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

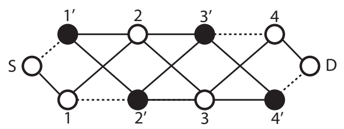

Figure 1: Example of the *exponential path exploration problem* for end-to-end monitoring. The source node S is communicating with destination D, and dotted links are faulty links of malicious routers (black nodes). Suppose shortest-path routing is used.

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 weak authenticator for each packet such that each forged packet has a *non-trivial probability* to be detected. In this way, if a malicious node modifies or injects more than a threshold number of (e.g., tens of) packets, the malicious activity will cause a detectable deviation on the counter values maintained at different routers. Essentially, $\mathrm{ShortMAC}$ traps an attacker into a dilemma: if the attacker inflicts damage worse than a threshold, it will be detected, which may lead to removal from the network; otherwise, the damage is limited and thus a guarantee on data-plane packet delivery is achieved.

Contributions. 1) We propose a data-plane fault localization protocol $\mathrm{ShortMAC}$ that achieves high security assurance with 100 - 10000 times lower detection delay and storage overhead than existing secure protocols.
2) We define and derive a provable guarantee on end-to-end data-plane packet delivery, by limiting adversarial activities instead of perfectly detecting every single malicious action which would incur high protocol overhead.
3) We theoretically derive the performance bounds of $\mathrm{ShortMAC}$ and evaluate $\mathrm{ShortMAC}$ via SSFNet-based [6] simulation and Linux/Click router implementation.

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 $f_1, f_2, \ldots, f_{d-1}$, 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

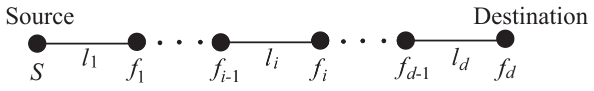

Figure 2: An example path and notation.

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 misrouting can be regarded as dropping packets along the original path and injecting them to the new path. A formal definition follows:

Definition 1 An $(x, y)-$***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).

2.2 Problem Statement

Our paper focuses on providing data-plane fault localization for a lower-bound guarantee on data-plane packet delivery. In this section, we define communication epochs, detection thresholds, faulty links, and finally we formalize fault localization.

Definition 2 An end-to-end communication is composed of a set of consecutive **epoch**s. 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 δ.

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

To achieve a guaranteed θ, 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 α percent of packets or injects β 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 α percent of the data packets on a single link, or injects more than β 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 δ.

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.

3.1 Strawman Approaches and Challenges

We highlight the design challenges of a secure fault localization protocol by sketching two intuitive but insecure strawman approaches below. In Section 9, we further show that a considerable number of existing fault localization protocols also suffer from the following vulnerabilities.

Strawman 1: Acknowledgment-based approach. Let us consider that the source S in Figure 2 sends out a data packet m towards the destination f_d. Upon receiving m at each hop in the path, router f_i must return an acknowledgment (ACK) to S authenticated with the secret key shared with S (assuming S and f_i have pre-established a secret key using Diffie-Hellman [20] as in Passport [39], or some other key exchange protocol). If S receives correct ACKs from routers $f_1, \ldots, f_{i-1}$ but not from router f_i, S concludes link l_{i-1} is faulty. In this approach however, a malicious router f_i can drop the *ACK* from another router, say f_{i+5}, without dropping other packets to frame l_{i+4} as malicious to the source (*framing attack*). To reduce the overhead of ACK packets, the source node may "sample" a subset of packets and only the sampled packets will require ACKs from the routers. In this approach however, if a malicious router f_m can distinguish between sampled and non-sampled packets, f_m can safely drop *all and only* non-sampled packets without being detected.

Strawman 2: Neighborhood-based approach. Routers in a path may employ "hop-by-hop" monitoring to detect packet delivery fault to reduce the overhead of ACK packets and address the framing attack in Strawman 1. For example in Figure 2, each router f_i asks for the ACK packets *only from* the 2-hop neighbor f_{i+2} in the path, and accuses l_i if f_i does not receive an ACK from f_{i+2} for a packet sent by f_i. In this approach however, if f_i is colluding with f_{i+1} and does not accuse f_{i+1} even if f_i does not receive the ACK from f_{i+2}, then f_{i+1} can safely drop packets without being detected.

3.2 ShortMAC High-level Protocol Steps

ShortMAC eschews the use of acknowledgments for individual packets and neighborhood-based monitoring, but instead monitors the packet *count* and *content* at each hop. Specifically, a router maintains per-path counters to record the number of received data packets originated from the source in the current epoch. To ensure that the packet count is a verifiable measure of the desired monitoring task, we require that both packet tampering and injection by malicious (colluding) routers affect counter values at benign nodes.

The high-level protocol steps are sketched in Figure 3. At the beginning of each epoch denoted by e_k, the source S sends a **path setup request** along a selected path p, requesting for (i) setting up monitoring counters at each router (ii) and retrieving the counters from the routers and destination from the previous epoch (not needed if path p was not used by S in the previous epoch). This path setup request is delivered over a secure channel as we will describe in Section 4. The routers and destination agree on counter setup, and send back a **confirmation report** to S over the same secure channel, containing the counter values of the routers and destination in the past epoch. S then performs fault detection based on the retrieved counters, and bypasses the detected faulty link (if any) by finding another path without the identified faulty link (e.g., via source routing,

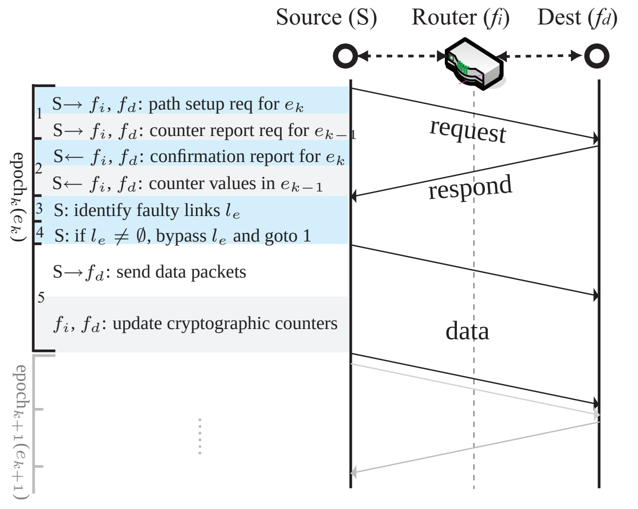
Source (S)
Router (f_i)
Dest (f_d)
1
S→ f_i, f_d: path setup req for e_k
S→ f_i, f_d: counter report req for e_{k-1}
2
S← f_i, f_d: confirmation report for e_k
S← f_i, f_d: counter values in e_{k-1}
3
S: identify faulty links l_e
4
S: if $l_e \neq \emptyset$, bypass l_e and goto 1
S→f_d: send data packets
5
f_i, f_d: update cryptographic counters
request
respond
data
epoch$_k(e_k)$
epoch$_{k+1}(e_{k+1})$

Figure 3: Protocol flow for Source(S), Router(f_i), and Destination(f_d).

path splicing [46], or pathlet routing [24]). Note that the detection result is only used by S itself (instead of sharing with other nodes, which is susceptible to framing attacks) for selecting its own routing path.

Finally, S sends packets along the selected working path p, with each packet carrying several ShortMAC authentication bits. The routers verify the authentication bits in each received packet based on the symmetric epoch key shared with the source node of p, increment locally stored counters for p accordingly, and forward only the authentic packets. Due to the ShortMAC *authentication bits*, modified/injected packets can result in an observable deviation in the counter values which enable fault localization by the source at the end of each epoch.

Although the high-level *epoch-based* protocol flow (nodes periodically send certain locally logged *traffic reports* to the source) bears great similarity with Fatih [45], AudIt [10], and Statistical FL with sketch [16], both Fatih and AudIt use simple counters or Bloom Filters without *keyed* hash functions as the traffic reports, thus remaining vulnerable to packet modification/injection attacks. In addition, the secure sketches used in Statistical FL as the traffic reports consume several hundreds of bytes per path. In contrast, ShortMAC efficiently addresses packet modification attack as sketched below.

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 amartized 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.

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_i, we use a Pseudo-Random Function (PRF) which constructs a k-bit string as a function of the packet m and key K_{si} shared between S and f_i. 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_{si} [44]. Each router f_i maintains two path-specific counters C_i^{good} and C_i^{bad} 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.

3.4 ShortMAC Example

We present a toy example in Figure 4 to provide intuition on how ShortMAC enables data-plane fault localization. Suppose the source node sends out 1000 packets in a certain epoch. The source uses a PRF taking a secret key as input which can map a packet into two bits (called 2-bit MAC) uniformly at random to anyone without knowledge of the secret key. The source computes the PRF four times for each packet, taking as input the epoch symmetric key shared with f_1, f_2, f_3, and the destination, respectively. Then the source attaches the resulting four 2-bit MACs to each packet.

Among the 1000 packets, suppose three packets are spontaneously dropped on the first link, and router f_1 receives the remaining 997 packets. f_1 computes the PRF on each of the received packets taking as input the epoch symmetric key shared with the source, and compares the resulting 2-bit MACs with the one embedded in each packet. All verifications are successful, so f_1 has $C_1^{good} = 997$ and $C_1^{bad} = 0$. Suppose the malicious router f_2 drops 100 good packets and injects 100

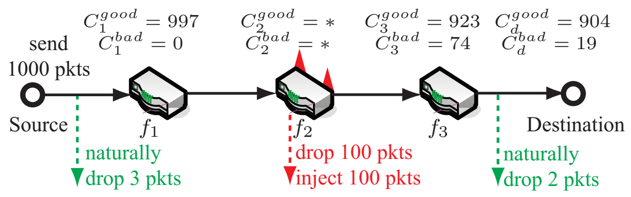

Figure 4: Fault localization example with ShortMAC using 2-bit MAC. f_2 is malicious.

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 the malicious data content. f_3 thus computes $C_3^{bad} = 100 - 26 = 74$ and $C_3^{good} = 997 - 100$ (dropped legitimate packets) $+ 26$ (bogus but undetected packets) $= 923$. Similarly, we can analyze the counters for the destination in Figure 4, assuming 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 θ

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+1} is abnormally lower than that of f_m based on the drop detection threshold T_{dr} that 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 θ.

4 ShortMAC Details

In this section we describe the ShortMAC protocol in detail, where the source can either guarantee that a high fraction θ 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.[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, given that a malicious node has a limited degree, after bypassing all its malicious links the source can eventually bypass that node.

4.1 ShortMAC Packet Format

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

$$trailer = \langle SN, \mathcal{M}_1, \ldots, \mathcal{M}_d \rangle, \tag{1}$$

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

$$\begin{aligned} \mathcal{M}_d &\leftarrow PRF_{K_{sd}}(IP_{invar}||SN||TTL_d) \\ \mathcal{M}_{d-1} &\leftarrow PRF_{K_{s(d-1)}}(IP_{invar}||SN||TTL_{d-1}||\mathcal{M}_d) \\ &\ldots\ldots \\ \mathcal{M}_i &\leftarrow PRF_{K_{si}}(IP_{invar}||SN||TTL_i||\mathcal{M}_{i+1}||\ldots||\mathcal{M}_d) \end{aligned} \tag{2}$$

where "$||$" denotes concatenation and $PRF_{K_{si}}(\cdot)$ denotes a PRF keyed by the epoch symmetric key K_{si} shared between S and f_i. 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) $\mathcal{M}_i$ also authenticates the downstream $\mathcal{M}_{i+1}, \ldots, \mathcal{M}_d$, so that if a malicious router f_m changes any of these downstream k-bit MACs, f_i can observe the inconsistency in $\mathcal{M}_i$ with a probability $1 - \frac{1}{2^k}$ and increase its C_i^{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:

$$\texttt{request} = \left(S, EpNum, p, \mathrm{Sig}_s[H(S||EpNum||p)]\right) \tag{3}$$

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 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

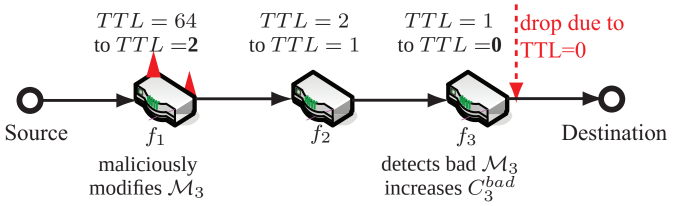
$TTL = 64$ to $TTL = \mathbf{2}$
$TTL = 2$ to $TTL = 1$
$TTL = 1$ to $TTL = \mathbf{0}$
drop due to TTL=0
Source
f_1
maliciously modifies $\mathcal{M}_3$
f_2
f_3
detects bad $\mathcal{M}_3$ increases C_3^{bad}
Destination

Figure 5: Illustration of framing attacks. If $\mathcal{M}_i$ 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 $\mathcal{M}_i$ in Eq.(2) had not authenticated the downstream $k-$bit MAC field, f_1 can maliciously modify $\mathcal{M}_3$ in the packets which causes f_3 to detect inconsistent $\mathcal{M}_3$ with a non-trivial probability and increase C_3^{bad}, thus framing the link between f_2 and f_3.

router f_i and the destination f_d to set up an epoch symmetric key K_{si} with S, and report the counter values (C_i^{bad} and C_i^{good}) along the reverse path p if p was used in the previous epoch. Then S expects Acknowledgment (ACK) packets from all the nodes in p containing the requested information authenticated with each node's private key and K_{si}.

Note that a spontaneous loss of request or ACK packets will prevent S from setting up epoch symmetric keys and learning the counter values by certain routers in the previous epoch. To preclude such damage, we use the following **hop-by-hop reliable transmission** approach: when f_i forwards either a request or an ACK packet to its neighbor, f_i tries up to r times (e.g., $r = 5$) until it gets a confirmation from the neighbor. In this way, the failure of receiving a request or ACK packet can only indicate malicious drops – more precisely, with the probability of $1-\rho^r$, where ρ is the natural loss rate of a link. Then thanks to the Onion ACK approach presented below, the source can immediately identify a malicious link that drops request or ACK packets and request for a new path excluding the detected link.

Stage 2: Report with Onion ACK
Upon receiving a request, f_i starts a timer whose value is the maximum round trip time from f_i to the destination.[2] At the same time, f_i constructs its local report $\mathcal{R}_i$ by:

$$\mathcal{R}_i = \left(f_i, K_{si}, p, EpNum, C_i^{good}, C_i^{bad}, \Delta_{ttl}\right) \tag{4}$$

where f_i is the node id, K_{si} is the generated epoch symmetric key, p is the requested path, $EpNum$ is the epoch number contained in the request in Eq.(3) to prevent replay attacks, and C_i^{good} and C_i^{bad} are the counter values from the previous epoch (null if the path was not used in the previous epoch). In addition, Δ_{ttl} indicates the amount with which f_i will decrease the TTL field. For intra-domain networks where each f_i is a physical router, Δ_{ttl} is expected to be 1; for inter-domain routing in the Internet where f_i maps to an Autonomous System (AS), Δ_{ttl} reflects the actual forwarding hops within AS f_i, so that S can estimate the TTL value TTL_i for each hop in order to compute $\mathcal{M}_i$ in Eq.(2). Once the report is constructed:

[2]We 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 Avramopolous et al. [12].

Case 1 If f_i receives an ACK $\mathcal{A}_{i+1}$ from neighbor f_{i+1} before the timer expires, f_i further commits $\mathcal{R}_i$ into a new ACK $\mathcal{A}_i$ by combining the received $\mathcal{A}_{i+1}$ via an **Onion ACK** approach:

$$\mathcal{A}_i = \left(f_i, \mathrm{Sig}_i[\mathcal{R}_i, \mathcal{A}_{i+1}, \mathrm{H}_{K_{si}}(\mathcal{R}_i||\mathcal{A}_{i+1})]\right), \tag{5}$$

$\mathrm{H}_{K_{si}}(\cdot)$ denotes a message authentication code computed with K_{si}. Then, f_i forwards $\mathcal{A}_i$ to f_{i-1} toward S.

Case 2 If f_i 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 request 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_m 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 $\mathcal{A}_1$ from f_1, S first iteratively retrieves $\mathcal{A}_1, \mathcal{A}_2, \ldots$ in the order of the certified public keys of $f_1, \ldots, f_d$, until it either completes at d or fails at j ($j \neq d$).[3] 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 addition, S extracts $\mathcal{R}_1, \ldots, \mathcal{R}_j$ 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 $C_i^{good} < C_{i-1}^{good}$. 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 $C_1^{bad}, C_2^{bad}, \ldots, C_j^{bad}$ values *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

[3]S can verify if a certain retrieved report $\mathcal{R}_i$ is valid by checking the embedded message integrity code $\mathrm{H}_{K_{si}}(\mathcal{R}_i||\mathcal{A}_{i+1})$.

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 $\mathrm{check\text{-}dropping}$ 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* ω, by bypassing all malicious links, S can eventually avoid a malicious node within a constant time ω. Once a path with no previously detected malicious links is established, the source node sends its packets with $\mathrm{ShortMAC}$ authentication shown in Eq.(2); and each node f_i examines its corresponding k-bit MAC $\mathcal{M}_i$ 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 $\mathrm{ShortMAC}$'s security against data-plane attacks by malicious routers. Section 6 provides theoretical proofs on $\mathrm{ShortMAC}$'s security. In our adversary model, a malicious router can drop and inject data packets, `request`s and `ACK`s, and can send arbitrary counter values in its reports. We show that $\mathrm{ShortMAC}$ 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 $C_{m-1}^{good} - C_m^{good}$ will exhibit a large discrepancy. Hence, either l_{m-1} or l_m will become suspicious. Moreover, if f_m injects/modifies packets, $\mathcal{M}_{m+1}$ will be inconsistent at f_{m+1} with high probability and cause a non-zero C_{m+1}^{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 `ACK`s or `request`s. Since the `ACK`s or `request`s are digitally signed, f_m cannot impersonate others or modify the content of the `ACK`s or `request`s. f_m cannot selectively drop the `ACK` reports due to the use of Onion `ACK`. Instead, f_m can only drop the `ACK`s or `request`s 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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ 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 `request`s 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 incriminating 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 $\mathrm{ShortMAC}$ 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^{good} and C_{m+1}^{good} F_m claims. (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

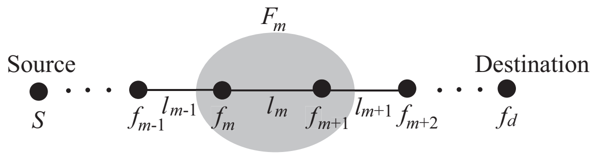

Figure 6: Security against colluding nodes – one base case with two adjacent colluding nodes f_m and f_{m+1} forming a virtual malicious node F_m.

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 δ on detection false negative and false positive rates, the injection detection threshold T_{in} can be set to $T_{in} = \frac{2\ln\frac{2d}{\delta}}{q^4}$, where d is the path length and $q = \frac{2^k-1}{2^k}$ is the probability that a fake packet will be inconsistent with the associated k-bit MAC. The number of fake packets β an adversary can inject on one of its malicious links without being detected is limited to:* $\beta = \frac{T_{in}}{q} + \frac{\sqrt{\left(\ln\frac{2}{\delta}\right)^2 + 8qT_{in}\ln\frac{2}{\delta}} + \ln\frac{2}{\delta}}{4q^2}$.

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 δ. 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 ρ fraction of packets. We first describe how the drop detection threshold T_{dr} is set when N and δ 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 ρ; otherwise, with a smaller N, the observed per-link drop rate can deviate further away from ρ, 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 δ. 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 δ.

Lemma 2 ***Dropping Detection*** *and (N, δ)-* ***Fault Localization:*** *Given the bound δ on detection false positive and negative rates and drop detection threshold T_{dr}, the detection delay N is given by: $N = \frac{\ln(\frac{2d}{\delta})}{2\left(T_{dr}-\rho\right)^2\left(1-T_{dr}\right)^d}$, where d is the path length. Correspondingly, the fraction of packets α an adversary can drop on one of its malicious links without being detected is limited to: $\alpha = 1-(1-T_{dr})^2 + \frac{\beta}{N(1-T_{dr})^d}$.*

Protocol	ShortMAC	PAAI-1	SSS	Sketch
Detect. Delay (pkt)	3.8×10^4	7.1×10^5	1.6×10^8	$\approx 10^6$
Comm. (extra %)	$< 10^{-5}$	1	1	$< 10^5$
Marking Cost (bytes)	2	0	0	0
Per-path State (bytes)	21	$2{\times}10^5$	4×10^3	≈ 500

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, $\delta = 1\%$, $\rho = 0.5\%$, $T_{dr} = 1.5\%$, 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 ρ 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}, δ, and path length d, we can achieve $(\alpha, \beta)_\delta-$forwarding security where α is given by Lemma 2 and β is given by Lemma 1. We also achieve (Ω, θ)-Guaranteed forwarding correctness with Ω equal to the number of malicious links in the network, and $\theta = (1-T_{dr})^d - \frac{\beta}{N}$. 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.[4]

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 $\lg N$-bit C^{good} counter, one $\lg \beta$-bit C^{bad} counter, one $\lg N$-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 denoted by *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

[4]The 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 $\mathrm{ShortMAC}$ is 1 extra ACK for every 3.8×10^4 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 $\mathrm{ShortMAC}$ on the widely used SSFNet simulator [6]. Since the communication and storage overhead of $\mathrm{ShortMAC}$ are both straightforward (see the theoretical results in Section 6), we launch extensive simulations mainly to study the detection delay and security of $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ security against other attack strategies than constant dropping/injection rates used in the theoretical analysis.

Evaluation scenario and attack pattern Recall that $\mathrm{ShortMAC}$ 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 ρ for each link to make the detection harder (easier for causing false positives). Since Section 5 elaborates $\mathrm{ShortMAC}$ 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 δ with per-link natural loss rate ρ 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

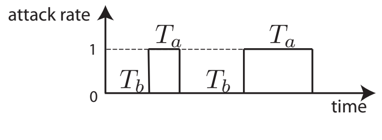

Figure 7: Attack pattern of random-period packet corruption, where T_b denotes the benign (non-attack) period and T_a denotes the attack period when the malicious node drops/modifies *all* legitimate packets. The durations for both periods are randomly generated.

dropping attacks with a dropping rate as low as 2%, $\mathrm{ShortMAC}$ 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 ($\mathrm{ShortMAC}$ 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 nonzero), 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_3 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 $\mathrm{ShortMAC}$'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 $\mathrm{ShortMAC}$'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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ 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

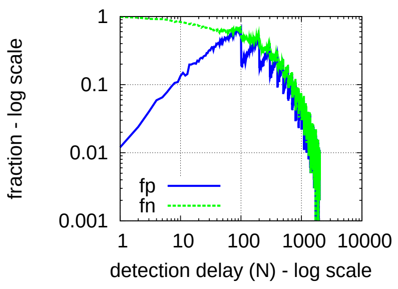

Figure 8: Detection delay N and false positive (fp) and negative (fn) rates in dropping attacks with malicious drop rate set to 2%, per-link natural loss rate ρ set to 0.5%, per-link drop detection threshold T_{dr} set to 1%, and f_3 as the malicious router.

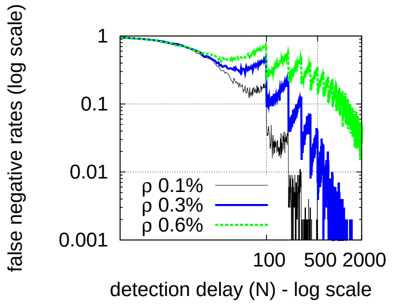

Figure 9: Effects of natural loss rates on detection delay N and false positive and negative rates δ, with malicious drop rate 5% at f_3 and per-link drop detection threshold $T_{dr} = 1\%$.

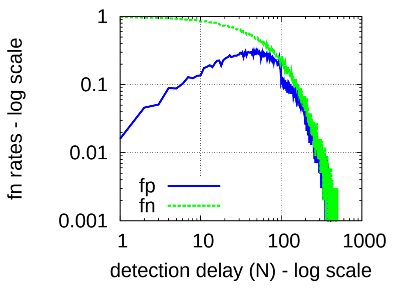

Figure 10: Detection delay N and false positive and negative rates in injection attacks with malicious injection rate set to 2% using 2-bit MICs, natural loss rate set to 0.5%, per-link drop detection threshold T_{dr} set to 1%, and f_3 as the malicious router.

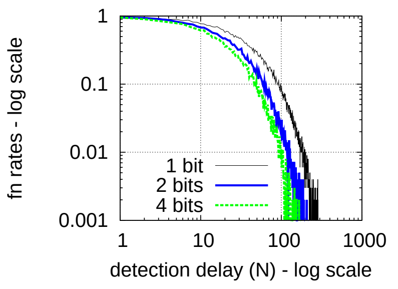

Figure 11: Effects of different k-bit MIC lengths on detection delay N and false positive and negative rates δ.

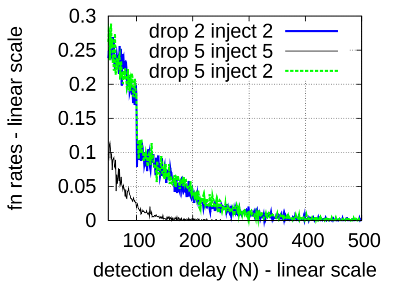

Figure 12: Combined attacks at malicious router f_3 against 2-bit MIC with varying packet dropping and injection rates. "drop p inject q" denotes the use of $p\%$ dropping rate and $q\%$ injection rate.

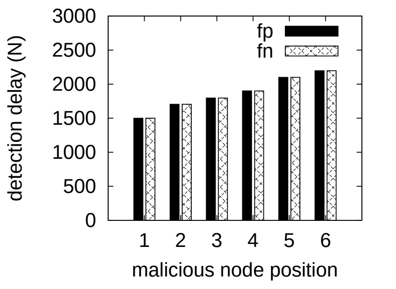

Figure 13: Variance on detection delay N in dropping attacks with required error rate $\delta < 1\%$, detection threshold $T_{dr} = 1\%$, per-link natural packet loss rate $\rho = 0.5\%$, and both malicious dropping and injection rates set to 5%.

the dropping detection as we explained above), which can also be theoretically proved.

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 - $max\{$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 5×10^4, 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.

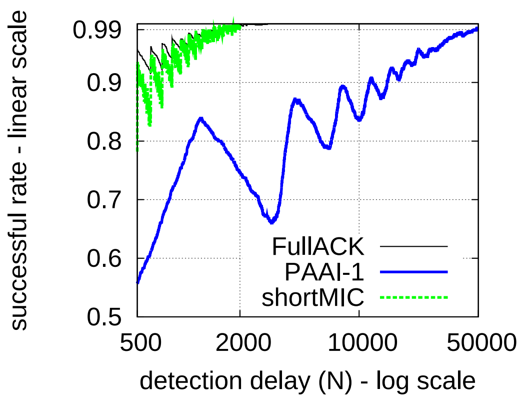

Figure 14: Comparison with the heavy-weighted fullACK scheme and efficient PAAI-1 scheme [59], with $\rho = 0.5\%$, $T_{dr} = 1\%$, and 5% malicious drop rate. PAAI-1 samples $\frac{1}{36}$ of total packets. The successful rate is defined as 1 - $max\{$false positive rate, false negative rate$\}$.

	ShortMAC	FullACK	PAAI-1
Detect. delay	20 sec	20 sec	8.3 min
Communication	0.01%	100%	5.6%

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.

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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ routers in user-space to evaluate $\mathrm{ShortMAC}$'s effects on communication throughput, which are mainly determined by $\mathrm{ShortMAC}$'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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ system.

We utilize the widely used Netperf benchmark [4] for our $\mathrm{ShortMAC}$ throughput evaluation, and write our own micro-benchmark for accurate latency evaluation. We evaluate $\mathrm{ShortMAC}$ with varying packet sizes by configuring the interface Maximum Transmission Unit (MTU) sizes. We evaluate the throughput of a $\mathrm{ShortMAC}$ router and a $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ 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* $\mathrm{ShortMAC}$ operations as the base line. The $\mathrm{ShortMAC}$ router runs as a *single user-space* process without exploring 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 $\mathrm{ShortMAC}$ 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 $\frac{53.84}{57.52}$=94% of the baseline throughput.

Source node throughput We further evaluate the throughput of a $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ source node represents the throughput bottleneck as the source needs to compute multiple k-bit MICs. However we demonstrate that by parallelizing the $\mathrm{ShortMAC}$ operations on readily-available multi-processor systems, the throughput of a $\mathrm{ShortMAC}$ 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* $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ source throughput with various packet sizes, and observe that in all cases $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ operations.

$\mathbf{ShortMAC}$ latency We also evaluate the additional latency incurred by a $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$ 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 $\mathrm{ShortMAC}$'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 $\mathrm{ShortMAC}$ is negligible.

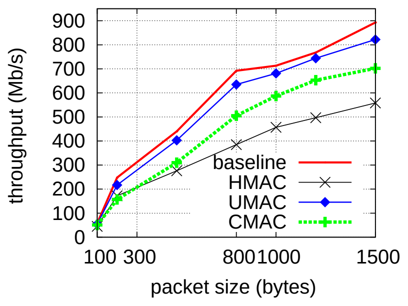

Figure 15: ShortMAC router throughput.

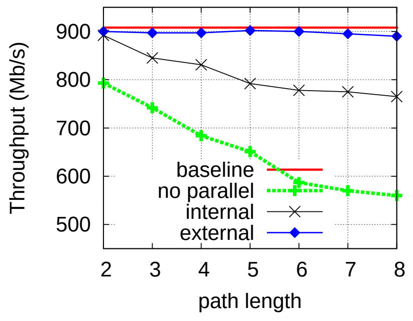

Figure 16: ShortMAC source node throughput.

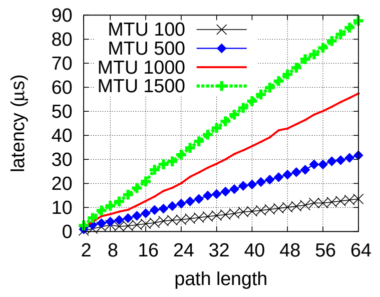

Figure 17: ShortMAC source node latency.

Path Length	Checksum (μs)	UMAC (μs)			
		100	500	1000	1500
2	0.0374	0.1771	0.4760	0.8892	1.4047
3	0.0378	0.3691	0.9557	1.7635	3.3025
4	0.0442	0.5239	1.4273	2.6357	4.0944
5	0.0415	0.7080	1.9018	3.5059	5.4566
6	0.0437	0.8723	2.3758	4.3839	6.8307
7	0.0445	1.0467	2.8530	5.2617	8.2019
8	0.0474	1.2206	3.3274	6.1285	9.5483

Table 3: ShortMAC source node latency breakdown (checksum updates and UMAC computation). All the data represents the average time of processing 50000 packets.

9 Related Work

Perlman [49] first introduced the idea of acknowledgment-based approaches to detect data-plane adversaries and achieve robust routing in the presence of Byzantine failures. In the more recent Sprout project [21], a source node monitors the end-to-end path performance and uses probabilistic route selection to find a working path if the current path is faulty. However, without secure fault localization, both schemes suffer from the exponential path exploration problem as Figure 1 shows. Given the importance of data-plane fault localization, several approaches have recently been proposed, which unfortunately suffer from the following limitations.

Security vulnerabilities. In WATCHERS [18, 28], AudIt [10] and Fatih [45], each router records a *traffic summary* implemented based on naive counters or Bloom Filters [17] updated with no secret keys for the packets it forwards, and periodically exchanges local summaries with others for distributed fault detection based on *flow reservation*. Without any authentication of the data packets, these schemes suffer from packet modification attacks; or if traditional authentication based on MACs or digital signatures is used, the protocol overhead is too high. Similarly, the recently proposed Network Confessional [11] is also vulnerable to packet modification/injection attacks due to the absence of efficient packet authentication. In ODSBR [14, 15] and Secure Traceroute [48], the source node monitors the end-to-end loss rate of the path; and only when the observed loss rate exceeds a certain threshold, the source starts probing specific nodes in the path soliciting acknowledgments for the *subsequent* packets the source sends. However, a malicious node can safely drop packets when the probing is not activated, while behaving "normally" when probing is invoked. Hence, the source can never catch the malicious nodes nor bound the malicious dropping rate, unless the probing is always activated which incurs high overhead. In addition, ODSBR employs binary search in the probing phase for dropping localization, until the algorithm converges to a specific link. Since the binary search algorithm proceeds on each packet lost (possibly due to natural loss), in the presence of natural packet loss the algorithm either does not converge or incurs high false positives by incriminating benign links. Liu et al. propose enabling neighboring (or two-hop) routers in the path to monitor each other [37] by using 2-hop acknowledgment packets. However, such a neighborhood-based detection scheme is vulnerable to colluding (neighboring) routers.

High protocol overhead. Among the known secure proposals, the protocol due to Avramopoulos et al. [12] incurs high overhead due to the acknowledgments from all routers in the path and multiple digital signature generation and verification operations for *each* data packet. Both Statistical FL [16] and PAAI-1 [59] achieve small communication overhead, but at the cost of high storage overhead and unacceptably long detection delays (Sections 6 and 7).

Applicability constraints (and security vulnerabilities). A recent proposal due to Wang et al. [53] for forwarding fault localization in sensor networks requires a special tree-like routing infrastructure where the communications take place only between a sensor node and the same trusted base station. Both Watchdog [43] and Catch [42] can identify and isolate malicious routers for wireless ad hoc networks, where a sender S verifies if the next-hop node f_i indeed forwards S's packets by *promiscuously* listening to f_i's transmission. Both approaches rely on the wireless broadcast medium and are thus inapplicable to wired networks. Furthermore, both Watchdog and Catch are vulnerable to collusion attacks, where a malicious node f_m drops the packets of a remote sender S which is out of the promiscuous listening range of f_m while the colluding neighbors in the promiscuous listening range of f_m intentionally do not report the packet dropping behavior of f_m.

10 Conclusion

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, compared to existing *secure* protocols, $\mathrm{ShortMAC}$ achieves several orders of magnitude lower detection delay and protocol overhead, which facilitates its practical deployment. Finally, we demonstrate that $\mathrm{ShortMAC}$'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 $\mathrm{ShortMAC}$ probabilistic authentication and efficient fault localization can become a basic building blocks for the construction of highly secure and efficient network protocols.

References

[1] Arbor networks: Infrastructure 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] Linux as IP router. `http://freedomhec.pbworks.com/f/linux_ip_routers.pdf`.

[4] Netperf benchmark. `http://www.netperf.org/netperf/`.

[5] The OpenMP API specification for parallel programming. `http://openmp.org/wp/`.

[6] Scalable simulation framework. `http://www.ssfnet.org/`.

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

[8] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proc. ACM SIGCOMM*, Seattle, WA, Aug. 2008.

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

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

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

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

[13] I. Avramopoulos and J. Rexford. Stealth probing: Efficient data-plane security for IP routing. In *USENIX*, 2006.

[14] 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.

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

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

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

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

[19] H. Chan and A. Perrig. Round-effcient broadcast authentication protocols for fixed topology classes. In *IEEE Symposium on Security and Privacy*, May 2010.

[20] W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, Mar. 1979.

[21] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy. Routing amid colluding attackers. In *IEEE ICNP*, 2007.

[22] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 180–197, London, UK, 1997. Springer-Verlag.

[23] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *ACM SIGCOMM*, 2009.

[24] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *Proceedings of the ACM SIGCOMM*, 2009.

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

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

[27] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: Secure path vector routing for securing BGP. In *Proceedings of ACM SIGCOMM*, Sept. 2004.

[28] 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.

[29] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure border gateway protocol (S-BGP) — real world performance and deployment issues. In *NDSS*, 2000.

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

[31] M. E. Kounavis, X. Kang, K. Grewal, M. Eszenyi, S. Gueron, and D. Durham. Encrypting the Internet. In *ACM SIGCOMM*, 2010.

[32] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), Feb. 1997.

[33] T. Krovetz and W. Dai. VMAC: Message Authentication Code using Universal Hashing. Internet-draft, 2007.

[34] N. Kushman, S. Kandula, D. Katabi, and B. M. Maggs. R-BGP: Staying Connected In a Connected World. In *USENIX NSDI*, 2007.

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

[36] P. Laskowski and J. Chuang. Network monitors and contracting systems: competition and innovation. In *Proceedings of ACM SIGCOMM*, 2006.

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

[38] X. Liu, X. Yang, and Y. Lu. To filter or to authorize: Network-layer dos defense against multimillion-node botnets. In *Proceedings of ACM SIGCOMM*, 2008.

[39] X. Liu, X. Yang, D. Wetherall, and T. Anderson. Efficient and secure source authentication with packet passports. In *USENIX SRUTI*, 2006.

[40] X. Liu, X. Yang, and Y. Xia. NetFence: Preventing Internet Denial of Service from Inside Out. In *ACM SIGCOMM*, 2010.

[41] A. Lysyanskaya, R. Tamassia, and N. Triandopoulos. Multicast authentication in fully adversarial networks. In *In IEEE Symposium on Security and Privacy*, pages 241–255, 2004.

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

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

[44] M.Bellare, R.Canetti, and H.Krawczyk. Pseudorandom functions revisited: the cascade construction and its concrete security. In *IEEE FOCS*, 1996.

[45] 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.

[46] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *ACM SIGCOMM*, 2008.

[47] S. L. Murphy and M. R. Badger. Digital signature protection of the OSPF routing protocol. In *NDSS*, 1996.

[48] 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.

[49] R. Perlman. *Network Layer Protocol with Byzantine Agreement*. PhD thesis, The MIT Press, Oct. 1988. LCS TR-429.

[50] A. Perrig, R. Canetti, D. Song, and D. Tygar. The TESLA broadcast authentication protocol. *RSA Cryptobytes*, 2002.

[51] E. T. Krovetz. UMAC: Message Authentication Code using Universal Hashing. RFC 4418, 2006.

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

[53] C. Wang, T. Feng, J. Kim, G. Wang, and W. Zhang. Catching packet droppers and modifiers in wireless sensor networks. In *IEEE SECON*, 2009.

[54] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford. Don't secure routing protocols, secure data delivery. In *Proc. of ACM Workshop on Hot Topics in Networks (Hotnets-V)*, Nov. 2006.

[55] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Trans. Netw.*, 7:502–513, August 1999.

[56] W. Xu and J. Rexford. MIRO: Multi-path Interdomain Routing. In *ACM SIGCOMM*, 2006.

[57] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *Proceedings of ACM SIGCOMM*, 2006.

[58] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-limiting network architecture. In *IEEE/ACM Transactions on Networking*, 2008.

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

A Proofs

Proof of Lemma 1: Recall from Section 4 that in $\mathrm{ShortMAC}$, the source finds the first C_i^{bad} such that $C_i^{bad} > T_{in}$, and identifies link l_i as malicious. In this proof, we first derive the upper bound β of malicious packet injection (which is based on T_{in}) according to the upper bound δ of false negative rate. Then we calculate the injection threshold T_{in} given the false positive upper bound δ.

With k-bit MICs, when f_{i-1} receives a fake packet, the probability that C_{i-1}^{bad} will be increased is $q = \frac{2^k-1}{2^k}$, 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_i^{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}^{bad}, and the false negative rate $\mathbb{P}_{fn}$ is given by:

$$\begin{aligned}\mathbb{P}_{fn} &= \mathbb{P}(C_{m+1}^{bad} < T_{in}) = \mathbb{P}\big((q-\epsilon)y < T_{in}\big) \\ &\leq 2e^{-2y\left(q-\frac{T_{in}}{y}\right)^2} \text{(Hoeffding's inequality)}\end{aligned} \tag{6}$$

where ϵ is the deviation and $0 \leq \epsilon \leq q$. To achieve the desired upper bound $\mathbb{P}_{fn} \leq \delta$, we set the threshold β such that

$2e^{-2\beta(q-\frac{T_{in}}{\beta})^2} = \delta$. Solving for β gives:

$$\beta = \frac{T_{in}}{q} + \frac{\sqrt{\left(\ln\frac{2}{\delta}\right)^2 + 8qT_{in}\ln\frac{2}{\delta}} + \ln\frac{2}{\delta}}{4q^2}. \tag{7}$$

(7) implies that if the adversary injects more than β packets on a single link l_{m+1}, C^{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 δ).

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 $C^{bad}_{m+1} < T_{in}$ but $C^{bad}_i \geq T_{in}$ (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 $\mathbb{P}_{fp}$ is:

$$\begin{aligned}\mathbb{P}_{fp} &:= \sum_{i=m+2}^{d} \mathbb{P}(C^{bad}_{m+1} < T_{in}, C^{bad}_i \geq T_{in} | l_i \text{ benign}) \\ &\leq d\cdot\mathbb{P}(C^{bad}_{m+1} < C^{bad}_{m+2}).\end{aligned} \tag{8}$$

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

$$C^{bad}_{m+1} = (q-\epsilon_1)\cdot y; \quad C^{bad}_{m+2} = \left(q(1-q)+\epsilon_2\right)\cdot y. \tag{9}$$

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

$$\begin{aligned}\mathbb{P}_{fp} &\leq 1 - \mathbb{P}(\epsilon \leq \frac{q^2}{2}) = \mathbb{P}(\epsilon > \frac{q^2}{2}) \\ &\leq 2e^{-2y(\frac{q^2}{2})^2}.\end{aligned} \tag{10}$$

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 $\mathbb{P}_{fp} \leq \delta$, we set the threshold T_{in} such that $2e^{-2T_{in}(\frac{q^2}{2})^2} = \delta$. Solving for T_{in} gives $T_{in} = \frac{2\ln\frac{2d}{\delta}}{q^4}$. ■

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 ρ^*_i, exceeds the drop detection threshold T_{dr}. To bound the total false positive rate below δ, it is sufficient to ensure that each ρ^*_i may exceed T_{dr} with a probability $\delta_i = \frac{\delta}{d}$ (since we need to ensure the overall false positive $\sum_i \delta_i \leq \delta$), i.e., $\mathbb{P}(\rho^*_i > T_{dr}) < \frac{\delta}{d}$, which is equivalent to:

$$\mathbb{P}(\rho^*_i - \rho > T_{dr} - \rho) < \frac{\delta}{d}. \tag{11}$$

By using Hoeffding's inequality, we have:

$$\begin{aligned}&\mathbb{P}\Big(\rho^*_i - \rho > T_{dr} - \rho\Big) < 2e^{-2C^{good}_{i-1}(T_{dr}-\rho)^2} \\ &\Rightarrow\ C^{good}_{i-1} \geq \frac{\ln(\frac{2d}{\delta})}{2(T_{dr}-\rho)^2}.\end{aligned} \tag{12}$$

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 $C^{good}_i \geq \frac{\ln(\frac{2d}{\delta})}{2(T_{dr}-\rho)^2}$ for any $i < m$. Since we also have $C^{good}_i \geq N(1-T_{dr})^i$ for $i < m$, we get:

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

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^{recv}_m data packets, but claims that it receives C^{good}_m data packets, and drops x fraction of the received C^{good}_m data packets on l_{m+1}. We first have the following facts:

$$C^{recv}_m \leq C^{good}_{m-1}, \quad C^{good}_{m+1} = (1-x)C^{recv}_m + \beta. \tag{14}$$

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

$$\begin{aligned}&\frac{C^{good}_m}{C^{good}_{m-1}} \geq 1 - T_{dr}, \quad \frac{C^{good}_{m+1}}{C^{good}_m} \geq 1 - T_{dr} \\ &C^{good}_{m-1} \geq (1-T_{dr})^{m-1}N \geq (1-T_{dr})^d N.\end{aligned} \tag{15}$$

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

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

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

$$\theta = \frac{\eta^{leg}}{N} = \frac{C^{good}_{d+1} - \eta^{fake}}{N}. \tag{16}$$

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

$$C^{good}_{d+1} \geq (1-T_{dr})^d N, \quad \eta^{fake} \leq \beta. \tag{17}$$

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 Ω 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 Ω epochs to find a working path. ■