ShortMAC: Efficient Data-Plane Fault Localization

Xin Zhang, Zongwei Zhou, Hsu-Chun Hsiao, Tiffany Kim, Patrick Tague, and Adrian Perrig

January 30, 2011

CMU-CyLab-11-007

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213
ShortMAC: Efficient Data-Plane Fault Localization

Xin Zhang, Zongwei Zhou, Hsu-Chun Hsiao, Tiffany Kim, Patrick Tague and Adrian Perrig
Carnegie Mellon University

Abstract
The rising demand for high-quality online services requires reliable packet delivery at the network layer. Data-plane fault localization is recognized as a promising means to this end, since it enables a source node to efficiently localize faulty links, find a fault-free path, and enforce contractual obligations among network nodes. Existing fault localization protocols cannot achieve a practical tradeoff between security and efficiency and they require unacceptably long detection delays, and require monitored flows to be impractically long-lived. In this paper, we propose an efficient fault localization protocol called ShortMAC which leverages probabilistic packet authentication and achieves 100 - 10000 times lower detection delay and overhead than related work. We theoretically derive a lower-bound guarantee on data-plane packet delivery in ShortMAC, implement a ShortMAC prototype, and evaluate its effectiveness on two platforms: SSFNet simulator and Linux/Click router. Our implementation and evaluation results show that ShortMAC causes negligible throughput and latency costs while retaining a high level of security.

1 Introduction
Performance-sensitive services, such as cloud computing, and mission-critical networks, such as the military and ISP networks, require high assurance of data delivery at the network layer. However, real-world incidents [2, 7, 9, 26, 35, 52] and research studies [10, 16, 45, 59] reveal the existence of compromised routers in ISP and enterprise networks, and demonstrate that current networks are surprisingly vulnerable to data-plane attacks: a compromised router or a dishonest transit ISP can easily drop, delay, inject or modify packets on the forwarding path to mount Denial-of-Service, surveillance, man-in-the-middle attacks, etc. Unfortunately, current networks do not provide any assurance of data delivery in adversarial environments, and lack a reliable way to identify misbehaving routers that jeopardize packet delivery. For example, a malicious or misconfigured router can “correctly” respond to ping or traceroute probes while corrupting other data packets.

Though end-to-end monitoring [13, 25] and multi-path routing [21,23,34,46,54,56,57] can mitigate data-plane attacks to some extent, they are proven to render poor performance guarantees [49,59]; without the exact knowledge of which link is faulty, a source node would need to explore an exponential number of paths in the number of faulty links in the worst case. As illustrated in Figure 1 where the default route from S to D is path \(1 , 2 , 3 , 4 \), end-to-end monitoring only indicates if the current path is faulty without localizing a specific faulty link (if any) of a compromised or misconfigured router on the path. In the worst case, \( S \) needs to explore \( 2^4 \) paths to find the path with no faulty links, i.e., path \( 1 , 2 , 3 , 4 \).

Therefore, data-plane fault localization has been widely recognized as a promising remedy for securing data delivery [10, 11, 16, 59]. In a nutshell, fault localization enables a source or transit node to monitor data forwarding at each hop and localize abnormally high packet loss, injection, and/or forgery on a certain link. Such information about link quality can be utilized for two vital purposes. First, by excluding detected poor links the source can select high-performance routing paths to carry its traffic, thus eliminating the exponential path exploration problem as depicted in Figure 1. Second, fault localization provides forwarding accountability which proves to be a necessary component for enforcing contractual obligations between participating nodes in a contractual networking service such as the Internet or wireless mesh networks, as demonstrated by Laskowski and Chuang [36].

Unfortunately, existing fault localization protocols suffer from security, efficiency, and agility challenges in the presence of strong adversaries: (i) Security and efficiency: As Section 3.1 illustrates, sophisticated attacks (e.g., framing and collusion attacks) and natural packet loss tend to make a fault localization protocol insecure or heavy-weight (to prevent sophisticated attacks). (ii) Agility: In addition, recent secure and relatively light-weight protocols [16,59] leverage packet sampling or flow fingerprinting to prevent packet modification attacks while reducing communication overhead. However, in addition to high storage overhead, these techniques result in long detection delays and thus require monitored paths to be long-lived (e.g., after monitoring \( 10^8 \) packets over the same path in Statistical FL by Barak et al. [16]), which is impractical for networks with short-lived flows and agile routing paths.

To fill the gap between the demanding requirements on reliable communication in modern networks and the absence of data delivery assurance, in this paper, we propose ShortMAC, an efficient fault localization protocol to provide a theoretically proven guarantee on end-to-end data-plane packet delivery even in the presence of sophisticated adversaries. More specifically, we aim to guarantee that, given a correct routing infrastructure, a benign source node can find a non-faulty path in time linear in the number of faulty links, along which a very high fraction of packets can be correctly delivered. Our key insights are two-fold as sketched below for achieving such a

![Figure 1: Example of the exponential path exploration problem for end-to-end monitoring. The source node S is communicating with destination D, and dotted links are faulty links of malicious routers (black nodes). Suppose shortest-path routing is used.](image)
provable guarantee via fault localization with dramatically improved efficiency compared to existing work.

**Insight 1.** We first observe that localizing data-plane faults along a communication path can be reduced to monitoring packet count (number of received packets) and packet content (payload of received packets) at each router on that path. Furthermore, if packets can be efficiently authenticated, packet count also becomes a verifiable measure of packet content, because forged packets (with invalid contents) will be dropped by the routers and manifest an observable deviation in the packet count. Thus, routers can reduce storage overhead by storing counters instead of packet contents.

**Insight 2.** We also observe that we can achieve a high packet delivery guarantee via fault localization by limiting the amount of malicious packet drops/modifications, instead of perfectly detecting each single malicious activity. Furthermore, strong per-packet authentication to achieve perfect detection of every single bogus packet is unnecessary for limiting the adversary’s ability to modify/inject bogus packets. Instead, the source can use much shorter packet-dependent random integrity bits as a way of defining malicious forwarding behavior. We identify packet dropping and packet injection as the two fundamental data-plane threats. Note that other data-plane attacks can be reduced to these two threats as follows: (i) packet modification is equivalent to dropping the original packet and injecting a fabricated packet, (ii) packet replay can be regarded as repeated packet injection, (iii) packet delay can be treated as dropping the original packet and later injecting it, and (iv) packet mis-routing can be regarded as dropping packets along the original path and injecting them to the new path. A formal definition follows:

**Definition 1** An \((x, y) - \text{Malicious Router}\) is a router that intentionally drops up to a fraction \(x\) of the legitimate data packets from a source \(S\) to a destination \(f_d\), and injects up to \(y\) spurious packets to \(f_d\), pretending that the packets originate from \(S\).

The misbehavior space of such a malicious router comprises (i) dropping packets, (ii) injecting packets on any of its adjacent links which we call malicious links (non-malicious links are called benign links), (iii) strategically claiming arbitrary local state (e.g., number of packets received) to its own advantage, or (iv) colluding with other malicious routers to perform the above attacks.

Such a strong attacker model is not merely out of theoretical curiosities, but has been widely witnessed in practice. For example, outsider attackers have leveraged social engineering, phishing [7], and exploration of router software vulnerabilities [2, 9] and weak passwords [26] to compromise ISP and enterprise routers [52]. Also, in a 2010 worldwide security survey [1], 61% of network operators ranked infrastructure outages due to misconfigured routers, which also fall under our attacker model, as the No. 2 security threat.

Furthermore, we assume that an adversary knows the cryptographic keys of controlled routers, and can eavesdrop and perform traffic analysis anywhere in the network. The protocol parameters are public; as a consequence, the adversary may attempt to bias the measurement results to evade detection or frame honest links. However, the adversary cannot control the natural packet loss rate on the links in the path, because this would constitute a physical-layer attack which can be dealt with through physical-layer protections. We consider attackers with polynomially bounded computational power which cannot break cryptographic schemes, e.g., encryption or Message Authentication Codes (MAC).

**2 Problem Statement and Setting**

For generality, we consider an abstract multi-hop network model where routers relay packets between sources and destinations. Different real-world networks can have varying granularity for defining these entities. In the Internet, for example, a router, a source, or a destination can correspond to a transit, a source, and a destination Autonomous System (AS), respectively; in an ISP or a corporate network, on the other hand, each entity corresponds to a physical router or a switch (in particular a source and a destination can correspond to a stub router which directly connects end hosts). Throughout the paper, we follow the notation as illustrated in Figure 2. We denote the routers in a path by \(f_1, f_2, \ldots, f_d\), the destination by \(f_d\), and the link between \(f_{i-1}\) and \(f_i\) by \(l_i\).

**2.1 Adversary Model**

The goal of an adversary who controls malicious routers is to sabotage data delivery at the forwarding path. Instead of considering an individual forwarding attack, we seek a general way of defining malicious forwarding behavior. We identify packet dropping and packet injection as the two fundamental data-plane threats. Note that other data-plane attacks can be reduced to these two threats as follows: (i) packet modification is equivalent to dropping the original packet and injecting a fabricated packet, (ii) packet replay can be regarded as repeated packet injection, (iii) packet delay can be treated as dropping the original packet and later injecting it, and (iv) packet mis-routing can be regarded as dropping packets along the original path and injecting them to the new path. A formal definition follows:

**Definition 2** An end-to-end communication is composed of a set of consecutive epochs. An epoch for an end-to-end path is defined as the duration of transmitting a sequence of \(N\) data packets by a source \(S\) toward a destination \(f_d\) along that path. The epochs are asynchronous among different paths.
Definition 3 Given a drop detection threshold $T_{dr}$ (i.e., fraction of dropped packets) and an injection detection threshold $T_{in}$ (i.e., number of injected packets), a link $l_i$ is defined as faulty iff: (i) more than $T_{dr}$ fraction of packets are dropped on $l_i$ by $f_i$ in an epoch, or (ii) more than $T_{in}$ packets are injected by $f_i$ in an epoch, or (iii) the adjacent router $f_i$ or $f_{i+1}$ makes $l_i$ appear faulty in an epoch.

When $T_{dr}$ and $T_{in}$ are carefully set based on the prior knowledge such that the natural packet loss and corruption are below $T_{dr}$ and $T_{in}$, respectively, a faulty link must be a malicious link.

Definition 4 $(N, \delta)$—Data-plane Fault Localization is achieved iff: given an end-to-end communication path $p$, after a detection delay of sending $N$ packets, the source node $S$ of path $p$ can identify a specific faulty link along that path (if any) with false positive or negative rate less than $\delta$.

Definition 5 $(\Omega, \theta)$—Guaranteed Forwarding Correctness (Guaranteed Data-Plane Packet Delivery) is achieved iff: after exploring at most $\Omega$ paths, a source node can find a non-faulty path (if any) along which all routers have correctly forwarded at least $\theta$ fraction of the source’s data packets sent along the path to $f_d$.

To achieve a guaranteed $\theta$, we need to bound (not necessarily eliminate) the adversary’s ability to drop packets and to inject packets so that if the adversary drops more than $\alpha$ percent of packets or injects $\beta$ bogus packets, it will be detected with a high probability. A formal definition follows.

Definition 6 For any epoch with a sufficiently large number of data packets from any source, $(\alpha, \beta)\delta$—Forwarding Security is achieved iff two conditions are simultaneously satisfied:
1. (Low False Negative Rate) When the adversary drops more than $\alpha$ percent of the data packets on a single link, or injects more than $\beta$ fake packets on a single link, the source will detect at least one of the malicious links under the adversary’s control with probability at least $1 - \delta$.
2. (Low False Positive Rate) The probability of falsely incriminating at least one benign link is at most $\delta$.

2.3 Scope and Assumptions

Since we focus on data-plane security at the network layer, we assume the following network control-plane and link-layer mechanisms, each of which represents a separate line of research orthogonal to ours. (i) Since we focus on data plane, we can borrow existing secure routing protocols [27, 29, 47] by which nodes can learn the genuine network topology. (ii) We assume secure neighbor identification so that a node upon receiving a packet knows which neighbor sends that packet, which can be achieved via link-layer authentication. (iii) We assume that each node in the network is pre-installed with a unique public key certified by a trusted Certification Authority, by which each node can obtain and verify any other node’s public key. In addition, when needed, a source node $S$ can set up a shared secret key $K_{si}$ with router $f_i$ using an existing key exchange protocol, e.g., Diffie-Hellman [20] as in Passport [39].

3 ShortMAC Overview

We first highlight the challenges of a secure fault localization protocol design, and then present our key ideas.
nodes, which is susceptible to framing attacks) for selecting its own routing path. Note that the detection bits. The routers verify the authentication bits in each received packet based on the symmetric epoch key shared with the source node of the received packets until the authentication key is later disclosed (longer than the end-to-end path latency). In addition, some recently proposed multicast/broadcast authentication schemes still require considerable communication overhead (e.g., up to hundreds of bytes per packet [41]) or multiple rounds for authenticating a message [19].

**2) Not necessary:** On the other hand, as we aim to limit the damage the adversary can inflict for a lower-bound guarantee on data-plane packet delivery, perfect per-packet authenticity is not necessary. Instead, our goal only requires the authenticity of a very large fraction of data packets.

**ShortMAC approach.** Based on these observations, we propose ShortMAC, a light-weight scheme trading per-hop overhead with the adversary’s ability to forge a few (e.g., tens) more packets. More specifically, in ShortMAC, the source attaches to each packet a *k*-bit random nonce, called *k*-bit MAC, for each node on the path, where the parameter *k* is significantly less than the length of a typical MAC. To construct the *k*-bit MAC for *f*<sub>i</sub>, we use a Pseudo-Random Function (PRF) which constructs a *k*-bit string as a function of the packet *m* and key *K*<sub>si</sub> shared between *S* and *f*<sub>i</sub>. We rely on the result that the output *k*-bit MAC is indistinguishable from a random *k*-bit string to any observer without the secret key *K*<sub>si</sub> [44]. Each router *f*<sub>i</sub> maintains two path-specific counters *C*<sub>i</sub><sup>good</sup> and *C*<sub>i</sub><sup>bad</sup> to record the numbers of received packets along that path with correct and incorrect *k*-bit MACs, respectively, in the current epoch. Such a scheme considerably reduces communication overhead compared to attaching entire MACs while retaining high security assurance and communication throughput, as shown later.

### 3.3 ShortMAC Packet Authentication

It is important that the integrity of the source’s data packets is ensured, in order to detect malicious packet modification during the forwarding path. Our approach is to turn packet count into a reliable measure of packet content so that routers only need to store space-efficient counters; otherwise a malicious router can always perform packet modification attacks without affecting the counter values, or inject bogus packets on behalf of the source to manipulate the counter values of the reporting routers (as is the case in Fatih [45] and AudIt [10]). Hence, we reduce the problem to how the source node can authenticate its packets to all the routers in the path. However, traditional broadcast authentication schemes provide high authenticity for each *single* message, which is neither necessary nor practical in our setting where the messages are line-rate packets:

1) **Not practical:** On one hand, perfectly ensuring the authenticity of every *single* data packet introduces high overhead in a high-speed network. For example, digital signatures or one-time signatures for per-packet authentication is either computationally expensive or bandwidth-exhaustive, and using amaritized signatures would either fail in the presence of packet loss [22] or incur high communication overhead [55]. Attaching a Message Authentication Code (MAC) for each node along the path (as is used by Avramopoulos et al. [12]) is too bandwidth-expensive (e.g., reserving a 160-bit MAC space for each hop). In addition, TESLA authentication [50] would require time synchronization and routers to cache the received packets until the authentication key is later disclosed (longer than the end-to-end path latency). Finally, some recently proposed multicast/broadcast authentication schemes still require considerable communication overhead (e.g., up to hundreds of bytes per packet [41]) or multiple rounds for authenticating a message [19].

**2) Not necessary:** On the other hand, as we aim to limit the damage the adversary can inflict for a lower-bound guarantee on data-plane packet delivery, perfect per-packet authenticity is not necessary. Instead, our goal only requires the authenticity of a very large fraction of data packets.
malicious packets. For each injected packet, \( f_2 \) needs to forge 2-bit MACs for both \( f_3 \) and the destination that “authenticate” the fabricated data content. However, since \( f_2 \) does not know the corresponding epoch symmetric keys of \( f_3 \) and the destination, \( f_2 \) can only guess the 2-bit MACs for its injected packets. Since the 2-bit MACs produced by the PRF are indistinguishable from random bits, \( f_2 \) can correctly guess each 2-bit MAC with probability \( \frac{1}{4} \). Since \( f_2 \) must guess two correct MACs, each forged packet will be accepted by the destination with probability \( \frac{1}{16} \). Suppose next that 26 of the 100 2-bit MACs that \( f_2 \) forged for \( f_3 \) happen to be valid with respect to the fabricated data content. \( f_3 \) thus computes \( C_{bad}^{good} = 100 - 26 = 74 \) and \( C_{good}^{good} = 997 - 100 \) (dropped legitimate packets) + 26 (bogus but undetected packets) = 923. Similarly, we can analyze the counters for the destination in Figure 4, assuming that 7 out of the 26 received bogus packets happen to be consistent with their 2-bit MACs at the destination.

3.5 Fault Localization and Guaranteed \( \theta \)

At the end of each epoch, routers and the destination report their counter values to the source using a secure transmission approach (details in Section 4). The source can identify excessive packet drops between \( f_m \) and \( f_{m+1} \) if the \( C_{m+1}^{good} \) value of \( f_m \) is abnormally lower than that of \( f_m \) based on the drop detection threshold \( T_{ds} \), which is carefully set based on the customized acceptable per-link drop rate. Moreover, this scheme can successfully bound the total number of spurious packets with fabricated \( k \)-bit MACs that the adversary can inject, because at least one of the downstream recipient routers will detect the inconsistency of the \( k \)-bit MACs with a non-trivial probability, thus having a non-zero \( C_{bad} \) value. For example in Figure 4, although \( f_2 \) can claim any values for its own counters, no matter what values \( f_2 \) claims, the source can notice excessive packet loss and a large number of fake packets either between \( f_1 \) and \( f_2 \), or \( f_2 \) and \( f_3 \). Hence one of \( f_2 \)'s malicious links will be detected by the source.

Once the source \( S \) bypasses all malicious links identified by ShortMAC, \( S \) can find a working path with no excessive packet corruption at any link, thus being assured with a guaranteed successful forwarding rate \( \theta \).

4 ShortMAC Details

In this section we describe the ShortMAC protocol in detail, where the source can either guarantee that a high fraction \( \theta \) of its data has been correctly forwarded if no malicious activities are detected, or can bypass the faulty links and find a working path after exploring a number of paths linear to the number of faulty links.\(^\text{1}\) In the following we first formalize the ShortMAC packet format and then detail the protocol.

\(^1\)It has been proved that forwarding fault localization protocols protocols can only identify faulty links, rather than identifying the nodes [16]. However,

4.1 ShortMAC Packet Format

A source node \( S \) adds a trailer to each data packet it sends:

\[
\text{trailer} = (SN, M_1, \ldots, M_d),
\]

where \( SN \) is a per-path sequence number to make each IP packet unique along the same path to prevent packet replay attacks, and \( M_i \) denotes the \( k \)-bit MAC computed for \( f_i \), which is constructed in a recursive way starting from \( f_d \):

\[
M_d \leftarrow \text{PRF}_{K_{sd}}(IP_{invar}||SN||TTL_d)
\]

\[
M_{d-1} \leftarrow \text{PRF}_{K_{(d-1)d}}(IP_{invar}||SN||TTL_{d-1}||M_d)
\]

\[
\ldots\ldots
\]

\[
M_i \leftarrow \text{PRF}_{K_{si}}(IP_{invar}||SN||TTL_i||M_{i+1}||\ldots||M_d)
\]

where “\( || \)” denotes concatenation and \( \text{PRF}_{K_{sd}}(\cdot) \) denotes a PRF keyed by the epoch symmetric key \( K_{sd} \) shared between \( S \) and \( f_1 \). As previously discussed, the output of this PRF can be guessed correctly with probability no larger than \( \frac{1}{2^k} \) by anyone without the secret key \( K_{si} \) [44]. In addition,

1) \( IP_{invar} \) denotes the invariant portion of the original IP packet that should not be changed at each router during forwarding, including the packet payload and IP headers excluding variable fields such as TTL, RecordRoute IP option, Timestamp IP option etc. If these invariant fields are unexpectedly changed during forwarding, each downstream router can detect inconsistency between the (modified) packet and embedded \( k \)-bit MAC with a non-trivial probability \( 1 - \frac{1}{2^k} \) and thus increase its \( C_{bad} \) counter.

2) \( TTL_i \) denotes the expected TTL value at router \( i \). Without authenticating this field in the \( k \)-bit MAC, a malicious router can strategically lower the TTL field to cause packet drop at a remote downstream router due to zero TTL value, thus performing framing attacks as illustrated in Figure 5.

3) \( M_i \) also authenticates the downstream \( M_{i+1}, \ldots, M_d \), so that if a malicious router \( f_m \) changes any of these downstream \( k \)-bit MACs, \( f_i \) can observe the inconsistency in \( M_i \) with a probability \( 1 - \frac{1}{2^k} \) and increase its \( C_{bad} \) value. Otherwise, the protocol is vulnerable to framing attacks as Figure 5 shows.

4.2 Protocol Details

Formally, ShortMAC consists of Request, Report, Identify, Bypass and Send stages, described as follows.

Stage 1: Request with hop-by-hop reliable transmission

At the end of each epoch (i.e., after sending every \( N \) data packets), the source \( S \) will send a digitally signed path setup request packet (denoted by request) along a path \( p = (f_1, \ldots, f_d) \) selected to be used in the next epoch:

\[
\text{request} = (S, EpNum, p, \text{Sig}_s[H(S||EpNum||[p])])
\]

where \( S \) is the source node ID, \( p \) is the path selected by \( S \), \( EpNum \) is a unique sequence number for each epoch to prevent replay attacks, and \( \text{Sig}_s \) is a digital signature with \( S \)'s private key. Also, since each field in the signature has an uniform size, no shift attack is possible. This request asks each ever, given that a malicious node has a limited degree, after bypassing all its malicious links the source can eventually bypass that node.
When the check fails at $i$, $S$ will immediately identify $l_i$ as faulty due to the use of reliable hop-by-hop transmission and Onion ACK. For example, if $S$ receives no report it will identify $l_j$ as faulty ($j = 1$).

In addition, $S$ extracts $R_1, \ldots, R_j$ in turn which include the $C_i^{bad}$ and $C_i^{good}$ values. A non-zero $C_i^{bad}$ implies the existence of malicious packet injection between $f_i$ and $S$. However, $S$ cannot blame $l_i$ simply whenever $C_i^{bad} > 0$, say, $C_i^{bad} = 1$. A possible scenario is that a malicious node $f_{i-2}$ injects a fake packet, but the $k$-bit MAC intended for $f_{i-1}$ “happens” to be consistent with the fake packet at benign node $f_{i-1}$ (e.g., when $k = 2$, this can happen with probability 0.25). In this case, $f_{i-1}$ will forward the fake packet which $f_i$ may detect and thus increase $C_i^{bad}$. Similarly, due to natural packet loss, $S$ cannot simply accuse link $l_i$ when $C_i^{good} < C_i^{good}$. Therefore, we leverage two detection thresholds $T_{in}$ and $T_{dr}$, where $T_{in}$ is the injection detection threshold for the number of injected packets on each link, and $T_{dr}$ is the drop detection threshold for the fraction of dropped packets on each link. As we will show in Section 6, these thresholds reduce false positives while limiting the adversary’s ability to corrupt packets and ensuring a lower bound on the successful packet forwarding rate. The detection thresholds are used in the following two detection procedures:

1. **check-injection**: $S$ checks the extracted $C_i^{bad}, C_i^{bad}, \ldots, C_i^{bad}$ in order. If $C_i^{bad} \geq T_{in}$ for some $i$, then $S$ identifies $l_i$ as faulty and the check-injection procedure stops.
2. **check-dropping**: If no fault is detected by check-injection, $S$ further checks the extracted $C_1^{good}, C_2^{good}, \ldots, C_j^{good}$ values

---

2We can expect a reasonable upper bound of link latency in benign cases, which can be used to compute the maximum round trip time on-the-fly according to the hop count from $f_i$ to the destination. The notion of such a timer is first introduced by Avrampolous et al. [12].

3$S$ can verify if a certain retrieved report $R_i$ is valid by checking the embedded message integrity code $H_{K_{S_i}}(R_i || A_{i+1})$. 

---

**Figure 5**: Illustration of framing attacks. If $M_i$ in Eq.(2) had not authenticated the TTL field, a malicious router can strategically change the TTL to an arbitrary value to frame others. For example, $f_1$ maliciously changes the TTL value in the packets to 2, instead of decrementing it by 1. This causes the packets to be dropped at $f_3$, thus framing the link between $f_2$ and $f_3$. Similarly, if $M_i$ in Eq.(2) had not authenticated the downstream $k$-bit MAC field, $f_1$ can maliciously modify $M_3$ in the packets which causes $f_3$ to detect inconsistent $M_3$ with a non-trivial probability and increase $C_3^{bad}$, thus framing the link between $f_2$ and $f_3$. **Case 1** If $f_1$ receives an ACK $A_{i+1}$ from neighbor $f_{i+1}$ before the timer expires, $f_2$ further commits $R_i$ into a new ACK $A_i$ by combining the received $A_{i+1}$ via an Onion ACK approach:

$$A_i = (f_i, \text{Sig}_i[R_i, A_{i+1}, H_{K_{S_i}}(R_i || |A_{i+1}|)])$$

$H_{K_{S_i}}(\cdot)$ denotes a message authentication code computed with $K_{S_i}$. Then, $f_i$ forwards $A_i$ to $f_{i-1}$ toward $S$.

**Case 2** If $f_1$ receives no ACK packet from $f_{i+1}$ before the timer expires, $f_i$ will initiate a new ACK with its local report and send it to $f_{i-1}$. Later we show the source node will conclude that a fault has occurred on link $l_{i+1}$, since reliable hop-by-hop transmission is used for the requests and ACKs.

The Onion ACK prevents the adversary from selectively dropping the requests or the reports of certain routers and framing a benign link [59], as illustrated in Strawman 1 in Section 3.1. Onion ACK eliminates such attacks since all the reports are combined and authenticated in one ACK packet at each hop so that a malicious node can only drop or modify the onion report from its immediate neighbors. If $f_{i-1}$ drops or modifies the received request or Onion ACK, one of its adjacent links will be pinpointed by the source node, in the identify stage described below.

**Stage 3: Identify**
Upon receiving an Onion ACK $A_1$ from $f_1$, $S$ first iteratively retrieves $A_1, A_2, \ldots$ in the order of the certified public keys of $f_1, \ldots, f_{j+1}$, until it either completes at $d$ or fails at $j (j \neq d)$.

When the check fails at $j (j \neq d)$, $S$ will immediately identify $l_j$ as faulty due to the use of reliable hop-by-hop transmission and Onion ACK. For example, if $S$ receives no report it will identify $l_1$ as faulty ($j = 1$).
in order. If \( C_{i}^{good} < (1 - T_{dr}) \cdot C_{i-1}^{good} \) (with \( C_{0}^{good} = N \)) holds for certain \( i \), then \( S \) identifies \( l_{i} \) as faulty and the check-dropping procedure terminates.

**Stage 4: Bypass and Send**

If Stage 2 outputs any malicious link \( l_{m} \), \( S \) selects a new path excluding the previously detected malicious links and goes back to Stage 1. Since a malicious router has a limited degree \( \omega \), by bypassing all malicious links, \( S \) can eventually avoid a malicious node within a constant time \( \omega \). Once a path with no previously detected malicious links is established, the source node sends its packets with ShortMAC authentication shown in Eq.(2); and each node \( f_{i} \) examines its corresponding \( k \)-bit MAC \( M_{i} \) in each packet to increase \( c_{i}^{good} \) or \( c_{i}^{bad} \) accordingly. In addition, each router remembers the last seen per-path \( SN \) embedded in the packets as shown in Eq.(1), and discards packets with older \( SN \) in that path.

**5 Security Analysis**

This section discusses ShortMAC’s security against data-plane attacks by malicious routers. Section 6 provides theoretical proofs on ShortMAC’s security. In our adversary model, a malicious router can drop and inject data packets, requests and ACKs, and can send arbitrary counter values in its reports. We show that ShortMAC is secure against a single malicious router (say, \( f_{m} \)) as well as multiple colluding nodes.

**Corrupting data packets.** Dropping legitimate data packets by \( f_{m} \) will cause a discrepancy of the counter values between \( f_{m} \) and its neighbors. For example, if \( f_{m} \) correctly reports \( C_{m}^{good} \), then \( C_{m}^{good} - C_{m+1}^{good} \) will exhibit a large discrepancy; if \( f_{m} \) reports a lower \( C_{m}^{good} \), then \( C_{m}^{good} - C_{m-1}^{good} \) will exhibit a large discrepancy. Hence, either \( l_{m-1} \) or \( l_{m} \) will become suspicious. Moreover, if \( f_{m} \) injects/modify packets, \( M_{m+1} \) will be inconsistent at \( f_{m+1} \) with high probability and cause a non-zero \( C_{m+2}^{bad} \). Hence, both dropping and injection attacks can be detected as long as the source can learn the correct counter values in the ACK packets sent by the nodes between \( f_{m} \) and the destination, which is described next.

**Corrupting ACKs or requests.** Since the ACKs or requests are digitally signed, \( f_{m} \) cannot impersonate others or modify the content of the ACKs or requests. \( f_{m} \) cannot selectively drop the ACK reports due to the use of Onion ACK. Instead, \( f_{m} \) can only drop the ACKs or requests from its immediate neighbors, which will again harm its incident links.

**Replay, reorder, and traffic analysis attacks.** To prevent replay and reorder attacks, each packet contains a per-path sequence number \( SN \) in Eq.(1) and each router discards packets with older \( SNs \). Hence, the replayed and reordered packets will be dropped at the next-hop benign node without influencing the counter values of benign nodes. Note that because ShortMAC runs on a per-path basis and a \( SN \) is a per-path sequence number providing natural isolation across different paths, packets along the same path are expected to maintain the same order during forwarding as they were sent by the source in benign cases. On the other hand, if \( f_{m} \) falsely reports a large \( SN \), \( f_{m+1} \) will drop the subsequent packets and \( l_{m} \) will be identified as malicious due to its high packet drop rate. Moreover, the per-path \( SN \) can prevent ShortMAC from traffic analysis attacks, where \( f_{m} \) attempts to find out the correct \( k \)-bit MAC of a packet \( m \) by re-sending \( m \) with different \( k \)-bit MACs and observing whether the next-hop \( f_{m+1} \) forwards the packet. Such traffic analysis is ineffective because \( f_{m+1} \) can detect packets with the same \( SN \) and each packet is unique due to the use of the per-path \( SN \), and thus \( f_{m} \) cannot send the same packet \( m \) with only the \( k \)-bit MAC changed.

**DoS attacks.** A malicious router \( f_{m} \) may launch bandwidth Denial-of-Service (DoS) attacks by generating an excessive amount of packets. However, this attack can be reduced to a packet injection attack and will be reflected by \( C_{m+1}^{bad} \). Also, \( f_{m} \) may launch computational DoS attacks by generating excessive requests with bogus signatures, hoping to deplete other routers’ computational resource by expensive signature verifications. However, during the transmission of a request message, at each hop the router will check if the signature is indeed from the source; if the verification fails, the router immediately drops the message without forwarding it. In either flooding or computational DoS attack, \( f_{m} \) can only overload its immediate neighbor \( f_{m+1} \), thus incorporating \( f_{m} \)’s adjacent link \( l_{m} \) by introducing a high drop or injection rate on \( l_{m} \).

A malicious router may also attempt to open many bogus flows with spoofed sources to exhaust other routers’ state. We can borrow existing work to provide source accountability and reliable flow/path identification [8, 58]. Also note that in our adversary model we consider malicious routers which threaten the communication between benign hosts. We do not consider DDoS attacks launched by malicious hosts (botnets), which other researchers have strived to defend against [38, 40, 58]. Hence in our problem setting, a link under DDoS attacks thus exhibiting high loss rate is simply considered a faulty link under our adversary model. Meanwhile, the path setup phase in ShortMAC can be naturally integrated with capability schemes [58] for DDoS limiting, and the per-path counters may also be used for per-path rate limiting.

**Collusion attacks.** Each of the colluding routers can commit any of the misbehavior discussed above. We can prove by induction that in any case, one of the malicious links of one of the colluding nodes is guaranteed to be detected. A proof sketch is given below.

**Proof:** First consider the base case where two nodes \( f_{m} \) and \( f_{m'} \) (\( m < m' \)) collude. Without loss of generality:

1. In the first case where \( f_{m} \) and \( f_{m'} \) are not adjacent (i.e., \( m' > m + 1 \)), the above security analysis still applies to \( f_{m} \) and one of \( f_{m} \)’s malicious links will become suspicious if \( f_{m} \) misbehaves. This is because if \( f_{m} \) commits the above attacks, such misbehavior will be reflected in the benign neighbor \( f_{m+1} \)’s counters which cannot be biased by \( f_{m'} \).

2. In the other case where \( f_{m} \) and \( f_{m'} \) are adjacent (\( m' = m + 1 \)), these two nodes can be regarded as one single “virtual” malicious node \( F_{m} \) with neighbors \( f_{m-1} \) and \( f_{m+2} \), as shown in Figure 6. (i) If \( f_{m} \) or \( f_{m+1} \) drops packets, a discrepancy will exist between \( C_{m-1}^{good} \) and \( C_{m+2}^{good} \), no matter what values of \( C_{m-1}^{good} \) and \( C_{m+2}^{good} \). (ii) If \( f_{m} \) or \( f_{m+1} \) injects packets, \( C_{m-2}^{bad} \) will become non-zero and make \( l_{m+1} \) suspicious. In any case, an adjacent link of \( F_{m} \) (which must be a malicious link) will become suspicious.

We now consider the general case with \( n \) colluding nodes. We can first group adjacent colluding nodes into virtual malicious nodes as in Figure 6, resulting in non-adjacent malicious nodes (including virtual malicious nodes). Then we can show
that non-adjacent malicious nodes can be detected based on the above analysis.

However, note that although the colluding attackers cannot corrupt packets more than the same thresholds as an individual attacker on any single link, they can choose to distribute packet dropping across multiple links. In this case, the total packet drop rate by colluding attackers increases (and is still bounded) linearly to the number of malicious links in the same path, as analyzed in Section 6.

6 Theoretical Results

We first prove the \((N, \delta)\)-data-plane fault localization (Definition 4) and \((\alpha, \beta, \delta)\)-forwarding security of ShortMAC (Definition 6), which in turn yield the \(\theta\)-guaranteed forwarding correctness (Definition 5). Proofs of the lemmas and theorems are provided in the Appendix.

**Lemma 1 Injection Detection:** Given the bound \(\delta\) on detection false negative and false positive rates, the injection detection threshold \(T_{in}\) can be set to \(T_{in} = \frac{2\ln(\frac{2d}{\rho})}{\delta}\), where \(d\) is the path length and \(q = \frac{2^{b} - 1}{2^{b}}\) is the probability that a fake packet will be inconsistent with the associated \(k\)-bit MAC.

The number of fake packets \(\beta\) an adversary can inject on one of its malicious links without being detected is limited to:

\[
\beta = \frac{T_{in}}{\delta} + (\frac{\ln(d)}{\delta})^{2} + b\ln T_{in} + \frac{\ln(\frac{1}{\rho})}{\delta} + \frac{\ln(\frac{1}{\beta})}{\delta}.\]

In Lemma 2, we derive \(N\), the number of data packets a source needs to send in one epoch to bound the detection false positive and false negative rates below \(\delta\). Due to natural packet loss, a network operator first sets an expectation based on her domain knowledge such that any benign link in normal condition should spontaneously drop less than \(\rho\) fraction of packets. We first describe how the drop detection threshold \(T_{dr}\) is set when \(N\) and \(\delta\) are given. Intuitively, by sending more data packets (larger \(N\)), the observed per-link drop rate can approach more closely its expected value, which is less than \(\rho\); otherwise, with a smaller \(N\), the observed per-link drop rate can deviate further away from \(\rho\), and the drop detection threshold \(T_{dr}\) has to tolerate a larger deviation (thus being very loose) in order to limit the false positive rate below the given \(\delta\). On the other hand, a small \(N\) is desired for fast fault localization. We define **Detection Delay** to be the minimum value of \(N\) given the required \(\delta\).

**Lemma 2 Dropping Detection and \((N, \delta)\)-Fault Localization:** Given the bound \(\delta\) on detection false positive and negative rates and drop detection threshold \(T_{dr}\), the detection delay \(N\) is given by:

\[
N = \frac{\ln(\frac{1}{1-\delta})}{\ln(1-\delta) T_{dr}},\]

where \(d\) is the path length. Correspondingly, the fraction of packets \(\alpha\) an adversary can drop on one of its malicious links without being detected is limited to:

\[
\alpha = 1 - (1 - T_{dr})^{2} + \frac{\beta}{N(1 - T_{dr})^{2}}.\]

Table 1: Theoretical comparison with PAAI-1 [59] and Stat. FL [16] (including two approaches SSS and sketch). Note that the details of sketch are not provided in the published paper [16], and the full version of [16] does not present the explicit bounds on detection delay. The above figures for sketch are estimated from their earlier work [25]. In this example scenario, \(d = 5, \beta = 1\%\), \(\rho = 0.5\%\), \(T_{dr} = 1.5\%\), a symmetric key is 16 bytes, and ShortMAC uses 2-bit MACs. PAAI-1 specific parameters include the “packet sampling rate” set to 0.01, the end-to-end latency set to 25 ms, the source’s sending rate set to 10^6 packets per second, each packet hash is 128 bits.

In practice, \(T_{dr}\) can be chosen according to the expected upper bound \(\rho\) of a “reasonable” normal link loss rate such that a drop rate above \(T_{dr}\) is regarded as “excessively lossy”.

**Theorem 1 Forwarding Security and Correctness** Given \(T_{dr}\), \(\delta\), and path length \(d\), we can achieve \((\alpha, \beta, \delta)\)-forwarding security where \(\alpha\) is given by Lemma 2 and \(\beta\) is given by Lemma 1. We also achieve \((\Omega, \theta)\)-Guaranteed forwarding correctness with \(\Omega\) equal to the number of malicious links in the network, and \(\theta = (1 - T_{dr})^{2} - \frac{\delta^{2}}{\delta}\), where \(N\) is derived from Lemma 2.

In Theorem 2, we analyze the protocol overhead with the following three metrics (we further analyze the throughput and latency in Section 8 via real-field testing):

1) The **communication overhead** is measured by the fraction of extra packets that need to be transmitted by each router.

2) The **marking cost** is defined as the number of extra bits embedded into each data packet (sent by the source).

3) The **per-path state** is defined as the per-path extra bits that a router stores for the security protocol in fast memory needed for per-packet processing.

**Theorem 2 Overhead:** For each router, the communication overhead is one packet for each epoch of \(N\) data packets. The marking cost is \(k \cdot d\) bits for the \(k\)-bit MACs where \(d\) is the path length. The per-path state comprises one \(\log\) \(N\)-bit \(C_{good}\) counter, one \(\log\) \(\beta\)-bit \(C_{bad}\) counter, one \(\log\) \(N\)-bit last-seen per-path \(SN\), and one epoch symmetric key.

Though Barak et al. proved the necessity of per-path state for a secure fault localization protocol [16], such a minimal per-path state in ShortMAC is viable for both intra-domain networks with tens of thousands of routers and the Internet AS-level routing among currently tens of thousands of ASes.

**Theoretical comparison of example results.** We compare the above theoretical results of ShortMAC with two recent proposals, PAAI-1 [59] and Stat. FL [16] (including two approaches SSS and sketch). Table 1 presents the numeric figures using an example parameter setting for intuitive illustration, while ShortMAC presents similarly distinct advantages in other parameter settings. In this example scenario shown in the table, the guaranteed data-plane packet delivery

---

4The buffering space needed for the Onion-ACK construction of report messages in ShortMAC is not a major concern, as the Onion-ACK is computed only once every epoch, which can be buffered in off-chip storage.
ratio is $\theta = 92\%$. The communication overhead for a router in ShortMAC is 1 extra ACK for every $3.8 \times 10^4$ data packets in an epoch; the marking cost is 10 bits for the 2-bit MACs in a path with 5 hops, and the per-path state at each router is 21 bytes (16-byte symmetric key, 2-byte $C^{good}$, 1-byte $C^{bad}$, and 2-byte per-path SN).

7 SSFNet-based Evaluation

We implement a prototype of ShortMAC on the widely used SSFNet simulator [6]. Since the communication and storage overhead of ShortMAC are both straightforward (see the theoretical results in Section 6), we launch extensive simulations mainly to study the detection delay and security of ShortMAC under various settings and with comparison to other recently proposed schemes (Section 8 further investigates its throughput and latency). These experimental results not only consolidate our theoretical findings, but more importantly also complement the theoretical results by shedding light on the average-case performance (while the theoretical results are derived for the worst case due to multiple mathematical relaxations such as Hoeffding inequality), and by showing ShortMAC security against other attack strategies as constant dropping/injection rates used in the theoretical analysis.

Evaluation scenario and attack pattern Recall that ShortMAC runs on a per-path basis (routers maintain state for each path separately), which provides a natural isolation across paths. Therefore in our evaluation we focus on a path with 6 hops (routers $f_1, f_2, f_3, f_4, f_5$ and the destination $f_6$), while we also experimented with other path lengths and obtained the same observations. We simulate both an (i) independent packet corruption pattern where a malicious node drops/injects each packet independently with a certain drop/injection rate, and (ii) random-period packet corruption pattern as shown in Figure 7. In both attack patterns, we control and vary the average packet drop/injection rates. Our results show that both attack patterns yield similar observations, thus we only show the results for the independent packet corruption pattern. Since SSFNet does not exhibit observable natural packet loss in normal cases, we intentionally infuse natural packet loss rate $\rho$ for each link to make the detection harder (easier for causing false positives). Since Section 5 elaborates ShortMAC security against colluding attacks, we only present the results for a single malicious node, and show the result variations with different malicious node positions in the end. For each simulation setting, we run the simulation 1000 times and present the average results.

Against various dropping attacks Figure 8 depicts the detection delay $N$ and error rates $\delta$ with per-link natural loss rate $\rho$ as 0.5%, drop detection threshold $T_{dr}$ as 1%, and a stealthy malicious drop rate of 2%. We see that: (i) even against stealthy dropping attacks with a dropping rate as low as 2%, ShortMAC can successfully localize a faulty link in < 2000 packets with an error rate $\delta < 1\%$, which is tens of times faster than the worst-case theoretical bound (Lemma 2). (ii) In addition, the false negative rate is always no lower than the false positive rate, because when false positive occurs (a benign link being falsely detected) the actual faulty link must have evaded detection for the current epoch (ShortMAC detects only one “faulty” link each epoch). (iii) When $N$ is large, the false positive and negative rates are almost identical, because a difference between these two rates is created only when no faulty link is detected (false positive is 0 while false negative is non-zero), which is unlikely to happen when $N$ is large.

Figure 9 further shows the different detection delays with different natural packet loss rates, from which we can see that larger $|T_{dr} - \rho|$ yields better detection accuracy and lower detection delay.

Against various injection attacks We let the malicious node $f_j$ inject packets at stealthy rate of 2% (relative to the legitimate packet sending rate). We generate the injected packets in SSFNet by flipping a single bit in the original packets. Figure 10 shows the corresponding results, from which we can see the error rates stay below 1% in a few hundred packets, indicating that an adversary can only inject up to around ten packets without being detected. We further investigate the effects of using different lengths of $k$-bit MICs; and Figure 11 shows that the detection delay and error rate dramatically diminish as $k$ increases.

Against combined attacks We launch various dropping and injection attacks simultaneously to validate ShortMAC’s security and efficiency. We experimented with various combinations of dropping and injection attack strategies presented before and obtained similar results to Figure 12 where the dropping/injection rates are chosen between 2% and 5%. In addition to the demonstration of ShortMAC’s security and efficiency, we also observe that the detection delay is majorly determined by the dropping detection process, which is much slower than the injection detection process. This also indicates a malicious node cannot gain any advantage (and actually can only harm itself) by injecting bogus packets in an attempt to bias the counter values to evade detection.

Variance due to different malicious node positions To investigate the performance variance of ShortMAC due to different positions where the malicious node resides, we create a longer path with 6 forwarding nodes $f_1, f_2, \ldots, f_6$. We place the malicious node at each position (1 to 6) in turn, and obtain the corresponding detection delays needed to limit the error rates below 1%. Figure 13 shows one representative scenario where we set both dropping and injection rates to 5%. We can see that (i) the dropping detection delay increases linearly when the malicious node is farther away from the source. This is because in the ShortMAC detection process, the source always inspects the closer links first and stops once the first “faulty” link is detected; and the false positive rate thus increases when more links exist between the source and the malicious node due to natural packet loss on each link. (ii) Furthermore, the slope for this linear increase can be obtained from Lemma 2 by taking a partial derivative on hop count $d$. (iii) In contrast, the injection detection delay exhibits little variance (cannot be seen from the figure since the detection delay is determined by
Figure 8: Detection delay $N$ and false positive (fp) and negative (fn) rates in dropping attacks with malicious drop rate set to 2%, per-link natural loss rate $\rho$ set to 0.5%, per-link drop detection threshold $T_{dr}$ set to 1%, and $f_3$ as the malicious router.

Figure 9: Effects of natural loss rates on detection delay $N$ and false positive and negative rates $\delta$, with malicious drop rate 5% at $f_3$ and per-link drop detection threshold $T_{dr}$ = 1%.

Figure 10: Detection delay $N$ and false positive and negative rates in injection attacks with malicious injection rate set to 2% using 2-bit MICS, natural loss rate set to 0.5%, per-link drop detection threshold $T_{dr}$ set to 1%, and $f_3$ as the malicious router.

Figure 11: Effects of different $k$-bit MIC lengths on detection delay $N$ and false positive and negative rates $\delta$.

Figure 12: Combined attacks at malicious router $f_3$ against 2-bit MIC with varying packet dropping and injection rates. "drop $\rho$ inject $q$" denotes the use of $\rho$% dropping rate and $q$% injection rate.

Figure 13: Variance on detection delay $N$ in dropping attacks with required error rate $\delta < 1\%$, detection threshold $T_{dr} = 1\%$, per-link natural packet loss rate $\rho = 0.5\%$, and both malicious dropping and injection rates set to 5%.

<p>|</p>
<table>
<thead>
<tr>
<th>ShortMAC</th>
<th>FullACK</th>
<th>PAAI-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detect. delay</td>
<td>20 sec</td>
<td>20 sec</td>
</tr>
<tr>
<td>Communication</td>
<td>0.01%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 2: Comparison of ShortMAC, FullACK, and PAAI-1 with a source send rate of 100 packets per second, based on the results from Figure 14.

Figure 14: Comparison with the heavy-weighted fullACK scheme and efficient PAAI-1 scheme [59], with $\rho = 0.5\%$, $T_{dr} = 1\%$, and 5% malicious drop rate. PAAI-1 samples $\frac{1}{10}$ of total packets. The successful rate is defined as $1 - \max\{\text{false positive rate, false negative rate}\}$.


Comparison with recently proposed protocols For comparison, we simulate both the recently proposed FullACK and PAAI-1 [59] schemes which present the lowest detection delays to date. FullACK is a heavy-weight fault localization protocol which requires an Acknowledgment packet from every forwarding node for every packet the source sent. In contrast, PAAI-1 employs packet sampling and only requires acknowledgments for the securely sampled packets, to reduce communication overhead while retaining desired detection delay. Since both FullACK and PAAI-1 only consider packet dropping attacks, we compare their dropping detection delays along a path with 5 forwarding nodes and $f_3$ as the malicious node. Figure 14 shows the results of an example scenario where per-link natural packet loss rate $\rho = 0.5\%$ and drop detection threshold $T_{dr} = 1\%$. To make the comparison clear, we use a metric of successful rate, which equals to $1 - \max\{\text{false positive rate, false negative rate}\}$. According to the results, the detection delay to achieve a successful rate $> 99\%$ for ShortMAC, FullACK, and PAAI-1 is 2000, 2000, and $5 \times 10^4$ respectively. With an example source sending rate of 100 packets per second, Table 2 further shows their detection delays in time and the extra communication overhead.

8 Linux Prototype and Evaluation

We demonstrate that the efficient cryptographic operations of ShortMAC, even implemented in user-space un-customized desktop OS, incur little communication degradation and negligible additional latency. It has also been demonstrated that using modern hardware implementation and acceleration the speed of PRF functions can be fundamentally improved [31].
Implementation We implement ShortMAC source and destination nodes as user-space processes running on Ubuntu 10.04 32-bit Desktop OS, listening to application packets via TUN/TAP virtual interfaces and appending k-bit MICs to the packets. We also implement ShortMAC routers using the Click Modular Router [30] running on Ubuntu 10.04 32-bit Desktop OS, which verify the k-bit MICs in each packet at each hop. To approach the realistic performance of commercial-grade routers, we implement the above elements on off-the-shelf servers with an Intel Xeon E5640 CPU (four 2.66 GHz cores with 5.86 GT/s QuickPath Interconnect, 256KB L1 cache, 1MB L2 cache, 12MB L3 cache, and 25.6 GB/s memory bandwidth) and 12G DDR3 RAM. The servers are equipped with Broadcom NetXtreme II BCM5709 Gigabit Ethernet Interface Cards.

Evaluation methodology We run the ShortMAC routers in user-space to evaluate ShortMAC’s effects on communication throughput, which are mainly determined by ShortMAC’s computational overhead due to one PRF operation to verify the corresponding k-bit MIC. We also test the throughput and the breakdown of the computational latency on a ShortMAC source node. Since source node needs to compute multiple k-bit MICs, and update TCP/IP checksum after appending the k-bit MICs in each packet, it is highly possible to become the bottleneck of ShortMAC system.

We utilize the widely used Netperf benchmark [4] for our ShortMAC throughput evaluation, and write our own micro-benchmark for accurate latency evaluation. We evaluate ShortMAC with varying packet sizes by configuring the interface Maximum Transmission Unit (MTU) sizes. We evaluate the throughput of a ShortMAC router and a ShortMAC source separately to better illustrate the throughput of each component, while the end-to-end path throughput can be easily derived by taking the minimum throughput of the two evaluation results. Then we evaluate the end-to-end latency with different path lengths. To optimize the epoch path setup speed, each router exchanges a master symmetric key with every other node in the network via their public keys during ShortMAC protocol bootstrapping, and uses this master symmetric key for later exchanging epoch symmetric keys for authenticating the request (3) and report (4) packets, which are shown in Section 4.2.

Router throughput with different PRF implementations We first evaluate the throughput of a user-level ShortMAC router with different PRF implementations using HMAC-SHA1 [32] and the two recently proposed, fast, and rigorously proved secure MAC algorithms, VMAC [33] and UMAC [51]. In the experiments the ShortMAC router connects a source machine and a destination machine, with the source sending TCP packets via Netperf as fast as possible to the destination to stress-test the router. For comparison, we use the Click router throughput without ShortMAC operations as the base line. The ShortMAC router runs as a single user-space process without exploiting parallelism, which can already match up the base line speed as shown below.

Figure 15 depicts the results with packet sizes ranging from 100 bytes to 1500 bytes. From the figure we can see that UMAC-based PRF implementation yields the highest throughput, which retains more than 90% of the baseline throughput (e.g., 92% with 1.5KB packet size and 96% with 1KB packet size). With a small packet size of 100 bytes, both the baseline and ShortMAC throughput dropped substantially (similar to other public testing results [3]), because the network drivers used in our experiments are running under interrupt-driven mode, which hampers throughput when packet receiving rate is high. However, UMAC-based PRF still retains 94% of the baseline throughput.

Source node throughput We further evaluate the throughput of a ShortMAC source node with different path length d, where for each path length the source needs to perform d − 1 UMAC-based PRF operations. Originally, it might seem that the ShortMAC source node represents the throughput bottleneck as the source needs to compute multiple k-bit MICs. However we demonstrate that by parallelizing the ShortMAC operations on readily-available multi-processor systems, the throughput of a ShortMAC source node can fully cope with the base line rate even with a path length of 8. Here, for comparison, we use the source node throughput without ShortMAC operations as the base line. We evaluate two different approaches of parallelizing implementation based on widely used OpenMP [5] API. Our first implementation (internal parallelism in short) uses multiple OpenMP threads to parallelize the computation of multiple k-bit MICs per packet. Our second implementation (external parallelism in short) assigns different packets to different OpenMP threads.

We evaluate the ShortMAC source throughput with various packet sizes, and observe that in all cases ShortMAC incurs negligible throughput degradation. Hence we only show the results with packet size set to 1500 bytes in Figure 16. We can see that external parallelism yields the best performance, which matches the baseline case where the source performs no ShortMAC operations.

ShortMAC latency We also evaluate the additional latency incurred by a ShortMAC source node for computing the k-bit MICs with different path lengths and packet sizes; while the end-to-end latency can be derived base on our results. This additional latency in ShortMAC includes PRF computation, k-bit MICs appending, and TCP/IP checksum updating. We write our micro-benchmark to derive the additional time delay for the source to send each packet compared to the baseline case where the source does not compute any k-bit MIC nor updates the checksums.

Figure 17 and Table 3 show the results. We can see that the latency incurred by the checksum computation is stable. It does not increase with the packet size because in our implementation we employ incremental checksum update for the short MIC appended to the packet, instead of recomputing the checksum over the entire packet. We do not observe sharp increase of checksum latency with increasing path length either due to ShortMAC’s efficient k-bit MIC authentication. In addition, the latency caused by the checksum computation is small compared to the latency introduced by UMAC-based PRF computation. The additional latency due to UMAC computation increases linearly to the path length under the same packet size, and also increases linearly to the packet size with a fixed path length due to the property of the UMAC algorithm. Finally, compared to the average end-to-end network latency which is on the order of milliseconds, the additional latency introduced by ShortMAC is negligible.
In this paper, we design, analyze, implement, and evaluate ShortMAC, an efficient data-plane fault localization protocol, which enables a theoretically proven guarantee on data-plane packet delivery and substantially outperforms related protocols in the following aspects. First, ShortMAC achieves high security assurance even in the presence of strong adversaries in control of colluding malicious routers that can drop, modify, inject, and misroute packets at the forwarding paths; whereas a majority of existing fault localization protocols exhibit security vulnerabilities under such a strong adversary model. Second,
comparing to existing secure protocols, ShortMAC achieves several orders of magnitude lower detection delay and protocol overhead, which facilitates its practical deployment. Finally, we demonstrate that ShortMAC’s efficient cryptographic operations, even if implemented in software, have negligible effects on the communication throughput via realistic testing on Gigabit Ethernet links. We anticipate that ShortMAC probabilistic authentication and efficient fault localization can become a basic building blocks for the construction of highly secure and efficient network protocols.

References

[10] K. J. Houle, G. M. Weaver, N. Long, and R. Thomas. Trends in denial of service attacks, even if implemented in software, have negligible effects on the communication throughput via realistic testing on Gigabit Ethernet links. We anticipate that

A Proofs

Proof of Lemma 1: Recall from Section 4 that in ShortMAC, the source finds the first $C_{\text{bad}}$ such that $C_{\text{bad}} > T_{m}$, and identifies link $l_{i}$ as malicious. In this proof, we first derive the upper bound $\beta$ of malicious packet injection (which is based on $T_{m}$) according to the upper bound $\delta$ of false negative rate. Then we calculate the injection threshold $T_{m}$, given the false positive rate $\epsilon$. The probability that $C_{\text{bad}}$ will be increased is $q = \frac{2^{k-1}}{m}$, since the adversary can only randomly generate a $k$-bit string for the fake packet without knowledge of the secret keys of other (benign) routers. Furthermore, the probability that $C_{\text{bad}}$ will be increased is $q(1 - q)$.

Malicious Injection Bound WLOG, suppose $f_{m}$ is a malicious router and $f_{m+1}$ is benign (there can be other malicious routers between the source and $f_{m}$). Suppose the malicious routers between the source and $f_{m}$ (including $f_{m}$) inject $y$ packets on link $l_{m+1}$. Then whether $l_{m+1}$ will be detected depends on the value of $C_{m+1}$, and the false negative rate $P_{f}$ is given by:

$$P_{f} = \mathbb{P}(C_{\text{bad}} < T_{m}) = \mathbb{P}(q - \epsilon < y < T_{m}) \leq 2e^{-2\epsilon y \left(\frac{q - \epsilon}{y}\right)^{2}}$$

where $\epsilon$ is the deviation and $0 \leq \epsilon \leq q$. To achieve the desired upper bound $P_{f} \leq \delta$, we set the threshold $\beta$ such that
\[2e^{-2\beta(q - \frac{T_{in}}{m})^2} = \delta.\] Solving for \(\beta\) gives:

\[\beta = \frac{T_{in}}{q} + \sqrt{\left(\ln \frac{2}{\delta}\right)^2 + 8qT_{in}\ln \frac{2}{\delta} + \ln \frac{2}{\delta}}.\] (7)

(7) implies that if the adversary injects more than \(\beta\) packets on a single link \(l_{m+1}\), \(C^{\text{bad}}_{m+1}\) will exceed \(T_{in}\) and \(l_{m+1}\) will be detected with a high probability \(\geq 1 - \delta\) (or a false negative rate lower than \(\delta\)).

**Injection Detection Threshold** WLOG, suppose \(f_m\) is a malicious router and \(f_{m+1}\) is benign (there can be other malicious routers between the source and \(f_m\)). Suppose the malicious routers between the source and \(f_m\) (including \(f_m\)) inject \(y\) packets on link \(l_{m+1}\). False positives occur when \(C^{\text{bad}}_{m+1} < T_{in}\) but \(C^{\text{bad}}_{i} \geq T_{in}\) (where \(i \geq m + 2\)). (WLOG, suppose \(f_{i-1}\) and \(f_i\) are honest.) Hence, a benign link \(l_i\) is falsely accused, and the false positive rate \(P_{fp}\) is:

\[P_{fp} := \sum_{i=m+2}^{d} \mathbb{P}(C^{\text{bad}}_{m+1} < T_{in}, C^{\text{bad}}_{i} \geq T_{in}, \text{denial of a benign link})\] (8)

The actual \(C^{\text{bad}}_{m+1}\) and \(C^{\text{bad}}_{m+2}\) values can be represented by:

\[C^{\text{bad}}_{m+1} = (q - \epsilon_1) \cdot y; \quad C^{\text{bad}}_{m+2} = (q(1 - q) + \epsilon_2) \cdot y.\] (9)

If we can bound \(\epsilon_1 = \epsilon_2 = \epsilon \leq \frac{d^2}{q}\), we can guarantee that \(C^{\text{bad}}_{m+1} > C^{\text{bad}}_{m+2}\). Therefore, we have:

\[P_{fp} \leq 1 - \mathbb{P}(\epsilon \leq \frac{d^2}{q}) = \mathbb{P}(\epsilon > \frac{d^2}{q}) \leq 2e^{-2\beta(q - \frac{T_{in}}{m})^2}.\] (10)

Note that in (10), we leverage Hoeffding’s inequality and the fact \(y \geq T_{in}\) in the false positive cases.

To achieve the desired upper bound \(P_{fp} \leq \delta\), we set the threshold \(T_{in}\) such that \(2e^{-2T_{in}(\frac{d^2}{q})^2} = \delta\). Solving for \(T_{in}\) gives \(T_{in} = \frac{2\ln \frac{2\delta}{\delta}}{q^2}\).

**Proof of Lemma 2: Drop Detection Threshold and Detection Space** False positives arise when the observed drop rate of a benign link \(l_i\), denoted by \(\rho_i^*\), exceeds the drop detection threshold \(T_{dr}\). To bound the total false positive rate below \(\delta\), it is sufficient to ensure that each \(\rho_i^*\) may exceed \(T_{dr}\) with a probability \(\delta_i = \frac{\delta}{2}\) (since we need to ensure the overall false positive \(\sum_i \delta_i \leq \delta\), i.e., \(\mathbb{P}(\rho_i^* > T_{dr}) < \frac{\delta}{2}\), which is equivalent to:

\[\mathbb{P}(\rho_i^* > T_{dr} - \rho) < \frac{\delta}{d}.\] (11)

By using Hoeffding’s inequality, we have:

\[\mathbb{P}(\rho_i^* > T_{dr} - \rho) < 2e^{-2\rho_i^*(T_{dr} - \rho)^2}\]\[\Rightarrow C^{\text{good}}_{i-1} \geq \frac{\ln \frac{2\delta}{\delta}}{2(T_{dr} - \rho)^2}.\] (12)

Recall that the check-dropping procedure will detect the malicious link with excessive drop rate closest to the source, denoted by \(l_m\). So we need to guarantee \(C^{\text{good}}_{i} \geq \frac{\ln \frac{2\delta}{\delta}}{2(T_{dr} - \rho)^2}\) for any \(i < m\). Since we also have \(C^{\text{good}}_{i} \geq N(1 - T_{dr})^i\) for \(i < m\), we get:

\[N = \frac{\ln \frac{2\delta}{\delta}}{2(T_{dr} - \rho)^2(1 - T_{dr})^d}.\] (13)

Analogously, we can also calculate the false negative rate, which yields the same result.

**Malicious Dropping Bound** Suppose a malicious node \(f_m\) closest to the source receives \(C^{\text{recv}}_{m}\) data packets, but claims that it receives \(C^{\text{good}}_{m}\) data packets, and drops \(x\) fraction of the received \(C^{\text{recv}}_{m}\) data packets on \(l_{m+1}\). We first have the following facts:

\[C^{\text{recv}}_{m} \leq C^{\text{good}}_{m-1}, \quad C^{\text{good}}_{m+1} = (1 - x)C^{\text{recv}}_{m} + \beta.\] (14)

To make neither of its incident links undetected, \(f_m\) must manage to satisfy:

\[\frac{C^{\text{good}}_{m}}{C^{\text{good}}_{m-1}} \geq \frac{1 - T_{dr}}{T_{dr}} \quad \frac{C^{\text{good}}_{m+1}}{C^{\text{good}}_{m}} \geq \frac{1 - T_{dr}}{T_{dr}} \quad \frac{C^{\text{good}}_{m}}{C^{\text{good}}_{m-1}} \geq (1 - T_{dr})^{m-1}N \geq (1 - T_{dr})^dN.\]

Solving (14) and (15), we have: \(x \leq 1 - (1 - T_{dr})^2 + \frac{\beta}{N(1 - T_{dr})} \Rightarrow x = \alpha.\]

**Proof of Theorem 1:** \((\alpha, \beta)\)-Statistical Security can directly follow Lemma 1 and Lemma 2. In the following, we will prove \((\Omega, \theta)\)-Guaranteed Forwarding Correctness.

Given \(N\) and \(\delta\), we can set the drop detection threshold \(T_{dr}\) from Lemma 2 and the injection bound \(\beta\) from Lemma 1. Let \(\eta^{\text{fake}}\) denote the fake data packets the destination has received but not detected yet, and \(\eta^{\text{leg}}\) denote the legitimate data packets the destination has received out of \(N\) data packets from the source. Then we have:

\[\theta = \frac{\eta^{\text{leg}}}{N} = \frac{C_{d+1}^{\text{good}} - \eta^{\text{fake}}}{N}.\] (16)

When no fault is detected in the identify stage, it satisfies:

\[C_{d+1}^{\text{good}} \geq (1 - T_{dr})^dN, \quad \eta^{\text{fake}} \leq \beta.\] (17)

By (16) and (17), we have \(\theta = (1 - T_{dr})^d - \frac{\beta}{N}\).

Finally, we can integrate ShortMAC with routing as follows. The control plane first provides a routing path \(p\) for the source \(S\), and then avoids faulty links using feedback (fault localization results) from the data plane. In this way, ShortMAC enables the source to identify the malicious links that reside in previously explored paths. In a network with \(\Omega\) malicious links, the source can bypass at least one of the malicious links after each epoch until a working path is found, resulting in an exploration of at most \(\Omega\) epochs to find a working path.