XShot: Light-weight Link Failure Localization using Crossed Probing Cycles in SDN

Hongyun Gao
Tianjin Key Lab. of Advanced Networking (TANKLab), College of Intelligence and Computing (CIC), Tianjin University
gaohongyun@tju.edu.cn

Laiping Zhao*
TANKLab, CIC, Tianjin University
laiping@tju.edu.cn

Huanbin Wang
TANKLab, CIC, Tianjin University
whb990422@tju.edu.cn

Zhao Tian
TANKLab, CIC, Tianjin University
tianzhao@tju.edu.cn

Lihai Nie
TANKLab, CIC, Tianjin University
nlh3392@tju.edu.cn

Keqiu Li
TANKLab, CIC, Tianjin University
keqiu@tju.edu.cn

ABSTRACT

Accurate and quick failure localization is critical for automatic network troubleshooting. While it is particularly difficult to solve the problem in the traditional network due to the uncertain routing, Software Defined Networking (SDN) enables the deterministic routing for packet transmission through the traffic engineering algorithm in the centralized controller.

Based on this, we propose to localize link failures quickly through the *cross connectivity verification* of multiple probing paths. We present `XShot`, a light-weight link failure localization system in SDN. It assigns a unique binary code for each link, and can pinpoint the exact faulty link within just one-round shot of probing. `XShot` is very light-weight, and its probing paths are derived by solving the optimization model of minimizing the probing packet cost and forwarding rule cost. Our model analysis gives the lower bound of the number of probing paths in any SDN topology. `XShot` is also capable of localizing *partial failures* that only cause long latency on packet transmission, by learning from the measured latency of crossed probing paths. We evaluate `XShot` using 63 real internet topologies, and find that while it can achieve accurate link failure localization within just one-round probing, it even uses 9.63% less probing packets and 6.15% less forwarding rules in 79.37% of the real topologies than the multi-round probing solution.

CCS CONCEPTS

• **Networks → Network measurement**; **Network monitoring**; *Network performance modeling*; *Network reliability*.

KEYWORDS

Network Troubleshooting, Software Defined Networking, Active Probing, Cross Verification

ACM Reference Format:
Hongyun Gao, Laiping Zhao, Huanbin Wang, Zhao Tian, Lihai Nie, and Keqiu Li. 2020. *XShot*: Light-weight Link Failure Localization using Crossed Probing Cycles in SDN. In *49th International Conference on Parallel Processing - ICPP (ICPP '20), August 17–20, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3404397.3404439

1 INTRODUCTION

To enable widespread support for massive higher-level services, modern communication networks continue growing rapidly in both scale and complexity, typically housing up to tens of thousands of network devices along with even more links [8, 15]. Inevitably, device or link failures have also occurred frequently for a variety of reasons, causing severe service outages or performance degradations [13, 26, 29]. Therefore, an essential issue faced by network operators is how to promptly detect and localize these failures for ensuring the reliable operation of networks.

Prior work on failure localization mainly relies on network monitoring. They can be classified into two approaches: *passive monitoring* [14, 16, 21, 26] and *active probing* [6, 13, 15, 18, 25, 28, 29, 33, 35, 42]. The *passive monitoring* approach leverages readily available metrics from production traffic to detect abnormalities. Since the direct alarm signals are often missed under failures [11], the *passive* approach may generate many false alarms and turn failure localization into a long-time lagging process [15]. The *active probing* approach involves injecting one or more probing packets (e.g., ping service) into the network to monitor the status of switches/links that the packets pass by. However, it can only provide a rough estimate of the failure position, due to the uncertain paths used for data transfer in traditional networks [13, 15].

Software Defined Networking (SDN) [10, 22] opens up an opportunity to overcome the limitation of *active probing*. It decouples the control plane from the data plane, and routes packets on predefined and deterministic paths. The deterministic routing paths make it possible to detect and localize the exact position of failures efficiently [35]. In particular, the *connectivity verification* method [4, 18, 27] can measure the up-or-down state of a path precisely

*Corresponding author.

according to the receiving state of probing packets. Based on *connectivity verification*, Kozat *et al.* [18] have proposed to localize failures in a multi-round binary-search way by organizing network links into a ring. Moreover, richer link metrics (e.g., link latency, packet drop rate) can be further derived through end-to-end performance measurements [3, 20, 28, 35].

Although effective for localizing link failures, these approaches either cannot distinguish *fail-stop* and *partial failures* (i.e., a faulty link may be dropping or delaying packets probabilistically [15, 26, 29], for example, random packet loss, link congestion) just by packet drops, or incur high cost including *additional hardware monitors* [3, 28, 35], *many probing packets and forwarding rules* [4, 20, 27, 28, 35] and *long probing time* [18]. Specifically, the multicast-based approaches [4, 20, 27, 35] typically requires $O(\|E\|)$ probing packets and installs $O(\|V\|)$ (or $O(\|E\|)$) forwarding rules on switches, where $\|E\|$ and $\|V\|$ denote the number of links and switches in a network, respectively. While the probing packets impose a large communication load on the network, the forwarding rules also take the expensive resources of ternary content-addressable memory (TCAM).

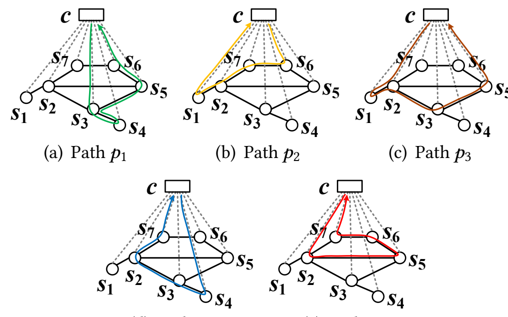
In this paper, we aim to pinpoint the exact faulty links in SDN in a more light-weight and quick manner. We explore to reduce the number of probing packets and forwarding rules significantly, and do not rely on the additional hardware monitors for saving cost. As partial failures caused during maintenance are fairly more common than complete failures [15], and packet loss or high latency induced by these failures affects users [29], the localization approach should be able to detect and localize all of them. There are two major challenges towards this problem:

(1) **How to formulate the probing cost in terms of packets and rules?** Probing packets and forwarding rules increase over the number of probing paths, and the probing paths need to cover all of the links. Hence, a large-scale network may occupy a lot of network resources for setting probing paths. To minimize the resource consumption, the probing paths should be crafted carefully.

(2) **How to identify partial failures from noisy measurements?** Given the minimum probing paths, the measured metrics (e.g., latency) are often noisy. It is difficult to recognize partial failures (especially those with high latency) from noises due to their similar uncertainty. Hence, how to improve the precision of failure localization under partial failures is also challengeable.

We present *XShot*, a quick and light-weight failure localization system in SDN. Inspired by the monitoring cycles in all-optical networks [36, 37, 39, 40], *XShot* employs a *cross verification* method to localize the faulty link within just one-round shot of crossed probing paths. That is, each link failure corresponds to one and only one binary code, which is defined based on the probing results of crossed paths. However, different with those in all-optical networks, *XShot* does not rely on any special hardware monitors and supervisory wavelengths. Moreover, while a monitoring cycle in the all-optical layer is not allowed to pass through a link twice due to the interference of light waves, we observe that they cannot localize failures in *one-cut* and *two-cut* networks. We solve this problem by re-formulating the constraint in probing path design.

XShot is designed to reduce probing packet cost and forwarding rule cost through minimizing the number and length of probing paths by solving an integer programming model. For identifying



(a) Path p_1 (b) Path p_2 (c) Path p_3

(d) Path p_4 (e) Path p_5

Link	Binary Code					Link	Binary Code				
	p_1	p_2	p_3	p_4	p_5		p_1	p_2	p_3	p_4	p_5
(s_1, s_2)	0	1	1	0	0	(c, s_1)	0	1	0	0	0
(s_2, s_3)	0	0	1	1	0	(c, s_2)	0	0	1	0	1
(s_2, s_5)	0	0	0	0	1	(c, s_3)	1	0	0	0	0
(s_2, s_7)	0	1	0	1	0	(c, s_4)	0	0	0	1	0
(s_3, s_4)	1	0	0	1	0	(c, s_5)	0	0	1	0	0
(s_3, s_5)	1	0	1	0	0	(c, s_6)	1	1	0	0	0
(s_5, s_6)	1	0	0	0	1	(c, s_7)	0	0	0	1	1
(s_6, s_7)	0	1	0	0	1						

(f) Cross verification code for each link failure

Figure 1: The probing paths for the network in Figure 3.

partial failures that only cause long latency, *XShot* further employs an unsupervised machine learning algorithm to differentiate them from noisy latency measurements. It scores the measured values of each path in real time, and the one with a high score is considered abnormal. In summary, we make the following contributions:

- A cross probing-based link failure localization method in SDN. It is light-weight and supports the localization of both fail-stop and partial failures within just one round probing.
- An integer linear programming (*ILP*) model for minimizing the number and length of probing paths, thereby reducing the packet and rule costs.
- A machine learning algorithm that learns to identify partial failures from noisy measurements.
- A detailed comparative evaluation of *XShot*, showing significant improvement in the localization precision and efficiency.

2 MOTIVATION

In this section, we show how *cross verification* can localize the faulty links efficiently.

2.1 Cross Verification

Figure 1 shows an example of an SDN, which consists of seven switches $\{s_1, s_2, ..., s_7\}$ and a centralized controller c. There are 15 links in the network, including 8 links in the data plane and 7 links between c and switches. We suppose there exists one link failure, and it may occur at any link. To pinpoint the exact faulty link, a straightforward method is probing each link directly, requiring at least 15 probing messages in this example. If we allow each link to be traversed by multiple messages, the combined measurement of these messages can possibly reduce the probing cost.

Since we have 15 links in the network, at least $\lceil \log_2(15+1) \rceil = 4$ bits of binary codes are required to uniquely identify each link (we use the "+1" operation in the formula because a *full*-0 code cannot

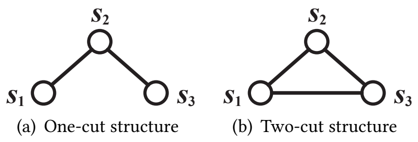


Figure 2: One-cut and two-cut network structure.

identify failures). As shown in Figure 1(a) - Figure 1(e), we design five routing paths whose start and end point are both the controller c, and install their forwarding rules using OpenFlow [23]:

- Path p_1: $c \to s_3 \to s_4 \to s_3 \to s_5 \to s_6 \to c$;
- Path p_2: $c \to s_1 \to s_2 \to s_7 \to s_6 \to c$;
- Path p_3: $c \to s_2 \to s_1 \to s_2 \to s_3 \to s_5 \to c$;
- Path p_4: $c \to s_7 \to s_2 \to s_3 \to s_4 \to c$;
- Path p_5: $c \to s_2 \to s_5 \to s_6 \to s_7 \to c$.

We send specially designed packets along these paths, and record if each of them arrives at the end point successfully: if yes, then all links along the path are normal, and we record "0", otherwise "1". Figure 1(f) shows the cross verification code for each link failure. For example, a failure at link (s_1, s_2) can be identified by two failed transmissions on path p_2 and p_3 while all other transmissions are normal.

2.2 Challenges in SDN

Cross verification has been utilized to localize failures in the optical layer [36, 37]. They set up a pair of optical transceivers and a hardware monitor in the network for probing. While a probing cycle in [36] limits that a node can only be traversed at most once by each cycle, Wu *et al.* [37] extend to support multi-traverse through a node. However, they both do not allow to traverse a link more than once in the same probing cycle due to the limit of light wave transmission. This leads to "*failed localization*" problem in network with one-cut and two-cut structures. As shown in Figure 2, *one-cut structure* refers to a subset of the network where the remove of a link (named one-cut link) will partition the network into two independent parts. Similarly, *two-cut structure* refers to a subset of the network, where the remove of two links (named two-cut link) will partition the network into two independent parts. If a link can only be traversed at most once in the same probing cycle, then the failures at links (s_1, s_2) and (s_2, s_3) in Figure 2(a) will not be distinguished from each other. Likewise, there is also the same problem in two-cut structure.

In SDN, probing cycles are allowed to traverse a link multiple times through the elaborately designed forwarding rules. Hence, how to derive probing cycles in SDN is a new challenge. Moreover, while the *binary code* of cross verification can only help to localize *fail-stop* failures in the optical layer, how to localize the partial failures in SDN is also a big challenge.

3 XSHOT DESIGN

In this section, we present the design of *XShot* based on the idea of *cross verification*.

3.1 System Overview

In SDN, the routing path for each packet is deterministic and is derived by the traffic engineering algorithm in the centralized controller. *XShot* designs a set of crossed probing paths, and localizes

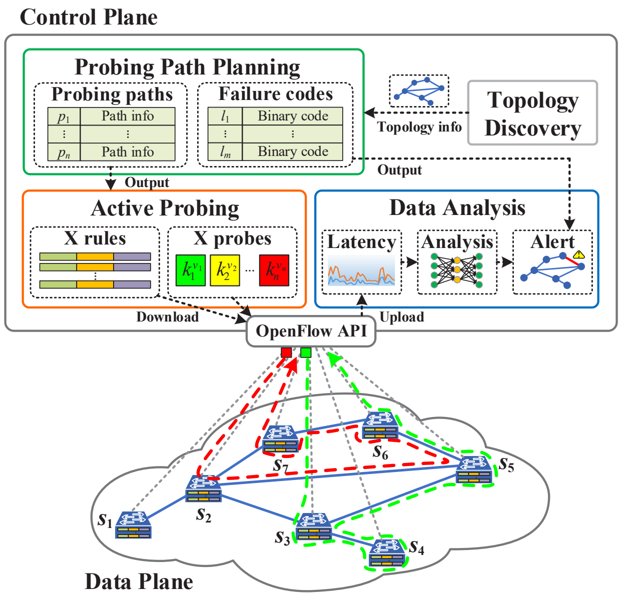


Figure 3: *XShot* architecture. It consists of three components: *probing path planning* designs a set of probing paths that start and end at the controller, *active probing* periodically sends probing packets along the paths, and *data analysis* collects the latency for transmitting probing packets and derives the failure positions.

network failures using a *cross verification* method, that is, each link failure can be uniquely identified by the connectivity verification results of the paths. For partial failures that only cause long delay (e.g., network congestion) rather than disconnection, *XShot* recognizes them through a measurement and analysis of end-to-end delays of the crossed paths. Figure 3 shows the overall design of *XShot*. There is a single centralized controller in the SDN network, and the controller is connected to switches through one or multiple dedicated links. *XShot* is deployed totally in the controller, and consists of three components: *probing path planning*, *active probing* and *data analysis*.

Given the network topology, *probing path planning* generates a list of probing paths which may intersect at links. Since end-hosts may belong to private individuals, it is not always possible to deploy monitoring agents on them. Moreover, running the monitoring agent on dedicated hardware is rather costly. Hence, *XShot* limits the start and end points of all probing paths at the controller itself, generating *probing cycles*. For reducing the probing cost, *XShot* minimizes the number of packets and forwarding rules used by the probing paths by solving a mathematical model of the *cost-minimization problem*.

Active probing installs the forwarding rules on switches according to the probing paths, and measures the end-to-end latency by sending packets along them. Since each probing cycle shares the same start and end point (i.e., the controller), the end-to-end latency can be derived directly by the interval between the time of sending and receiving the packet. If the probing packet does not arrive at the destination successfully, it implies the occurrence of *fail-stop* failures on the path. If the delay is just very high, it implies that congestion may occur in the network. Probing packets are created

Table 1: Notation list

Notation	Definition
$G(V, E)$	An undirected graph of SDN.
V	The set of switches in the SDN.
E	The set of links in the SDN, and $E = E_c \cup E_d$.
E_c	The set of links between controller and switches.
E_d	The set of forwarding links in data plane.
c	The SDN controller.
p_i	The ith probing path.
n	The number of probing paths.
c_{pkt}	The probing packet cost.
c_{rule}	The forwarding rule cost.

periodically, and its frequency can be decided depending on the characteristic of faults the administrator aims to localize.

Data analysis component collects the measured latency of probing packets, detects the path status using an unsupervised learning algorithm, and pinpoints the exact faulty link according to the unique binary code. As the latency data may be blended with noises due to the contention on bandwidth or controller resources, we further explore an accelerated detection method to improve the localization precision for partial failures that cause high latency.

3.2 Probing Path Planning

Denote by $G = (V, E)$ an SDN network, where V and E represent the set of switches and links in the network, respectively, and $E = E_c \cup E_d$, where E_c is the set of links between the controller and the data plane, and E_d denotes the set of links in the data plane. We seek a probing solution for minimizing the probing cost. Since we do not rely on specialized monitoring hardware, the probing cost mainly refers to the number of probing packets and forwarding rules. We define four variables for formulating the problem:

- $e^i_{\overrightarrow{xy}}$: a nonnegative integer variable, indicating the times of link (x, y) crossed by path p_i from x to y. It is set to "0" if the directed link $x \longrightarrow y$ is not on path p_i.
- $q^i_{\overrightarrow{xy}}$: a nonnegative integer variable for ensuring that the derived path is feasible. It takes a positive integer if the directed link $x \longrightarrow y$ is on path p_i, and "0" otherwise.
- l^i_{xy}: a binary variable, which has the value "1" if link (x, y) is on path p_i, and "0" otherwise.
- $h^i_{xy,ab}$: a binary variable, and $h^i_{xy,ab} = l^i_{xy} || l^i_{ab}$ ("OR" operation).

Each probing cycle requires a probing packet, and they all share the same start and end point (i.e., the controller). Denote by c the centralized controller, then the probing packet cost (c_{pkt}) can be defined as below,

$$c_{pkt} = \sum_i \sum_{(c,y)\in E_c} e^i_{\overrightarrow{cy}}. \tag{1}$$

For the cost of forwarding rules, as each directed transmission requires a forwarding rule, we define c_{rule} as below,

$$c_{rule} = \sum_i \sum_{(x,y)\in E_d} (e^i_{\overrightarrow{xy}} + e^i_{\overrightarrow{yx}}) + \sum_i \sum_{(x,c)\in E_c} e^i_{\overrightarrow{xc}}. \tag{2}$$

The problem can be formulated as an integer linear programming (ILP) problem as below.

Objective:

$$\min \qquad \omega \times c_{pkt} + c_{rule} \tag{3}$$

where ω is a weight used to balance the cost c_{pkt} and c_{rule}. In general, $\omega \geq 1$ because the probing packets must be periodically sent to track the network status all the time whereas the forwarding rules are statically installed.

Subject to:

$$e^i_{\overrightarrow{xy}} \leq 1, \quad \forall i, \forall (x,y) \in E \tag{4}$$

$$e^i_{\overrightarrow{yx}} \leq 1, \quad \forall i, \forall (x,y) \in E \tag{5}$$

$$\sum_{(c,y)\in E_c} e^i_{\overrightarrow{cy}} \leq 1, \quad \forall i \tag{6}$$

$$\sum_{(x,c)\in E_c} e^i_{\overrightarrow{xc}} \leq 1, \quad \forall i \tag{7}$$

$$\sum_{(x,y)\in E} (e^i_{\overrightarrow{xy}} - e^i_{\overrightarrow{yx}}) = 0, \quad \forall i, \forall x \in V \tag{8}$$

$$e^i_{\overrightarrow{xy}} \leq q^i_{\overrightarrow{xy}} \leq \varphi \times e^i_{\overrightarrow{xy}}, \quad \forall i, \forall (x,y) \in E \tag{9}$$

$$e^i_{\overrightarrow{yx}} \leq q^i_{\overrightarrow{yx}} \leq \varphi \times e^i_{\overrightarrow{yx}}, \quad \forall i, \forall (x,y) \in E \tag{10}$$

$$\sum_{(x,y)\in E} (q^i_{\overrightarrow{xy}} - q^i_{\overrightarrow{yx}}) \geq \frac{1}{\varphi} \sum_{(x,y)\in E} e^i_{\overrightarrow{xy}}, \quad \forall i, \forall x \in V \tag{11}$$

$$\frac{1}{2}(e^i_{\overrightarrow{xy}} + e^i_{\overrightarrow{yx}}) \leq l^i_{xy} \leq e^i_{\overrightarrow{xy}} + e^i_{\overrightarrow{yx}}, \quad \forall i, \forall (x,y) \in E \tag{12}$$

$$\sum_i l^i_{xy} \geq 1, \quad \forall (x,y) \in E \tag{13}$$

$$\frac{1}{2}(l^i_{xy} + l^i_{ab}) \leq h^i_{xy,ab} \leq l^i_{xy} + l^i_{ab}, \quad \forall i, \forall (x,y) \neq (a,b) \tag{14}$$

$$\sum_i (2h^i_{xy,ab} - l^i_{xy} - l^i_{ab}) \geq 1, \quad \forall (x,y) \neq (a,b) \tag{15}$$

$$l^i_{xy}, l^i_{ab}, h^i_{xy,ab} \in \{0,1\}, \quad \forall i, \forall (x,y) \neq (a,b) \tag{16}$$

$$e^i_{\overrightarrow{xy}}, e^i_{\overrightarrow{yx}}, q^i_{\overrightarrow{xy}}, q^i_{\overrightarrow{yx}} \in Z^{\geq}, \quad \forall i, \forall (x,y) \in E \tag{17}$$

$$i \in \{1, 2, \ldots, ||E||\}. \tag{18}$$

- Constraints (4) and (5) limit that every link is crossed by a path p_i in either direction at most once, so as to prevent the *endless transmission* of probing packets. Note that we allow each link to be traversed twice but in different directions, for localizing failures in *one-cut* and *two-cut* networks.
- Constraints (6) and (7) make sure that every path crosses the links in E_c in either direction at most once.
- Constraint (8) enforces that the egress of a switch x on p_i equals to its ingress.
- Constraints (9), (10) and (11) together make sure that the derived path is feasible. In particular, constraint (9), (10) and (17) limits the range for $q^i_{\overrightarrow{xy}}$, where φ is a constant and $\varphi \geq 2||E||$: If $e^i_{\overrightarrow{xy}} = 0$, then there must be $q^i_{\overrightarrow{xy}} = 0$. Constraints (11) ensures that the sums of $q^i_{\overrightarrow{xy}}$ of all out-going links on p_i from switch x is greater than those of in-coming links. While constraint (7) does not allow disconnected link

existing in the path p_i, constraint (11) further avoids the possibility of "fake cycle" whose start and end point are the same switch in the data plane.

- Constraint (12) enforces the correct relationship between $e^i_{\overrightarrow{xy}}$ and l^i_{xy}.
- Constraint (13) makes sure that each link is crossed by at least one path.
- Constraint (14) together with constraint (16) enforce the correct relationship between $h^i_{xy,ab}$ and l^i_{xy}, l^i_{ab}.
- Constraint (15) makes sure that each link has a unique failure code.
- Constraints (16), (17) and (18) refer to domain constraints, and $Z^{\geq}$ represents non-negative integers.

We solve the ILP model using IBM-CPLEX [7], and output the probing paths to the *active probing* component. Figure 1 shows the five probing paths we have derived for the network in Figure 3. Since the model only requires the network topology as input, it is activated whenever an event of network topology change occurs, which is rare in reality.

3.3 Bound Analysis

Theorem 1. *Given an SDN topology $G(V, E)$, which has $\|E\|$ links including $\|E_c\|$ control links and $\|E_d\|$ forwarding links, the lower bound of the number of probing paths (n) is,*

$$n \geq \max\{\lceil \log_2(\|E\| + 1) \rceil, \lceil \frac{2}{3}\|E_c\| \rceil\}. \tag{19}$$

Proof. To distinguish the failures of all links in E using binary code, we need at least $\lceil \log_2(\|E\|) \rceil$ bits. While the *full*-0 code cannot be used for identifying failures, we can derive the first lower bound:

$$n \geq \lceil \log_2(\|E\| + 1) \rceil. \tag{20}$$

Next, we prove the second part of the bound. Let m_i be the number of control links traversed by exactly i paths, then,

$$\sum_{i=1}^{n} m_i = \|E_c\| \tag{21}$$

$$m_i \leq C_n^i \tag{22}$$

where C_n^i denotes the number of possibilities for selecting i out of n. Let $T(m_i)$ be the traversed times of these m_i links. According to constraints (6) and (7), we know that the n paths can cross the control links in E_c $2n$ times. Thus,

$$\sum_{i=1}^{n} T(m_i) = 2n. \tag{23}$$

In addition, because a link can be traversed two times by the same path in the opposite directions, we have,

$$T(m_i) \geq im_i. \tag{24}$$

Then, we discuss the lower bound in two conditions:
(i) **When $\|E_c\| \geq C_n^1$ is established**, there must be an integer k in the range of $[2, n]$ such that $\sum_{i=1}^{k-1} C_n^i \leq \|E_c\| \leq \sum_{i=1}^{k} C_n^i$. Given the integer k and the above formulas of (21)-(24), we have,

$$\begin{aligned}
\sum_{i=1}^{n} T(m_i) &= \sum_{i=1}^{k-1} T(m_i) + \sum_{i=k}^{n} T(m_i) \\
&\geq \sum_{i=1}^{k-1} im_i + \sum_{i=k}^{n} im_i \\
&\geq \sum_{i=1}^{k-1} im_i + k\sum_{i=k}^{n} m_i \\
&= \sum_{i=1}^{k-1} im_i + k(\|E_c\| - \sum_{i=1}^{k-1} m_i) \\
&= k\|E_c\| - \sum_{i=1}^{k-1} (k-i)m_i \\
&\geq k\|E_c\| - \sum_{i=1}^{k-1} (k-i)C_n^i \\
&= \sum_{i=1}^{k-1} iC_n^i + k(\|E_c\| - \sum_{i=1}^{k-1} C_n^i).
\end{aligned} \tag{25}$$

Thus, we have,

$$2n \geq \sum_{i=1}^{k-1} iC_n^i + k(\|E_c\| - \sum_{i=1}^{k-1} C_n^i). \tag{26}$$

It is easy to derive that the lower bound is surely established if $n = 1$. Because we have $\sum_{i=1}^{k-1} C_n^i \leq \|E_c\|$, and when $n \geq 2$, $C_n^1 + 2C_n^2 = n^2 \geq 2n$ is established, we derive that k **must be** 2. Thus, we have,

$$2n \geq C_n^1 + 2 \times (\|E_c\| - C_n^1). \tag{27}$$

As n is an integer, we have,

$$n \geq \lceil \frac{2}{3}\|E_c\| \rceil. \tag{28}$$

(ii) **When $\|E_c\| \geq C_n^1$ is not established**, that is $n > \|E_c\|$, the lower bound of $n \geq \lceil \frac{2}{3}\|E_c\| \rceil$ is still established.

To sum up, no matter whether $\|E_c\| \geq C_n^1$ is established or not, we have $n \geq \lceil \frac{2}{3}\|E_c\| \rceil$. Combined with Formula (20), we have $n \geq \max\{\lceil \log_2(\|E\| + 1) \rceil, \lceil \frac{2}{3}\|E_c\| \rceil\}$. Proof end. □

3.4 Active Probing

Given the derived probing cycles, *active probing* installs their corresponding rules in the data plane using OpenFlow. Rules are first encapsulated into individual OpenFlow messages and then written into TCAM of the designated switches via OpenFlow API (e.g., *write()* in Floodlight [9]). Once installed, these rules remain static unless the network topology changes. For example, Table 2 shows the five forwarding rules required for the path p_1 in Figure 1: $c \rightarrow s_3 \rightarrow s_4 \rightarrow s_3 \rightarrow s_5 \rightarrow s_6 \rightarrow c$. In particular, $vlan_1$ is the Virtual Local Area Network (VLAN) tag [5] of path p_1, and $port_i^j$ denotes the port on switch s_i directed towards switch s_j.

Then, *active probing* starts the probing process for localizing failures, which consists of three steps:

Table 2: Forwarding rules for path p_1

Switch	**Forwarding Rule**	
	Match Fields	***Actions***
S_3	VLAN == $vlan_1$, inport == CONTR[a]	output = $port_3^4$
	VLAN == $vlan_1$, inport == $port_3^4$	output = $port_3^5$
S_4	VLAN == $vlan_1$, inport == $port_4^3$	output = INPORT
S_5	VLAN == $vlan_1$, inport == $port_5^3$	output = $port_5^6$
S_6	VLAN == $vlan_1$, inport == $port_6^5$	output = CONTR

[a]CONTR represents CONTROLLER.

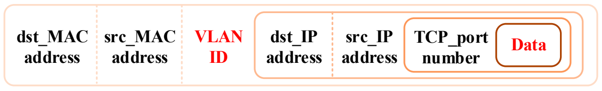


Figure 4: The format of probing packets.

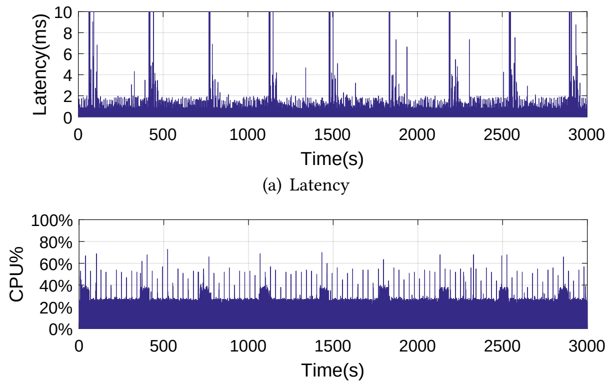
Step 1: Constructing data packets: Each probing cycle requires a single probing packet. We construct a TCP packet for each of them. Since all paths share the same start and end point (i.e., the controller), the source and destination IP address of the packet are the same. We utilize the *VLAN ID* [5] to distinguish the packets of different paths. Figure 4 shows the overall format of data packets. We record the sending time of the packet in the *data* field.
Step 2: Periodically sending packets: *Active probing* sends the constructed packets simultaneously and periodically following probing paths. Highly frequent probing could improve the localizing efficiency, but also increase the network burden. This is a trade-off. We suggest that, if we need to localize partial failures that have a short-duration, the sending interval could be in milliseconds; if we are required to localize the *fail-stop* failures, the sending interval could be in minutes.
Step 3: Measuring latency: We record the sending and receiving time of each packet, and derive the end-to-end latency of them. If a packet does not come back, we mark its probing cycle failed.

3.5 Data Analysis

When the probing packets return to the controller in forms of *packet-in* messages, the *data analysis* module is triggered to parse the contents of these incoming messages, and record the latency using: *latency = receiving time - sending time*. *XShot* then localizes the failure position using the measured latency.

Packet loss: *Packet loss* may be caused by *fail-stop* or *partial failures*. *XShot* infers the link failures in case of packet loss by observing whether the probing packets are received or not. Specifically, the *data analysis* module initialize the latency as 0. Upon a probing packet is received, the corresponding latency is updated to the actual measurement. Before the next probing, if the latency remains 0, this module determines packet drops occur on the cycle. According to the failure codes, the faulty link with packet drops can be localized directly.

High latency: It is difficult to identify the *partial failures* that only cause *high latency*, due to the noises in measured data. *XShot* chooses Donut [38], an unsupervised anomaly detection algorithm based on VAE, to infer partial failures. Since it does not rely on data label for training the detection model, it meets our demand well as the link failures data is far less than the normal data. Concretely, it gives a score to each of the measure data as the evaluation result, and



(a) Latency

(b) CPU usage

Figure 5: Time-varying fluctuations in terms of latency and CPU usage.

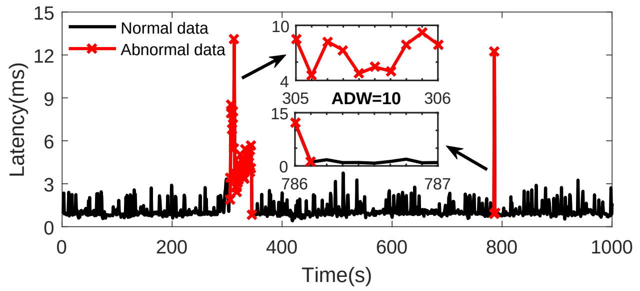


Figure 6: Data analysis using Donut with $ADW = 10$.

select an appropriate threshold according to the score distribution and the true labels. If the score of latency at a certain time exceeds the selected threshold, Donut identifies the latency is abnormal. Donut is highly sensitive for the spikes in latency and can capture them quickly and efficiently.

We measure the latency of probing packets when there are no failures in the network, and find that transient unexpected fluctuations exist in the measured data. As shown in Figure 5(a), the unexpected spikes seriously affects the detection accuracy of high-latency failures. Figure 5(b) shows the reason of this abnormal situation. We find that the CPU usage of the controller (i.e., Floodlight) has the same fluctuation frequency as the latency. Due to the sudden increase of CPU usage, the packet-processing speed of the controller slows down, resulting in an undesirable increase in measured latency.

To solve this problem, we introduce an *accelerated detection window* (ADW) into Donut (Figure 6): (i) When an anomaly is detected by Donut in the normal probing intervals, *XShot* drives the *active probing* component to send a certain number (i.e., the detection window) of additional probing packets in a higher frequency to verify the detection result before the next normal probing. (ii) If there are more detected anomalies in the window than a preset threshold, the detection result of Donut is true positive. (iii) Otherwise, it is false positive and removed. Through this method, the detection accuracy is greatly improved.

4 IMPLEMENTATION AND DISCUSSION

We have implemented *XShot* on Floodlight [9] controller using Java. It supports the OpenFlow protocol [23] (OpenFlow v.1.3) to connect the data plane and works with physical and/or virtual OpenFlow switches.

Unidirectional failures: Since the constraint (13) in ILP only ensures that each link is traversed by at least one path, it does not guarantee that both directions can be covered by probing paths. Hence, *XShot* can localize the failures that affect the data transmission in both directions, or part of those unidirectional failures that just in the probing cycle direction. For example, the fail-stop failures caused by catastrophic events like fiber cuts and complete hardware failures can all be localized precisely. For some partial failures, like network congestion in unidirectional transmission, *XShot* may miss them. A simple solution is activating a second round of probing, but in the opposite direction along the original probing cycle. In this way, all unidirectional failures will be covered by the probing process, at the cost of at most double of forwarding rules.

Switch failures: While our discussions so far focus on the link failures (including links themselves and switch port failures), *XShot* can be easily extended to localize failures in switches (excluding switch ports), by treating links and switches reversely. That is, suppose all links are normal, and pinpoint the faulty switch using the same *cross verification* method.

Multi-simultaneous failures: The current design of *XShot* can only localize a single failure at a time. If there exist multi-simultaneous failures in the network, *XShot* cannot distinguish between them. The network administrator can localize the failures using more network monitoring information or more monitoring paths. This will be part of our future work.

Extremely short-lived failures: *XShot* can detect the partial failures that last for a short period. However, it cannot detect failures whose lifetime is much shorter than the probing interval. While it is possible that *μbursts* (lasting under a millisecond or even a single RTT) exist in production networks, prior work [41] has developed a high-resolution measurement framework to poll switch statistics at a microsecond-level granularity. We do not consider this problem because such intensive monitoring behaviors are usually accompanied with substantial dedicated overhead. And moreover, if the bursts are too short-lived, they may be gone before network operators take actions to mitigate.

Taking into account these factors, we choose to monitor the SDN networks at a moderately coarser granularity than microseconds. More generally, *XShot* is more effective on detecting severe or sustained network events.

5 EVALUATION

In this section, we present the evaluation of *XShot*.

5.1 Setup

5.1.1 Experimental environment. We choose Floodlight [9] as the SDN controller, and employ Mininet [19] to create an OpenFlow-enabled network connected by virtual elements (e.g., Open vSwitch [34]). We collected 63 available network topologies from the Internet Topology Zoo [17], and set a centralized controller which is connected to only one randomly selected switch (i.e., $\|E_c\| = 1$). We derive the probing paths for all of them. The probing interval is 1 second. When a network anomaly is detected, we increase the probing frequency to once every 0.1 second.

5.1.2 Compared approaches. We compare *XShot* with Link Layer Discovery Protocol (LLDP) [24] and Logical Ring [18]. LLDP localizes failures using multicast messages, i.e., SDN controller periodically multicasts LLDP packets to switches. Upon receiving the packets, switches further multicast them to their respective neighbours, followed by returning back to controller. Once the packets on a link are dropped, the controller marks the link complete failure.

Logical Ring [18] proposes a binary search-based multi-round exploration process for localizing failures. It transforms the network topology into a bidirectional logical ring, then sending packets along the ring in a binary search way. If a packet did not return in a round, there must be a failure in the middle of the current path.

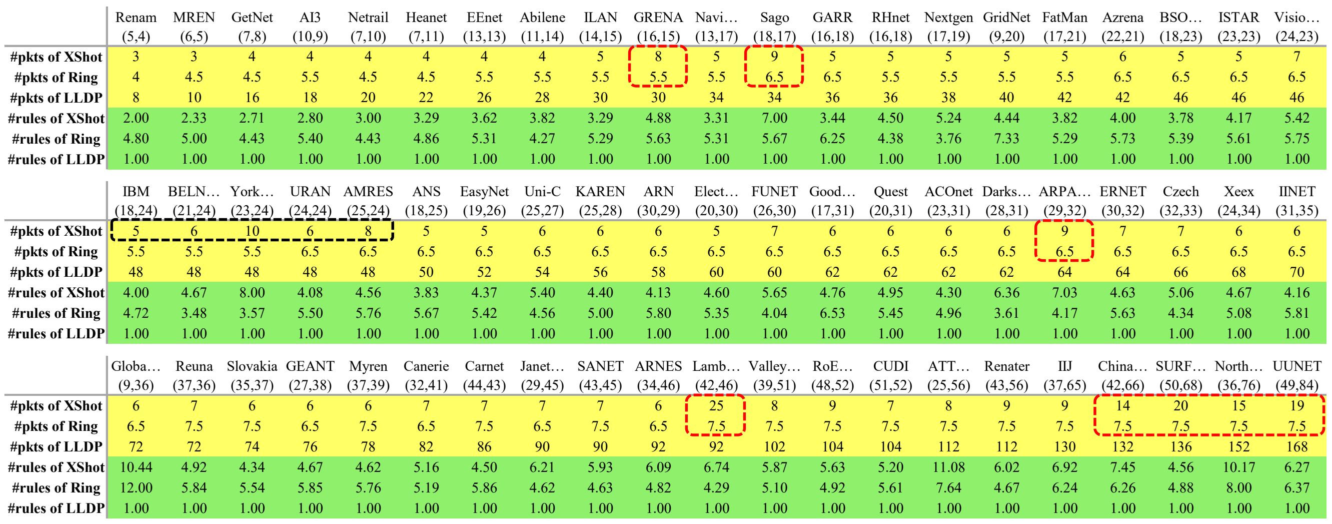
Clearly, the Logical Ring method may take much longer time to localize a failure. The probing time of each packet consists of two parts: propagation delay and transmission delay. For simplicity, let d_p and d_t be the average propagation and transmission delay respectively on each link. Then the probing time of packet i, can be denoted by $h_i(d_p + d_t)$, where h_i is the number of hops on the path traversed by packet i. In each round of probing, *XShot* injects all probing packets in parallel. It can thus localize a failure at once, and the localization delay d_{xshot} incurred by *XShot* is the probing time of the last received packet, that is, $h_{max}(d_p + d_t)$. According to the constraints (4) and (5), $h_{max} \leq 2\|E\|$ and then $d_{xshot} \leq 2\|E\|(d_p + d_t)$. Unlike *XShot*, Logical Ring sends probing packets sequentially in the binary search manner, making its localization delay d_{ring} up to $(3L_{ring} - 2) \times (d_p + d_t)$ even in the best case. Given $\|E\| \leq L_{ring} \leq 2\|E\|$, then $d_{ring} \geq (3\|E\| - 2) \times (d_p + d_t)$. In a word, the localization delay of Logical Ring is at least 1.5 times higher than that of *XShot*. Although the delay of Logical Ring can be mitigated by parallelizing the search, consuming more probing packets and/or forwarding rules is inevitable.

5.1.3 Metrics. We use four metrics to evaluate the performance of *XShot*: (1) The number of probing packets, i.e., the number of probing paths. (2) The number of forwarding rules installed in data plane. (3) The failure detection precision including: $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$, where TP, FP and FN refer to *true positive*, *false positive* and *false negative*. (4) Controller overhead, including CPU and memory usage.

5.2 Results

5.2.1 Number of Probing Packets. The yellow part in Figure 7 presents the number of probing packets required by three approaches for accurately localizing failures in the 63 network topologies. Since the number of packets required by Logical Ring varies according to the position on the ring of each failed link, we take the average number for all links as the overhead of Logical Ring.

We see that the packet cost for probing of either *XShot* or Logical Ring is low, and generally has shown a log linear increase with the topology size of the data plane (i.e., $O(\log \|E_d\|)$). The packet cost of LLDP-based method is the largest, which is the double of $\|E_d\|$. In 79.37% of topologies, *XShot* averagely requires 9.63% less number of probing packets than Logical Ring. However, in 20.63% of topologies, *XShot* needs more probing packets to locate link failures.



	Renam (5,4)	MREN (6,5)	GetNet (7,8)	AI3 (10,9)	Netrail (7,10)	Heanet (7,11)	EEnet (13,13)	Abilene (11,14)	ILAN (14,15)	GRENA (16,15)	Navi… (13,17)	Sago (18,17)	GARR (16,18)	RHnet (16,18)	Nextgen (17,19)	GridNet (9,20)	FatMan (17,21)	Azrena (22,21)	BSO… (18,23)	ISTAR (23,23)	Visio… (24,23)
#pkts of XShot	3	3	4	4	4	4	4	4	5	8	5	9	5	5	5	5	5	6	5	5	7
#pkts of Ring	4	4.5	4.5	5.5	4.5	4.5	5.5	5.5	5.5	5.5	5.5	6.5	6.5	5.5	5.5	5.5	5.5	6.5	6.5	6.5	6.5
#pkts of LLDP	8	10	16	18	20	22	26	28	30	30	34	34	36	36	38	40	42	42	46	46	46
#rules of XShot	2.00	2.33	2.71	2.80	3.00	3.29	3.62	3.82	3.29	4.88	3.31	7.00	3.44	4.50	5.24	4.44	3.82	4.00	3.78	4.17	5.42
#rules of Ring	4.80	5.00	4.43	5.40	4.43	4.86	5.31	4.27	5.29	5.63	5.31	5.67	6.25	4.38	3.76	7.33	5.29	5.73	5.39	5.61	5.75
#rules of LLDP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

	IBM (18,24)	BELN… (21,24)	York… (23,24)	URAN (24,24)	AMRES (25,24)	ANS (18,25)	EasyNet (19,26)	Uni-C (25,27)	KAREN (25,28)	ARN (30,29)	Elect… (20,30)	FUNET (26,30)	Good… (17,31)	Quest (20,31)	ACOnet (23,31)	Darks… (28,31)	ARPA… (29,32)	ERNET (30,32)	Czech (32,33)	Xeex (24,34)	IINET (31,35)
#pkts of XShot	5	6	10	6	8	5	5	6	6	6	5	7	6	6	6	6	9	7	7	6	6
#pkts of Ring	5.5	5.5	5.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5
#pkts of LLDP	48	48	48	48	48	50	52	54	56	58	60	60	62	62	62	62	64	64	66	68	70
#rules of XShot	4.00	4.67	8.00	4.08	4.56	3.83	4.37	5.40	4.40	4.13	4.60	5.65	4.76	4.95	4.30	6.36	7.03	4.63	5.06	4.67	4.16
#rules of Ring	4.72	3.48	3.57	5.50	5.76	5.67	5.42	4.56	5.00	5.80	5.35	4.04	6.53	5.45	4.96	3.61	4.17	5.63	4.34	5.08	5.81
#rules of LLDP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

	Globa… (9,36)	Reuna (37,36)	Slovakia (35,37)	GEANT (27,38)	Myren (37,39)	Canerie (32,41)	Carnet (44,43)	Janet… (29,45)	SANET (43,45)	ARNES (34,46)	Lamb… (42,46)	Valley… (39,51)	RoE… (48,52)	CUDI (51,52)	ATT… (25,56)	Renater (43,56)	IIJ (37,65)	China… (42,66)	SURF… (50,68)	North… (36,76)	UUNET (49,84)
#pkts of XShot	6	7	6	6	6	7	7	7	7	6	25	8	9	7	8	9	9	14	20	15	19
#pkts of Ring	6.5	7.5	7.5	6.5	7.5	6.5	7.5	6.5	7.5	6.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
#pkts of LLDP	72	72	74	76	78	82	86	90	90	92	92	102	104	104	112	112	130	132	136	152	168
#rules of XShot	10.44	4.92	4.34	4.67	4.62	5.16	4.50	6.21	5.93	6.09	6.74	5.87	5.63	5.20	11.08	6.02	6.92	7.45	4.56	10.17	6.27
#rules of Ring	12.00	5.84	5.54	5.85	5.76	5.19	5.86	4.62	4.63	4.82	4.29	5.10	4.92	5.61	7.64	4.67	6.24	6.26	4.88	8.00	6.37
#rules of LLDP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Figure 7: Cost evaluation of three failure localization approaches under the setting of $\|E_c\| = 1$. The part in 🟨 denotes the overhead of probing packets and that in 🟩 denotes the average number of forwarding rules per switch. "Renam(5,4)" represents *topology_name*(#*nodes*, #*links*) **in the data plane.**

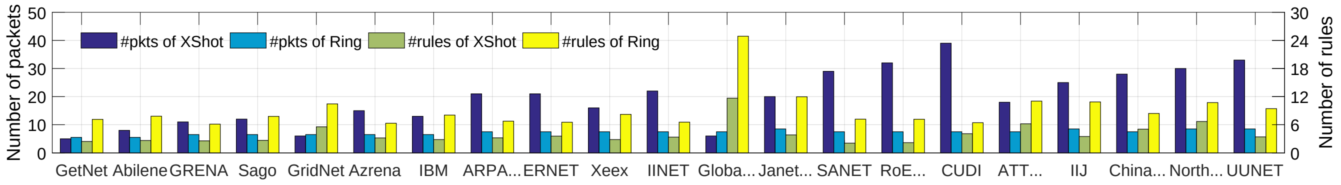


Figure 8: Cost evaluation of XShot and Logical Ring under out-of-band setting over varied real-world topologies.

This is because each of these topologies contains many one-cut or two-cut links. For example, all the links in GRENA and Sago are either one-cut or two-cut, while *XShot* requires 8 and 9 probing cycles in the two topologies, Logical Ring only requires an average of 5.5 and 6.5 probing cycles. This can also explain why the derived results (circled by black dotted lines in the figure) for the topologies with the similar number of links are quite different.

5.2.2 Number of Forwarding Rules. The green part in Figure 7 shows the evaluation results on the average number of forwarding rules per switch. Based on the multicast manner, LLDP approach requires only one rule on each switch for detecting failures. *XShot* and Logical Ring require roughly the same number of forwarding rules, and the average number of rules per switch is mostly between 4 and 7, which commonly occupy less than 0.1% of TCAM resources. More specifically, the total average rule number per switch for all topologies of *XShot* is 5.04, and that of Logical Ring is 5.37, that is *XShot* is 6.15% less than Logical Ring.

We further evaluate the rule cost by changing the control link setting into *out-of-band setting*, i.e., each switch has a dedicated link connected to the controller. Figure 8 shows the packet cost and forwarding rule cost of *XShot* and Logical Ring over different real-world topologies. Compared with the Logical Ring, *XShot* always requires less number of forwarding rules. The average number of static rules per switch of XShot is between 2 and 6 for most of the evaluated topologies, while that of Logic Ring is between 6 and 11. *XShot* saves an average of 56.26% rules than Logical Ring. In case of packet cost, *XShot* requires more packets than Logical Ring due to

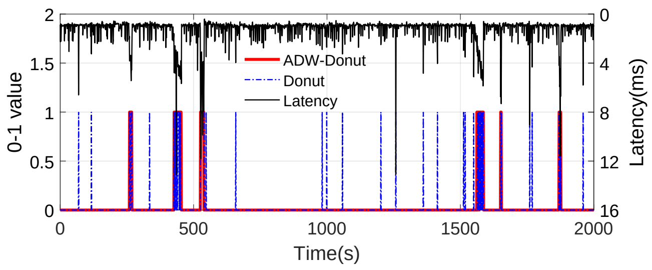


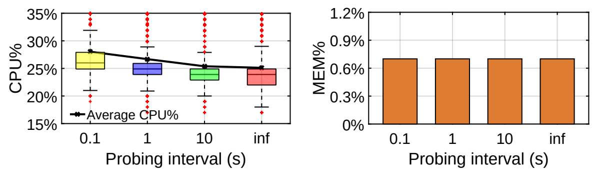
Figure 9: Failure detection using Donut with $ADW = 10$ **at** $interval = 1s$**: 0 represents the normal state, and 1 represents anomaly.**

the increase in the number of control links. Hence, *XShot* not only saves much time in failure localization, but also requires much less TCAM resources when $\|E_c\|$ is relatively large.

5.2.3 Failure Detection Performance. We evaluate the detection performance of *XShot* for high-latency failures by manually injecting failures using high-intensive load generated by iPerf [12]. Figure 9 shows the detection results of original Donut and the improved Donut with $ADW = 10$ at a probing interval of 1 second. Due to the fluctuations in measured latency, Donut yields more false positive results and has a poor detection precision. After we added the accelerated detection window into Donut, the probability of false positive is greatly reduced and the precision is increased to more than 94% as shown in Table 3. The recall is less than 90% implying that there are still some false negatives. The main reason is that the

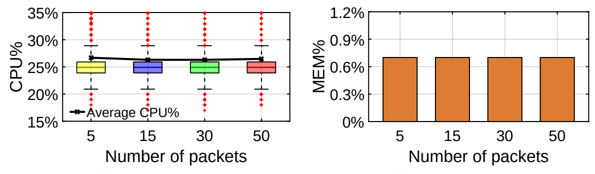
Table 3: Detection performance of Donut and ADW-Donut under different durations of congestion

	$\leq 5s$	$\leq 10s$	$\leq 20s$
Donut recall	76.87%	86.17%	87.07%
ADW-Donut recall	80.48%	87.57%	88.53%
Donut precision	75.24%	79.57%	81.56%
ADW-Donut precision	94.83%	96.28%	96.61%



(a) CPU usage (b) MEM usage

Figure 10: Controller overhead under different probing intervals when $\#pkts = 5$.



(a) CPU usage (b) MEM usage

Figure 11: Controller overhead under different number of probing packets when $interval = 1s$.

perceived latency has a slow growth in the start stage of congestion, and thus the early abnormalities are too weak to be detected. However, in the middle or later period of congestion, the recalls of Donut and ADW-Donut achieve more than 95%. If we increase the duration of network congestion, the detection performance is clearly improved in Table 3. For the short-term high-latency failures, the probing interval and/or detection window can be adjusted properly to further improve the precision and recall.

5.2.4 Overhead. We further measure the controller overhead in terms of the memory and CPU. Figure 10 and Figure 11 show the entire CPU usage and memory usage of Floodlight under different probing intervals ($interval = inf$ means that *XShot* is not working) and under different number of probing packets. In Figure 10(a), we see that the CPU usage increases slowly with the 10x growth of probing frequency, and *XShot* can increase the average CPU usage by less than 3%, compared with the *XShot*-not-working situation ($interval = inf$). In case of changing the number of probing packets, the CPU usage has barely changes (Figure 11(a)). For the memory usage in Figure 10(b) and Figure 11(b), we see that Floodlight is very steady and consumes only around 0.7% memory, little of which is caused by *XShot*.

6 RELATED WORK

Passive monitoring. To monitor the network health, the passive approach generally deploys monitoring agents on network devices [8], and periodically records the available information of each device using various statistical technologies (e.g., SNMP statistics) [15]. In particular, NetSight [14] diagnoses network problems by capturing and analyzing the log of each packet passing through each switch. The massive log capturing and analysis can result in high bandwidth and processing overhead, which is not scalable in large-scale networks [2]. Other systems [16, 21] collect and analyze the snapshots of the entire network state for monitoring at a coarse-grained timescale, but it is difficult to capture the consistent network state, and the alarms may be untrustworthy [26].

Active probing in traditional networks. Active probing measures the network state or performance using probing packets [8]. In traditional networks, while it is not trivial to identify the exact routing path for each packet, prior work seeks to estimate the failure location through a massive data analysis of probing packets [15]. For example, Pingmesh [13] leverages *pings* to measure round-trip network latency between every pair of peer severs, and uses the latency data to estimate relevant packet drop rate. PTPmesh [25] shows how to derive both one-way delay and packet loss ratio in networks from Precision Time Protocol (PTP) statistics based on UDP ping.

One way to find the path of a flow in traditional networks is to employ something like a traceroute. 007 [2] analyses the traceroute packets, and derives a ranking of links that most likely drop the packet on a TCP flow in a datacenter. However, this may be infeasible when ICMP is disabled in some intermediate nodes for security [35]. NetBouncer [29] detects both device and link failures leveraging the IP-in-IP technology, which is a hardware feature only supported in some modern switches, and it is only suitable for structured network topologies (e.g., CLOS network).

In the all-optical network, one can use monitoring cycles [1, 36, 37, 39, 40] or trails [1, 30–32] to detect and localize network failures. The cycles/trails are preconfigured monitoring paths, each assigned with individual optical transceivers and supervisory wavelengths. The monitoring cycles include simple cycles that traverse a device at most once [36], and nonsimple cycles that traverse a device multiple times [37]. However, the interference of light wave transmission makes the cycles unable to pass through the same link. To solve this problem, open monitoring structures are required, such as link-based and trail-based monitoring [1, 30–32]. Yet, they rely on many hardware monitors, incurring high cost.

Active probing in SDN. The emergence of SDN [10, 22] brings opportunities to tackle the challenges for failure localization in traditional networks. The default failure localization approach in SDN is to use LLDP [24, 27], which periodically multicasts LLDP packets to switches. Similarly, StaF [4] presents a stateful flooding method to localize link failures. It works by every node flooding its adjacent links and reporting their states to the monitor. However, these two approaches impose large amounts of communication load on the network, and further degrade the network performance. Kozat *et al.* [18] design a Logical Ring probing scheme to find faulty links with less probing packets and forwarding rules. It employs a multi-round biunary-search way for failure localization.

Failure localization can also be achieved through identifying link metrics [3, 20, 28, 35], which infer the switch/link state from end-to-end performance measurements, such as latency and packet drop rate. LLMP [20] modifies LLDP to support link latency measurements. While the link metrics can be obtained by measuring the performance of specified paths [3, 28, 35]. These approaches often need to deploy multiple monitors, or set up multiple paths via many rules, which also consume a lot of network resources (e.g., bandwidth, TCAM).

7 CONCLUSION

In this paper, we propose the design and implementation of *XShot*, a quick and light-weight link failure localization system in SDN. Based on the cross verification of multiple probing paths that start and end at SDN controller, *XShot* pinpoints the exact faulty link within just one-round shot of probing. By solving the ILP model of cost-minimization problem, *XShot* reduces the number of probing packets to $O(\log \|E\|)$ and the number of forwarding rules to $O(\|V\|)$. *XShot* identifies the high-latency failures by learning from the measured latency of crossed probing paths, and has a detection precision of more than 94%. In the future, we would like to extend the cross verification to the localization of multi-simultaneous link failures, and explore the failure localization approach in hybrid SDN networks.

ACKNOWLEDGMENTS

This work is supported in part by the National Key Research and Development Program of China No. 2019QY1302; the NSFC-General Technology Basic Research Joint Funds under Grant U1836214; NSFC Grant 61872265; the New Generation of Artificial Intelligence Science and Technology Major Project of Tianjin under Grant 18ZXZNGX00190, 19ZXZNGX00010.

REFERENCES

[1] Satyajeet S Ahuja, Srinivasan Ramasubramanian, and Marwan Krunz. 2008. SRLG failure localization in all-optical networks using monitoring cycles and paths. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*. IEEE, 700–708.

[2] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically Finding the Cause of Packet Drops. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, Sujata Banerjee and Srinivasan Seshan (Eds.). USENIX Association, 419–435.

[3] Alon Atary and Anat Bremler-Barr. 2016. Efficient Round-Trip Time monitoring in OpenFlow networks. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9.

[4] Akbari Indra Basuki and Fernando Kuipers. 2018. Localizing link failures in legacy and SDN networks. In *10th International Workshop on Resilient Networks Design and Modeling, RNDM 2018, Longyearbyen, Svalbard, Norway, August 27-29, 2018*. IEEE, 1–6.

[5] Kenneth K Chan, Philip W Hartmann, Scott P Lamons, Terry G Lyons, and Argyrios C Milonas. 1989. Virtual local area network. US Patent 4,823,338.

[6] Lu Cheng, Xuesong Qiu, Luoming Meng, Yan Qiao, and Raouf Boutaba. 2010. Efficient Active Probing for Fault Diagnosis in Large Scale and Noisy Networks. In *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*. IEEE, 2169–2177.

[7] IBM ILOG Cplex. 2009. V12. 1: User Manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.

[8] Ayush Dusia and Adarshpal S. Sethi. 2016. Recent Advances in Fault Localization in Computer Networks. *IEEE Communications Surveys and Tutorials* 18, 4 (2016), 3030–3051.

[9] Project Floodlight. 2015. Floodlight is an Open SDN Controller. http://www.projectfloodlight.org/Floodlight.

[10] Open Networking Foundation. 2012. Software-defined networking: the new norm for networks. *ONF white paper* (2012).

[11] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul (Eds.). ACM, 350–361.

[12] Vivien Gueant. 2017. iPerf-The TCP, UDP and SCTP network bandwidth measurement tool. *Iperf. fr. Np* (2017).

[13] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, David A. Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye (Eds.). ACM, 139–152.

[14] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, Ratul Mahajan and Ion Stoica (Eds.). USENIX Association, 71–85.

[15] Herodotos Herodotou, Bolin Ding, Shobana Balakrishnan, Geoff Outhred, and Percy Fitter. 2014. Scalable near real-time failure localization of data center networks. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani (Eds.). ACM, 1689–1698.

[16] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, Steven D. Gribble and Dina Katabi (Eds.). USENIX Association, 113–126.

[17] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. 2011. The Internet Topology Zoo. *Selected Areas in Communications, IEEE Journal on* 29, 9 (october 2011), 1765 –1775.

[18] Ulas C. Kozat, Guanfeng Liang, Koray Kokten, and János Tapolcai. 2016. On Optimal Topology Verification and Failure Localization for Software Defined Networks. *IEEE/ACM Trans. Netw.* 24, 5 (2016), 2899–2912.

[19] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks. HotNets 2010, Monterey, CA, USA - October 20 - 21, 2010*, Geoffrey G. Xie, Robert Beverly, Robert Tappan Morris, and Bruce Davie (Eds.). ACM, 19.

[20] Yang Li, Zhi-Ping Cai, and Hong Xu. 2018. LLMP: Exploiting LLDP for Latency Measurement in Software-Defined Data Center Networks. *J. Comput. Sci. Technol.* 33, 2 (2018), 277–285.

[21] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, Brighten Godfrey, and Samuel Talmadge King. 2011. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul (Eds.). ACM, 290–301.

[22] Nick McKeown. 2009. Software-defined networking. *INFOCOM keynote talk* 17, 2 (2009), 30–32.

[23] Nick McKeown, Thomas E. Anderson, Hari Balakrishnan, Guru M. Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker, and Jonathan S. Turner. 2008. OpenFlow: enabling innovation in campus networks. *Computer Communication Review* 38, 2 (2008), 69–74.

[24] Farzaneh Pakzad, Marius Portmann, Wee Lum Tan, and Jadwiga Indulska. 2016. Efficient topology discovery in OpenFlow-based Software Defined Networks. *Computer Communications* 77 (2016), 52–61.

[25] Diana Andreea Popescu and Andrew W Moore. 2017. Ptpmesh: Data center network latency measurements using ptp. In *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 73–79.

[26] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C. Snoeren. 2017. Passive Realtime Datacenter Fault Detection and Localization. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 595–612.

[27] Sayed Mohd Saquib, Eswara Chinthalapati, and Dilip Kumar. 2017. Efficient topology failure detection in SDN networks. US Patent 9,742,648.

[28] Megumi Shibuya, Atsuo Tachibana, and Teruyuki Hasegawa. 2014. Efficient performance diagnosis in OpenFlow networks based on active measurements. In *Proc. ICN*. 268–273.

[29] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 599–614.

[30] János Tapolcai, Pin-Han Ho, Lajos Rónyai, and Bin Wu. 2012. Network-wide local unambiguous failure localization (NWL-UFL) via monitoring trails. *IEEE/ACM Trans. Netw.* 20, 6 (2012), 1762–1773.

[31] János Tapolcai, Pin-Han Ho, Péter Babarczi, and Lajos Rónyai. 2013. On achieving all-optical failure restoration via monitoring trails. In *2013 Proceedings IEEE*

INFOCOM. IEEE, 380–384.

[32] János Tapolcai, Bin Wu, and Pin-Han Ho. 2009. On Monitoring and Failure Localization in Mesh All-Optical Networks. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil.* IEEE, 1008–1016.

[33] Niels L. M. van Adrichem, Christian Doerr, and Fernando A. Kuipers. 2014. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*. IEEE, 1–8.

[34] Open vSwitch. 2013. Production Quality, Multilayer Open Virtual Switch. http://www.openvswitch.org/.

[35] Xiong Wang, Mehdi Malboubi, Zhihao Pan, Jing Ren, Sheng Wang, Shizhong Xu, and Chen-Nee Chuah. 2018. ProgLIMI: Programmable LInk Metric Identification in Software-Defined Networks. *IEEE/ACM Trans. Netw.* 26, 5 (2018), 2376–2389.

[36] Bin Wu and Kwan L. Yeung. 2006. M2-CYCLE: an Optical Layer Algorithm for Fast Link Failure Detection in All-Optical Mesh Networks. In *Proceedings of the Global Telecommunications Conference, 2006. GLOBECOM '06, San Francisco, CA, USA, 27 November - 1 December 2006*. IEEE.

[37] Bin Wu, Kwan L Yeung, and Pin-Han Ho. 2009. Monitoring cycle design for fast link failure localization in all-optical networks. *Journal of lightwave technology* 27, 10 (2009), 1392–1401.

[38] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*. 187–196.

[39] Hongqing Zeng and Changcheng Huang. 2004. Fault detection and path performance monitoring in meshed all-optical networks. In *Proceedings of the Global Telecommunications Conference, 2004. GLOBECOM '04, Dallas, Texas, USA, 29 November - 3 December 2004*. IEEE, 2014–2018.

[40] Hongqing Zeng, Changcheng Huang, and Alex Vukovic. 2006. A Novel Fault Detection and Localization Scheme for Mesh All-optical Networks Based on Monitoring-cycles. *Photonic Network Communications* 11, 3 (2006), 277–286.

[41] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference, IMC 2017, London, United Kingdom, November 1-3, 2017*, Steve Uhlig and Olaf Maennel (Eds.). ACM, 78–85.

[42] Yao Zhao, Yan Chen, and David Bindel. 2006. Towards unbiased end-to-end network diagnosis. In *Proceedings of the ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pisa, Italy, September 11-15, 2006*, Luigi Rizzo, Thomas E. Anderson, and Nick McKeown (Eds.). ACM, 219–230.