

I. Introduction

ISP and enterprise networks demand reliable data delivery to support performance-critical services, thus requiring the data plane to correctly forward packets along routing paths. However, *real-world* incidents [2], [4], [6], [23], [28], [42] reveal the existence of compromised routers in ISP and enterprise networks sabotaging network data delivery. Also, in a 2010 worldwide security survey [1], 61% network operators ranked infrastructure outages due to misconfigured network equipment such as routers as the No. 2 security threat. *Fault localization* (FL), which identifies *faulty links* of compromised or misconfigured routers that suppress or forge packets during forwarding, is recognized as a key building block for achieving reliable data delivery [7], [9], [10], [12], [24], [30], [34], [35], [47]: by removing the identified faulty links from the network, end-to-end communication carries on via non-faulty paths.

Barak et al. recently proved the lower bound overhead of secure FL protocols in the *current* network infrastructure [12], which is impractical for large-scale ISP/enterprise/datacenter networks. Specifically, the lower bound states that a router *must* share some secret (e.g., cryptographic keys) with each source sending traffic traversing that router, making the key storage overhead at an intermediate router *linear in the number of end nodes*. In addition, secure FL protocols run at the granularity of entire end-to-end paths, requiring each intermediate router to store *per-path state* and the paths to be long-lived (e.g., transmitting at least 10^6 packets, which would hinder agile load-balancing and traffic engineering) [12], [47]. These fundamental limitations exist in traditional network infrastructure due to the lack of *any trust relationships* among nodes. Hence, a source node needs to *directly* check or monitor all intermediate routers (thus sharing secret keys and state) in the routing path to ensure the routers behave correctly.

Furthermore, in existing secure FL protocols, a node n which detects a faulty link l can only remove l from n's *local* routing table but cannot share the detection result with other nodes, otherwise a potentially malicious n make false accusation of other benign links (*slander* attacks). This retards the *network-wide* detection/failure recovery process, and causes inconsistent routing tables at different nodes (faulty links excluded from the routing tables of some but not all nodes). Inconsistent routing tables violate the requirements of certain routing protocols such as link-state routing. The lack of trust among network nodes also inhibits the global sharing of local detection result.

In light of the secure FL limitations in current network infrastructures, we explore how trusted computing technology can enable a network architecture with intrinsic *trust of correct data delivery* among nodes with fundamentally better performance than the proven boundaries [12] in a traditional network architecture. Our key insight is that *remote code attestation* provided by trusted computing enables a node to verify if a remote communicating node runs a trusted (or expected) version of software/protocol via authenticated "code measurements". *Isolation* further ensures that critical code execution and data are isolated from all other code and devices on the local system. Jointly, these properties provide **transitivity of verification**, i.e.: *if A verifies B's code integrity (via attestation and isolation) and B verifies C, then A believes in C's code integrity as well without needing to verify C's code integrity, because A knows B's code has correctly verified C*. Transitivity of verification, when applied to secure network protocol designs, enables *each* node to perform verification and monitoring *only with 1-hop neighbors*, building a *chain* of verification over the *end-to-end* path with reduced overhead, i.e., only requiring *per-neighbor* (as opposed to per-node or per-path) state at each router. In short, transitivity of verification eliminates the need of establishing *direct point-to-point* validation between any two nodes in the network which incurs high storage overhead and obstructs key management.

Though useful, current trusted computing technologies are by no means a panacea when directly applied to the realm

This research was supported by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389, W911NF-09-1-0273, and MURI W 911 NF 0710287 from the Army Research Office, and by support from NSF under the awards CNS-0831440 and CNS-1040801. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, NSF or the U.S. Government or any of its agencies.

of computer networks. Although several researchers propose Trusted Platform Module (TPM)-based protocols for securing general distributed systems (e.g., BIND [39]) and specific network applications (e.g., Not-a-Bot [22]), fundamental challenges render these approaches ineffective in securing data delivery at the network layer: (i) existing approaches cannot "attest" raw command-line configuration for which an expected "measurement" for remote attestation is hard to define, (ii) the extensive network stack would swell the size of the Trusted Computing Base (TCB) and it is challenging to abstract a small-sized, invariant "critical code", and (iii) a large ISP network can contain different routing instances with different implementation versions [29], which obstructs the use of a consistent "code measurement" for attestation.

The TrueNet design answers these challenges of applying trusted computing. Instead of strictly attesting the *semantics* of the huge, intertwined network stack itself, TrueNet attests the *behavior* of the network stack, i.e., whether it has successfully delivered the data or not. On one hand, the success of data delivery guarantees that *all* of the network-layer components have worked correctly, regardless of their implementation variations. On the other hand, if *any* of the network-layer components misbehaves, failures will arise in data delivery by which the faulty link(s) can be detected. Correspondingly, our approach in TrueNet is to monitor *1-hop* data delivery behavior (behavior of the network-layer protocol stack) with a small **monitoring module** as the critical code at each hop, and attest, isolate, and protect only the particular monitoring module with trusted computing. Thus, TrueNet requires only a small amount of critical code (the small monitoring module) as the TCB. Such a small TCB size (i) supports different network stack implementations and flexible protocol updates, (ii) makes the attestation of the small critical code efficient, and (iii) enables applying formal analysis [17] on the small critical code to ensure the TCB is indeed trustworthy.

The small TCB on each TrueNet router forms a *logical protected path* overlayed on the physical machines and an untrusted network stack between a source and destination, along which data delivery is monitored and ensured. As a result, TrueNet achieves efficient FL with **small router state** (only per-neighbor state), support for **dynamic/short-lived paths** (no requirements on the minimum number of packets transmitted along a path since monitoring is performed only between neighbors), and **global sharing** of detection results while eliminating slander attacks. As a proof of concept, we implement a TrueNet prototype in Linux using existing trusted computing technology and a TPM, and demonstrate that TrueNet provides high throughput while achieving the desired security properties. We also launch real trace-based measurements to show that the router state in TrueNet is up to *five* orders of magnitude less than related work [12], [47].

Contributions. We design, implement, and evaluate TrueNet, which, assuming trusted hardware, achieves secure FL with properties (i.e., per-neighbor router state, dynamic path support, and global sharing of FL results while avoiding slander or framing attacks) that invalidate the previously proven performance boundaries in traditional networks [12]. TrueNet still provides benefits for partial adoption, enabling incremental deployment, and can be deployed in inter-domain settings with the recently proposed SCION architecture [46]. Finally, TrueNet explores the role trusted computing might play in securing network protocols, shows the possibility of using trusted computing to break traditional performance boundaries, and could spark future research.

II. Problem Statement

We study the *network layer* for an ISP, enterprise, or datacenter network under a single administrative domain, where an administrator exists for configuring network nodes when necessary (e.g., installing public keys on the nodes, etc). We consider a router-level topology, where a *node* refers to a router and particularly a *source* (*destination*) refers to a source (sink) router. We denote the monitoring module at a router A as $\mathbf{MM}_A$ and a link between routers A and B as l_{AB}. TrueNet aims to achieve secure FL as follows.

Definition 1: We denote by $\delta_{AB} = \{\delta^d_{AB}, \delta^f_{AB}\}$ *the number of original packets dropped and misrouted (*δ^d_{AB}*),* and *the number of packets injected, modified, and reordered (*δ^f_{AB}*) on* l_{AB}*. A link* l_{AB} *is* faulty *if* δ_{AB} *is larger than a certain* accusation threshold $\{T^d, T^f\}$ *set by the network administrator, i.e.:*

$$\delta^d_{AB} > T^d, \quad \textbf{\textit{or}} \quad \delta^f_{AB} > T^f. \tag{1}$$

Definition 2: ***Aggregate FL*** *is achieved iff* given a routing path p, δ_{AB} *can be accurately learned for each link* l_{AB} *in* p*.* ***Per-packet FL*** *is achieved iff* given the routing path p *the failure of delivering a single packet in* p *can be immediately localized to a specific link in* p*.*

Adversary Model. We follow the trusted computing literature and assume the adversary can compromise the router OS, install malware on the routers, and launch remote software-based attacks; but the adversary cannot compromise hardware or manipulate the physical network infrastructure, nor defeat trusted computing primitives (code attestation and isolation). Such a remote attacker model is consistent with real-world router-based attacks. For example, most documented router compromises in ISP and enterprise networks are due to phishing [4] and remote exploitation of router software vulnerabilities [2], [6] and weak passwords [23] by remote hackers [42]. In addition, a majority of network operators in a recent security survey [1] listed *router misconfiguration*, which also falls under our software-based attack model, as an important cause of outages; and documented router software misconfiguration has led to network partitioning [28]. Finally, software-based attacks are usually more stealthy and large-scale than hardware-based attacks, since a hardware-based attacker usually needs physical proximity to targeted routers and will likely leave physical evidence, making the attack more auditable and less scalable.

The adversary controls multiple malicious routers which can drop, modify, inject, reorder, and misroute packets on

links incident to malicious nodes in control. Furthermore, the adversary can launch *collusion* attacks where multiple malicious routers can coordinate and conspire to evade FL or incriminate a benign link. However, the adversary has polynomially bounded computational power and cannot break cryptographic primitives.

Scope. TrueNet focuses on achieving secure FL against malicious routers. We do not consider control-plane or routing attacks and endhost- or source-based attacks such as DoS, while TrueNet complements existing secure routing [15], [16], [20] or DoS prevention schemes [43], [45]. In addition, TrueNet aims to demonstrate via secure FL the fundamental benefits trusted computing offers to secure network protocols, and we anticipate future work will utilize trusted computing to solve other network security problems including DoS attacks.

III. Fundamental Challenges

We further elaborate on the fundamental challenges in directly applying code attestation and isolation to secure data delivery in large-scale networks.

Large protocol stack. The network layer contains numerous interacting software components, i.e., (i) topology discovery, (ii) path selection from the topology, (iii) converting routing tables to forwarding tables, (iv) forwarding table lookup, etc. The incorrect operation of *any* of these components will hamper the correctness of the eventual network data delivery; therefore, straightforward attestation of the entire protocol stack would require attesting tens of thousands of lines of code. For example, the IPv4 subsystem in the Linux 2.6.37 kernel contains more than 66K lines of code, and the IP-related elements in the Click modular router [27] contain more than 15K lines of code. This swells the TCB size and thus broadens the surface for potential vulnerabilities.

Diverse implementations and complex dependencies. In practice, there can be many co-existing protocol implementations and instances [29] within the same large ISP or enterprise network. Furthermore, due to the intrinsic and obscure interactions between network-layer components, it is highly challenging to distill an invariant, small, infrequently updated critical code as TCB to be attested.

Securing raw user input/configuration. In addition to the network protocol stack, data delivery also depends on human command-line input and configuration. Unfortunately, user configurations are hard to attest due to the flexibility of the configuration language, but can be utilized by the attackers to launch attacks to sabotage data delivery. Since the current Cisco IOS provides rich command-line interfaces to drop and alter packets, an attacker can cause damage without even modifying the network stack.

Hence in this paper, we strive to address these challenges by ascertaining *the minimal, invariant critical code for securing network data-plane packet delivery*, along with its minimal configuration parameters.

IV. Design Building Blocks

Remote attestation, isolation, and sealed storage are the high-level primitives that trusted computing offers pertaining to our purpose of securing network data delivery.

Trusted computing primitives. By remotely *attesting* a selected piece of "critical code", a node X can verify if a remote node Y is executing the expected, correct version of the critical code. In conjunction with *isolation*, attestation can ensure that the execution of the critical code occurs untampered by any potentially present malicious code including the OS. Specifically, with attestation of the 1-hop monitoring module as the critical code in TrueNet, a node X can convince another node Y that X is indeed executing the correct monitoring module in an isolated fashion. Furthermore, sealed storage binds a piece of sensitive data to a particular piece of software, ensuring that only the software that originally sealed the data accesses it. In TrueNet, sealed storage can seal a monitoring module's secret keys so that only the *same* monitoring module can access the secrets.

These trusted computing primitives have been widely deployed on commodity computers [21], [25]. In the remainder of the paper, we first use these trusted computing primitives *conceptually* for presenting the TrueNet protocol. Then we delineate and implement a TrueNet router architecture incorporating the trusted computing primitives in Sections X and XI.

Security properties. Remote attestation and sealed storage can be used to set up *secure channels* and *transitivity of monitoring results* as the security properties leveraged by TrueNet for efficiently achieving FL.

1) Secure channel: The above trusted computing primitives enable a monitoring module MM_A to generate and convey its public key to a remote MM_B [33], based on which MM_A and MM_B can establish a shared secret key. By performing cryptographic operations using the secret keys *sealed* and only known by the trusted monitoring modules at network routers, a compromised router OS or malware cannot impersonate the monitoring module by forging signatures or performing encryption/decryption based on those sealed keys. This builds a secure communication channel among the monitoring modules at different routers.

2) Transitivity of monitoring results: End-to-end monitoring can now be achieved via a chain of 1-hop monitoring between every two adjacent neighbors while eliminating slander and collusion attacks. This is because if a node X verifies via code attestation that its neighbor Y is executing the correct monitoring module MM_Y, X knows that the monitoring results reported by MM_Y are correct, and that MM_Y is correctly monitoring Y's neighbor, which recursively ensures the entire end-to-end path is being correctly monitored.

V. TrueNet Overview

We give an overview of TrueNet with Figure 1 as an example topology. The shaded areas denote the monitoring modules isolated and protected by trusted computing at each router and

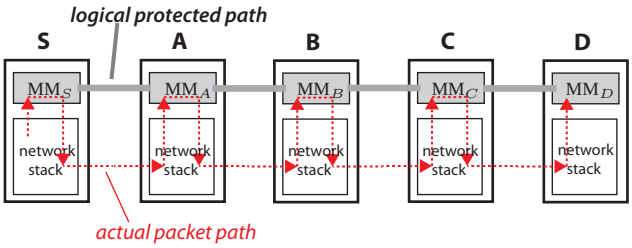

Fig. 1. An example topology to illustrate the operation of TrueNet. The solid line represents the logical protected path of packets implemented by the secure channels between the trusted monitoring modules.

thus reside in the TCB. *A router's network stack (including the OS, network interfaces, and other related programs) is untrusted*.

The logical protected path. In TrueNet, each packet is *supposed* to pass through the monitoring module MM$_i$ at each hop i. The MMs on the logical path are protected by trusted computing mechanisms and are thus trusted. The dashed line in Figure 1 depicts the **actual packet path** comprising the physical machines and network stack, originated from node S and destined to D. In contrast, the *secure channels* between adjacent trusted monitoring modules along the actual packet path form a **logical protected path** overlayed on the untrusted network stack. Every two neighboring MM$_A$ and MM$_B$ on the logical protected path share a secret key K_{AB} that is sealed by and only accessible to the same MM$_A$ or MM$_B$. Nodes (i.e., monitoring modules) in the logical protected path can thus communicate with *secrecy* and *authenticity* using the shared and sealed secret keys, and the untrusted network stack cannot inject or forge authenticated messages in the logical protected path. Nodes in the logical protected path can also attest to each other that the MMs are indeed intact and trusted.

The formation of this logical protected path requires only *per-neighbor* key storage yet greatly facilitates secure FL. Specifically, each MM$_i$ maintains a local data structure (e.g., a counter) to reflect the reception of each packet as the "packet footprint". In this way, each packet should leave a certain footprint at each hop's monitoring module *iff* the packet is successfully delivered along the logical protected path. Later by comparing the packet footprints left at every two neighbors MM$_A$ and MM$_B$ in the logical protected path, it either confirms that the packets have been successfully delivered (if the footprints match) or some problem occurs between MM$_A$ and MM$_B$ (if the footprints do not match). The secrecy and authenticity properties of the logical protected path ensure that the footprints reported by each MM$_i$ will not be forged or injected by a malicious network stack or malware.

Localizing a faulty *link*. Note that TrueNet detects a faulty *link* between two adjacent MMs, instead of a specific malicious router. In this way, MMs do not rely on the untrusted network stack or NIC to correctly deliver packets to the MMs: if the NIC or network stack of a router M drops or modifies packets before sending to MM$_M$, faults will be localized between MM$_M$ and its neighboring MMs. For example in Figure 1, if the malicious OS or a malware in router A corrupts or drops the packet before it reaches MM$_A$, then the footprint that packet leaves at MM$_A$ will differ from that at MM$_S$, thus causing link l_{SA} to be detected as we show shortly.

Small TCB. The TCB in TrueNet only includes the trusted computing primitives and the protected MM. Due to the challenges outlined in Section III, it is impractical to include the entire network stack and NIC in the TCB or for code attestation. Due to those challenges, one *cannot* simply use attestation to determine if the local OS or NIC is compromised and stop any malicious system.

TrueNet FL phases. From a high level, TrueNet consists of setup, 1-hop monitoring, and global accusation phases as sketched below.

1) Setup: During protocol setup, an administration entity of the network installs a public/private key pair, a public key K_{admin} of the administration entity, and a neighbor list to each node. Every two neighbors A and B establish a shared secret key K_{AB}, which is used to authenticate the messages exchanged between MM$_A$ and MM$_B$ in the logical protected path. The administration entity signs the neighbor list along with a version number using its private key K^{-1}_{admin}. The node private key, MAC key and K_{AB} are sealed by and only accessible to the local monitoring module.

2) 1-hop monitoring: To implement the secure channel between neighboring MMs in the logical path, a MM$_A$ computes a Message Authentication Code (MAC) for each packet sent to the next-hop MM$_B$ in the logical protected path using K_{AB}. By verifying the MAC, MM$_B$ can be convinced that its neighbor A is running the correct monitoring module *otherwise K_{AB} cannot be retrieved for authentic MAC generation*. Similarly, by authenticating the footprint reports, a node can be convinced that its neighbors are telling the correct footprints *and* having correctly monitored their neighbors in the logical protected path, *otherwise the sealed key cannot be retrieved for authenticating the reports*. This *chain of 1-hop monitoring* ensures all links in a logical protected path have been correctly monitored.

TrueNet provides two types of 1-hop monitoring primitives in the monitoring modules, namely, *per-packet monitoring* and *aggregate monitoring* for achieving per-packet FL and aggregate FL, respectively. These two monitoring approaches differ in the footprint data structure and how frequently footprints are compared between neighbors. In per-packet monitoring, a monitoring module MM$_B$ maintains an identifier (e.g., a sequence number) for each received packet with a correct MAC computed by MM$_A$, and sends back an acknowledgment (ACK) to MM$_A$ for each received packet from MM$_A$ *immediately*. In aggregate monitoring in contrast, MM$_B$ increments a *counter* if a packet received from the neighbor MM$_A$ contains a correct MAC computed by MM$_A$. Then MM$_B$ exchanges the counters with its neighbor across the logical protected paths *periodically*. Hence, aggregate monitoring reduces the communication overhead and tells *how many* packets have been dropped or corrupted between every two neighbors in the

logical protected paths, while per-packet monitoring provides more fine-grained and immediate information about *which packets* have been corrupted between two neighbors in the logical protected paths, enabling instant *failure recovery* (e.g., by immediately retransmitting the corrupted packets at the *network layer* on a per-link basis). In both monitoring approaches, MMs add additional *per-neighbor* sequence numbers for the data packets, which are used to prevent replay and reordering attacks and identify dropped packets.

3) Global accusation: A monitoring module MM_A constantly asks for the footprint reports from each neighbor MM_B to learn δ_{AB}. If MM_A observes an abnormally large δ_{AB} on a link l_{AB} in the logical protected path, MM_A sends out an accusation message to its 1-hop neighbors in the logical protected path which can verify and accept the message based on authentic MACs. Similarly, the neighbors of MM_A in the logical protected path further tell their neighbors about the accusation. This process recursively achieves network-wide *trustworthy broadcasting* (Section VIII). Hence, *all* the network nodes remove faulty links from their routing tables upon identification. Such consistency of routing tables further accelerates network-wide failure recovery, enabling the use of link-state routing which remains the de facto routing protocol for contemporary intra-domain networks.

Small router state and support for dynamic paths. Note that in any phase, attestation and authentication are only performed between two neighbors; thus each node only maintains *per-neighbor* state. Such 1-hop operations also eliminate the need for long-lived and stable paths, facilitating load balancing.

The following sections detail each phase of TrueNet.

VI. TrueNet Setup

In the setup phase, a local network administrator remains responsible for setting up and updating a router with appropriate cryptographic keys and its neighbor list as follows.

Day Zero setup. The first time a router i physically joins a network, the network administrator (i) launches a monitoring module MM_i on router i and ensures that MM_i is securely loaded and protected by the trusted computing primitives on router i. (ii) The administrator installs a public key K_{admin} of the administration entity of the network into MM_i and ensures that MM_i has correctly loaded and protected K_{admin} for verifying future messages from the administrator. (iii) The administrator creates and installs a public/private key pair K_i/K_i^{-1} and a neighbor list NL_i for router i, along with a version number and a signature created using its private key K_{admin}^{-1}. The private key K_i^{-1} is sealed and only accesible to MM_i. (iv) Each router i exchanges a secret key K_{ij} with each of its neighbors j using their public/private key pairs [33]. K_{ij} is sealed and only accessible to MM_i and MM_j, and is used for constructing the secure channel between MM_i and MM_j.

Incremental updates. After Day Zero setup, the administration entity uses the public key K_{admin} to authenticate all its update messages to the routers (e.g., when updating NL_i or K_i). These control messages from the administration entity will be protected by per-packet monitoring as we describe below. The MMs run at routers are responsible for verifying the authenticity of these updates messages using K_{admin}. The neighboring nodes i and j can periodically update their shared secret key K_{ij}. However, this paper omits the details of handling these updates due to space limitation.

VII. TrueNet 1-Hop Monitoring

Given an end-to-end communication path p, 1-hop monitoring in TrueNet ensures that the data sent by the source will be correctly delivered to the destination along p, otherwise a faulty link in p that tampers with correct data delivery will be localized and accused. Thus, we assume the source node can learn path p (e.g., from link-state routing, source routing, or recent centralized routing protocols like 4D [20], SANE [16] or ETHANE [15]), which is a common requirement for all existing secure FL schemes. We first detail each of per-packet and aggregate monitoring, and then discuss their usage scenarios in Section VII-C.

A. Per-packet Monitoring

We use Figure 1 as an example to illustrate TrueNet per-packet monitoring. Table I shows the interactions between the source S and the first hop router A for transmitting and protecting a single packet. Subsequent routers in path p will perform identical operations as router A.

Packet generation. Upon receiving a packet m with path p embedded from the network stack (OS_S) of the source S, the trusted monitoring module MM_S wraps the packet into $\mathcal{M}_S$ with a *per-neighbor* sequence number N_{SA}^S for the next-hop router A, and a MAC computed over m and N_{SA}^S with the secret key K_{SA} shared between MM_S and MM_A (Table I S2). Meanwhile, router A maintains a per-link sequence number N_{SA}^A remembering the last sequence number for the packets sent from S to A. Note that only one MAC for the next hop is attached (as opposed to attaching one MAC for each router in the path), because the transitivity of verification provided by trusted computing enables the chaining of trusted 1-hop verifications to achieve end-to-end guarantees.

As it transmits the packet, MM_S starts a timer, expecting to receive an ACK from the next-hop receiver MM_A within the allocated time, allowing MM_S to determine whether MM_A successfully received the packet. For this purpose, N_{SA}^S is temporarily stored as the packet identifier until the timer expires (Table I S3). MM_S then increments N_{SA}^S for the next packet to be sent to prevent packet replay and reordering attacks (Table I S4), and sends $\mathcal{M}_S$ back to OS_S, which in turn forwards $\mathcal{M}_S$ to router A.

Packet reception. Each received packet is expected to be passed through the monitoring module at each hop. At router A, MM_A first validates the received packet $\mathcal{M}_S$ via `validatePkt` (Table I A2), which includes checking the sequence number, the next hop, and the MAC as follows:

1) `validatePkt` first checks if the per-neighbor sequence number N_{SA}^S contained in $\mathcal{M}_S$ matches the locally stored per-neighbor N_{SA}^A value. If the values differ, indicating a replay,

	Source			Router A	
S1)	$\text{OS}_S \rightarrow \text{MM}_S$:	packet m			
S2)	MM_S `genPkt`:	$\mathcal{M}_S \leftarrow m, N^S_{SA}, [m\|\|S\|\|N^S_{SA}]_{K_{SA}}$			
S3)	MM_S `awaitACK`:	store N^S_{SA}, start timer			
S4)	MM_S `incrSN`:	$N^S_{SA} \leftarrow N^S_{SA} + 1$			
S5)	$\text{MM}_S \rightarrow \text{OS}_S$:	$\mathcal{M}_S$	$\overset{\mathcal{M}_S}{\Longrightarrow}$	A1)	$\text{OS}_A \rightarrow \text{MM}_A$ $\mathcal{M}_S$
				A2)	MM_A `validatePkt`: if $\mathcal{M}_S$ invalid, accuse l_{SA}
				A3)	MM_A `genACK`: $ack_{AS} \leftarrow A, N^A_{SA}, [A\|\|N^A_{SA}]_{K_{SA}}$
				A4)	MM_A `incrSN`: $N^A_{SA} \leftarrow N^S_{SA}$
S6)	$\text{OS}_S \rightarrow \text{MM}_S$:	ack_{AS}	$\overset{ack_{AS}}{\Longleftarrow}$	A5)	$\text{MM}_A \rightarrow \text{OS}_A$: ack_{AS}
S7)	MM_S `verifyACK`:	if ack_{AS} invalid, accuse l_{SA}		A6)	MM_A `updatePkt`: $\mathcal{M}_A \leftarrow m, N^A_{AB}, [m\|\|A\|\|N^A_{AB}]_{K_{AB}}$
				A7)	MM_A `awaitACK`: store N^A_{AB}, start timer
				A8)	MM_A `incrSN`: $N^A_{AB} \leftarrow N^A_{AB} + 1$
				A9)	$\text{MM}_A \rightarrow \text{OS}_A$: $\mathcal{M}_A \Longrightarrow$ further sent to router B

TABLE I
TRUENET PER-PACKET MONITORING. SHADED INSTRUCTIONS ARE FUNCTIONS OF THE MONITORING MODULE MM_i WHICH IS IN THE TCB. $[m]_K$ DENOTES A MESSAGE AUTHENTICATION CODE (MAC) COMPUTED OVER m USING THE SYMMETRIC KEY K.

re-ordering, or packet injection, `validatePkt` terminates (skipping the following checks) and returns "invalid".

2) `validatePkt` then retrieves the next hop from path p embedded in $\mathcal{M}_S$, and checks if the local router A is indeed the next hop in p for the current communication flow. An inconsistency indicates the previous router's OS used a wrong interface (packet misrouted), and `validatePkt` terminates returning "invalid".

3) `validatePkt` finally checks the MAC in $\mathcal{M}_S$, and returns "invalid" if the MAC is incorrect.

If `validatePkt` outputs "valid", MM_A generates an ACK including N^A_{SA} as the packet identifier with a MAC (Table I A3), which MM_S awaits. MM_A then increments the local per-neighbor sequence number N^A_{SA} (Table I A4) to prevent packet replay and reordering attacks. If `validatePkt` returns "invalid", MM_A believes that forwarding misbehavior occurs between MM_S and MM_A (denoted by l_{SA}). MM_A generates an accusation if the failure rate remains high with efficient trustworthy broadcasting (Section VIII), or signals MM_S in the ACK for instant failure recovery as we show shortly.

Packet forwarding. If the packet validation succeeds, the original MAC embedded in the received packet $\mathcal{M}_S$ is replaced with a new one computed for the next hop MM_B using the sealed secret key K_{AB} shared between MM_A and MM_B; and the per-neighbor sequence number is also replaced with the one (N^A_{AB}) for traffic between MM_A and MM_B (Table I A6). Right before the updated packet $\mathcal{M}_A$ departs MM_A, MM_A also starts a timer and expects an authenticated ACK from the next-hop MM_B (Table I A7). Finally, MM_A increments the per-neighbor sequence number N^A_{AB} for the next-hop B to prevent packet replay and reordering attacks (Table I A8).

ACK reception and failure recovery. Upon receiving an ACK ack_{AS} from a neighbor router A (Table I S6), MM_S checks if the corresponding packet identifier (N^S_{SA} in this case) is still stored indicating the timer has not expired. Then MM_S checks if the MAC is correct. If any check fails, MM_S can either re-transmit the particular corrupted packet up to r times for *instant failure recovery*, or globally accuses l_{SA} for failing to deliver any of the $r+1$ packets corresponding to N^S_{SA} via trustworthy broadcasting. The number of re-transmissions r is introduced and set to tolerate spontaneous packet loss. E.g., assuming an upper bound ρ (probability) of packet loss rate and an upper bound ϵ of allowed false positive rate, we should set $r \geq \frac{\ln \epsilon}{\ln \rho} - 1$.

Optimization. Similar to the TCP acknowledgement mechanism, a sender MM can send data packets *asynchronously* to the ACKs within a certain *sliding window* of w packets, before the ACKs for previous packets have been received. Accordingly, a receiver node can send *one single* ACK for all the w packets in the previous sliding window to reduce communication overhead.

B. Aggregate Monitoring

In aggregate monitoring, packet forwarding at each hop is divided into consecutive **monitoring intervals**, which are *asynchronous* among network nodes. A monitoring interval **from** A **to** B refers to the aggregate monitoring for packets *sent from A to B* in that interval.

Different from per-packet monitoring where MM_S starts a timer and expects an *immediate* ACK from MM_A for *each packet* sent from MM_S to MM_A, in aggregate monitoring, MM_S increments a local *monitoring counter* C^S_{SA} for each packet sent to A. Our key observation is that, due to packet authentication by 1-hop MACs, *packet count becomes a verifiable measure of the packet payload as well*, because a modified packet payload will result in an invalid MAC and cause the packet to be dropped without polluting the counter. Correspondingly, MM_A also increments a local monitoring counter C^A_{SA} for each *valid* packet received from MM_S; and increments another per-neighbor counter $\overline{C}^A_{SA}$ for each *invalid* packet received from A, as Figure 2 depicts. These counters can later be compared to reflect $\delta_{SA} = \{\delta^d_{SA}, \delta^f_{SA}\}$, i.e.:

$$\delta^d_{SA} = |C^S_{SA} - C^A_{SA}|, \;\; \delta^f_{SA} = \overline{C}^A_{SA} \tag{2}$$

Similarly, MM_A sets a counter C^A_{AB} for the next hop B, and this process recursively builds a trusted chain of 1-hop

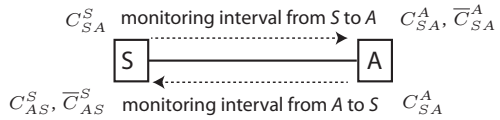

Fig. 2. Router state in TrueNet aggregate monitoring: three counters for each neighbor.

aggregate monitoring over the entire end-to-end path, while each node only has per-neighbor state (monitoring counters).

Periodically, neighbors exchange local monitoring counters in a "request-and-reply" manner to learn δ_{AB} for each link l_{AB} and accuse any link with δ_{AB} larger than a pre-set accusation threshold. Specifically, each monitoring interval consists of sending η packets (e.g., 10^4 packets). MM_S counts the number of packets sent in each monitoring interval I from S to A. Each time η packets have been sent indicates the end of interval I, and MM_S generates a counter request $\mathcal{R}_{SA}$ including the requester S, the next-hop requestee A, the interval number I to prevent replay attacks, and a MAC computed for the next hop MM_A. Then similar to per-packet monitoring, MM_S stores I and C^S_{SA}, starts a timer to wait for the counter report from MM_A, increments the interval number I, and zeros C^S_{SA} for the next interval. Finally, the request $\mathcal{R}_{SA}$ and the report $\mathcal{A}_{SA}$ proceed in the same way as in per-packet monitoring. Based on the received $\mathcal{A}_{SA}$, MM_S can calculate δ_{SA} (Equation 2) and accuse a faulty link if any.

C. Per-Packet vs. Aggregate Monitoring

Per-packet monitoring enables instant FL and failure recovery by re-transmitting the corrupted packets immediately, at the cost of an additional ACK per packet (or per w packets in a sliding window) on each link. Aggregate monitoring reduces the communication overhead by sending one counter report for all the packets in each monitoring interval (with η packets), at the cost of additional FL delay (one monitoring interval).

In TrueNet, per-packet monitoring is used to protect critical *control-plane messages*, e.g., the router configuration messages from the network administrator to each router as we mentioned earlier, global accusation message via trustworthy broadcasting as we show in Section VIII, or flow setup packets in TCP. Accordingly, aggregate monitoring would be used to protect *line-rate* data packets for the sake of lower overhead, and the network can rely on *transport layer* protocols (such as TCP) for retransmitting and recovering the lost or corrupted packets on an end-to-end basis.

VIII. TrueNet Trustworthy Broadcasting

TrueNet trustworthy broadcasting achieves *reachability*, *integrity*, and *trustworthiness* of the broadcasted message. Specifically, when a certain node O broadcasts a certain message m, (i) every node in the network will receive the message as long as the malicious nodes do not cause a *graph partition* in the network topology (**reachability**), (ii) the broadcast message received by each node is the same as the original one

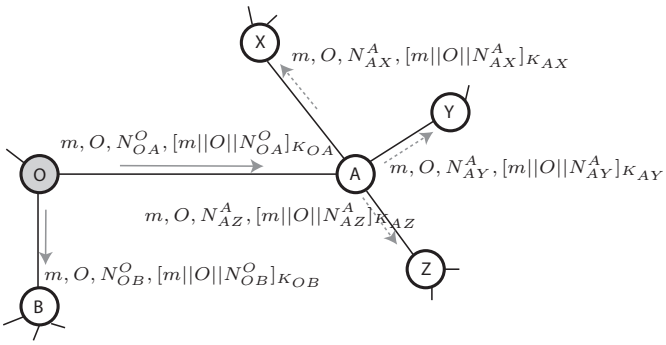

Fig. 3. TrueNet trustworthy broadcasting example. Node O is the originator of the broadcast message and other nodes use per-packet monitoring to protect the broadcast message.

(**integrity**), (iii) and the broadcast message is trusted, e.g., the accused link is indeed faulty (**trustworthiness**).

TrueNet trustworthy broadcasting is built on top of per-packet monitoring to achieve the above security properties. When a node O originates a broadcast message m, it uses per-packet monitoring (Table I) to convince O's neighbors that the message has not been modified from the original one thus preserving integrity, and the message is generated by the correct monitoring module thus preserving trustworthiness. Figure 3 shows an example of how a broadcast message propagates using per-hop monitoring (not showing the ACKs). The per-packet authenticated ACK in per-packet monitoring assures a sender that its neighbors have received the correct message thus achieving reachability, also run the correct monitoring modules, and thus will faithfully keep broadcasting the message to their neighbors and so on.

Duplicate suppression. Numerous methods exist to ensure that the broadcast message traverses each link only a single time. Due to limited space we defer detailed protocol design and analysis to future work. However, a simple method for suppressing duplicate broadcast messages is for each MM to keep state to detect duplicate messages it may later receive. To recover the state, messages can contain time stamps and nodes can be loosely time synchronized, thus only requiring storage for the maximum clock skew plus the maximum duration for the message to reach all nodes.

Global accusation. Once a node's MM detects faults, the MM generates an accusation and disseminates it *inside* certain network-wide, *periodic* beacon messages, such as the periodical routing updates (or link state announcements in link state routing) or keep-alive messages between neighbors. In TrueNet, each router R's MM_R expects to receive every neighbhor's beacon after every t seconds, otherwise MM_R accuses its neighbor which does not send a beacon on time (hence a malicious router OS cannot prevent the locally generated accusations from being sent to its neighbors). A beacon from a neighbor MM_N contains any accusation generated by MM_N and is protected using per-packet monitoring. If a beacon from MM_N contains an accusation, this beacon automatically becomes a broadcast message and is further

propagated using the trustworthy broadcasting.

IX. TrueNet FL Analysis

This section analyzes TrueNet FL delay, security and overhead, while Section XI presents real-field implementation and evaluation.

A. FL Delay

The FL delay in per-packet monitoring equals the packet re-transmission time r: only when all $r+1$ packets fail to be delivered (failure recovery fails) will a link accusation be made. Theorem 1 states the lower bound of r. During aggregate monitoring, at the end of a monitoring interval, a router A can learn the *accurate* δ_{AB} for each local link l_{AB} (thus achieving aggregate FL), *regardless* of the interval length η (number of packets sent in that interval). Hence, the value of η is set based on the desirable tradeoff between detection delay and communication overhead. For example, a smaller η enables faster detection but increases the number of counter reports (one report required for every η packets). Furthermore, since faulty links are defined and detected based on the accusation threshold, the value of η is also determined by the accuracy of the threshold-based faulty link accusation. Specifically, a too small η will introduce considerable noise in the observed link loss rate, given by $\frac{\delta^d_{AB}}{\eta}$, due to the existence of spontaneous packet loss. Theorem 1 states the lower bound of η for achieving a sufficiently high accusation accuracy.

Theorem 1: Suppose the natural packet drop rate is ρ on a link, the accusation threshold $T_d = \rho + \epsilon$ where $T_d \in (0,1)$[1], and the allowed false positive and negative rate is σ. Then the FL delay or packet re-transmission time for failure recovery in per-packet monitoring is at least $r = \frac{\ln\sigma}{\ln\rho} - 1$. The FL delay or a monitoring interval length is at least $\eta = \frac{\ln(\frac{2}{\sigma})}{(T_d-\rho)^2}$.

Proof: We assume each link has a natural drop rate ρ.

Per-packet monitoring. The probability that a benign link "naturally" drops all $r+1$ packets (including r re-transmissions), or the false positive fp, is given by $fp = \rho^{r+1}$. Since we require $fp \le \sigma$, we have $r \ge \frac{\ln\sigma}{\ln\rho} - 1$.

Aggregate monitoring. We study how many packet transmissions are required to estimate the drop rate of a single link l_{ij} within a certain *accuracy interval*. Suppose that the true value of the drop rate of l_{ij} is θ^*_{ij}, and the estimated drop rate of l_{ij} is θ_{ij}. We compute the number of packets needed to achieve a (ϵ,σ)-accuracy for θ_{ij}:

$$Pr(|\theta_{ij} - \theta^*_{ij}| > \epsilon) < \sigma \tag{3}$$

i.e., with probability $1-\sigma$ the estimated θ_{ij} is within $(\theta^*_{ij} - \epsilon, \theta^*_{ij} + \epsilon)$. We define each time a data packet is sent over link l_{ij} as a random trial, and thus each monitoring interval has η random trials. Then using *Hoeffding's inequality*, we have:

$$Pr(|\theta_{ij} - \theta^*_{ij}| > \epsilon) < 2e^{-2\eta\epsilon^2} \tag{4}$$

[1]To simplify the mathematical formula, we denote T_d as a *fraction* of packets dropped, instead of the absolute number of dropped packets as the original T^d denotes.

Then by Equation 3, we have:

$$2e^{-2\eta\epsilon^2} \le \sigma \Rightarrow \eta \ge \frac{\ln(\frac{2}{\sigma})}{2\epsilon^2} \tag{5}$$

Since $\epsilon = T_d - \rho$, we further have: $\eta \ge \frac{\ln(\frac{2}{\sigma})}{2(T_d-\rho)^2}$ ■

Finally, the *network-wide* faulty link detection process is accelerated in TrueNet since a faulty link detected by one node will be removed from the routing tables of all other nodes; whereas in existing protocols a node cannot share others' accusation because of slander attacks.

B. Security analysis

TrueNet achieves per-packet and aggregate FL via per-packet and aggregate monitoring, respectively. Recall that the adversary can drop, modify, inject, replay, re-order, and misroute packets at links under control.

Per-packet FL. Packet dropping, modification, and injection attacks between MM_A and MM_B will cause MM_A or MM_B to fail to generate authentic ACKs for the original packets; thus the link l_{AB} that corrupts the packets will be localized. Packet replay and re-ordering attacks from MM_A to MM_B will cause packets to be dropped at MM_B thanks to the use of per-neighbor sequence numbers, because MM_B stores and only expects a packet with the *most recent* per-neighbor sequence number. Finally, packet misrouting attacks are impossible because the source embeds the expected path p in the packets, and routers will perform next-hop checking based on the path and will drop any packets that are misrouted.

Aggregate FL. Without loss of generality, we consider a monitoring interval from A to B for example. Upon receiving the counters C^B_{AB} and $\overline{C}^B_{AB}$ from B (otherwise MM_A can immediately accuse l_{AB} for not sending a correct counter report), MM_A can first be convinced that the counter values were reported by the correctly running MM_B and *are thus correct*. Then MM_A can estimate δ_{AB} and detect any fault. Similar to the analysis of per-packet FL above, packet dropping will increase δ^d_{AB}, and packet modification, injection, replay, re-ordering, and misrouting will increase δ^f_{AB}.

We give one interesting note about packet misrouting attack using Figure 1 as an example topology. The malicious node B can first misroute the packets to a colluding neighbor C' (not shown in the figure), which then *transparently* forwards the packet back to C (the legitimate next hop of B in path p) *without passing the packet through* $\text{MM}_{C'}$. TrueNet treats this as a legitimate case which does *not* violate aggregate FL, because *in the logical protected path* the packets still traverse from MM_B to MM_C in order. This packet detouring is only possible between *colluding neighbors* which can be treated as *one logical* malicious entity, and is akin to detouring packets inside the same malicious router.

C. Overhead Analysis

Storage overhead. We focus on the router state required for *per-packet* processing which needs to reside in on-chip memory or cache and usually becomes the system scalability bottleneck. A router state in TrueNet includes (i) per-neighbor

secret keys (e.g., 16 bytes per neighbor) for both per-packet and aggregate monitoring, and (ii) *three* monitoring counters (e.g., 3×8 bytes) in aggregate monitoring as Figure 2 shows. Since per-packet monitoring is used for infrequent (compared to the link rate) packets, such state can be either stored in the adequate off-chip DRAM, or stored in a small cache (storing up to w packets in a sliding window at any time).

Communication overhead. The extra communication overhead in TrueNet per-packet monitoring includes one ACK per packet or per sliding window with w packets. The communication overhead in aggregate monitoring is one counter report per monitoring interval (e.g., with 10^4 packets). When per-packet monitoring is only used for protecting infrequent (compared to the line rate) control messages such as flow setup in TCP and link-state routing updates, the extra communication overhead amortized on each data packet is small.

X. TrueNet Router Architecture

We present a TrueNet router architecture leveraging a *dedicated hypervisor* and TPM chip to implement the trusted computing primitives (remote attestation, isolation, and sealed storage), and modern mainstream router hardware to speed up time-critical operations in TrueNet.

Anatomy of a TrueNet router. Modern routers commonly use a switch-based router architecture with fully distributed processors [11] and the network interfaces perform almost all the critical data-path operations for a normal packet. Figure 4 shows the architecture of a TrueNet router, where the shaded components are those added in a TrueNet router but not present in a standard modern router and also constitute the TCB for TrueNet. As Figure 4 shows, each TrueNet router is equipped with a TPM chip and CPUs with hardware virtualization support (e.g., AMD SVM [5], or Intel TXT [25]), and installs a dedicated hypervisor such as TrustVisor [32]. The dedicated hypervisor isolates MM from the rest of the router system (e.g., router OS, peripheral devices, etc.), enables remote attestation and sealed storage with the support of TPM chip, and protects MM's execution integrity, data integrity and secrecy. Similar to TrustVisor [32], the TPM operations are only needed when the dedicated hypervisor boots to ensure the hypervisor's integrity, while afterwards the dedicated hypervisor performs attestation and storage sealing to improve the efficiency.

For better performance, we anticipate on every network interface, there is a *trusted* hardware MAC Module (MACM) to perform the MAC operations in MM as described earlier. A MACM has a piece of private memory space and a high-speed MAC computation module. The private memory of MACM is mapped to the main memory residing in the CPU subsystem, and shared with the local MM. The dedicated hypervisor also protects this piece of main memory from the rest of the CPU subsystem, so that only the MM can read from and write to this main memory region. However, MACM can also be implemented inside the software MM as we described earlier, which we used for our prototyping (Section XI-A).

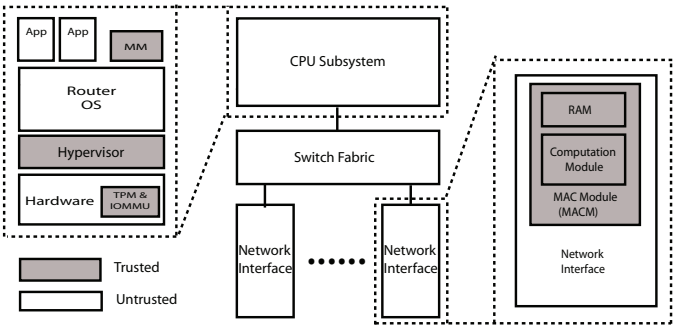

Fig. 4. TrueNet router architecture.

Software monitoring module MM. A MM handles all *control-plane* operations that are not time-critical, or infrequent in a TrueNet system. First, the local MM negotiates secret keys with the MMs on the neighboring routers, and writes the secret keys into the main memory region that maps the private memory of MACM. Secret key negotiation only happens periodically according to the cryptographic key lifetime. Secondly, MMs on the source nodes also handle packets originating from the connected end-hosts by adding the entire routing path into the packets (Section VII) for 1-hop monitoring. Thirdly, the MM is also responsible for generating accusations to be broadcasted in the beacon messages. In addition, MM also periodically checks the locally stored timers for awaiting ACKs from neighbors to detect and remove any expired entries.

Dedicated MAC module. The dedicated MAC module (MACM) is responsible for all data-plane operations to achieve high packet processing throughput. A MACM verifies the MAC in the packet, validates the correct presence of the local router in the embedded path, computes the new MAC using the shared secret key for the next hop router, updates per-neighbor sequence numbers and monitoring counters, and attaches the new MAC to the packet on a per-packet basis. To achieve high throughput in MAC computation, We can use parallelizable MAC algorithms such as XOR-MAC [13], XECB-MAC [18], PMAC [14], or high speed hardware implementations [31], [38], [41], [44] which can obtain more than 62.6 Gbps throughput.

XI. Implementation and Evaluation

In this section, we evaluate both TrueNet's *computational* overhead based on our Linux prototype of a TrueNet router and TrueNet's *storage* overhead based on real-world ISP topologies and traffic traces.

A. Prototype and Computational Overhead

We implement a TrueNet router prototype in Linux with TPM chip to evaluate per-packet cryptographic computational overhead of a TrueNet router. We show the performance of a TrueNet intermediate router which performs two MAC operations per packet (verification of the previous-hop MAC and generation of the next-hop MAC) *inside the software*

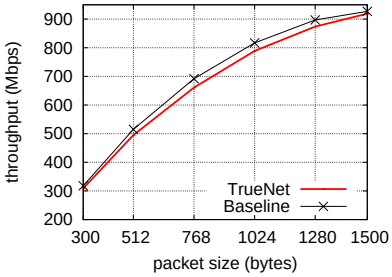

Fig. 5. TrueNet router throughput.

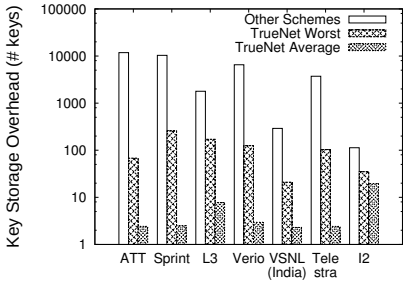

Fig. 6. Key storage overhead of a single router on ISP topologies.

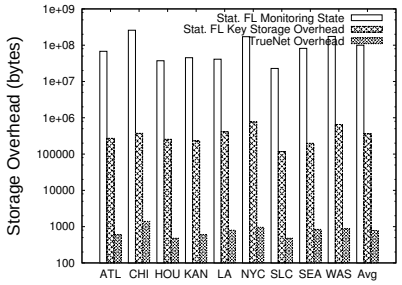

Fig. 7. Overhead comparison based on Internet2 topology and traffic traces.

MM. We observe that the TrueNet per-packet cryptographic operations, even implemented in TrueNet software module MM without any hardware acceleration, can fully cope with gigabit link-rate processing of data packets, and are fully scalable to higher performance with more CPUs. We anticipate the dedicated hardware MACM (Section X) can further boost the TrueNet router throughput.

Platform. We performed all experiments on off-the-shelf servers with one Intel Xeon E5640 CPU (four 2.66 GHz cores, 256KB L1 cache, 1MB L2 cache, 12MB L3 cache), 12G DDR3 RAM with 25.6 GB/s memory bandwidth. This CPU supports new Intel AES-NI instructions [26] for high speed AES computation. The servers are equipped with TPM chips and Broadcom NetXtreme II BCM5709 Gigabit Ethernet Interface Cards, and runs Ubuntu 10.04 32-bit Desktop OS.

Prototype. In our TrueNet prototype, we modify TrustVisor [32] as our dedicated hypervisor. We run Ubuntu Linux OS on top of our hypervisor and implement a TrueNet intermediate router as a multi-threaded user-space process. A TrueNet router process includes the secure software module MM and untrusted network stack. The untrusted network stack consists of two threads: a receiver thread that listens to network packets via TUN/TAP virtual interfaces and puts received packets to an input packet queue, and a forwarder thread in charge of sending the packets in the output packet queue to their appropriate next-hop routers. Multiple MMs run as child threads, constantly poll the input packet queue, copy the new incoming packets to a shared output packet queue, and perform MAC computations. We use the CMAC-AES-128 MAC algorithm to leverage the new AES-NI instructions on Intel CPUs.

Our software module MM performs similar per-packet cryptographic operations as the hardware module MACM proposed in Section X in software manner, while maintaining same security guarantees. The MM child threads are running inside the secure and isolated execution environment provided by dedicated hypervisor ever since threads start. The dedicated hypervisor also protects the memory region of input packet queue as accessible by both the untrusted network stack and MMs, and the output packet queue as writable by MMs but only readable by untrusted network stack. This memory configuration assures MM's execution integrity. Finally, the TPM securely boots and late-launches the dedicated hypervisor to guarantee its integrity, as described in the TrustVisor proposal [32].

	Packet Size (Byte)			
	1500	1000	500	100
MAC Computation	4.2	2.9	1.6	0.4
Others	1.3	1.1	0.7	1.1
Total	5.5	4	2.3	1.5

TABLE II
TRUENET SOFTWARE MODULE MM'S LATENCY OVERHEAD BREAKDOWN. ALL THE DATA IS THE AVERAGE TIME (MICROSECONDS) IN 50000 PACKET PROCESSING TRIALS.

Throughput and Latency Breakdown. We tested the throughput of our software TrueNet router prototype using the widely adopted network performance benchmarking tool Netperf [3]. Figure 5 shows the test result. The baseline performance in the figure is obtained by using a main thread to receive packets, *two* MM threads to move packets to the output packet queue *without any other operations*, and one forwarder thread to send packets out to next-hop routers. For TrueNet prototype, the test setting is similar to baseline performance test with the only difference that MM threads perform TrueNet packet validation and MAC computations for every packet. As Figure 5 shows, TrueNet prototype incurs *negligible* throughput degradation when compared with the baseline throughput (maximum degradation in our test is (817-789)/817=4.5% when packet size is 1024 bytes, most degradation rates are under 2%).

We also shows a latency overhead breakdown of executing software module MM's per-packet process. From Table II, we know that, leveraging the new AES-NI instruction, MAC computations are highly efficient (on average 3 CPU cycles per packet byte). In our prototype, AES key setup time is negligible since each TrueNet router only needs to hold one session key per neighboring router in a session key life time, and we can pre-compute all AES sub-keys.

B. Storage Overhead Measurement

TrueNet's ability to deliver strong security properties (instant failure recovery with per-packet FL, global accusation, etc) with less state than previous attempts [12], [47] follows

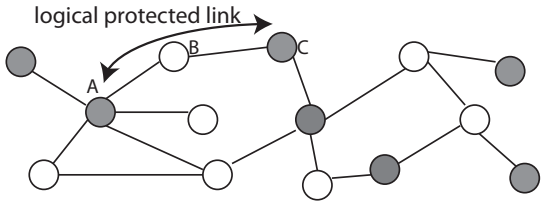

Fig. 8. Incremental deployment of TrueNet. The shaded nodes have deployed TrueNet and form logical trust links between each other.

logically. Still, measurements under real-world conditions provide an exact assessment of TrueNet's strength.

Rocketfuel-based measurements. The Rocketfuel topologies [40] of various top-tier ISPs extend from the ISPs' peering routers to approximately the first hop within a customer's network. We count the node degree for each router in the topology to assess TrueNet's overhead and compare it to the number of nodes in the network, representing the recently proposed Statistical FL [12] and PAAI [47]'s key storage overhead. Figure 6 suggests that TrueNet incurs on average two orders of magnitude less overhead in the worst case (considering the maximum node degree in the topologies), and three order less overhead for the average case (considering the average node degree).

Internet2-based measurements. The Internet2 provides similar topology data for its core routers, which Figure 6 also illustrates (labeled as "I2"). Since this topology only includes core routers, TrueNet does not deliver the orders of magnitude less overhead achieved with the Rocketfuel topologies, providing an 83% savings in the average case and 69% in the worst case. Conveniently, the Internet2 also provides Netflow data, allowing for measurement of TrueNet's and Statistical FL's monitoring state overhead. These Netflow files capture 1/100 packets seen over a five minute interval. In Statistical FL, the router incurs an around 500-byte "secure sketch" [19] for each path (identified as each unique source and destination in our measurement). In contrast, a TrueNet router maintains three counters (24 bytes) for each neighbor. Figure 7 shows that TrueNet requires approximately five orders of magnitude less monitoring state overhead. Additionally, these flow data allow for a more accurate estimation of key storage overhead in Statistical FL (number of sources with traffic concurrently traversing the same router), also shown in Figure 7 (the key storage overhead in TrueNet is still one key per neighbor).

XII. Discussion

A. Incremental Deployment

Although we argue it is feasible to upgrade all routers with trusted computing primitives within *a single* administrative domain, we note that partial deployment of TrueNet can still benefit the early adopters. Specifically, when only a subset of routers in a network are equipped with TrueNet, the monitoring modules still constitute logical protected paths where a *logical protected link* between two MMs may consist of multiple physical links. Figure 8 shows an example where the shaded nodes have deployed TrueNet and a logical protected link consists of l_{AB} and l_{BC}. Hence, FL is still achieved on each logical protected link (though not an exact physical link), which helps localizing the failure to a bounded region and facilitates network diagnosis. Furthermore, the more densely the MMs are deployed, the more accurate the failure localization can be, which incents incrementally deploying TrueNet.

B. Interdomain Deployment

TrueNet mainly targets intra-domain networks such as ISP and enterprise networks, where sophisticated hardware attacks can be precluded since the *remote* attacker (the adversary model we considered) does not have physical access to the routers. However, it is ineffective to deploy TrueNet in the *current* inter-domain setting where each Autonomous System (AS) represents a node in TrueNet, because a selfish or malicious AS has physical access to its routers and can thus subvert the hardware (e.g., TPM chips) upon which trusted computing primitives rely. Fortunately, the recently proposed SCION [46] inter-domain architecture groups the ASes into different *trust domains*, within which strong contractual or legislative regulation can be enforced. Hence, an AS tampering with the hardware can be legally penalized by the containing trust domain. This architecture naturally enables the wide deployment of TrueNet (or trusted computing primitives in general) across different ASes within a trust domain. Meanwhile, TrueNet also serves as an example of how to technically achieve enforceable accountability within a trust domain in SCION.

XIII. Related Work

Many efforts in trusted computing focus on efficient implementation of remote attestation, sealed storage, and secure boot for bootstrapping trust on commodity computers [32], [36]. A few proposals also consider utilizing trusted computing to address network security plagues [22], [37], [39]. However, BIND [39] focuses on routing security and cannot secure against raw user input and configurations. Not-a-Bot [22] leverages trusted computing and TPM to mitigate DDoS attacks but not to secure the network layer. Recently, Saroiu et al. [37] propose the design of TPM-based "trusted sensors" via remote attestation to secure a broad range of mobile applications.

The network fault localization problem has been extensively studied in the literature with traditional network infrastructure, which turns out to be a surprisingly challenging task indicated by the security vulnerabilities of many existing protocols [7], [9], [10], [24], [30], [34], [35] (both Barak et al. [12] and Zhang et al. [47] have summarized the vulnerabilities). Among the known secure proposals, the protocol due to Avramopoulos et al. [8] incurs high computational and communication overhead, because it requires acknowledgments and multiple digital signature operations from all routers in the path for *each* data packet. Both Statistical FL [12] and PAAI-1 [47] require per-node secret key storage and per-path monitoring state, and incur unacceptably long detection delays (e.g., after

sending at least 10^6 packets) due to their use of probabilistic data structures for packet fingerprinting to reduce storage overhead.

XIV. Conclusion

Using secure FL as a case study, we demonstrate that trusted computing enables transitivity of verification and eliminates the need of establishing *direct point-to-point* trust between any two nodes in the network which incurs high storage overhead and obstructs key management. TrueNet employs only a small TCB to achieve secure FL with small router state, dynamic path support, and global accusation that are proven impossible in traditional networks. We hope TrueNet can spark future research on utilizing trusted computing to efficiently address other network security problems such as DDoS defense, access control, resource allocation, etc.

References

[1] Arbor networks: worldwide security survey 2010. http://www.arbornetworks.com/sp_security_report.php.

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

[3] Netperf benchmark. http://www.netperf.org/netperf/.

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

[5] Advanced Micro Devices. AMD 64 architecture programmer's manual: Volume 2: System programming. AMD Publication no. 24593 rev. 3.14, Sept. 2007.

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

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

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

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

[11] J. Aweya. IP router architecture: An overview. *International Journal of Communication Systems*, 14(5):447–475, June 2001.

[12] B. Barak, S. Goldberg, and D. Xiao. Protocols and lower bounds for failure localization in the internet. In *Proceedings of EUROCRYPT*, 2008.

[13] M. Bellare, R. Guerin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *CRYPTO '95*, pages 15–28. Springer-Verlag, 1995.

[14] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication(PMAC). In *EUROCRYPT '02*, volume LNCS 2322, pages 384–397. Springer-Verlag, 2002.

[15] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker. Ethane: Taking control of the enterprise. In *SIGCOMM*, 2007.

[16] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. Mckeown, and S. Shenker. Sane: A protection architecture for enterprise networks. In *USENIX Security*, 2006.

[17] S. Chaki, E. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in C. In *IEEE Transactions on Software Engineering*, 2004.

[18] V. D. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *Fast Software Encryption '01*, volume LNCS 2355, pages 92–108. Springer-Verlag, 2001.

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

[20] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. In *ACM SIGCOMM CCR*, 2005.

[21] T. C. Group. TPM specification version 1.2, 2009.

[22] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-bot: Improving service availability in the face of botnet attacks. In *Usenix NSDI*, 2009.

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

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

[25] Intel Corportation. Intel trusted execution techonology – software development guide. Document number 315168-005, June 2008.

[26] Intel Mobility Group, Israel Development Center, Israel. Intel advanced encryption standard (aes) instructions set, Jan. 2010. http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/.

[27] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 2000.

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

[29] F. Le, G. G. Xie, and H. Zhang. Theory and new primitives for safely connecting routing protocol instances. In *ACM SIGCOMM*, 2010.

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

[31] Y. Lu, G. Shou, Y. Hu, and Z. Guo. The research and efficient fpga implementation of ghash core for gmac. In *International Conference on E-Business and Information System Security*, pages 1–5, 2009.

[32] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB reduction and attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.

[33] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri. Minimal TCB code execution (extended abstract). In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2007.

[34] 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.

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

[36] B. Parno, J. M. McCune, and A. Perrig. Bootstrapping trust in commodity computers. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.

[37] S. Saroiu and A. Wolman. I am a sensor, and i approve this message. In *HotMobile*, 2010.

[38] A. Satoh, T. Sugawara, and T. Aoki. High-performance hardware architecture for galois counter mode. *IEEE Transactions on Computers*, 58(7):917–930, 2009.

[39] E. Shi, A. Perrig, and L. V. Doorn. BIND: A time-of-use attestation service for secure distributed systems. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2005.

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

[41] H. Technology. Aes based authentication cores. http://www.heliontech.com/aes_auth.htm.

[42] R. Thomas. Isp security bof, nanog 28. http://www.nanog.org/mtg-0306/pdf/thomas.pdf.

[43] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *IEEE Symposium on Security and Privacy*, 2004.

[44] B. Yang, R. Karri, and D. A. McGrew. A high-speed hardware architecture for universal message authentication code. *IEEE Journal on Selected Areas in Communications*, 24(10):1831–1839, 2006.

[45] X. Yang, D. Wetherall, and T. Anderson. Tva: A dos-limiting network architecture. In *IEEE/ACM Transactions on Networking*, to appear.

[46] 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 *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, May 2011.

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