Computer Networks 158 (2019) 158–174

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

RFL: Robust fault localization on unreliable communication channels

Bo Wu[a, b], Ke Xu[a, b, *], Qi Li[b, c], Bingyang Liu[d], Shoushou Ren[d], Fan Yang[a], Meng Shen[e], Kui Ren[f, g]

[a] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[b] Beijing National Research Center for Information Science and Technology, Beijing, China
[c] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
[d] Huawei Technologies, Beijing, China
[e] School of Computer Science, Beijing Institute of Technology, Beijing, China
[f] Institute of Cyber Security Research, Zhejiang University, Hangzhou, China
[g] School of Computer Science and Engineering, Zhejiang University, Hangzhou, China

ARTICLE INFO

Article history:
Received 27 June 2018
Revised 13 January 2019
Accepted 23 April 2019
Available online 25 April 2019

Keywords:
Source and path verification
Fault localization
Symmetric key distribution

ABSTRACT

The current Internet is vulnerable to various attacks, e.g., source spoofing and flow hijacking attacks, which are incurred by misconfigurations or attacks. Either users or network operators are unable to easily localize these faults. Existing fault localization mechanisms can detect such attacks under an assumption that localization is performed upon reliable communication channels. Unfortunately, the assumption does not always hold. The forwarding paths of localization are not always reliable. Packets are usually dropped for some reasons. In particular, adversaries can interfere with fault localization by maliciously dropping packets. In this paper, we relax the assumption and propose a robust data-plane fault localization protocol named RFL that can localize faults and achieve source authenticity and path compliance even if communication channels in the network are not reliable. RFL samples and verifies packets in each network entity so that the packet source can efficiently localize faults of packet forwarding by verifying the sampled packets. By leveraging packet acknowledgment, packet sampling based fault localization is not impacted by packet loss in the communication channels. In particular, RFL leverages a symmetric key distribution scheme to implement robust key distribution among different entities, which ensures that packet sources can always correctly fresh their keys to perform correct localization. Our security and theoretical analysis demonstrates the robustness of RFL protocol. We implement the RFL prototype on Click routers. The experiment results with the prototype demonstrate that RFL achieves more than 99.5% localization accuracy while incurring only 10% throughput degradation.

1. Introduction

Reliable and secure data delivery in networks is highly desirable, especially for security-critical services enabled in a set of administrative domains (e.g., campuses, enterprises ISP, IXP and datacenter networks) [1], which ensures correct packet delivery along desired forwarding paths and with authentic origins. However, the current design of the Internet does not have such a property and always suffers from various attacks, e.g., packet source spoofing, traffic hijacking [2], and even DDoS attacks [3,4]. It is difficult to identify such attacks in networks in real time. IETF and ITU-T all illustrate the higher requirements for fault indication and localization, especially on a service provider network [5,6]. Existing troubleshooting tools, e.g., ping and traceroute, cannot effectively detect and localize the faults incurred by attacks, either. Thus, it is necessary to develop a data-plane fault localization to deter attackers and ensure correct packet forwarding. In order to address this issue, end-to-end source authentication [7,8] and path validation [9–11] have been extensively studied. They verify packet origins and forwarding paths by embedding cryptographic tags (or markings) to localize faults in packet forwarding. However, they assume that the packets used to verify markings can be correctly delivered over reliable communication channels. In fact, it is not always true because the channels to deliver packets are not always reliable due to attacks or network failures [12–14]. In particular, an adversary can drop the localization packets and interfere with the fault localization. Therefore, none of the existing schemes can accurately localize faults in unreliable networks without the help of centralized

* Corresponding author.
E-mail addresses: wub14@mails.tsinghua.edu.cn (B. Wu), xuke@tsinghua.edu.cn (K. Xu), qi.li@sz.tsinghua.edu.cn (Q. Li), liubingyang@huawei.com (B. Liu), renshoushou@huawei.com (S. Ren), y-f14@tsinghua.org.cn (F. Yang), shenmeng@bit.edu.cn (M. Shen), kuiren@zju.edu.cn (K. Ren).

https://doi.org/10.1016/j.comnet.2019.04.023
1389-1286/

servers. For example, OPT [2] and OSV [15] perform source authentication and path validation by verifying received packets. Without packet acknowledgment from intermediate entities and packet receivers, the packet sources cannot correctly localize the misbehaved entities. Existing fault localization mechanisms [13,16] cannot localize malicious entities if the localization packets are delivered over unreliable transmission channels. Meanwhile, centralized localization mechanisms [17,18] can only identify attacks and localize faults by leveraging central controllers, which is not easy to achieve in practice.

Therefore, the robust data-plane fault localization that can tolerate unreliable communication channels is not well addressed. We observe that source authentication and path validation is still an effective approach to localize faults in networks. However, it is still challenging to realize accurate and efficient fault localization if packets are delivered on unreliable channels. As we mentioned above, traditional fault localization cannot be tolerant to unreliable communication channels. Specifically, they are unable to tolerate interference from various adversaries, e.g., packet dropping, modification, and packet hijacking, which aim to bypass localization. Moreover, fault localization should not incur significant communication overhead in networks so that it will not significantly impact the performance of packet forwarding.

In this paper, we propose RFL, a robust fault localization, which ensures source authenticity and forwarding path compliance, even if localization is performed under unreliable communication channels. RFL leverages a packet marking mechanism to sample and verify the packets, which allows packet sources to efficiently acknowledge and verify the sent packets. In particular, it enables robust key sharing among different entities so that they can always have the correct keys to perform localization. RFL maintains a timer for each entity to ensure the robustness of localization, which allows them to request correct packets and drop unsolicited packets if the packets are dropped, modified, and hijacked during secret key distribution and packet forwarding verification. Thereby, each entity can effectively perform source and path verification with correct keys. Moreover, RFL uses a probabilistic sampling function to sample and verify packets at each hop, which efficiently verifies packets and localizes faults while significantly reducing the overhead incurred by verification. We qualitatively analyze the overhead of RFL. The theoretical analysis shows RFL introduces small overhead. For example, it only incurs around 6.03% communication overhead, which significantly outperforms existing schemes. We prototype RFL upon Click Modular Router and use experimental results to demonstrate the performance of RFL. The experimental results show that RFL achieves more than 99.5% localization accuracy, and obtains more than 90% throughput and 85% goodput. Therefore, RFL provides robust fault localization, while incurring negligible performance overhead.

The contributions of this paper are four-fold:

- We propose RFL, a scheme ensures fault localization for reliable data delivery, which tolerates interference from unreliable communication channels and does not require the help of central servers.
- We develop a robust secret key sharing scheme (RSKey) that achieves secure and robust symmetric key setup and distribution over unreliable communication channels.
- We design algorithms to verify source authenticity and correct packet forwarding paths, which can defend against various source spoofing and traffic hijacking attacks.
- We perform the security and theoretical analysis of RFL, and use real experiment upon RFL prototype to demonstrate the feasibility of RFL.

Compared with our previous paper [19], we propose an improved fault localization protocol such that it can tolerate unreliable communication channels. In particular, it resists interferences from adversaries, e.g., modifying, dropping and redirecting localization packets. Moreover, we develop a robust symmetric key distribution scheme called RSKey to ensure secure key sharing among entities even under reliable communication channels. We also evaluate the performance of RSKey. We analyze the fault localization accuracy and the analysis results are consistent with our experimental results.

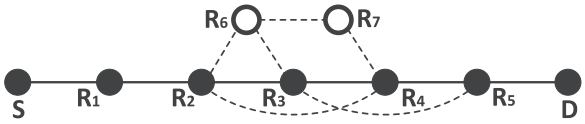


Fig. 1. Adversary model where R_2 is the misbehaved entity and the purposed forwarding path is $\Psi = \langle R_1, R_2, R_3, R_4, R_5 \rangle$ between S and D.

The remainder of this paper is organized as follows: in Section 2, we present our problem statement, including the adversary model and design goals. In Section 3, a high-level overview of RFL protocol is provided. In Sections 4, 5 and 6, we introduce the design details of RFL protocol, including robust symmetric key distribution, source and path verification, and fault localization. We respectively make some security analysis and theoretical analysis of RFL protocol in Sections 7 and 8. In Section 9, the experimental performance and evaluation are presented. In Section 10, we discuss a number of issues related to RFL. We cover the related work in Section 11. Finally, we conclude this paper in Section 12.

2. Problem statement

In this section, we present the adversary model and design goals. In this paper, we consider fault localization for end-to-end communication in multi-hop networks, where packets are delivered through a set of intermediate routers R_i $(1 \leq i \leq n)$ between a source S and a destination D, where n is the path length (not including S and D), and S, D and R_i are network entities in a network. Under an unreliable communication channel, a network entity may drop, modify, and hijack forwarding packets, which is usually incurred by attacks or network failures (e.g., misconfiguration and link failure). Both compromised and misconfigured entities are treated as misbehaved entities because they incur packet forwarding anomalies.

2.1. Adversary model

In this paper, we focus on fault localization by performing packet delivery verification, which ensures packet source authenticity and correct packet forwarding. Intuitively, Fig. 1 shows examples of attacks. We assume R_2 is the misbehaved entity, and $\Psi = \langle R_1, R_2, R_3, R_4, R_5 \rangle$, where Ψ is the desired forwarding path. In this case, R_2 can modify source address of IP packets originating from S for launching *source spoofing* attack. Besides, R_2 can also make the packets delivered along a path that differs from Ψ, e.g., $\langle R_2, R_6, R_3 \rangle$, $\langle R_2, R_6, R_7, R_4 \rangle$ and $\langle R_2, R_4 \rangle$ for the purpose of *path inconsistency*[1] attack. Moreover, if more than one misbehaved entities occur on Ψ, the packets can be transmitted along unordered entities, e.g., $\langle R_2, R_4, R_3, R_5 \rangle$.

As packets may be forwarded on unreliable communication channels, in particular, adversaries may interfere with fault localization, it is difficult to identify misbehaved entities. The misbehaved entity can launch various *sophisticated attacks* to evade to

[1] Actually, traffic hijacking is one of the instantiations of forwarding path inconsistency attack.

be localized. In details, the attacks can be classified into two categories. Firstly, the misbehaved entity can destroy or interfere with fault localization by unexpectedly discarding, modifying and redirecting some messages used to localize the fault. For example, when S tries to establish symmetric keys with intermediate entities on Ψ, R_2 can drop *request* packets (from S to R_i) or *ack* packets (from R_i to S) to destroy this procedure of key distribution; or when R_3 reports *ack* packet to S, R_2 can modify this packet data to interfere with fault localization. Secondly, the misbehaved entity can frame benign entities of their unrealistic misbehavior by launching frame attacks. For example, when receiving a packet, R_2 greatly reduces TTL value to 2, causing an illusion that R_4 is regarded as the fault due to its dropping packets.

2.2. Design goals

To defend against the above adversary model, the following desired properties should be satisfied in fault localization.

2.2.1. Source and path verification

Each entity on Ψ can perform source and path verification. Once source spoofing or path inconsistency occurs, the entity can identify and filter the unreliable packets.

2.2.2. High accuracy of fault localization

The packet source can effectively localize the fault with a high localization accuracy if any error occurs during symmetric key distribution or packet forwarding verification.

2.2.3. Robustness and lightweight

The fault localization can be robust, which no longer relies on reliable communication channels. Each entity is expectedly lightweight that does not store symmetric keys shared with other network entities.

In this paper, we focus on the data-plane fault localization by verifying source authenticity and path compliance. Considering the unreliable communication channels, we hope a robust fault localization scheme should be proposed, especially for the administrative domains that may be campus, ISP, datacenter or IXP networks. Then, it can be likely to be extended as a solution for fault localization in the general Internet. We will treat the detected entity and its neighbor as misbehaved entities because it is impossible to achieve the accuracy to a concrete entity of fault localization as the research [20] describes. As the current Internet, including network devices and end-hosts, are all unreliable, we hope the Internet users that try to pursue a higher security requirement for their outbound traffic should have the privilege to detect and localize the fault in this paper. Besides, this fault localization can be suitable for a set of various administrative domains (e.g., campus, ISP, datacenter and IXP networks) by making two pairs of vantage points periodically localize the offending routing entity. We assume end hosts (S and D) are trusted because it is meaningless for a malicious source to detect faked packet source and wrong forwarding path. Also, S should know the packet forwarding path Ψ, which can be learned by the existing control-data plane routing protocols [21,22], or it can be achieved by source routing [23]. Meanwhile, each entity on Ψ has long-lived public and private keys, and the public keys can be retrieved and verified by others, which is similar to existing secure routing protocols [2,13,16]. For an administrative domain, this can be achieved more easily under the centralized security management [17].

3. Overview of RFL

We now present an overview of RFL protocol that can accurately localize faults even on unreliable transmission channels in the presence of an adversary, e.g., dropping packets for bypassing localization. Similar to existing fault localization mechanisms [13,16,18], RFL performs fault localization based on various consecutive *epochs* that vary with different end-to-end communication sessions and different phases of the sessions. In each session, the *epoch* value will be updated after S sends a number of packets to D or a period of time interval expires. For an administrative domain (e.g., campus, ISP, datacenter and IXP network), the end-hosts (S and D) can also be the trusted pairs that are selected for monitoring network traffic and localizing the fault. RFL allows each entity to perform hop-by-hop verification during each epoch. In particular, RFL requires each entity to maintain a timer such that the entity can acknowledge the verification results if the timer expires, which ensures tolerance of unreliable transmission channels.

Fig. 2 shows the workflow of RFL protocol. At the beginning, RFL enables the source S to establish symmetric keys with all the other entities on Ψ. Using the symmetric keys, the packets verification and fault localization will then be carried out. Concretely, before each packet's departure, S firstly initializes RFL packets. Each entity will verify and probabilistically sample the packets on receiving them. At the end of each epoch, S will perform fault localization according to the sampling information from entities.

3.1. Robust symmetric key distribution

RFL introduces a robust symmetric key distribution called RSKey to guarantee secure key establishment and allocation between S and network entities in a forwarding path. As shown in Fig. 2, S sends a request packet (denoted by ReqKey) along Ψ towards D. Each entity computes and temporarily stores the symmetric key. Upon receiving ReqKey, D initializes and sends an ack packet (denoted by AckKey) back to S, which will gradually record the encrypted symmetric keys and signatures of each entity on Ψ hop by hop during AckKey delivery to S. Based on the received AckKey, S decrypts and obtains symmetric keys.

3.2. Lightweight source and path verification

After obtaining the symmetric keys, S pre-computes a marking for each entity on Ψ before sending out a data packet. All markings are inserted into a new header called RFL header between IP header and TCP header. During the packet transmission, each entity R_i performs packet forwarding verification by recalculating its own marking using symmetric key K_{S,R_i} once receiving packets, and comparing it with the inserted one on RFL header. Only these two values are equal, can R_i forward the packet to downstream entities. Note that each intermediate entity R_i does not require to store symmetric keys for each flow, which introduces lightweight RFL routers.

3.3. Robust fault localization

RFL enables entities to sample packets for fault localization. S uses the packet sampling information of each entity on Ψ to localize the fault. Our developed probabilistic packet sampling function $\mathcal{F}$ determines which entity will sample this packet in one epoch. The packet sampling results of each entity can be only predicted by S, but unknowable to others. Concretely, S uses $\mathcal{F}$ to learn which entities will sample this packet. Upon receiving a packet, the entities (R_i and D) perform $\mathcal{F}$ to sample this packet and store the results in a bloom filter. At the end of each epoch, S sends a request probing packet, called ReqProb, to ask all entities on Ψ for their sampling results of this epoch. The ack message (denoted by AckProb) carrying the encrypted sampling results will be delivered from D to S. Thereby, based on the received AckProb, S determines where the fault occurs.

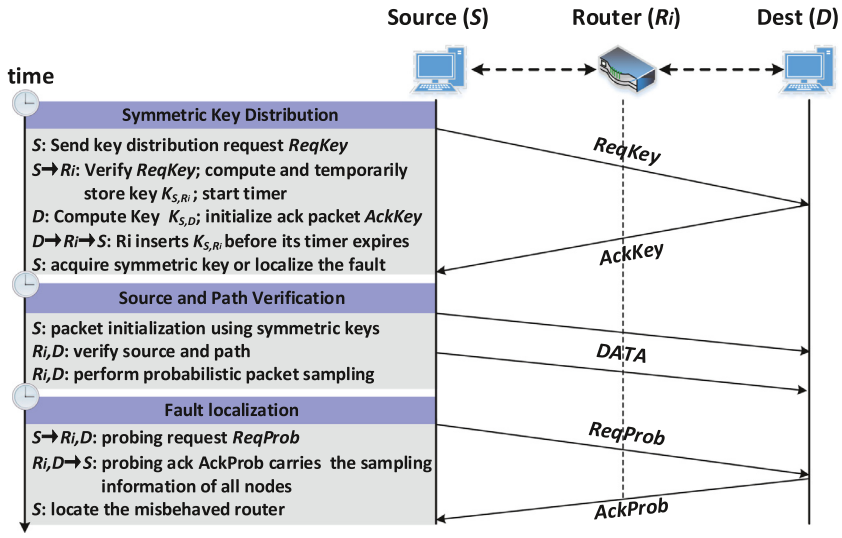


Fig. 2. The workflow of RFL protocol for the source (S), intermediate entities (R_i) and the destination (D), where ReqKey and AckKey (ReqProb and AckProb) respectively represent the request and acknowledgment messages for secret key distribution (fault localization) probing.

However, it is challenging to implement RFL because of the following issues:

3.4. Corrupting symmetric key distribution

The symmetric key is an important guarantee to correctly perform packet forwarding verification and fault localization. If an adversary maliciously corrupts symmetric key distribution (e.g., modify, drop and redirect keys). RFL will not be able to perform localization correctly.

3.5. Heavyweight verification overhead

To verify packet forwarding, each intermediate entity may suffer high verification overhead, especially the overhead to store various keys for verification. Thus, it is necessary to design a verification mechanism that does not require storing keys in the intermediate entities.

3.6. Corrupting fault localization

An adversary can interfere with the transmission of request or ack packet to corrupt RFL. For example, a misbehaved entity does not correctly respond S with its symmetric key or packet sampling results, incurring a failure of fault localization. Also, the misbehaved entity can also launch frame attacks to corrupt RFL, which can trigger benign entities to drop packets by mistake, e.g., because of the TTL values manipulated by the misbehaved entity.

4. Robust symmetric key distribution

In this section, we present our proposed RSKey, a robust symmetric key sharing between S (or a vantage entity within an administrative domain) and each entity on Ψ, which allows different entities to obtain correct keys so that it can perform correct localization. Similar to DRKey [2], RSKey provides the stateless operation on routers and enables each router not to store symmetric keys. More especially, it is tolerant to unreliable communication channels during symmetric key distribution, which does not rely on routing entities to work normally. RSKey scheme offers robustness against the interference of misbehaved entities and enables S to localize the fault even facing an offending entity that may be under attack or behave malevolently.

4.1. ReqKey transmission from S to D

To achieve sharing symmetric keys with other entities, RSKey enables the request packet i.e., ReqKey, to be transmitted from S to D, as Algorithm 1 shows. S firstly initializes ReqKey packet and

Algorithm 1 ReqKey Transmission from S to D.

function ReqKey Initialization by S ( )
Obtain: Ψ={S, R_1,...R_n, D}, $PubK$={K_1, ... K_n}
Measure: T_{start}
Compute:
  $SessionID = H(K_S||K_D||T_{start})$
  $Sign_S = Sign_{K_S^{-1}}(H(D||\Psi SessionID||PubK)$
$ReqKey = \{\Psi, SessionID, PubK, Sign_S\}$
Start timer $\mathcal{T}_S$
end function
function ReqKey Transmission from S to R_n( )
Forward ReqKey from S to each router R_i along Ψ
Check $Sign_S$
Record TTL value ttl_i^r of ReqKey
Compute and temporarily store $K_{S,R_i} = PRF_{LSI_i}(SessionID||S)$
Start timer $\mathcal{T}_i$
end function
function ReqKey Transmission from R_n to D( )
Forward ReqKey from R_n to D
Check $Sign_S$
Record TTL value ttl_d^r of ReqKey
Compute $K_{S,D} = PRF_{LSI_d}(SessionID||S)$
end function

delivers it to D through Ψ. ReqKey contains Ψ, session identifier $SessionID$, the public key set $PubK$ of all entities on Ψ and S's signature $Sign_S$, as Eq. (1) shows.

$$ReqKey = \{\Psi, SessionID, PubK, Sign_S\}, \quad (1)$$

where $SessionID$ is the session identifier that is the hash over the public keys of end-hosts (K_S and K_D) and the start time of this session (T_{start}). It can be computed by the following equation:

$$SessionID = H(K_S||K_D||T_{start}). \quad (2)$$

As Eq. (3) shows, $Sign_S$ uses the destination address (D) as one of the inputs, contributing to defending against redirection or hijacking attack by modifying the destination address of ReqKey packet.

Certainly, $Sign_S$ is also keyed with K_S^{-1} that makes ReqKey away from source spoofing.

$$Sign_S = Sign_{K_S^{-1}} H(\mathrm{D}||\Psi||SessionID||PubK). \quad (3)$$

In addition to checking $Sign_S$, each intermediate entity R_i creates and temporarily stores the symmetric key K_{S,R_i}, which is calculated using pseudo-random function (PRF) keyed by local secret information (LSI_i), only known by R_i (see Eq. (4)).

$$K_{S,R_i} = PRF_{LSI_i}(SessionID||\mathrm{S}). \quad (4)$$

R_i starts a timer $\mathcal{T}_i$ when ReqKey is delivered to the downstream entity, where $\mathcal{T}_i$ is a timer maintained on R_i. It starts when ReqKey arrives and expires after a certain timeout, called *timer threshold* that can be obtained by *round-trip time* (RTT). The created key K_{S,R_i} is temporarily stored until AckKey comes or $\mathcal{T}_i$ expires. More considerately, RSKey makes each entity temporarily record the TTL value ttl_i^r when receiving ReqKey, which helps to avoid TTL attack (detailed shortly).

4.2. AckKey transmission from D to S

To reply ReqKey, AckKey is initialized by D and then sent back through intermediate entities towards S, whose construction is as Eq. (5) shows.

$$AckKey = \{ReqKey, EncK_{S,D}, SignK_D\}. \quad (5)$$

$EncK_{S,D}$ and $SignK_D$ are the encrypted $K_{S,D}$ and D's signature, respectively (detailed in Algorithm 2).

Algorithm 2 AckKey Transmission from D to S.

function AckKey Initialization by D ( )
Compute:
  $EncK_{S,D} = Enc_{K_S}(K_{S,D})$, $ttl_d = ttl_d^r$
  $SignK_D = Sign_{K_D^{-1}}(H(K_S||ttl_d||ReqKey||EncK_{S,D}))$
  $AckKey$={$ReqKey$,$EncK_{S,D}$,$SignK_D$}
end function
function AckKey Transmission from D to S ( )
Forward $AckKey$ from D to each router R_i along Ψ
  if $\mathcal{T}_i$ <timer threshold **then**
    Check $SignK_{i+1}$ and record TTL value ttl_i^a of $AckKey$
    Compute:
    $Enck_{S,R_i} = Enc_{K_S}(K_{S,R_i})$, $ttl_i = ttl_i^r + ttl_i^a$
    $EncKset_i = \{EncK_{S,R_i}, \ldots, EncK_{S,D}\}$
    $SignKset_i = \{SignK_{i+1}, \ldots, SignK_D\}$
    $SignK_{R_i} = Sign_{K_i^{-1}}(H(K_S||ttl_i||ReqKey||EncKset_i||SignKset_i))$
    $AckKey = \{ReqKey, EncKset_i, SignKset_i, SignK_{R_i}\}$
  else
    Compute:
    $Enck_{S,R_i} = Enc_{K_S}(K_{S,R_i})$
    $SignK_{R_i} = Sign_{K_i^{-1}}(H(K_S||ttl_i||ReqKey||EncK_{S,R_i}))$
    $AckKey = \{ReqKey,EncK_{S,R_i},SignK_{R_i}\}$
  end if
Forward $AckKey$ from R_1 to S
end function

During AckKey transmission, each entity adds its encrypted symmetric key $EncK_{S,R_i}$ and signature $SignK_{R_i}$ into AckKey. Note that when computing the signature $SignK_{R_i}$, the encrypted symmetric keys and signatures of downstream entities (i.e., $R_{i+1}, \ldots, R_D$) will also be added as the input. In this case, each router R_i only checks $SignK_{R_{i+1}}$ of its 1-hop downstream R_{i+1}, which can also defend against frame attacks. Therefore, AckKey has the following construction (Eq. (6)) when it arrives at R_i:

$$AckKey = \{ReqKey, EncKset_i, SignKset_i, SignK_{S,R_i}\}, \quad (6)$$

where $EncKset_i$ and $SignKset_i$ are the sets of encrypted symmetric keys and signatures, respectively. Thus, AckKey can record the encrypted symmetric keys of all entities when it arrives at S, as case I in Fig. 3 shows.

To deal with the malicious discarding of either ReqKey or AckKey packet by the misbehaved entity, each entity (including S) starts the timer $\mathcal{T}_i$ when ReqKey passes through. If $\mathcal{T}_i$ expires, i.e., R_i does not receive AckKey within timer threshold, R_i initializes a new AckKey, which contains $EncK_{S,K_{R_i}}$ and $Sign_{K_{R_i}}$. Thus, if any misbehaved entity drops ReqKey or AckKey, its upstream entity close to S will forward its AckKey to S. This is as case II in Fig. 3 shows, where the misbehaved entity R_3 can modify, drop and redirect ReqKey or AckKey packet for interfering with S obtaining symmetric keys from all intermediate entities. In this case, S can achieve fault localization to identify the misbehaved entity (described shortly). To avoid TTL attack, we use TTL values of both received ReqKey and AckKey as the input to compute $SignK_{R_i}$. When AckKey is verified at each hop, R_i obtains purposed TTL value of ReqKey and AckKey of other entities based on Ψ, and then use them to check others' signatures. If the misbehaved entity launches TTL attack, the signatures of its downstream entities will not be verified successfully.

4.3. Symmetric key acquisition

The source S needs to retrieve the symmetric keys shared with other entities on Ψ, which can be used to perform packet forwarding verification and fault localization (see Algorithm 3). In RFL, S

Algorithm 3 Symmetric Key retrieval by S.

function Symmetric Key Retrival by S ( )
Require: $AckKey$
Compute:
  if $\mathcal{T}_S$ <timer threshold **then**
    for $1 \le i \le n+1$ **do**
      if Check$SignK_{R_i}$ **then**
        $K_{S,R_i} = Dec_{K_S^{-1}}(EncK_{S,R_i})$
      elseLocate $\langle R_{i-1},R_i\rangle$ as the fault.
      end if
    end for
  elseLocate R_1 as the fault.
  end if
end function

can obtain symmetric keys from the received AckKey packet. Concretely, once receiving AckKey, S firstly checks the signatures and decrypts the encrypted keys to obtain symmetric keys. For each entity from R_1 to D on Ψ, if its signature is verified successfully using its public key, S can obtain the symmetric key by decrypting the encrypted key using S's private key. In this way, the symmetric key set (denoted by $\mathbb{L}$) can be obtained:

$$\mathbb{L} = \langle K_{S,R_1}, K_{S,R_2}, \ldots, K_{S,D}\rangle. \quad (7)$$

Obviously, as there may be misbehaved entities to interfere with the transmission of ReqKey and AckKey, S may only gain the subset (denoted by $\mathbb{L}'$) of $\mathbb{L}$, i.e., $\mathbb{L}' \subseteq \mathbb{L}$:

$$\mathbb{L}' = \langle K_{S,R_1}, K_{S,R_2}, \ldots, K_{S,R_m}\rangle. \quad (8)$$

Then S affirms confidently there is at least one misbehaved entity between R_m and R_{m+1} on Ψ. Another case is that S never receives $AckKey_{(i)}$ before $\mathcal{T}_S$ expires, which illustrates R_1 did not initialize and send $AckKey_1$ back. In this case, R_1 will be localized as the fault. With the function of fault localization, our proposed RSKey provides more robust symmetric key distribution, where the misbehaved entity has to behave normally for bypassing fault localization.

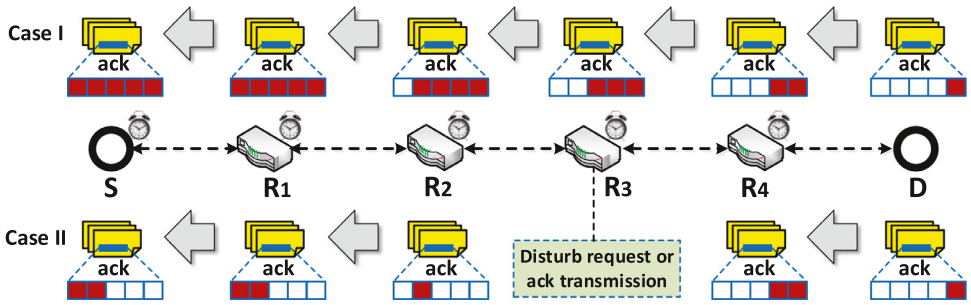


Fig. 3. RSKey can ensure the robustness of symmetric keys distribution even facing unreliable communication channels. The red rectangle represents the ack of each network entity. Case I shows the AckKey packet can be securely delivered to S, while case II depicts the misbehaved entity R_3 can interfere with ReqKey or AckKey packet transmission.

5. Lightweight source and path verification

In this section, we present lightweight source and path verification to detect packet forwarding anomalies by verifying packets forwarding at different entities, which does not require retrieving and storing keys in intermediate entities for verification. RFL allows S (or a vantage entity) to initialize RFL protocol by using the keys obtained during the key distribution phase such that different intermediate entities can perform packet verification. It uses the keys to pre-compute markings for each entity on Ψ and inserts these markings into packets. During the packet delivery towards D, each entity dynamically recomputes the secret key and then verifies the packet by recalculating their own marking.

5.1. RFL protocol initialization

RFL protocol is initialized to make markings for each entity calculated and inserted in the packet header. RFL enables S to compute the markings using symmetric keys for all entities, and generates a new packet header called RFL header to include these markings. Inspired by some current secure network architectures (e.g., HIP [24,25] and DONA [26]), the RFL header locates at 3.5 layer between IP header and TCP header, which is conducive to hop-by-hop packet verification. Before sending packets, S initializes RFL headers with the following structure:

$$RFL\ header = \{SessionID, epoch, PacketID, M_{Path}\}. \tag{9}$$

PacketID is the unique packet identifier, calculated based on Eq. (10):

$$PacketID = H(SessionID||epoch||IP_{cst}), \tag{10}$$

where IP_{cst} represents the constant portion of an IP packet (excluding variable fields, such as TTL and checksum) during forwarding. M_{Path} (in Eq. (9)) denotes the pre-computed marking sequence for later verification at each entity, as Eq. (11) shows.

$$M_{Path} = \langle M_1, M_2, \ldots, M_n, M_D \rangle, \tag{11}$$

where M_i is pre-calculated for each entity using PRF keyed with shared symmetric keys, as Eq. (12) shows, where M^{in}_{cst} is the splice of constant input: $SessionID||epoch||PacketID||S||D$. Adding source and destination address (denoted by S and D) as the input can also help to defend against both source spoofing and traffic redirection that are caused by modifying the source and the destination address of an IP packet, respectively.

In Eq. (12), TTL_i and TTL_D respectively denote the purposed TTL value when the packet arrives at R_i and D. The marking is pre-computed for each entity on the reverse Ψ. More especially, if there is a misbehaved routing entity on the forwarding path, it can maliciously modify the inserted markings that are used for packet verification at downstream entities. This can bring illegal packet losses when the packet is delivered to downstream entities, who will be localized as the fault. In order to address this issue, RFL supports the integrity protection for RFL header, especially for inserted markings,[2] i.e., M_{path}. In Eq. (12), the markings of downstream entities are firstly computed and R_i's marking M_i is pre-computed by using its downstream entities markings as inputs. Therefore, if an intermediate entity maliciously modifies the marking(s) of downstream entities, this malicious entities' next hop(s) will drop the delivered packet(s), preventing frame attacks caused by malicious modification of the pre-inserted markings on RFL header.

$$\begin{aligned} M_D &= PRF_{K_{S,D}}(M^{in}_{cst}||TTL_D||R_n). \\ M_i &= PRF_{K_{S,R_i}}(M^{in}_{cst}||TTL_i||R_{i\text{-}1}||M_{i+1}||\ldots||M_n||M_D). \end{aligned} \tag{12}$$

The fields *SessionID, epoch* and *PacketID* in RFL header respectively occupies 128 bits, 16 bits and 128 bits. Each marking occupies 32 bits, whose rationality analysis is shown in Section 8.

5.2. Source and path verification

RFL allows each intermediate entity to verify packet source and forwarding path by verifying the markings encoded in RFL header. Note that RFL enables each entity on Ψ not to store any symmetric key, which can be calculated according to Eq. (4). This makes entities lightweight and can protect the state of exhaustion (e.g., DoS) attack. During transmission, the packet is verified for its origin and forwarding path at each hop. Concretely, each entity uses Eq. (12) to recalculate the marking M'_i. If the computed M'_i equals to M_i in RFL header, it illustrates that the source and the forwarding path are all correct up to the current entity. Else, i.e., the verification fails, the packet will be dropped at this hop. This verification by each entity for the received packet can also prevent the state exhaustion attacks on D.

6. Robust fault localization

In this section, we present a robust fault localization that can accurately localize faults even if an adversary interferes with the localization. RFL allows each entity to probabilistically sample forwarding packets and send sampling results to S (or a vantage entity within administrative domains). By using a positive-ratio-bound fault localization algorithm, S can efficiently and accurately localize the fault, which also tolerates unreliable communication channels. For every epoch, S respectively establishes one *bloom filter* [28] for each entity. We define $\mathcal{B}^e_i$ and $\mathcal{B}^e_D$ as the bloom filter with L-bits length for the packet sampling on R_i and D for an epoch, which is established by S.

Before the departure of each packet, S uses probabilistic packet sampling function $\mathcal{F}$ (detailed below) to learn which entity will

[2] In fact, integrity protection is crucial for the design and deployment of any network protocols and architecture [27].

sample this packet. For example, if R_η will sample this packet according to $\mathcal{F}$, S samples this packet in $\mathcal{B}^e_\eta$. In other words, $\mathcal{F}$ determines the packet sampling on R_i, which is only known by S and R_i. Concretely, with symmetric keys and *PacketID* in RFL header, S computes and gains the 128-bit hash value:

$$H_{sampling} = H(K_{S,R_i}||PacketID). \tag{13}$$

We define ω as the number of selected bottom bits of $H_{sampling}$. If ω binaries are all equal to 0, i.e., no 1 appears in this lower ω bits binary, this packet will be sampled on the corresponding entity. In this case, ρ-th binary in the corresponding $\mathcal{B}^e_i$ or $\mathcal{B}^e_D$ will be switched from 0 to 1, where $\rho = PRF(H_{sampling}), 0 \le \rho < L$. For different packets that have been sampled, RFL might result in a same value of ρ, which is called sampling collision in this paper. To address this issue, RFL enables the entity to switch the next bit, i.e., (ρ+1)-th bit, from 0 to 1 until no sampling collision occurs. At the same time, each entity establishes two local bloom filters, one ($\mathcal{B}^e_{R_i}$ or $\mathcal{B}^e_d$) for the current epoch and another ($\mathcal{B}^{e+1}_{R_i}$ or $\mathcal{B}^{e+1}_d$) for the next. Note that the storage overhead is analyzed in Section 8. Then packet sampling is carried out at each entity according to $\mathcal{F}$, where the result is stored in the local bloom filter and can only be known by the corresponding entity and S.

At the end of each epoch, S tries to obtain the packet sampling information of all entities for localizing the fault by sending a request packet (ReqProb), which is initialized as Algorithm 4 shows.

Algorithm 4 ReqProb and AckProb Initialization.

function *ReqProb* INITIALIZATION BY S ( )
Require: Ψ, *SessionID*, *epoch*, *S*, *D*
Compute:
  $M^{req}_{cst} = \Psi||SessionID||epoch||S||D$
  $M^{req}_{D} = PRF_{K_{S,D}}(M^{req}_{cst}||TTL_D||R_n)$
  for i from n to 1 **do**
    $M^{req}_{i} = PRF_{K_{S,D}}(M^{req}_{cst}||TTL_i||R_{i-1}||M^{req}_{i+1}||\ldots||M^{req}_{n}||M^{req}_{D})$
  end for
  $M^{req}_{Path} = \langle M^{req}_1, \ldots, M^{req}_n, M^{req}_D\rangle$
  $ReqProb = \{\Psi, SessionID, epoch, M^{req}_{Path}\}$
  S delivers *ReqProb* to intermediate entities towards D.
end function
function ACKPROB INITIALIZATION BY D ( )
Require: Ψ, *SessionID*, *epoch*, $\mathcal{B}^e_d$, K_S, K_D^{-1}
Compute:
  $Enc\mathcal{B}^e_d = Enc_{K_{S,D}}(\mathcal{B}^e_d)$
  $Sign\mathcal{B}^e_d = Sign_{K_D^{-1}}(H(K_S||\Psi||SessionID||epoch||TTL^{ack}_d||Enc\mathcal{B}^e_d))$
  $AckProb = \{\Psi, SessionID, epoch, Enc\mathcal{B}^e_d, Sign\mathcal{B}^e_d\}$
Forwarding:
  D delivers AckProb to intermediate entities towards S.
end function

On receiving ReqProb, R_i firstly performs source and path verification via recomputing M^{req}_i, the pre-inserted marking for R_i in ReqProb. Then R_i starts a timer $\mathcal{T}_i$ and forwards ReqProb to the downstream entity towards D. When receiving ReqProb, D initializes a probing ack packet (AckProb) according to Algorithm 4, in which the encrypted $\mathcal{B}^e_d$ and D's signature are all added. Then, D sends AckProb back through entities on Ψ until it arrives at S. During AckProb transmission, each entity R_i firstly checks all signatures in AckProb. If no error occurs, R_i inserts its encrypted $\mathcal{B}^e_{R_i}$ and signature in AckProb. Thus, the architecture of AckProb is as Eq. (14) shows when it arrives at R_i and is overwritten by R_i.

$$AckProb = \{\Psi, SessionID, epoch, Enc\mathcal{B}^e_d, Sign\mathcal{B}^e_d \\ Enc\mathcal{B}^e_{R_n}, Sign\mathcal{B}^e_{R_n}, \ldots, Enc\mathcal{B}^e_{R_i}, Sign\mathcal{B}^e_{R_i}\}, \tag{14}$$

where $Sign\mathcal{B}^e_{R_i}$ uses all encrypted bloom filters of entities from R_i to D as the calculation input, as shown in Eq. (15), to prevent the previously encrypted bloom filters from malicious modification and avoids frame attack. If the forwarding path is asymmetric,[3] R_i will create AckProb$_i$ to send its sampling information back to S when $\mathcal{T}_i$ on R_i expires. On receiving AckProb$_i$, R_j ($0 < j < i$) will also check the signatures and add its $\mathcal{B}^e_{R_j}$ and $Sign\mathcal{B}^e_{R_j}$ to AckProb$_i$.

$$Sign\mathcal{B}^e_{R_i} = Sign_{K_i^{-1}}(H(K_S||\Psi||SessionID||epoch \\ ||TTL^{ack}_i||Enc\mathcal{B}^e_d||Enc\mathcal{B}^e_{R_n}||\ldots||Enc\mathcal{B}^e_{R_i})). \tag{15}$$

6.1. Positive-ratio-bound fault localization

Targeting at the data-plane fault localization, RFL tries to identify the difference between the actual and pre-computed sampling results for determining which routing entity is misbehaved. This is because S shares symmetric keys with intermediate routing entities, which unify the way for pre-computed and actual packet sampling. In order to express this difference and easily localize the fault, we introduce *positive ratio* denoted by $\mathcal{P}_i$ and $\mathcal{P}_D$ for R_i and D, which illustrates the probability that the corresponding entity is misbehaved. Meanwhile, considering the possibility of a natural loss of a link or routing entity, we introduce *positive ratio threshold* (denoted by ζ) that illustrates the probability of abnormal sampling only caused by packet natural loss. When $\mathcal{P}_i$ is larger than ζ, and $\mathcal{P}_1, \ldots, \mathcal{P}_{i\text{-}1}$ are all less than ζ, we can identify R_τ or $R_{i\text{-}1}$ as the misbehaved entity. As Algorithm 5 shows, when receiving AckPorb,

Algorithm 5 *Positive-ratio-bound* Fault Localization.

function FAULT LOCALIZATION ( )
Require: *AckPorb*, K_i, $\mathcal{B}^e_i$ ($1 \le i \le n$), K_D, $\mathcal{B}^e_D$, ζ
  for $1 \le i \le n$ **do**
    if !($CheckSig_{K_i}(Sign\mathcal{B}^e_{R_i})$ & $CheckSig_{K_D}(Sign\mathcal{B}^e_{R_D})$) **then**
      R_1 is localized as the misbehaved entity.
    end if
    Compute: $\mathcal{B}^e_{R_i} = Dec_{K_{S,R_i}}(Enc\mathcal{B}^e_{R_i})$, $\mathcal{B}^e_{i,R_i} = \mathcal{B}^e_i \bigoplus \mathcal{B}^e_{R_i}$
  end for
  Compute: $\mathcal{B}^e_d = Dec_{K_{S,D}}(Enc\mathcal{B}^e_d)$, $\mathcal{B}^e_{D,d} = \mathcal{B}^e_D \bigoplus \mathcal{B}^e_d$
  for $1 \le i \le n$ **do**
    $\mathcal{C}_i$:*Count binary 1 in* $\mathcal{B}^e_i$, $\mathcal{C}'_i$:*Count binary 1 in* $\mathcal{B}^e_{i,R_i}$
    Compute: $\mathcal{P}_i = \mathcal{C}_i/\mathcal{C}'_i$
  end for
  $\mathcal{C}_D$:*Count binary 1 in* $\mathcal{B}^e_D$, $\mathcal{C}'_D$:*Count binary 1 in* $\mathcal{B}^e_{D,d}$
  Compute: $\mathcal{P}_D = \mathcal{C}_D/\mathcal{C}'_D$, $\mathcal{P}_0 = 0$, $\mathcal{P}_{D\text{-}1} = \mathcal{P}_n$
  for τ *from* 1 *to* n *and* D **do**
    if $\mathcal{P}_\tau \ge \zeta$ & $\mathcal{P}_{\tau\text{-}1} < \zeta$ **then**
      $\langle R_{\tau\text{-}1}, R_\tau\rangle$ is localized as the misbehaved entity.
    end if
  end for
end function

S firstly checks the signatures $Sign\mathcal{B}^e_{R_i}$ ($1 \le i \le n$) and $Sign\mathcal{B}^e_D$ to verify the validity and authenticity of AckPorb. If any error occurs, S can localize R_1 as a misbehaved entity, because R_1 still forwards an invalid AckProb to S instead of verifying and drop it. Concretely, if R_ε ($1 < \varepsilon \le n$) on Ψ modified AckPorb, $R_{\varepsilon\text{-}1}, \ldots, R_1$ will all discard AckPorb when receiving it. In this case, if AckPorb with error signatures arrives at S, R_1 either modify it or ignores the signature verification for collusion attack. Note that if no AckProb is received, S can directly localize R_1 as the fault, because R_1 neither forwards AckPorb to S nor initializes it AckProb$_1$.

After all signatures in AckProb are verified by S, $\mathcal{B}^e_{R_i}$ and $\mathcal{B}^e_d$ can be obtained through decrypting $Enc\mathcal{B}^e_{R_i}$ and $Enc\mathcal{B}^e_d$. We define

[3] Actually, many forwarding paths in today's Internet are asymmetric [29], which will be discussed in Section 10.

$\mathcal{C}_i$ ($\mathcal{C}_d$) as the number of packet sampling difference between pre-computed sampling $\mathcal{B}_i^e$ ($\mathcal{B}_D^e$) and actual sampling $\mathcal{B}_{R_i}^e$ ($\mathcal{B}_d^e$), which actually is the number of binary 1 in $\mathcal{B}_i^e \bigoplus \mathcal{B}_{R_i}^e$ ($\mathcal{B}_D^e \bigoplus \mathcal{B}_d^e$). In RFL protocol, *positive ratio* $\mathcal{P}_i$ ($\mathcal{P}_D$) is the ratio of $\mathcal{C}_i$ ($\mathcal{C}_d$) to bloom filter length L. S tries to find $\mathcal{P}_\tau$ (τ=1,..., n, D) that meets $\mathcal{P}_\tau \geq \zeta_\tau$ and $\mathcal{P}_{\tau\text{-}1} < \zeta_{\tau\text{-}1}$, where ζ is the threshold of positive ratio of entity (detailed in Section 9). In this case, $\langle R_{\tau\text{-}1}, R_\tau \rangle$ is localized as the misbehaved entity, because one of $R_{\tau\text{-}1}$ and R_τ modified packet origin and path, causing the actual packet sampling in $\mathcal{B}_{R_\tau}^e$ is disturbed. Note that if $\mathcal{P}_D \geq \zeta_D$ and $\mathcal{P}_n < \zeta_n$, R_n is then localized by S. More generally, when S receives AckProb$_i$ instead of AckProb, actual sampling results of $R_{i+1}, \ldots, R_n$, D will be regarded as empty set, i.e., $\mathcal{B}_{R_{i+1}}^e = \ldots = \mathcal{B}_{R_n}^e = \mathcal{B}_d^e = \varnothing$. Then the fault can also be localized according to Algorithm 5.

7. Security analysis

In this section, we make security analysis of RFL protocol, especially against data-plane attacks (i.e., source spoofing and path inconsistency) from misbehaved entities and sophisticated attacks resulting from unreliable detection channels. In our adversary model, the misbehaved entity can modify source address and forwarding path, interfere with secret key distribution and try to corrupt fault localization. We show RFL protocol is secure against both a single misbehaved entity and multiple colluding entities even over unreliable channels.

7.1. Source spoofing

Modifying the source address of an IP packet by R_τ will introduce marking discrepancies between the pre-inserted (by S) and the recomputed (by downstream entities) values. For example, if R_τ corrupts the packet source address, all downstream entities (e.g., $R_{\tau+1}$) will drop this received packet, because the recalculated marking $M'_{\tau+1}$ (according to Eq. (12)) is not equal to the inserted value $M_{\tau+1}$ in RFL header as the source address also acts as an input of marking calculation.

7.2. Forwarding path inconsistency

As packet forwarding is verified hop by hop, corrupting packets forwarding path will cause non-correspondences between the pre-inserted and the recomputed markings for downstream entities on the actual forwarding path. For example, if R_τ delivers the packets to $R\varphi$ instead of $R_{\tau+1}$, $R\varphi$ will discard this packet, because the recalculated marking M'_φ does not equal $M_{\tau+1}$ in RFL header. As for collusion attacks, we will discuss it shortly.

7.3. Corrupting symmetric keys distribution

The misbehaved entity can drop, modify and redirect ReqKey or AckKey packet to destroy symmetric keys establishment and distribution, which is a precondition of the data-plane source and path verification in RFL protocol. In our proposed RSKey, the expired timer $\mathcal{T}_i$ on each entity enables R_τ's upstream entities to send the created symmetric keys to S. Besides, each entity will also verify ReqKey and AckKey packet, such as the integrity and authenticity verification, to avoid the modification or redirection of these two types of packets. More importantly, based on acknowledgements in AckKey$_{(i)}$ packet, S can localize the misbehaved entity if any error occurs during the distribution of symmetric keys, which makes R_τ have to behave normally to avoid the fault localization.

7.4. Corrupting fault localization

There are three methods for R_τ to corrupt RFL protocol. Firstly, R_τ can disturb the packet verification and the sampling operation by means of tampering the markings of downstream entities, which is pre-inserted in RFL header. For example, R_τ modifies $M_{\tau+2}$ in RFL header, causing $R_{\tau+2}$ to drop this packet. This can frame $\langle R_{\tau+1}, R_{\tau+2} \rangle$ as a misbehaved entity. In RFL protocol, each entity R_i uses the markings of downstream entities ($M_{i+1}, \ldots, M_D$) in RFL header as inputs to recompute the marking M'_i, as shown in Eq. (12). In this case, the recomputed markings differ from the pre-inserted markings. Thus, $R_{\tau+1}$ will drop the packet and $\langle R_\tau, R_{\tau+1} \rangle$ will be regarded as the fault if R_τ corrupts $M_{\tau+\Delta}$ ($2 \leq \Delta \leq n\text{-}\tau$) in RFL header.

Secondly, R_τ can drop, modify and redirect ReqProb and AckProb packet to prevent S from obtaining sampling information of entities. In RFL protocol, if *timer* $\mathcal{T}_{\tau\text{-}1}$ expires, AckProb$_{\tau\text{-}1}$ packet will be initialized and sent back to S. According to the *positive-ratio-bound* fault localization, $\mathcal{P}_\tau \geq \zeta$ and $\mathcal{P}_{\tau\text{-}1} < \zeta$ can make $\langle R_{\tau\text{-}1}, R_\tau \rangle$ easily localized.

Thirdly, R_τ can frame other entities on Ψ by launching TTL attack. For example, R_τ lowers TTL value of ReqProb packet to a smaller value (say, κ) with the purpose that $R_{\tau+\kappa}$ drops this packet and can be then localized as the fault. RFL protocol can deal with TTL attack, in which TTL value is added to compute signatures on AckProb$_{(i)}$ packet during sampling information transmission and fault localization, and to calculate the markings on ReqProb (see Eq. (12)) at RFL header initialization stage. Therefore, if R_τ launches TTL attack, the packet will be discarded at downstream entities. In this case, R_τ and one of its neighbor on Ψ will be localized as the fault.

7.5. Tunnel-based misbehavior

The forwarded packets can be bypassed by the misbehaved entity using a tunnel implemented at any layer, which seems to be able to evade the hop-by-hop packet verification in RFL. That makes the packets with spoofed source address or inconsistent forwarding path not be dropped. However, this tunnel-based misbehavior can be discovered in this paper. RFL enables each routing entity to perform packet sampling operations and periodically send sampling results to the source. In this case, if some packets are bypassed and forwarded to another destination using a tunnel-based attack, these packets will not be sampled in the downstream entities. As the probabilistic packet sampling function (in Section 6) is determined by the established symmetric key, the sampling results can be only known by the source and some routing entity, where others would not know the results. In this case, the sampling results of downstream entities will not be consistent with the sources calculated results. So, the source will localize the misbehaved entity that launches a tunnel-based attack.

7.6. Collusion attacks

All attacks discussed above can also be launched by more than one colluding entities. In this case, we can prove by induction method that RFL protocol works well to defend against collusion attacks. We give a proof sketch as following:

We assume there is another entity (denoted by R_σ, $\tau < \sigma \leq n$) on Ψ colluding with R_τ. Without loss of generality:

1) In the first case where R_τ is not adjacent to R_σ (i.e., $\sigma > \tau$+1), (i) if R_τ launches the above source spoofing or path inconsistency attacks while the packets are forwarded, the intermediate entities between R_τ and R_σ will also perform the verification for source and path, and then drop the corrupted packets. (ii) If R_τ corrupts

fault localization in any case of three methods described above, R_τ and its one neighbor will be localized as the fault.

2) In the other case where R_τ and R_σ are adjacent to each other (i.e., $\sigma = \tau+1$), we can regard these two entities as one single "virtual" misbehaved entity R_v with upstream entity $R_{\tau\text{-}1}$ and downstream entity $R_{\tau+2}$. (i) If R_v launches the above source spoofing or path inconsistency attack described above, $R_{\tau+2}$ will find the verification for packet origin and path fails, and then drop the corrupted packet. (ii) If R_v corrupts the fault localization, S can also obtain the packet sampling information of $R_1, \ldots, R_{\tau\text{-}1}$ (i.e., $\mathcal{B}^e_{R_1}, \ldots, \mathcal{B}^e_{R_{\tau-1}}$). In this case, at least one entity of R_τ and R_σ will be localized as the misbehaved entity.

8. Theoretical analysis

In this section, we provide the theoretical analysis for fault localization accuracy, communication and storage overhead, and some key parameters. Meanwhile, we analyze the rationalization of RFL's source and path verification, bloom filter size and S's detection interval.

8.1. Fault localization accuracy

In this section, we make an in-depth theoretical analysis about fault localization accuracy. In RFL, no matter what happens in these attacks: source spoofing, forwarding path inconsistency, packet modification and redirection, the packets will be certainly dropped as RFL enables every entity to perform packet forwarding verification. Therefore, we can regard these abnormal actions as dropping packets. We respectively define θ_{na} and θ_{mis} as the probability of natural loss and malicious loss (mis-loss) of the entity (as well as its upstream neighbored link), where $\theta_{mis} > \theta_{na}$. Thus, the positive ratio threshold ζ_i of R_i is as Eq. (16) shows, where no malicious loss occurs during packet forwarding. From Eq. (16), we can learn the ζ_i of R_i is positively related to θ_{na}, where more natural losses can cause a larger positive ratio threshold.

$$\zeta_i = 1 - (1 - \theta_{na})^i. \tag{16}$$

When the misbehaved entity R_i drops packets with the probability θ_{mis}, its positive ratio $\mathcal{P}_i$ is as Eq. (17) shows, where a larger θ_{na} can result in more observed packet losses for the routing entity R_i.

$$\mathcal{P}_i = 1 - (1 - \theta_{na})^{i-1} \cdot (1 - \theta_{mis}). \tag{17}$$

Thereby, the fault localization accuracy (denoted by δ) that identifies R_i as the misbehaved entity is as Eq. (18) shows, where $\mathrm{P}(\cdot)$ denotes the probability that satisfies some constraint conditions in brackets. We can learn the higher positive ratio brings about the higher accuracy of fault localization. Based on Eqs. (16) and (17), we also learn both the smaller θ_{na} and the larger θ_{mis} contribute to easily localizing the fault.

$$\delta = P(\mathcal{P}_i > \zeta_i \ \&\ \mathcal{P}_1 \le \zeta_1 \ \&\ \ldots \ \&\ \mathcal{P}_{i\text{-}1} \le \zeta_{i\text{-}1}). \tag{18}$$

8.2. Communication overhead

In RFL protocol, RFL header is the additional communication overhead. From Section 5.1, we can learn (*38+4n*) bytes are occupied in RFL header, where n is the length of Ψ. According to the research Huffaker et al. [30], the average end-to-end path length of the Internet is 13.11 hops, i.e., $n = 13.11$, causing the communication overhead of 90.44 bytes in RFL protocol. It is worth mentioning that about 85% data of the Internet is transmitted by large ($>$1400 bytes) packet [31]. We can adjust the packet sizes by configuring the interface of Maximum Transmission Unit (MTU). In this case, with packet size $P_{size} = 1500$ bytes, RFL communication overhead accounts for 6.03% of the entire IP packet.

Table 1
Communication overhead comparison.

	RFL	OPT	ICING	OSV	Faultprints
Com-overhead (Byte)	90.44	277.76	563.62	134.22	160.88
Com-overhead ratio (%)	6.03	18.52	37.57	8.95	10.73

Com-overhead is short for communication overhead.

Compared with other related mechanisms [2,13,15,32], RFL protocol outperforms in terms of communication overhead and its ratio under average path length and large packet (1500 bytes) of Internet, as shown in Table 1.

8.3. Verification rationalization

From Section 5.1, we can learn each pre-computed marking M_i occupies 32 bits. The verification at each hop is mainly performed by employing *PRF* to recompute the marking M_i. We must accept that although the inputs are different, there is also a probability, donated by φ, to result in the same output of *PRF* as the hash collision occurs. As every bit has an equal collision probability (i.e., 0.5), φ is the probability that collision of all 32 bits occurs at the same time, i.e., $\varphi = \frac{1}{2^{32}}$. This illustrates sending 2^{32} packets or $2^{32} \cdot 1500 > 2^{42} = 4\,\mathrm{TB}$ data (almost impossible for the normal end-to-end communication) will only cause this verification collision, averagely. Therefore, RFL protocol can perform rationally verification for source authenticity and path compliance as φ is small enough.

8.4. Bloom filter size

In RFL, end-hosts and intermediate entities all store at least two *bloom filters*, which is an important factor in determining their storage overhead. Although RFL tries to decrease storage requirements, blindly reducing *bloom filter* size is not advisable. On one hand, *bloom filter* should be sufficient to store the packet sampling information of one *epoch*, where the usage rate ξ^e_i of $\mathcal{B}^e_i$ does not exceed its the usage rate threshold ξ^e_{th}. This is also related to the link bandwidth (detailed in Section 8.6). On the other hand, false positive rate increases with the decrease of *bloom filter* size. For one *epoch e*, at most $L{\cdot}\xi^e_{th}$ packets will be sampled, causing $\mathrm{C}(L, L{\cdot}\xi^e_{th})$ sampling results, where $\mathrm{C}(\cdot)$ donates the number of combinations of L and $L{\cdot}\xi^e_{th}$. Therefore, the false positive rate is $\mathbb{F} = C(L,\ L \cdot \xi^e_{th})^{-1}$.

In this paper, we set *bloom filter* size $L = 1$ Kb and usage rate threshold $\xi^e_{th} = 0.8$. In this case, $\mathbb{F} \ll 0.001\%$, which is reasonable in RFL protocol.

8.5. Storage overhead

We make an analysis for RFL's storage overhead of routing entities and end hosts (S and D), and compare it with some related fault localization schemes (e.g., Faultprints [13], ShorMAC [16] and DynaFL [18]) in Table 2. We can learn our proposed RFL achieves the same storage overhead for both routing entities and end hosts, compared with Faultprints that is the best-of-the-art for inter-domain fault localization. In contrast, ShortMAC and DynaFL, although outperforming in some aspects (e.g., the sources storage of ShortMAC), does not provide robust fault localization on unreliable communication channels. Besides, DynaFL assumes a trusted central authority for localizing the fault, which easily becomes compromised due to malicious attacks in practice. More detailed analysis for RFL's storage overhead is shown as follows.

Table 2
Storage overhead comparison.

	RFL	Faultprints	ShortMAC	DynaFL
Routing entity (R_i)	1*key+2*NL*	1*key+2*NL*	*N**key+2*NL*	key**#neighbors*+2*L*
Source (S)	(n+1)*key+2(n+1)*L*	(n+1)*key+2(n+1)*L*	(n+1)*key	N/A
Destination (D)	1*key+2*L*	1*key+2*L*	1*key+2*L*	N/A

N is the number of sessions in each router per-second; *#neightbors* denotes the number of neighboring ASes or routing entities; *L* in this table is the size of bloom filter or counter.

8.5.1. Router storage overhead

While performing symmetric key distribution, RSKey requires intermediate entities to store an established symmetric key (16 bytes) temporarily before its timer expires. Note that RSKey is performed only in the initial period of RFL, and enables each intermediate router to store 16-byte shared key for a short duration.[4] After the shared key is obtained by S, each intermediate router does not have to store the symmetric key for per-path or per-source during packet delivery verification. Thus, RFL provides a lightweight router in terms of the symmetric key storage.

To sample the forwarded packets, R_i establishes two *bloom filters* for the current and next epoch of each session. Therefore, the storage overhead of router is *2L* bits for per-session and *2NL* bits i.e., O(N) with the fixed bloom filter size *L* for all sessions, where N is the number of sessions in each router per-second. According to the CAIDA results [31], 12.91K application sessions, on average, are observed in a router per-second. Therefore, each routing entity storage overhead is (12.9K · 2L) bits. Due to 1 MB = 2^{20} B = 2^{20}·8 bits, we can obtain each routing entity storage overhead in RFL is (12.9K · 2L) bits = (12.9K · 2L)/(2^{20}·8) MB for storing bloom filters.

8.5.2. End-host storage overhead

For one session, S stores the symmetric keys shared with entities on Ψ. With 128 bits length of each secret key, 16(n+1) bytes is occupied to store these keys. For each entity, two *bloom filters* are established, resulting in extra storage overhead of 2(n+1)*L* bytes in S. So the source storage overhead for one session is (16+2*L*)(*n*+1) bytes. On the destination host, only two *bloom filters* should be stored for packet sampling and localization, introducing its storage overhead of $\frac{2L}{8} = 0.25L$ bytes. Thus, S and D have storage overhead of O(n) and O(1) for the fixed bloom filter size *L* per-session. With the average path length of the Internet (i.e., $n = 13$ hops), their storage overhead is 146.28 bytes and 3.28 bytes, respectively.

8.6. Detection interval

When S switches *epoch* value, ReqProb packet is forwarded to all entities on Ψ for requiring their packet sampling information. We respectively define T_{MI} and W_B as the detection interval and link bandwidth. As the existence of other sessions, there are at most $\frac{W_B}{P_{size}}$ packets of current session and $\frac{W_B}{2^{\omega}\cdot P_{size}}$ packets of each epoch, which can be delivered and sampled by entities per-second. Eq. (19) shows the *bloom filter* is consumed to the usage rate threshold ξ_{th}^{e} after time interval T_{MI}, where the sampling capacity (denoted by *SC*) is defined as the total sizes of all sampled packets during each epoch.

$$\frac{W_B}{2^{\omega}\cdot P_{size}}\cdot T_{MI} \;=\; L\cdot\xi_{th}^{e} \;\Rightarrow\; T_{MI} = \frac{L\cdot\xi_{th}^{e}\cdot 2^{\omega}\cdot P_{size}}{W_B} = 2^{\omega}\cdot\frac{SC}{W_B} \tag{19}$$

From Eq. (19), we can learn the detection interval is affected by several factors in the proposed RFL protocol. The number (denoted by ω) of selected bottom bits determines the sampling probability

[4] The duration can be equal to the timer threshold, which is evaluated by a round-trip-time (RTT).

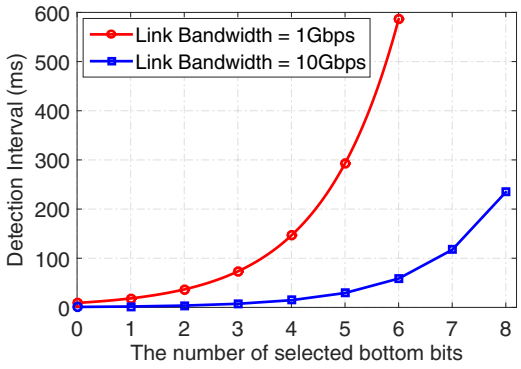


Fig. 4. Detection interval T_{MI} *vs.* the number of selected bottom bits ω.

(detailed in Section 6), where 2^{ω} also has a positive correlation to the detection interval. When $L = 1$ Kb, $\xi_{th}^{e} = 80\%$, $P_{size} = 1500$ bytes and W_B =1 Gbps, we can learn T_{MI} increases as the selected bottom bits ω increases at the link bandwidth of both 1 Gbps and 10 Gbps, as Fig. 4 shows. According to Basescu and Co-authors [13,33], with the average value of 225 ms, T_{MI} between 100 ms and 350 ms conforms to the realistic network. The detection interval of RFL protocol is carefully set mainly based on prior measurements in realistic networks, which can also be achieved by adjusting the value of ω in RFL. In this case, $\omega = 5$, $T_{MI} = 292.97$ ms when W_B =1 Gbps and $\omega = 8$, $T_{MI} = 234.38$ ms when W_B =10 Gbps meet the requirements of detection interval in realistic network, respectively.

8.7. Computation overhead

During packet transmission, each routing entity in the proposed RFL will perform the following operations. (i) Compute the symmetric key K_{S,R_i} shared with the source. This is based on Eq. (4) with O(1) PRF operation. (ii) Verify the source and path of received packets. Each routing entity will recompute the marking using the symmetric key and compare the calculated value with the inserted one in RFL header. This process is based on Eq. (12) with O(n-i) PRF operation, where i is the location of the routing entity. iii) Perform probabilistic packet sampling. When the packet passes the verification (in Section 5.2), routing entity will sample this packet according to a sampling probability (in Section 6). In this process, O(1) hash operation (based on Eq. (13)) and O(1) PRF operation (to compute the sampling location in a bloom filter) will be introduced. Therefore, based on the above analysis, the computation overhead of each routing entity is O(n). Note that the computation overhead has a significant effect on forwarding efficiency, which is evaluated in Section 9.3.

9. Performance evaluation

In this section, we will evaluate RFL's performance, including its localization accuracy, the performance of key sharing and packet forwarding with RFL.

Experiment setup. We use simulation network to evaluate the positive ratio threshold (Section 9.1) and fault localization accuracy (Section 9.2), and employ our implemented RFL prototype

based on Click router for evaluating packet forwarding (Section 9.3) and the performance of key sharing (Section 9.4). The simulation network is implemented by leveraging OMNeT++ that is an extensible, modular, component-based C++ simulation library and framework for building network simulators [34]. We implement RFL protocol described in Section 4, 5 and 6 with RSKey as a user-level application for carrying out secret key distribution, source and path verification, and fault localization. The prototype of the router, called RFL router, is implemented by using Click Modular Router [35], which runs on Ubuntu Linux 12.04 with Intel(R) Core(TM) i5-4590, CPU @ 3.30 GHz, 16GB memory and NIC of 1000 Mbps[5] We achieve communications between two computers acting as the source and the destination (Intel(R) Core(TM) i5-4200U, CPU @ 1.6GHz/2.3 GHz, 12.0 GB memory) through RFL router. In this evaluation, we use SHA-3 algorithm to compute hash values of long strings, such as the calculation of PacketID (see Eq. (10)). For computing the PRF value, we use HMAC based on SHA-1. We respectively truncate the value of SHA-3 and HMAC to meet our requirements, such as from 256 bits to 128 bits for computing PacketID and from 160 bits to 32 bits for computing M_i. In RSKey, RSA algorithm is employed to compute signatures, asymmetric encryption, and decryption. To evaluate the effect of fault localization, such as localization accuracy, we implement a multi-hop simulation network, where the source communicates with the detestation through more than one intermediate routing node.

9.1. Positive ratio threshold

We evaluate positive ratio threshold ζ_i of R_i through the simulation network with the path length of 20 hops, the longest forwarding path in end-to-end communication according to a CAIDA research [30]. The positive ratio illustrates the probability that the corresponding entity is misbehaved. When $\mathcal{P}_i$ is larger than ζ_i, and $\mathcal{P}_1, \ldots, \mathcal{P}_{i\text{-}1}$ are respectively less than $\zeta_1, \ldots, \zeta_{i\text{-}1}$, we can identify R_i or $R_{i\text{-}1}$ as the misbehaved entity (detailed in Section 6). In RFL, the attacks (including source spoofing, path hijacking, frame, and collusion attack) all cause downstream entities to drop the packets.

To evaluate the positive ratio under different router locations and packet natural loss probabilities, we have considered several different actual packet loss probabilities, which is denoted by θ in this paper. Note that θ can reflect three types of packet loss: no packet loss ideally ($\theta = 0$), natural packet loss ($\theta = \theta_{na}$) and malicious packet loss ($\theta = \theta_{mis}$). In order to be able to represent these three types of packet losses more generally, we select θ as the parameter/notation for evaluating positive ratio in Fig. 5. Fig. 5(a) shows the relationship between router location and its positive ratio with the variation of packet loss probability. To obtain a more accurate result, we run our simulation over 500 times for each result. The red line depicts the scenarios only with natural packet loss, which is also a threshold line that helps identify the misbehaved entity. According to the research [36], many network scenarios for real-time communications indicate specific requirements of Qualify of Service (QoS), including packet loss less than 0.1%. More especially, the network performance requirements between consumer network edge to a large company's network edge must be less than 0.1% packet loss during any 15s interval. In this case, we set the value of natural packet loss probability as 0.001, i.e., ($\theta_{na} = 0.001$). When the packet loss probability is 0.00, the positive ratio $\mathcal{P}$ is lower than the threshold value, because the lower packet loss probability brings about less packet loss during transmission. We set one misbehaved router at different locations from R_1 to R_{20}. The blue line ($\theta = 0.10$) and the green dotted line ($\theta = 0.05$) respectively show the positive ratio $\mathcal{P}$ of the misbehaved router in different locations. We learn that the positive ratio $\mathcal{P}_i$ will exceed the threshold ζ_i when the malicious router behaves abnormally with the probability of $\theta = 0.10$ or $\theta = 0.05$. According to this threshold, S can locate the misbehaved router who has different mis-loss probabilities.

We also evaluate the relationship between positive ratio and natural packet loss probabilities under different misbehaved packet loss probability. Fig. 5(b) shows the positive ratio of the middle entity (i.e., R_7) with the average Internet forwarding path length, i.e., $n = 13$ hops. With the increment of natural packet loss probability, the positive ratio threshold becomes larger, because the higher loss probability can easily introduce more packet loss. When the misbehaved packet loss probability is larger than the natural loss probability, the corresponding ratios exceed the threshold value. Therefore, under different natural packet loss probabilities, the fault can also be localized in RFL protocol.

9.2. Fault localization accuracy

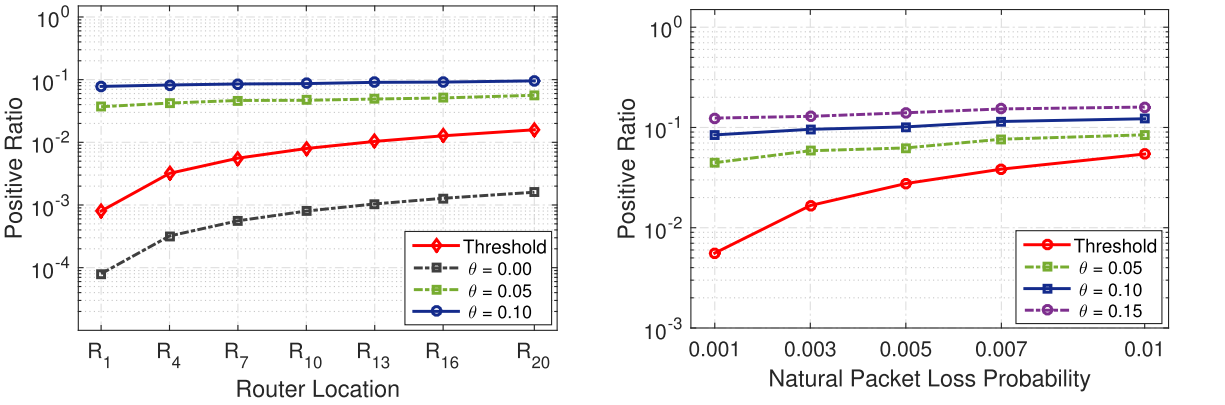
Based on the positive ratio threshold above, we evaluate fault localization accuracy denoted by δ through a simulation scenario with 13-hop forwarding path, which is the average forwarding path length during packet transmission [30]. We set one router of random location on forwarding path as the misbehaved router, that can launch both source spoofing and path inconsistency attacks. Besides, this misbehaved router can also disturb RFL protocol, which finally introduces the packets dropping.

We first evaluate fault localization accuracy with the variation of packet mis-loss probability of misbehaved router, just as Fig. 6(a) shows. We can learn the network function over 2% packet loss over a period of time is a strong indicator of problems [37]. Thus, we try to adjust the value of mis-loss probability around 2% (from 0.01 to 0.03) for evaluating the localization accuracy. From Fig. 6(a), we can learn fault localization accuracy of RFL becomes higher with the increase of packet mis-loss probability. This is because more packet mis-loss results in higher positive ratio than the threshold, and the higher mis-loss probability make RFL easier to identify the fault. We respectively take the value of natural packet loss probability as 0.001, 0.003 and 0.005, which introduce different positive ratio thresholds (described in Section 9.1). From Fig. 6(a), we know less natural packet loss brings about higher localization accuracy, as the lower positive ratio threshold makes it easier to localize the fault. This is in line with our theoretical analysis in Section 8.1.

Then the localization accuracy trends over different natural packet loss probabilities are evaluated in Fig. 6(b). For a fixed packet mis-loss probability (such as 1.5%), the localization accuracy of RFL will become lower when the natural packet loss increases. This is because the more natural losses will introduce a larger positive ratio threshold (according to Eq. (16)). In this case, the localization accuracy will become lower for a constant mis-loss probability, which can be theoretically learned from Eq. (18). Fortunately, under the smaller natural packet loss probability, such as 0.001 or 0.003, RFL achieves the fault localization with the accuracy of over 99.5% when the mis-loss probability is 0.020 or more.

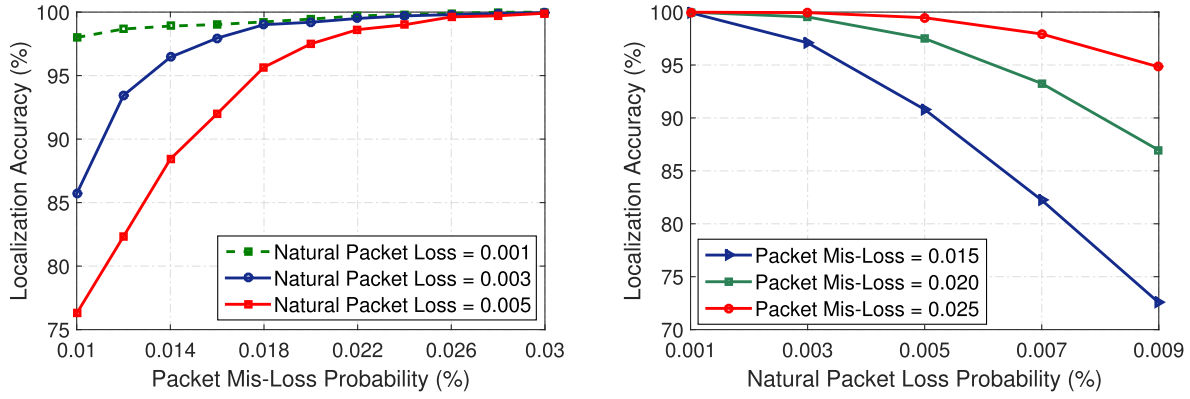
We define β as the ratio between packet mis-loss and natural loss probabilities, i.e., $\beta = \theta_{mis} / \theta_{na}$. Based on the observations mentioned above, we make a further evaluation for the localization accuracy in terms of different values of β, as Fig. 7 shows, where 0.001, 0.003, 0.005 are set as the value of θ_{na}, respectively. We can learn the localization accuracy is positively correlated to the ratio β, where fault localization becomes more accurate when the value of β increases. This is because an increased β can introduce

[5] Of course, it is also feasible for the similar packet processing as RFL to be implemented along fast TCAM path or in commercial router [13].



(a) Positive ratio *vs.* router locations. (b) Positive ratio *vs.* natural loss probabilities.

Fig. 5. The positive ratio for different router location and packet natural loss probabilities.



(a) Localization accuracy for different packet mis-loss probabilities. (b) Localization accuracy for different packet natural loss probabilities.

Fig. 6. The Localization accuracy for different packet mis-loss or natural loss probabilities.

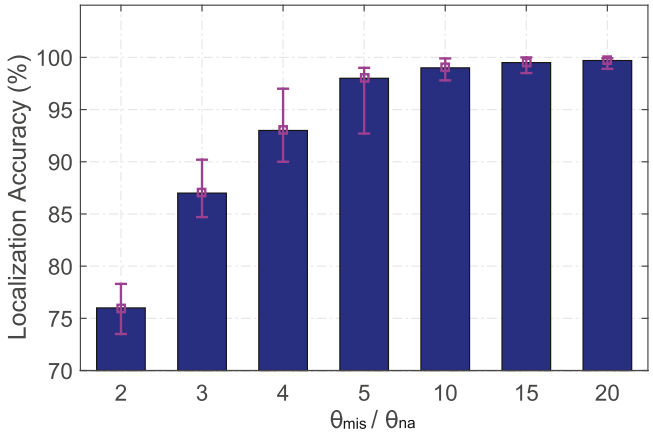


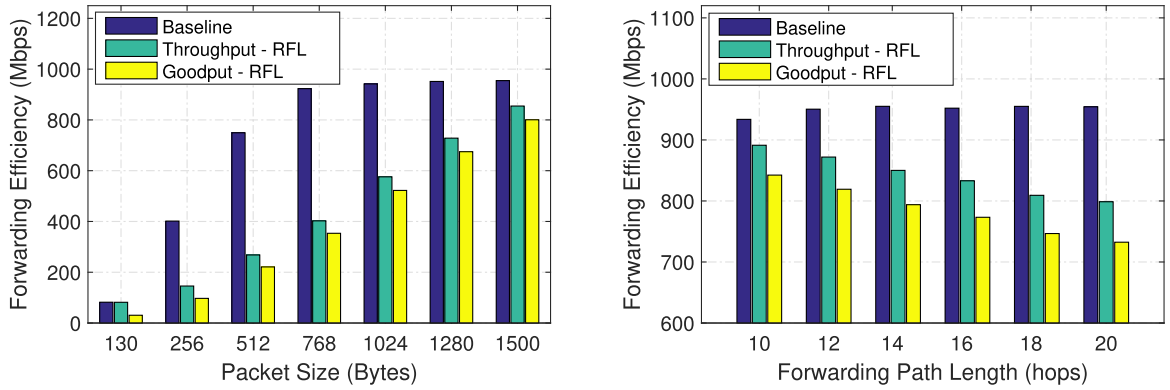
Fig. 7. The localization accuracy for different ratios β between θ_{mis} and θ_{na}.

a larger θ_{mis} than θ_{na}, which keeps consistent with the analysis in Section 8.1.

9.3. Router throughput and goodput

We implement the prototype of RFL router to evaluate the forwarding efficiency, including throughput and goodput, which can be used to demonstrate the technical feasibility in real experiments. In order to evaluate the packet forwarding efficiency at routing entities, we use iperf [38] to achieve the communications between the source and the destination through RFL router. RFL router performs the following operations when delivering packets: source and path validation, probabilistic packet sampling and storing sampling results.

From Eq. (12), we learn the computation overhead of RFL router increases linearly from R_n to R_1 when recomputing the markings. Namely, RFL routers close to S have higher computation overhead than the router close to D as the longer input. In this case, we evaluate the performance of middle router $R_{\lfloor \frac{n}{2} \rfloor}$ for the average-case analysis with different path lengths and packet sizes. Fig. 8(a) shows the relationship between packet size and forwarding efficiency with the average Internet path length of 13 hops. We calculate goodput as the valid throughput of useful packet data, excluding RFL header. Note that the smallest packet size is 130 bytes, including 90-byte RFL header and 40-byte IP/TCP header. We adjust packet size of 130 bytes to 1500 bytes by configuring the interface Maximum Transmission Unit (MTU) sizes. From Fig. 8(a), we can know both throughput and goodput of RFL router increase with the improvement of packet size. Especially for the large packet of 1500 bytes, RFL router can achieve over 90% throughput and about 85% goodput of baseline. From [30], we learn that the path length of end-to-end communication is 15.3 ± 4.2 hops for IPv4 packets. Thus, we evaluate the packet forwarding efficiency with different



(a) Forwarding efficiency *vs.* packet sizes with the average Internet path length of 13 hops.

(b) Forwarding efficiency *vs.* path lengths with the packet sizes of 1500 bytes.

Fig. 8. Forwarding efficiency of RFL router for different packet sizes and path lengths.

Table 3
RSKey evaluation - communication overhead.

	RSKey	DRKey	ICING
Source (S)	2	2	4^*n+4
Router (R_i)	2	2	4^*n+4
Destination (D)	2	2	4^*n+4

Here, the communication overhead is evaluated by the number of extra packets during symmetric key distribution.

Table 4
RSKey evaluation - ReqKey and AckKey packet latency.

	Entity	Path length	Latency
ReqKey	Source (S)	Irrelevant	653 µs
	Router (R_i)	Irrelevant	548 µs
	Destination (D)	Irrelevant	627 µs
AckKey	Source (S)	11	13,128 µs
		13	16,534 µs
		15	20,176 µs
		17	24,051 µs
		19	28,815 µs
	Router (R_i)	Irrelevant	903 µs
	Destination (D)	Irrelevant	754 µs

path lengths (from 10 hops to 20 hops) for the same packet size of 1500 bytes in Fig. 8(b). We can learn that RFL router's throughput and goodput all decrease when the forwarding path length increases because more downstream routers' markings are added as the input of marking recomputation. The longer input of marking recomputation incurs lower forwarding efficiency. Concretely, with path length increment of 1 hop, the throughput and goodput will reduce by 9.26 Mbps. Fortunately, the throughput and goodput respectively exceed 850 Mbps and 800 Mbps in the networks of 13-hop path length, more than 90% and 85% compared to the baseline. Thus RFL incurs only 10% forwarding efficiency while guaranteeing the robustness of fault localization, which is an incomparable advantage of other packet verification mechanism.

9.4. Performance and overhead of RSKey

To demonstrate the feasibility of RSKey in the early stage of RFL protocol, we perform RSKey performance evaluation during symmetric key distribution. The evaluation results show that RSKey introduces low communication overhead and packet latency.

Table 3 provides the results of communication overhead (evaluated by the number of extra packets) at the source (S), the router (R_i) and the destination (D). We can learn RSKey achieves the same lower communication overhead with the state-of-the-art scheme DRKey [2], and outperforms than ICING [32]. Note that RSKey and DRKey all require only 2 additional packets as the communication overhead for the symmetric key establishment, while ICING needs 4^*n+4 extra packets on all entities.

More importantly, RSKey achieves a more secure and robust symmetric key distribution than the state-of-the-art scheme DRKey. On one hand, if any misbehaved router disturbs secret key distribution by means of dropping, modifying or redirecting *request* or *acknowledgement* packet, DRKey scheme will fail. However, RSKey enables each entity to verify the *request* or *acknowledgement* message. If any failure occurs or the timer expires, the entity will send its encrypted symmetric key back to S. In this case, S can still obtain reliable entities' symmetric key even if there is any router trying to disturb secret key establishment. On the other hand, if any misbehaved router disturbs key distribution, DRKey will be stranded, resulting in S's wasting time to wait for acknowledgment message. However, RSKey can realize it as soon as possible with the help of timer, and localize the fault who disturbs secret key distribution. In this case, S can adjust the policies (e.g., avoiding the localized misbehaved router or correcting the fault) based on the result of fault localization.

To evaluate packet latency during RSKey processing, we implement the prototype of RSKey on the source host, the RFL router, and the destination host. RSKey scheme contains two stages: ReqKey stage and AckKey stage. From Section 4.1, we can learn the packet latency on different entities is irrelevant to path length on ReqKey stage. In this case, we will ignore the effect of path length changes on entities' performance.

Table 4 shows the packet latency of ReqKey stage and AckKey stage on the source, intermediate routers, and the destination. We can learn the packet latency of RSKey on the source is affected by path length, especially on AckKey stage. That is mainly because the source will check all the signatures and decrypts the encrypted symmetric keys when receiving AckKey packet, leading to higher computation overhead in proportion to path length. This result also shows the packet latency on RFL router is smaller than at least one of end hosts, which enables more cycles to be used to distribute secret keys by the source or the destination.

10. Discussion

10.1. Asymmetric paths

We mainly describe the fault localization for the packet verification with symmetrical paths, while many forwarding paths between the source and the destination in the current Internet are asymmetric [29,39,40]. RFL can also provide the compatibility for asymmetric paths. With the asymmetric paths, the timer $\mathcal{T}_i$ will expire as R_i does not receive AckKey or AckProb packets from its downstream entities. In this case, R_i then creates and initializes $AckKey_i$ or $AckProb_i$, which will be delivered to the source. Based on the encrypted symmetric keys or sampling information, the source can also identify and localize the misbehaved entity (see Algorithm 5).

10.2. Forwarding path instability

RFL enable the source to pre-insert the markings of entities on Ψ into RFL header for later packet verification. There is still a probability that the forwarding path Ψ changes due to the link failure, network congestion, and misconfiguration, making both the packets dropped incorrectly and the fault localized wrongly. Fortunately, the network end-to-end communications keep stable from tens of minutes to several days [41,42]. Besides, the stable forwarding paths (longer than 6 hours) will be chosen to transmit packets for 96% of times [41]. To further address this problem, the source and perform RSKey several times until it can obtain symmetric keys of all entities on the latest purposed path Ψ. Using the newly obtained symmetric keys, RFL provides a better compatibility for network instability.

10.3. Changed natural packet loss

The proposed RFL protocol enables the source to compute the positive ratio threshold (Eq. (16)) using the natural loss probability that is based on the prior observed knowledge for network communications. This can be obtained by using several existing methods, such as ping, traceroute, Paessler [43], VoIP Spear [44] and Appneta [45], which is beyond the research scope of this paper. Due to the dynamic network traffic that may incur a changed natural loss probability, RFL allows the source to update the positive ratio threshold for adapting to the change of natural packet loss rate. In this case, RFL can avoid false detection for a misbehaved router even though packet loss changes due to network traffic load instability.

10.4. Tradeoff between robustness and overhead

In RFL, there is a conflict between robustness and overhead. On one hand, to ensure the robustness against unreliable communication channels, RFL introduces the timer for each entity, which also incurs extra overhead. Fortunately, RFL enables each entity to use its timer only receiving request packets (i.e., ReqKey and ReqProb). This can help to lower the overhead of each entity to a certain extent. On the other hand, each entity has to store bloom filters for sampling packets. However, we think the packet sampling operation is an essential element for constructing robust fault localization. Through our analysis in Section 8.5, the average storage overhead of each entity is within an acceptable range, fortunately.

10.5. Incremental deployment

The proposed RFL protocol in this paper is compatible with the presence of routing entities that do not deploy RFL protocol, and provide a strong adaptability for the incremental deployment from the following aspects. i) RFL keeps the current IP header unchangeable and employs RFL header for packet delivery verification and fault localization, in which RFL-less entities can still forward RFL's packets based on the information in IP header without any disturbs due to the presence of RFL header. ii) Although RFL protocol introduces hop-by-hop packet verification for ensuring source authenticity and path compliance, the illegal packets with spoofed source(s) or redirected forwarding path(s) can be still filtered as long as there is only one RFL-compatible routing entity on the actual forwarding path. This is because this entity can recalculate its marking (Eq. (12)) and compare it with inserted one in the RLF header. iii) The proposed RFL protocol mainly focuses on the fault localization during packet transmission. The routing entities that have not installed RFL protocol will not affect the packet sampling of other RFL-compatible entities, whose sampling information can be still delivered to S at the end of each epoch. Using a portion of sampling results only from RFL-compatible entities, RFL enables S to narrow the scope of the fault based on Algorithm 5.

10.6. Comparison between centralized and distributed solutions

The centralized methods usually rely on an authority that collects diagnostic results and performs fault localization, which can be easily deployed (especially under the control of a single entity) [18,46] and apply to more complex networks (using machine learning [47] and big data techniques [48]). However, they ignore users security requirements for monitoring their traffic and tolerate the possibility that the compromised authority occurs, which can result in an incorrect localization or a false warning. By contrast, the distributed methods can avoid a compromised authority and achieve hop-by-hop packet verification, while introducing frequent interactions among network entities, which ignores the unreliable communication channels. The proposed RFL can achieve robust fault localization even facing unreliable communication channels, which can be easily deployed especially in administrative domains and does not rely on any third-party authority.

11. Related work

11.1. Secure routing and forwarding

Routing security has been widely studied to ensure correct packet forwarding on the Internet [11,21,49–54]. IETF RFC 6480 proposes the RPKI architecture as an infrastructure to support secure Internet routing [49]. S-BGP [11] verified the authenticity of announced routing paths by signing them, which incurs significant computation and communication overhead. In order to reduce the costs, a large amount of variants have been proposed. For example, So-BGP [51] ensured the correctness of announced routing paths by leveraging network topologies. IRV [50] validated the correctness of the announced routing paths by establishing an additional IRV server in each AS, which limited its deployment ability. RADAR [55] can be used to mitigate DDoS attacks introduced by vulnerabilities of current TCP/UDP. However, all these approaches did not address the security of routing data plane.

11.2. Source and path verification

IETF RFC 7039 and RFC 5210 provide a source address validation improvement (called SAVI) and architecture (called SAVA) for defending against source spoofing attacks [56,57]. RFC 6483 introduces validation of route origination using the resource certificate

PKI and route origin authorizations [58]. Origin and Path Trace (OPT) protocol [2,59] allows each router to verify delivered packets so as to verify the correctness of packet source and forwarding paths. It reduced storage overhead in routers, which prevents state exhaustion attack. Naous et al. [32] proposed a Path Verification Mechanism (PVM) to validate whether the packets correctly forwarded their forwarding paths. Cai and Wolf [15] performed source authentication and path validation by leveraging a set of orthogonal sequences instead of lightweight cryptographic operations. Wu et al. [60] introduced probabilistic packet marking to enable an efficient verification scheme, called PPV, for ensuring source authenticity and path compliance. Unfortunately, these mechanisms cannot localize the detected faults. Although Passport [7] and SNAPP [20] did not have such a problem, they were vulnerable to source spoofing or path deviation attacks. Qi et al. [61] presented a dynamic packet forwarding verification scheme called DynaPFV that can detect various attacks using SDN-related technologies for securing packet forwarding.

11.3. Fault localization

The proposed RFL can be classified as an active monitoring technique for the fault localization according to the recent surveys [62,63]. The prior schemes [64–67] are trying to select one or more probing stations for diagnosing the fault(s) in networks. They mainly rely on the selected stations that is assumed as trustworthy entities while performing fault localization. In fact, the current Internet, including network devices and end-hosts, are all unreliable. In this case, the selected localization stations are easily becoming an attack target, where the compromised stations can result in an incorrect localization or a false warning. Faultprints [13] was the first secure inter-domain fault localization scheme. It could localize the misbehaved links that drop, delay, modify packets at a high speed. ShortMAC [16] leveraged probabilistic packet authentication to locate the illegal network links, which achieved low detection delay and incurred small overhead. DynaFL [18] proposed the secure neighborhood-based fault localization (FL) protocol to cope with dynamic traffic patterns and routing path with a small router state. TrueNet [68] leveraged trusted computing technology to build a trusted network-layer architecture, and implemented a small TCB to address secure FL with small router state. However, these schemes might fail to localize faults if the generated acknowledgment packets are maliciously dropped by colluding entities. Our proposed protocol well addresses this issue by setting the timer on each entity, where the entity would send its sampling information towards S once the timer expires.

12. Conclusion

In this paper, we propose a robust data-plane fault localization protocol called RFL, which achieves high accuracy of localizing misbehaved entities and tolerates unreliable communication channels. RFL samples packets and embeds cryptographic markings in the packets so as to verify packet source and forwarding paths, and achieve fault localization, which ensures accuracy and robustness of the protocol even in the presence of interference from adversaries. In particular, RFL leverages a robust symmetric key sharing scheme to ensure that all entities can synchronize their verification keys to perform localization. We use real experiments with the RFL prototype to demonstrate the performance of RFL. The results show RFL achieves around 99.5% localization accuracy, and incurs small communication overhead, e.g., more than 90% throughput and 85% goodput compared to the baseline.

Declaration of interests

None.

Acknowledgements

The work in this paper was supported by National Key R&D Program of China under Grant 2018YFB0803405, the National Natural Foundation of China under Grants 61572278, 61772236, and 61602039, China National Funds for Distinguished Young Scientists under Grant 61825204, the Beijing Natural Science Foundation under Grant 4192050, and the Huawei Technologies Entrustment Project under Grant HF2019015003.

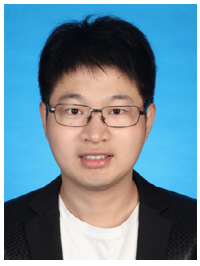
Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2019.04.023.

References

[1] H. Zeng, P. Kazemian, G. Varghese, N. McKeown, Automatic test packet generation, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, ACM, 2012, pp. 241–252.
[2] T.H.-J. Kim, C. Basescu, L. Jia, S.B. Lee, Y.-C. Hu, A. Perrig, Lightweight source authentication and path validation, in: Proceedings of ACM SIGCOMM, 2014.
[3] Z. Liu, H. Jin, Y.C. Hu, et al., MiddlePolice: Toward enforcing destination-defined policies in the middle of the Internet, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 1268–1279.
[4] Z. Liu, H. Jin, Y.C. Hu, et al., Practical proactive DDoS-attack mitigation via endpoint-driven in-network traffic control, IEEE/ACM Trans. Networking 26 (4) (2018) 1948–1961.
[5] L. Andersson, H. van Helvoort, R. Bonica, et al., Guidelines for the Use of the "OAM" Acronym in the IETF, IETF, 2011.
[6] Cisco, ITU-T Y.1731 Performance Monitoring in a Service Provider Network, 2011 (https://www.cisco.com/c/en/us/td/docs/ios/cether/configuration/guide/ce_y1731-perfmon.html#wp1062502).
[7] X. Liu, A. Li, X. Yang, D. Wetherall, Passport: secure and adoptable source authentication, NSDI, 2008.
[8] A. Perrig, R. Canetti, D. Song, J.D. Tygar, Efficient and secure source authentication for multicast, NDSS, 2001.
[9] B. Parno, A. Perrig, D. Andersen, Snapp: stateless network-authenticated path pinning, in: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ACM, 2008, pp. 168–178.
[10] M. Zhao, S.W. Smith, D.M. Nicol, Aggregated path authentication for efficient BGP security, ACM CCS, 2005.
[11] S. Kent, C. Lynn, K. Seo, Secure border gateway protocol (S-BGP), IEEE J. Sel. Areas Commun. 18 (4) (2000) 582–592.
[12] R.E. Miles, C.C. Snow, Causes of failure in network organizations, Calif. Manage. Rev. 34 (4) (1992) 53–72.
[13] C. Basescu, Y.-H. Lin, H. Zhang, A. Perrig, High-speed inter-domain fault localization, in: Security and Privacy (SP), 2016 IEEE Symposium on, IEEE, 2016, pp. 859–877.
[14] A. Schrank, J. Whitford, The anatomy of network failure, Sociol. Theory 29 (3) (2011) 151–177.
[15] H. Cai, T. Wolf, Source authentication and path validation with orthogonal network capabilities, in: Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on, IEEE, 2015, pp. 111–112.
[16] X. Zhang, Z. Zhou, H.-C. Hsiao, T.H.-J. Kim, A. Perrig, P. Tague, Shortmac: efficient data-plane fault localization., NDSS, 2012.
[17] P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, Y. Zhang, Mind the gap: monitoring the control-data plane consistency in software defined networks, in: Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies, ACM, 2016, pp. 19–33.
[18] X. Zhang, C. Lan, A. Perrig, Secure and scalable fault localization under dynamic traffic patterns, in: Security and Privacy (SP), 2012 IEEE Symposium on, IEEE, 2012, pp. 317–331.
[19] B. Wu, K. Xu, Q. Li, F. Yang, Robust and lightweight fault localization, in: IEEE International Performance Computing and Communications Conference (IPCCC), 2017.
[20] B. Barak, S. Goldberg, D. Xiao, Protocols and lower bounds for failure localization in the internet, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2008, pp. 341–360.
[21] Y.-C. Hu, A. Perrig, M. Sirbu, Spv: Secure path vector routing for securing bgp, in: ACM SIGCOMM Computer Communication Review, vol. 34, ACM, 2004, pp. 179–192.
[22] S.L. Murphy, M.R. Badger, Digital signature protection of the ospf routing protocol, in: Network and Distributed System Security, 1996., Proceedings of the Symposium on, IEEE, 1996, pp. 93–102.

[23] Wikipedia, Source Routing, 2017 (https://en.wikipedia.org/wiki/Source_routing).
[24] R. Moskowitz, P. Nikander, Host Identity Protocol (HIP) Architecture, 2006, doi:10.17487/RFC4423. (RFC 4423)
[25] R. Moskowitz, P. Jokela, T. Henderson, P. Nikander, Host Identity Protocol, 2008, doi:10.17487/RFC5201. (RFC 5201).
[26] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. Hyun Kim, S. Shenker, I. Stoica, A data-oriented (and beyond) network architecture, ACM SIGCOMM Comput. Commun. Rev. 37 (2007) 181–192, doi:10.1145/1282427.1282402.
[27] Q. Li, X. Zhang, Q. Zheng, R. Sandhu, X. Fu, Live: lightweight integrity verification and content access control for named data networking, IEEE Trans. Inf. Forensics Secur. 10 (2) (2015) 308–320.
[28] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun ACM 13 (7) (1970) 422–426.
[29] W. John, M. Dusi, K.C. Claffy, Estimating routing symmetry on single links by passive flow measurements, in: Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, ACM, 2010, pp. 473–478.
[30] B. Huffaker, M. Fomenkov, D.J. Plummer, D. Moore, K. Claffy, Distance metrics in the internet, in: Proc. of IEEE International Telecommunications Symposium (ITS), 2002.
[31] CAIDA, Passive Monitor: Equinix-Chicago, (http://www.caida.org/data/monitors/passive-equinix-chicago.xml).
[32] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, A. Seehra, Verifying and enforcing network paths with icing, in: Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies, ACM, 2011, p. 30.
[33] CAIDA, Round-Trip Time Internet Measurements, (http://www.caida.org/research/performance/rtt/walrus0202/).
[34] Wikipedia, Omnet++, 2018. (https://omnetpp.org).
[35] E. Kohler, R. Morris, B. Chen, J. Jannotti, M.F. Kaashoek, The click modular router, ACM Trans. Comput. Syst. 18 (3) (2000) 263–297.
[36] S. Peak, Silver Peak Application Performance Torture Test, 2018 (https://www.silver-peak.com/sites/default/files/infoctr/packet-loss-torture-test-041417.pdf).
[37] P. Tools, Whats Normal for Latency and Packet Loss?, 2018 (http://www.pingman.com/kb/42).
[38] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, K. Gibbs, Iperf: The TCP/UDP Bandwidth Measurement Tool, 2005. http://dast.nlanr.net/Projects.
[39] W. de Vries, J.J. Santanna, A. Sperotto, A. Pras, How asymmetric is the internet? in: IFIP International Conference on Autonomous Infrastructure, Management and Security, Springer, 2015, pp. 113–125.
[40] S. Wassermann, P. Casas, B. Donnet, G. Leduc, M. Mellia, On the analysis of internet paths with disnetperf, a distributed paths performance analyzer, in: Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on, IEEE, 2016, pp. 72–79.
[41] Í. Cunha, R. Teixeira, C. Diot, Measuring and characterizing end-to-end route dynamics in the presence of load balancing., in: PAM, vol. 11, Springer, 2011, pp. 235–244.
[42] M.S. Kang, S.B. Lee, V.D. Gligor, The crossfire attack, in: Security and Privacy (SP), 2013 IEEE Symposium on, IEEE, 2013, pp. 127–141.
[43] Paessler, Packet Loss Test Software, 2018 (https://www.paessler.com/packet_loss_test).
[44] VoIPspear, Cut to the Heart of Your Voip Problems, 2018 (https://www.voipspear.com).
[45] AppNeta, Completed End-to-End Network Performance Monitoring Software, 2018 (https://www.appneta.com).
[46] K. Xu, X. Wang, W. Wei, H. Song, B. Mao, Toward software defined smart home, IEEE Commun. Mag. 54 (5) (2016) 116–122.
[47] M.X. Cheng, W.B. Wu, Data analytics for fault localization in complex networks, IEEE Internet Things J. 3 (5) (2016) 701–708, doi:10.1109/JIOT.2015.2503270.
[48] G. Reali, M. Femminella, L. Monacelli, Probabilistic codebook-based fault localization in data networks, IEEE Trans. Netw. Serv. Manage. PP (2018) 1, doi:10.1109/TNSM.2018.2795083.
[49] M. Lepinski, S. Kent, An Infrastructure to Support Secure Internet Routing, IETF, 2012.
[50] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P.D. McDaniel, A.D. Rubin, Working around BGP: an incremental approach to improving security and accuracy in interdomain routing., in: NDSS, vol. 23, 2003, p. 156.
[51] J. Ng, et al., Extensions to BGP to Support Secure Origin BGP (soBGP), Technical Report, Internet Draft, Apr, 2004.
[52] C. Candolin, J. Lundberg, H. Kari, Packet level authentication in military networks, in: Proceedings of the 6th Australian Information Warfare & IT Security Conference, 2005.
[53] G. Hu, K. Xu, J. Wu, Y. Cui, F. Shi, A general framework of source address validation and traceback for ipv4/ipv6 transition scenarios, IEEE Netw. 27 (6) (2013) 66–73.
[54] R. Lychev, S. Goldberg, M. Schapira, BGP security in partial deployment: is the juice worth the squeeze? in: ACM SIGCOMM Computer Communication Review, vol. 43, ACM, 2013, pp. 171–182.
[55] J. Zheng, Q. Li, G. Gu, J. Cao, D.K. Yau, J. Wu, Realtime ddos defense using cots sdn switches via adaptive correlation analysis, IEEE Trans. Inf. Forensics Secur. 13 (7) (2018) 1838–1853.
[56] J. Wu, J. Bi, X. Li, et al, Source Address Validation Improvement (SAVI) Framework, IETF, 2013.
[57] J. Wu, J. Bi, X. Li, et al, A Source Address Validation Architecture (SAVA) Testbed and Deployment Experience, IETF, 2008.
[58] H. G, M. G, Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs), IETF, 2012.
[59] F. Zhang, L. Jia, C. Basescu, T.H.-J. Kim, Y.-C. Hu, A. Perrig, Mechanized network origin and path authenticity proofs, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 346–357.
[60] B. Wu, K. Xu, Q. Li, Z. Liu, Y.-C. Hu, M.J. Reed, M. Shen, F. Yang, Enabling efficient source and path verification via probabilistic packet marking, IEEE/ACM International Symposium on Quality of Service (IWQoS), 2018.
[61] Q. Li, X. Zou, Q. Huang, J. Zheng, P.P. Lee, Dynamic packet forwarding verification in SDN, IEEE Trans. Depend. Secure Comput. (2018).
[62] M. Sol-Sim, V. Munts-Mulero, A. Rana, G. Estrada, Survey on Models and Techniques for Root-Cause Analysis, 2017.
[63] A. Dusia, A. S. Sethi, Recent advances in fault localization in computer networks, IEEE Commun. Surv. Tut. 18 (2016) 3030–3051, doi:10.1109/COMST.2016.2570599.
[64] M. Brodie, I. Rish, S. Ma, Optimizing Probe Selection for Fault Localization, 2001, pp. 88–98, doi:10.3990/2.9.
[65] M. Natu, A. S. Sethi, Probabilistic Fault Diagnosis Using Adaptive Probing, vol. 4785, 2007, pp. 38–49, doi:10.1007/978-3-540-75694-1_4.
[66] A. X. Zheng, I. Rish, A. Beygelzimer, Efficient Test Selection in Active Diagnosis via Entropy Approximation, 2012.
[67] Y. Tang, E.S. Al-shaer, R. Boutaba, Active integrated fault localization in communication networks, in: IEEE/IFIP Integrated Management, 2005, pp. 543–556.
[68] X. Zhang, Z. Zhou, G. Hasker, A. Perrig, V. Gligor, Network fault localization with small TCB, in: 19th IEEE International Conference on Network Protocols (ICNP), IEEE, 2011, pp. 143–154.



Bo Wu received his bachelor degree from the school of software of Shandong University, China in 2014. He is working toward his Ph.D. degree supervised by Prof. Ke Xu in the Department of Computer Science & Technology at Tsinghua University. His research interests include network architecture, network security, next generation Internet and Blockchain.

Ke Xu received his Ph.D. from the Department of Computer Science & Technology of Tsinghua University, Beijing, China, where he serves as a full professor. He has published more than 100 technical papers and holds 20 patents in the research areas of next-generation Internet, P2P systems, Internet of Things (IoT), network virtualization and optimization, and network security. He is a member of ACM and has guest-edited several special issues in IEEE and Springer Journals.

Qi Li received his B.Sc. and Ph.D. degrees from the Department of Computer Science & Technology of Tsinghua University, Beijing, China, in 2003 and 2012, respectively. His research interests include network architecture, protocol design, and system and network security.

Bingyang Liu received a B.S. degree in computer software from Tsinghua University, China. He was a joint Ph.D. student in the Department of Computer Science, Duke University. He received a Ph.D. degree in computer science from Tsinghua University, China. He was ever a postdoctoral researcher at the Institute for Network Sciences and Cyberspace, Tsinghua University. He is now a researcher in Huawei Technologies. His research fields include Internet architecture, network security, DDoS defense, and software-defined networking (SDN).

Shoushou Ren received his B.S. degree in Communication Engineering from Beijing University of Posts and Telecommunication, Beijing, China, in 2011, and the Ph.D. degree in computer science from the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, in 2016. He is now a researcher in Huawei Technologies. His research interest includes network security, mobile network caching and heterogeneous network.

Fan Yang received her masters degree from the Department of Computer Science & Technology of Tsinghua University, Beijing, China, in 2017. Her research interests include network architecture and network security.

Meng Shen received his B.Eng. degree in computer science from Shandong University, Jinan, China, in 2009, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2014. He is currently an Assistant Professor with the Beijing Institute of Technology, Beijing, China. His research interests include privacy protection of cloud-based services, network virtualization, and traffic engineering. He was a recipient of the Best Paper Runner-Up Award at IEEE IPCCC 2014.

Kui Ren is a professor of the Institute of Cyber Security Research and School of Computer Science and Engineering at Zhejiang University, Hangzhou, China. He received his Ph.D. degree from Worcester Polytechnic Institute and both BE and ME degrees from Zhejiang University. Kui's research interests include Cloud Security, Wireless Security, and Smartphone-enabled Crowdsourcing Systems. His research has been supported by NSF, DoE, AFRL, MSR, and Amazon. He is a recipient of NSF CAREER Award in 2011 and Sigma Xi Research Excellence Award in 2012. Kui has published 135 peer-review journal and conference papers. Kui received several Best Paper Awards including IEEE ICNP 2011. Kui currently serves as an associate editor for IEEE Transactions on Information Forensics and Security, IEEE Wireless Communications, IEEE Internet of Things Journal, IEEE Transactions on Smart Grid, IEEE Communications Surveys and Tutorials, Elsevier Pervasive and Mobile Computing, and Oxford The Computer Journal. Kui is a Fellow of IEEE, a member of ACM, a Distinguished Lecturer of IEEE Vehicular Technology Society, and a past board member of Internet Privacy Task Force, State of Illinois.