# Rapid Detection and Localization of Gray Failures in Data Centers via In-band Network Telemetry

Chenhao Jia†, Tian Pan†‡, Zizheng Bian†, Xingchen Lin†, Enge Song†, Cheng Xu†, Tao Huang† and Yunjie Liu†
†State Key Laboratory of Networking and Switching Technology, BUPT, Beijing 100876, China
‡Purple Mountain Laboratories, Nanjing 211111, China
{564822241, pan, imagine, linxingchen, songenge, xu_cheng, htao, liuyj}@bupt.edu.cn

***Abstract*—Network reliability becomes increasingly important in modern data center networks (DCNs). The DCNs are expected to work sustainably under internal failures and assist network operators in troubleshooting them rapidly. However, some network failures will happen silently with packets discarded without producing any explicit notification before causing tremendous damage to the network. To troubleshoot these "gray failures", in this work, we present a rapid gray failure detection and localization mechanism based on the recently proposed In-band Network Telemetry (INT). Specifically, we leverage simplified INT probe packets to conduct network-wide telemetry to help the servers under ToR switches obtain all the feasible paths between sources and destinations. Once a network failure occurs, the affected thus unavailable paths will immediately be detected and flushed out of the path information table at each server by a timeout mechanism. Hence, servers can proactively perform source routing-based fast traffic reroute to avoid massive packet loss and retain uninterrupted quality of experience. At the meantime, all the aged path entries will be uploaded to a remote controller for centralized failure localization by identifying common path elements. To verify the feasibility of our design, we build a virtual network testbed with software P4 switches and a Redis database. Evaluation shows that our system can successfully detect network gray failures and reroute the affected traffic in no time while complete failure localization within only a few seconds.**

## I. Introduction

Data centers play an essential role in today's information ingestion, dissemination, computation, storage and management [1]. Meanwhile, the data center networks (DCNs) evolve to become mega-scale and high-density to meet diverse demands from heterogeneous user applications [2]. Specifically, DCNs are expected to achieve huge throughput to carry a great amount of aggregated traffic [3], ultra-low latency for fast response [4] and high security to protect private user data [5]. In addition, reliability of DCNs is also of great concern from both users and network operators that stirs up tremendous research interests [6–13]. It is expected that the DCNs can work uninterruptedly under random failures and help the network operators identify and resolve them rapidly, because the damage caused by failures is often evaluated according to the time duration before the failures get repaired [14].

Network failures can be divided into two categories. The first category of failures, including sudden power lost of the network equipment, abnormal link disconnection, *etc*, always occurs with corresponding alarms, making it straightforward to detect and locate. However, the other category of failures called *gray failures* (or silent failures or black holes) is more sophisticated and harmful [6]. The causes of gray failures range from the misconfiguration of network devices to the flaws in network protocol design or bugs in software design, all of which will make data packets discarded silently without any explicit notification. Therefore, it usually takes hours or even days to detect and locate gray failures while they have already made enormous damage. Hence, it is crucial to work out a mechanism to rapidly detect and locate gray failures.

In-band Network Telemetry (INT) [15] is a fine-grained monitoring architecture designed for collecting real-time network states in the data plane without requiring much intervention from the control plane CPU. In the INT model, a dedicated probe packet is used to traverse the network, while the INT-capable devices along the probing path will insert device-internal states into the probe packet. The probe packet can be collected by a remote centralized controller which will extract and analyze the telemetry metadata from the probe packet for network-wide congestion visualization, network event detection and network troubleshooting, *etc* [11].

In this work, we present a rapid gray failure detection and localization mechanism based on INT. First, servers will periodically send simplified INT probes to collect the end-to-end path information. The probes will be multicasted to reach all the servers except the sender, covering all feasible paths between sources and destinations. Second, each probe collector, which is also a server, will store these feasible paths into a path information table. Once gray failures occur, the affected path entries will be timed out due to the absence of the periodical probes. The server will get aware of this and proactively conduct traffic reroute via source routing to bypass the infeasible paths. Third, all the timed-out path entries will be uploaded to a remote controller for centralized failure localization. Since a single failure point can affect multiple paths, the controller will find out the commonality among these paths for failure localization.

Our major contributions are summarized as follows:

This work is supported by the National Key Research and Development Program of China (No. 2019YFB1802600), the National Natural Science Foundation of China (NSFC) (No. 61702049), the Fundamental Research Funds for the Central Universities, and the Research and Development Program in Key Areas of Guangdong Province (No. 2018B010113001). Corresponding author: Tian Pan.

978-1-7281-4973-8/20/$31.00 

- We raise a mechanism to rapidly detect network gray failures in DCNs based on INT. Specifically, we leverage a slim probe packet to collect all the feasible paths in the network and detect the network failures. Furthermore, we use source routing to reroute the traffic to bypass the infeasible paths once network failures are detected.
- We introduce a remote centralized controller to collect all the path entries which are timed out due to network failures from the servers. Then, the controller will find out the commonality among these timed-out paths to fast and precisely locate the network failure point.
- We build a network emulation system with software P4 switches to verify the feasibility of our design. The source code is available at our git repository [16]. Evaluation shows that our system can successfully detect network gray failures and reroute the traffic in no time, and complete the failure localization within a few seconds.

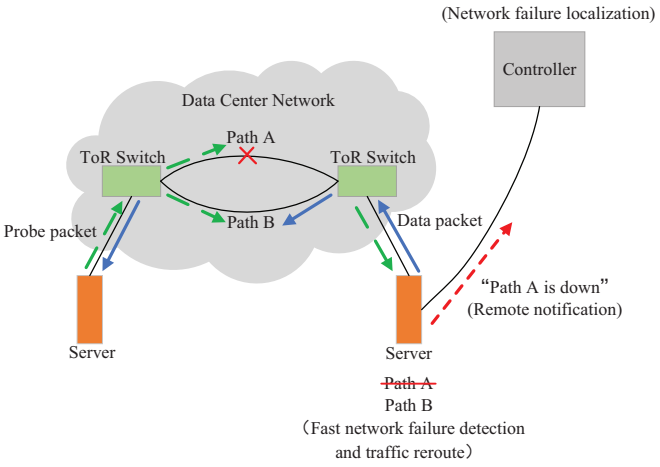



Fig. 1. Architecture of INT-based gray failure detection and localization.

## II. Challenging Issues

**Lightweight network-wide telemetry.** Gray failures may occur at any port of any device in DCNs, causing silent link breakdown and packet loss. To find out potential failures at the corners of the network, the entire network should be exhaustively monitored by the probe packets. A straightforward solution to achieve this is simply flooding the probe packets into every corner of the network. However, we still need to consider two vital constraints. First, the telemetry traffic will also contribute to considerable link bandwidth occupation, causing potential network congestion. To this end, a lightweight network-wide telemetry solution is appealing with the telemetry overhead reduced as much as possible. Second, for some specific DCN topologies, arbitrarily flooding the probes may cause probing loops. To this end, we need to carefully design specific multicast strategies for specific topologies to ensure loop-free telemetry.

**Prompt reaction to network failures.** Gray failures in DCNs are like black holes, which bring about silent loss of a huge number of packets. Even though end-host users or network operators may finally notice the abnormal traffic behaviors, it usually takes rather long time to troubleshoot the exact locations of gray failures before rerouting the traffic to bypass those black holes. The longer the reaction time, the heavier the loss of traffic. Besides, the loss of traffic will further trigger packet retransmission at the end hosts, causing even more congestion in the network. Conventionally, the data plane is the first witness of the gray failure. Then, it reports the failure to the centralized controller for further identification before the global reroute decision is generated by the controller. The traffic reroute will finally take place after the install of the updated forwarding rules. However, the above controller-switch interactions involve considerably large latency before the gray failure resolution. It is a challenging issue to promptly react to network failures (*e.g.*, by data plane fast reroute) to tremendously reduce the packet loss.

**Precise localization of network failures.** With network-wide telemetry and local reroute decision, the data plane can directly identify the broken paths and reroute the traffic to bypass these paths. However, the network operators will still want to figure out the root cause of the path failures (*e.g.*, which link/port is broken?), because reroute is only a temporary solution to reduce packet loss while the final solution should be the identification and elimination of the exact failure point. Considering that the device with the internal gray failure will not proactively report its bad condition, we have to rely on probe packets injected from outside as well as their abnormal behaviors or feedbacks to determine the exact failure point location. To summarize, fast and precise localization of network failures is also a research topic worth studying.

## III. System Design

### A. Overall Architecture

Our design is divided into three subsystems, namely, network-wide telemetry, network failure detection and fast traffic reroute at the local servers, network failure notification and localization at the remote controller (as shown in Fig. 1). First, servers will periodically send simplified INT probes (green arrows) to inspect all the feasible paths between sources and destinations. The collected information in the INT probes includes the identities of the switches they pass through and the corresponding ingress and egress port IDs. The probe packets will be multicasted to cover all feasible paths and reach all other servers except the sender. Second, each probe packet collector, which is also a server, will store these feasible paths in a *path information table* with an aging time for each path entry. Based on the path information table, servers can proactively send data packets onto the feasible paths via source routing (SR) [17] (blue arrows). When network failures occur, the probes will not be able to pass through the affected paths, making the relevant entries in the path information table aged/timed out thus the network failures can be detected promptly by the local servers. Given the affected path entries are aged, the subsequent traffic will be rerouted to other feasible paths to prevent packet loss. Third, since the path breakdown information at a single server is

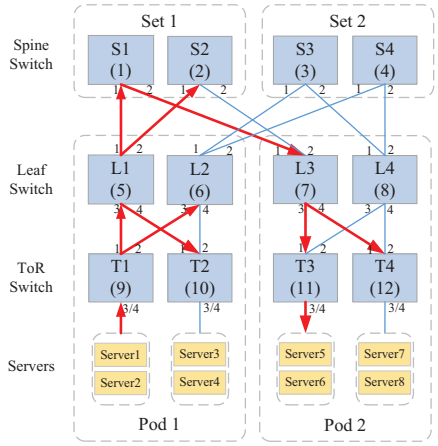



Fig. 2. Multicast-based routing of probe packets in a fat tree topology.

insufficient for precise network failure point localization, the controller needs to collect the path breakdown information from all the affected servers for centralized analysis (red arrows). More specifically, the controller will find out the commonality among these aged/timed-out paths to locate the exact failure point of a single link between two devices. We elaborate the three subsystems as follows.

### *B. In-band Network-Wide Telemetry*

**Probing path planning for network-wide coverage.** It is expected that the generated probe packets will cover all network paths for exhaustive on-path link state inspection and reach the servers at the other side for end-to-end path detection. To achieve this, we install dedicated forwarding rules into the switches, so that the switches will always send the probe packets to all the outgoing ports except the ingress port from which the probe packet comes in. This forwarding behavior ensures that the probe packets will traverse the whole network without triggering any forwarding loops. Fig. 2 shows how the probe packets generated from Server1 are forwarded in a 3-level fat tree topology [18]. The forwarding rules in each switch along the probing path are detailed as follows.

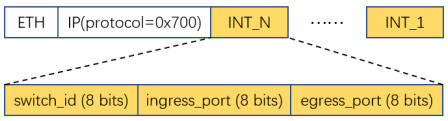
- For the ToR switch, when it receives a probe packet from a server, it will forward it to all the connected leaf switches (*e.g.*, Sever1 $\rightarrow$ T1 $\rightarrow$ L1 and L2). When it receives a probe packet from a leaf switch, it will forward the probe packet to the underneath servers (*e.g.*, L3 $\rightarrow$ T3 $\rightarrow$ Server5 and Server6).
- For the leaf switch, when it receives a probe packet from a ToR switch, it will forward the probe packet to all the ports except the ingress port (*e.g.*, T1 $\rightarrow$ L1 $\rightarrow$ S1, S2 and T2). When it receives a probe packet from a spine switch, it will send it to the underneath ToR switches (*e.g.*, S1 $\rightarrow$ L3 $\rightarrow$ T3 and T4).
- For the spine switch, it can only receive the probe packet from the leaf switch, and it will send it to all the other leaf switches it connects with (*e.g.*, L1 $\rightarrow$ S1 $\rightarrow$ L3).

In other words, the switches should forward the probe packet to the corresponding multicast group based on the switch type as well as the ingress port, which can be pre-configured by the controller. There is no need to update the data plane configuration unless the topology changes.

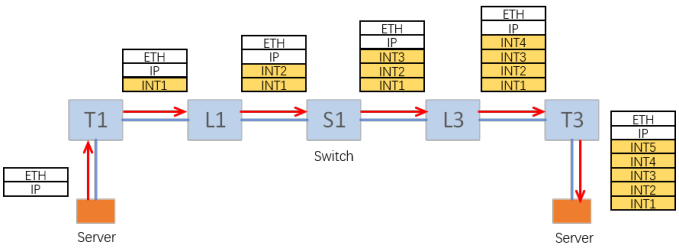
**Probe packet generation at the server side.** In our design, distinct from the previous work HULA [19] that generates probe packets via the ToR switches, we generate probe packets via the end servers, or more specifically, via the smart NICs [20] attached to the servers. Actually, deploying probe generators at the server side leads to many benefits. First, our design can provide complete end-to-end path monitoring covering the links between the switches as well as the links between the servers and the ToR switches (*i.e.*, the first and the last hop). By contrast, HULA probes are generated by the ToR switches and can only inspect the links between the switches and any gray failure occurring at the first/last hop (*e.g.*, server NIC port breakdown) cannot be quickly discovered. Second, in our design, end servers are aware of the gray failures in the network because they are at the endpoints of the telemetry paths. As a result, they can promptly react to manage the traffic they generate by proactively conducting traffic engineering (*e.g.*, using source routing). However, end servers in HULA are unaware of what happens in the network, therefore cannot take any effective measure to timely respond to network failures. While gaining so many benefits, deploying probe generators at the server side also brings about huge telemetry performance overhead, that is, the probe packets generated by so many end servers will aggregate and heavily consume the limited network bandwidth. To reduce the telemetry overhead, under each ToR switch, we allow only one server at a time for probe generation, which is based on our observation that, under the fat tree topology, the probe packets generated from all the servers under the same ToR switch share the exactly same probing path starting from the ToR switch. To inspect all the links between the servers and their ToR switch, we periodically change the chosen server for probe packet generation.

**Probe packet forwarding at the switch side.** In the original INT model [15, 21], the INT-capable device is expected to expose sufficient hardware-internal states to the probe packets, including the switch identity, the ingress/egress port ID, queue depth and queuing latency. With all these states extracted, the centralized controller can obtain the real-time congestion distribution across the network and conduct the corresponding traffic scheduling. To achieve prompt failure detection and localization in this work, however, not all above internal states (*e.g.*, queuing behaviors) are needed because we only care about whether the link is up or down rather than its congestion status. Besides, less extracted device states make the probe packet more slim and eat up less network bandwidth. The part of hardware-internal states extracted by the probe packet are listed and explained as follows.

- *switch_id (8-bit):* The identity of the switch. Each switch is assigned with a unique ID by the controller.
- *ingress_port (8-bit):* The ingress port ID from which the probe packet goes into the switch.
- *egress_port (8-bit):* The egress port ID from which the probe packet leaves the switch.




(a) Porbe packet format.




(b) Probe packet forwarding.

Fig. 3. The format and forwarding of the simplified/slim INT probe.

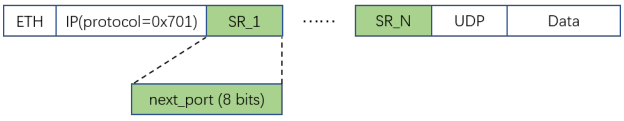
| Destination | Path |
|---|---|
| Server2 | sw9_3→sw9_4 |
| ..... | ..... |
| Server7 | sw9_4→sw9_1→sw5_3→sw5_1→sw1_1→sw1_2→sw7_1→sw7_4→sw12_1→sw12_3 |
| | sw9_4→sw9_1→sw5_3→sw5_2→sw3_1→sw3_2→sw7_1→sw7_4→sw12_1→sw12_3 |
| | sw9_4→sw9_2→sw6_3→sw6_1→sw2_1→sw2_2→sw8_1→sw8_4→sw12_2→sw12_3 |
| | sw9_4→sw9_3→sw6_3→sw6_2→sw4_1→sw2_2→sw8_2→sw8_4→sw12_2→sw12_3 |
| Server8 | sw9_4→sw9_1→sw5_3→sw5_1→sw1_1→sw1_2→sw7_1→sw7_4→sw12_1→sw12_4 |
| | sw9_4→sw9_1→sw5_3→sw5_2→sw3_1→sw3_2→sw7_1→sw7_4→sw12_1→sw12_4 |
| | sw9_4→sw9_2→sw6_3→sw6_1→sw2_1→sw2_2→sw8_1→sw8_4→sw12_2→sw12_4 |
| | sw9_4→sw9_3→sw6_3→sw6_2→sw4_1→sw2_2→sw8_2→sw8_4→sw12_2→sw12_4 |

Fig. 4. Server1's path information table.

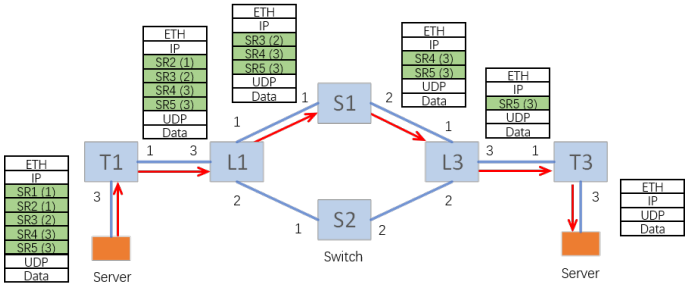
As shown in Fig. 3, the initial probe packet generated from the end server is composed of an Ethernet header and an IP header. We assign 0x700 as its protocol number to denote it is a simplified INT probe. As the probe packet passes through the network, the switches along the path will insert the INT information into the probe just behind its IP header. Finally, at the other endpoint of the telemetry path, the INT information is arranged as a stack inside the probe packet with the first piece of INT information residing at the end of the probe.

### C. Network Failure Detection and Fast Traffic Reroute

**Path information table.** After receiving the probe packets, the INT information carried by these probe packets will be parsed and stored locally in a path information table in each server at the receiving side. The path information table of Server1 is shown in Fig. 4, which illustrates all the servers that can be reached by Server1 and all the paths that can be taken. Each path entry consists of a group of switches as well as the corresponding ingress and egress ports in the probing path, along which the data packet can reach the specific destination server (because it shares exactly the same path with the probe packet). For example, if a data packet is sent from Server1 to Server2, the only path it can take is to go into sw9 from port 3 (sw9_3) and then leave the switch from port 4 (sw9_4) according to the illustrated path information table.




(a) Data packet format.




(b) Data packet forwarding.

Fig. 5. The format and forwarding of the data packet.

**Gray failure detection via path aging.** Each entry in the path information table is appended with an aging time tag. The table entry will be aged/timed out after a certain period of time unless it is updated by a newly arrived relevant probe packet with the aging time reset. If network failures occur, some probe packets will be dropped, prevented from reaching the destination server. Consequently, the relevant entries will be timed out. In other words, an entry will always exist in the table if the relevant probe packets are continuously received by the server while any timed-out entries indicate that there are some path failures occurring in the network. Since the intermediate switches cannot get aware of the gray failures by themselves to proactively generate notifications, we have to detect the gray failures using the above probe-based approach. There is one more question to clarify that how to differentiate gray failures from network congestion in our approach, since network congestion will also cause buffer overflow and packet loss. Actually, we can configure a bit longer aging time (*e.g.*, 1s) to filter out some light congestion (because light congestion will not last too long due to the fast decrease of the congestion window size). As for more severe congestion, paths could still be timed out but we think this can reasonably be regarded as some kind of network failures for dedicated treatment.

**Fast traffic reroute via source routing.** When some path breakdown is detected, the corresponding server will promptly reroute the affected data traffic according to the latest path information table using source routing (SR) [17]. SR is a hop-by-hop routing mechanism for the packet sender to specify the exact path along which the data packets pass through the network. As shown in Fig. 5, the data packet with SR metadata embedded is labelled with 0x701 as its protocol number, and all the egress ports along the specific path are arranged into the SR metadata located behind the IP header. When the data packet passes through the network, the switch will forward the data packet to the egress port completely according to the first parsed (*i.e.*, left-most) SR information. At the same time, the used SR information will be removed, ensuring that it will not be used again by the next switch. The SR information

embedded in the data packets is optimally selected based on the real-time available paths in the path information table. Therefore, the timed-out path entries due to network failures will no longer be used and the affected traffic will be rerouted to the available paths at once.

### D. Network Failure Localization at the Remote Controller

**Network failure remote notification.** Although network path failures can be detected by servers with traffic rerouted promptly to bypass the affected paths, it is still far away from network failure elimination unless the exact port/link breakdown can be located. However, since there is no global network view shared among the distributed servers, the timed-out path entries in a single server are insufficient for locating the exact network failure point. Here, following the SDN design paradigm [22], we introduce an external controller to collect path failure notifications sent from all the distributed servers and conduct centralized network failure localization. Specifically, each server should notify the remote controller about the broken paths when it deletes the timed-out entries.

**Centralized network failure localization.** In data center networks, multiple paths can be affected even by a single point failure and this observation can be utilized for precise network failure localization. By identifying the commonality among all the timed-out path entries, the scope of network failure will be gradually narrowed to a single link between two devices. Specifically, when the controller receives two timed-out path entries for the first time, it will find the commonality between them, which is considered as the result of the first round. Then, the subsequently received timed-out path entry will be compared with the result of the last round for finding the commonality between them. The above process will iterate until the link breakdown position is finally located with enough path failure information. For example, all the paths affected by the link failure between T1 and L1 are shown in Fig. 6. Since the timed-out path entries are sent to the controller from distributed servers, the order of path entries received by the controller is uncertain. If the 39th and the 40th path entries are received by the controller first, the commonality between them will be sw12_4 → sw12_1 → sw7_4 and sw5_3 → sw9_1 → sw9_4. And then if the controller further receives the 1st path entry, which is sw9_3 → sw9_1 → sw5_3 → sw5_4 → sw10_1 → sw10_3, the commonality between the newly received path entry and the previous result will be sw5_3 → sw9_1 (sw5_3 → sw9_1 and sw9_1 → sw5_3 are essentially the same link). Finally, the network failure will be located between port 3 of L1 (sw5_3) and port 1 of T1 (sw9_1). The above example also shows that there is no need to receive all the timed-out path entries before locating the failure point.

## IV. Implementation

### A. Network Testbed Setup

We use Mininet [23] to create the virtual network environment, which includes four spine switches, four leaf switches, four ToR switches and eight servers as shown in Fig. 2. To allow arbitrary packet header customization of INT, we use the simple switch model (a targeted switch architecture) of BMv2 (a software-based P4 switch) [24] to implement the virtual programmable switches in our testbed. Although the simple switch model of BMv2 might has some difference with the real hardware P4 switch, it is good enough for our design verification. In addition, we implement a remote controller using a separate OS process, which receives notifications from the servers for network failure localization. The controller communicates with the servers through the network socket. We write about 400 lines of P4 and Python code in total for creating such virtual network testbed, which is completely available in our git repository [16].

| No. | Source | Destination | Path |
|---|---|---|---|
| 1 | Server1 | Server3 | sw9_3→sw9_1→sw5_3→sw5_4→sw10_1→sw10_3 |
| 2 | Server1 | Server4 | sw9_3→sw9_1→sw5_3→sw5_4→sw10_1→sw10_4 |
| 3 | Server1 | Server5 | sw9_3→sw9_1→sw5_3→sw5_1→sw1_1→sw1_2→sw7_1→sw7_3→sw11_1→sw11_3 |
| ..... | ..... | ..... | ..... |
| 38 | Server8 | Server1 | sw12_4→sw12_1→sw7_4→sw7_2→sw3_2→sw3_1→sw5_2→sw5_3→sw9_1→sw9_3 |
| 39 | Server8 | Server2 | sw12_4→sw12_1→sw7_4→sw7_1→sw1_2→sw1_1→sw5_1→sw5_3→sw9_1→sw9_4 |
| 40 | Server8 | Server2 | sw12_4→sw12_1→sw7_4→sw7_2→sw3_2→sw3_1→sw5_2→sw5_3→sw9_1→sw9_4 |

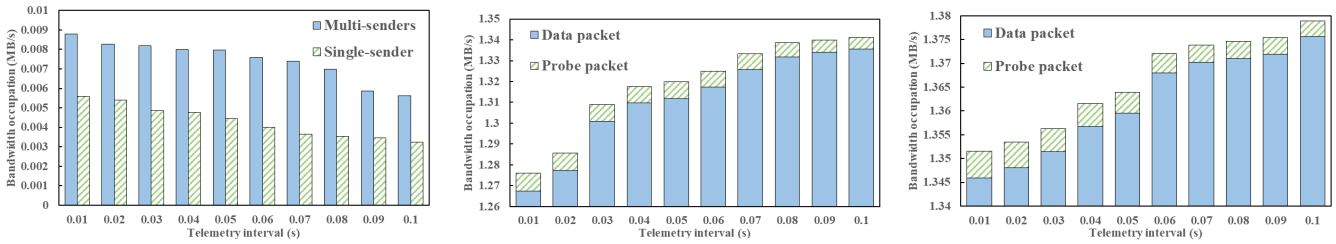
Fig. 6. Paths affected by the link failure between T1 and L1.

### B. Probe Generation, Collection and Storage

In the network created by Mininet, each server is a separate process, which means we can further spawn child processes to implement subtasks such as sending and receiving probe/data packets, and interacting with the remote controller. We leverage Python's Scapy library to write scripts for probe packet generation and collection. In the probe generation script, we can alter the telemetry frequency by adjusting the process sleeping time, which is set to 0.01s by default. After receiving the probe packets, we will parse them and store the telemetry results into a Redis database allocated on each server. Redis is an in-memory key-value database that has a high data read/write speed as well as good support for concurrency, available for high-speed probe packet handling in real time. Similarly, background data traffic is sent and received with the Scapy scripts as well. We write around 250 lines of Python code for probe/data traffic manipulation.

### C. Network Failure Notification and Localization

In Redis database, changes to the database including data addition, modification and aging operations can be published to specific channels while other agents can quickly get aware of these changes by subscribing the channels. In our system, we set an aging time for every entry of the path information table in the database. Once any table entry ages, it will be published to a specific channel. At the same time, we start a separate OS process at each server to subscribe the channel and send a notification to the remote controller about the aged entry through the network socket. The controller will further analyze the received notifications for centralized network failure localization by identifying common path elements. This part of logic contains about 30 lines of Python code.




(a) Bandwidth occupation of probe packets. (b) Bandwidth occupation in multi-senders. (c) Bandwidth occupation in single-sender.

Fig. 7. Bandwidth occupation of probe packets and data packets under single-sender and multi-senders modes.

## V. Evaluation

### A. Experimental Setting

In this section, we will evaluate our design in the following three perspectives through extensive experiments:

- What performance impact does periodical network-wide telemetry have on the original network?
- Can network devices detect and locate the gray failures rapidly by using the mechanisms we propose?
- How to make our system work more efficiently under different network conditions by tuning parameters?

We conduct experiments on an x86 server with the CPU of Intel Xeon E5-2603 v4 @ 1.70GHz, 6 cores and the memory of 32GB, running Ubuntu Linux 16.04.

### B. Telemetry Overhead

Before conducting telemetry, we need to select the probe packet generators from all the servers and set the appropriate probing interval. As mentioned earlier, we have two possible ways to deploy probe generators. The first is called multi-senders, which requires all the servers to send probe packets. The second is single-sender, which uses only one server at a time and changes the server periodically under the same ToR switch for probe generation. We compare the two approaches for probe generation and also study the relation between the telemetry interval and the network resource occupation.

We monitor the traffic load on the link between L1 and T1, whose bandwidth is limited to 12Mbps. At first, we adjust the data traffic sending rate at the servers, ensuring that the data traffic occupies the entire network bandwidth. After that, we inject probe packets into the network. As shown in Fig. 7(a), as the telemetry interval increases, the probes will occupy less bandwidth in both multi-senders and single-sender approaches. Since the number of the probe senders in multi-senders is twice that of single-sender, the probes in multi-senders occupy around twice as much bandwidth as that in single-sender.

Fig. 7(b) and Fig. 7(c) show the performance impact of introducing probe packets into the original traffic under two deployment modes. With the increase of the telemetry frequency, more and more probe packets get into the network, which will at the same time take up part of the link bandwidth thus make the bandwidth occupied by the data packets lower. In addition, higher telemetry frequency makes the total bandwidth occupation of both data packets and probe packets less than 12Mbps (*i.e.*, 1.5MB/s), and the reduction is more significant under multi-senders mode. By analyzing the CPU usage in servers, we figure out that the probe generation will consume CPU resource and the higher telemetry frequency means more CPU clock cycles will be used for sending probes, resulting in a lower packet sending rate in total.

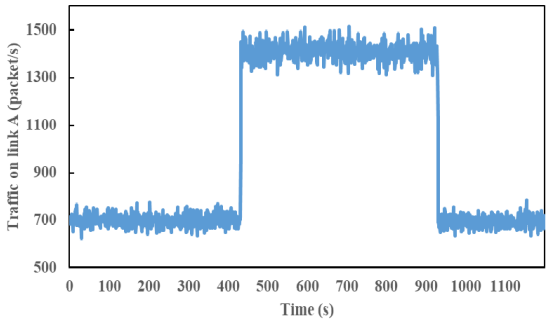


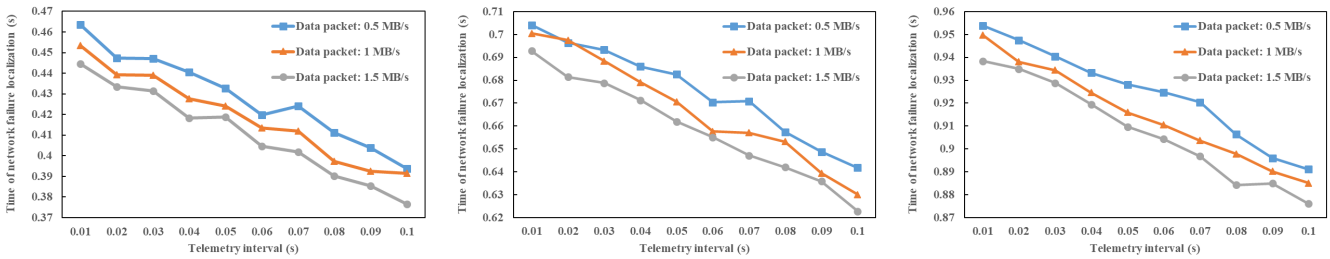

Fig. 8. Fast traffic reroute.

It can be inferred that with the increase of the number of servers under each ToR switch, the performance advantage of single-sender will be even more obvious, although it needs additional efforts for periodical server scheduling as well.

### C. Fast Reroute Triggered by Failure Detection

We verify whether the traffic can be rapidly rerouted after the network failure is detected by monitoring the traffic changes on the link between T1 and L1 (*i.e.*, link A). We also name the link between T1 and L2 as link B. To make the result clear and distinct, Server1 and Server2 will continuously send data packets while other servers will not. Most packets sent from Server1 and Server2 (except those sent between Server1 and Server2) will pass through link A or link B with roughly equal probabilities, which means the traffic conditions in these two links should be similar. We monitor the traffic on link A for 20 minutes, and then artificially create a network failure on link B during this period of time, to figure out whether all the traffic will pass through the feasible path instead of the path with the network failure at once.

As shown in Fig. 8, there are about 700 packets per second on link A from 0 to 450s and the similar traffic on link B. At 450s, we disconnect link B, then the number of packets per second on link A rapidly increases to 1400 in no time, indicating that the traffic which should have passed through link B is now passing through link A. It means the servers can quickly reroute the traffic to the feasible paths when network




(a) Aging time: 0.5s. (b) Aging time: 0.75s. (c) Aging time: 1s.

Fig. 9. Time of network failure localization under different aging times.

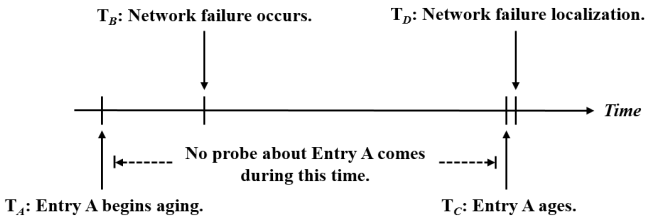



Fig. 10. Timeline of network failure localization.

failures occur under the help of telemetry. At 900s, the link between T1 and L1 is reconnected, the number of packets per second on link A is now rapidly decreasing to 700, which is the same as that before the network failure. Since the telemetry is continuously performed by the servers, the link reconnection can be fast detected by the servers and then the newly feasible paths will be used for data packet sending. The result shows that the telemetry together with SR can help servers detect network failures and reroute the traffic in a very short time.

### *D. Timeline of Network Failure Localization*

Once the controller receives enough aged path entries from servers, the network failure can be located. The faster the failure is located and repaired, the less damage it will cause. Here, we study the impact of aging time, telemetry interval and data packet sending rate on the efficiency of network failure localization, which is defined as the time between network failure occurrence and its final localization.

Fig. 9 shows that the aging time roughly determines the time of network failure localization, since the network failure can be discovered only after the aging of the corresponding path entries. Therefore, the shorter the aging time is, the more timely the network failure can be detected and located. Besides, the time of network failure localization is shorter than the aging time. This can further be explained through the timeline of network failure localization as shown in Fig. 10.

As mentioned earlier, the aging time of each entry in the path information table will be reset once the relevant probe packets arrive. When failure occurs, as some entries begin aging (at $T_A$) before the occurrence of the network failure (at $T_B$), no relevant probe packet can reach the servers and reset the aging time of these entries, making them aged at $T_C$. Since the controller only needs to make a simple calculation with the received aged path entries (this will not take a long time), hence the time between $T_C$ and network failure localization ($T_D$) is very short. Therefore, the time between $T_B$ and $T_D$ is usually shorter than that between $T_A$ and $T_C$, indicating the time of failure localization will be shorter than the aging time.

It is also shown in Fig. 9 that the time of network failure localization goes smaller during the increase of the telemetry interval. The reason is that when the network failure occurs, it will prevent the subsequent probe packets from reaching the servers while some probe packets that have already passed the faulty links and been buffered in the switches before the occurrence of the network failure can still reach the servers. These probe packets will continuously reset the aging time of the relevant entries in the path information tables after the occurrence of the network failure, making the servers mistakenly believe these paths are still feasible. However, as the telemetry interval increases, there will be less probe packets in the network and the entries in the path information table will be less updated. As a consequence, the server can detect the network failure much earlier. Similarly, a larger data packet sending rate can also accelerate the localization of network failures. Since a larger data packet sending rate will delay the sending of probe packets or even cause probe packet loss, resulting in a less updated path information table.

### *E. Fake Notification*

Although a lower telemetry frequency or larger data packet sending rate will make network failure localization faster, it will also put servers at the risk of generating fake notifications because probe packets cannot reach servers timely. We count the number of failure notifications received by the controller within 10 minutes when there is no real network failure. The result shown in Fig. 11 indicates that a smaller telemetry interval should be picked under the condition of a high data packet sending rate. Otherwise, fake notifications are likely to occur. Furthermore, the aging time of each entry in the path information table should be at least larger than the telemetry interval so as not to generate fake notifications.

### *F. System Parameter Tuning*

There are three important system parameters involved, namely, the aging time, the telemetry interval and the background traffic rate, which have significant impact on the failure resolution efficiency. Among the three parameters, the first two

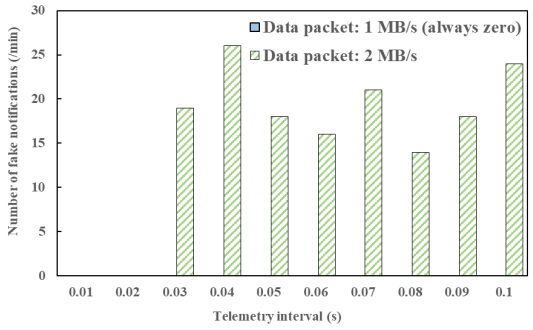



Fig. 11. Number of fake notifications.

are tunable while the third one is determined by the real traffic. Based on the experiments above, we propose the following heuristics for system parameter tuning.

First, we should estimate or directly measure the Round-Trip Time (RTT) in the real network environment. RTT is the time taken for a packet to go from the starting point to the destination and then back again to the starting point, which is an important indicator for network congestion [25]. If RTT is large, high telemetry frequency should be guaranteed so as to prevent the fake notification, although it will also cause high performance overhead and large delay of failure detection. Network operators need to make trade-offs between them. If RTT is small, a low telemetry frequency is satisfactory. The aging time of each entry in the path information table should be at least larger than the telemetry interval, ensuring that there is no fake notification as much as possible. It is noted that the aging time cannot be set too large as well, which will affect the timely detection and localization of network failures.

## VI. Related Work

**Protocol-independent forwarding and derived telemetry applications.** The rapid detection and localization of gray failures in our proposal rely heavily on the In-band Network Telemetry (INT) primitive proposed in [15]. In INT, a probe packet can extract device-internal states when it travels through the network devices and modify the probe packet header to accommodate the extracted states. Such ability of arbitrarily modifying the packet header is guaranteed by the protocol-independent forwarding architecture proposed in [26]. In protocol-independent forwarding, switch data plane can be programmed using the P4 language for adding customized network functions. The original INT specification defines several types of device-internal states to extract including forwarding latency and queue depth [21]. While in our solution, we tailor the original INT probe to carry only the switch ID and the ingress/egress port. Such simplified/slim probe packet format is dedicated to network troubleshooting with much reduced bandwidth overhead. For network-wide telemetry, INT-path [11] proposes a high-level orchestration of INT to generate non-overlapped probing paths with a minimum path number that cover the entire network. INT-path is theoretically perfect but with a deployment flaw that any topology change will trigger the replanning of the probing paths, which is inadaptable to failure-prone DCNs. Besides, servers in INT-path are unaware of the in-network congestion status, which means the remote controller is required for both network failure detection and localization by analyzing the collected telemetry results. HULA [19] is another telemetry application based on P4 and the INT-like primitive. It introduces a multicast-based probing method for specific DCN topologies. HULA has made an attempt for network failure detection. Specifically, HULA probe packets are sent between switches to monitor the links between them. The switches will reroute the traffic immediately after network failures are detected. However, if network failures occur at the links between the ToR switches and the underneath servers, they will not be detected by the HULA probes. Furthermore, HULA does not address the network failure localization problem.

**Network troubleshooting of gray failures.** The size expansion of DCNs leads to the rising frequency of gray failures. [8] defines and models the gray failures, and proposes an idea to detect and locate them, that is, leveraging the observations from a large number of different components that are complementary to each other to help uncover gray failures rapidly. Our failure localization scheme actually follows this design philosophy by identifying common path elements from multiple notifications. Pingmesh [12] uses TCP or HTTP pings between end hosts to infer whether the links along the path are broken according to the connection establish time. It can detect most network failures while the limitation is that it cannot precisely locate the faulty device during silent packet drops unless further using traceroute. NetBouncer [10] relies on servers to send probe packets through IP-in-IP tunneling [27], and the probe packets will return through the original path after reaching the destination switch. The probe packets will be collected by a processor to infer where the network failures are most likely to occur by a dedicated algorithm. Since NetBouncer is deployed in production environment without using P4 switches thus there are no switch IDs along the probing path embedded in the probe packets, probabilistic inferences according to observations are inevitable for failure localization. Besides, servers in NetBouncer cannot fast reroute the traffic when failures are detected. [6] also uses end-to-end probing to detect, and spatial correlation to infer the failure, without relying on the fine-grained, hop-by-hop probing ability provided by the programmable switch.

## VII. Conclusion

In this work, we propose a system for rapid detection and localization of gray failures in data center networks based on INT. It allows the servers to obtain the real-time path information through periodical network-wide telemetry. Once a failure occurs, the affected servers will detect the failure and make fast traffic reroute. At the same time, information of the broken paths will be uploaded to a remote controller for centralized network failure localization. By carefully tuning parameters, our system can achieve network failure detection, traffic reroute in no time, and network failure localization within only a few seconds, with minor performance overhead.

## REFERENCES


[1] A. D. JoSEP, R. KAtz, A. KonWinSKi, L. Gunho, D. PAttERSon, and A. RABKin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010.

[2] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[4] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012, pp. 253–266.

[5] M. Okuhara, T. Shiozaki, and T. Suzuki, "Security architecture for cloud computing," *Fujitsu Sci. Tech. J*, vol. 46, no. 4, pp. 397–402, 2010.

[6] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "Detection and localization of network black holes," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 2180–2188.

[7] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren, "Passive realtime datacenter fault detection and localization," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 595–612.

[8] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray failure: The achilles' heel of cloud-scale systems," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. ACM, 2017, pp. 150–155.

[9] H. Zeng, R. Mahajan, N. McKeown, G. Varghese, L. Yuan, and M. Zhang, "Measuring and troubleshooting large operational multipath networks with gray box testing," *Mountain Safety Res., Seattle, WA, USA, Rep. MSR-TR-2015-55*, 2015.

[10] C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang, "Netbouncer: Active device and link failure localization in data center networks." in *NSDI*, 2019, pp. 599–614.

[11] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, and Y. Liu, "Int-path: Towards optimal path planning for in-band network-wide telemetry," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 487–495.

[12] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[13] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 350–361, 2011.

[14] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[15] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.

[16] "Int-detect repository," https://github.com/graytower/INT_DETECT, 2019.

[17] C. A. Sunshine, "Source routing in computer networks," *ACM SIGCOMM Computer Communication Review*, vol. 7, no. 1, pp. 29–33, 1977.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[19] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, 2016, pp. 1–12.

[20] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 51–66.

[21] "In-band network telemetry (int) - p4.org," https://p4.org/assets/INT-current-spec.pdf, 2018.

[22] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[23] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[24] "P4 switch behavioral model," https://github.com/p4lang/behavioral-model, 2018.

[25] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," in *ACM SIGCOMM Computer Communication Review*, vol. 17, no. 5. ACM, 1987, pp. 2–7.

[26] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[27] W. Simpson, "Ip in ip tunneling," 1995.