Network Telemetry by Observing and Recording on Programmable Data Plane

Wen-Hong Lin
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China
958310048@qq.com

Wai-Xi Liu*
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China
lwx@gzhu.edu.cn

Gui-Feng Chen
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China

Song Wu
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China

Jin-Jiang Fu
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China

Xing Liang
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China

Sen Ling
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China

Zhi-Tao Chen
department of electronic and communication engineering
Guangzhou University
Guangzhou, P. R. China

***Abstract*—Fine-grained, real-time, and accurate monitoring data can better help detect equipment failure and perform traffic engineering. However, existing in-band network telemetry (INT) implementations still exhibit a few drawbacks such as lack of real-time monitoring, relatively high overheads due to per-packet operation, and limited monitoring range. This paper proposes an INT+PDP-based fine-grained real-time telemetry scheme by observing and recording on the programmable data plane (PDP), referred to as O&R. The key idea lies in designing some registers on data plane to observe the states of packets forwarded by it as well as adding a customized header on a normal data packet to record how it is forwarded on its routing path. Except for measuring some conventional performance parameters such as end-to-end delay, jitter, throughput, and packet loss rate, O&R designs a clock offset elimination algorithm to realize the time synchronization of two adjacent switches, based on which we can complete more fine-grained measurement such as queuing delay, processing delay, transmission delay, and propagation delay on any hop. O&R also can measure the queue state that includes real-time queue depth and how many flows share the queue. Extensive experimental results for the K=4 fat-tree data-center network demonstrate the effectiveness of O&R in terms of higher accuracy, better real-time performance, less overheads, and better fine-graining compared to existing schemes. The measurement accuracy of O&R is 46.3% higher than that of INT-like method. The measurement delay of O&R is ~1 ms, while INT-like method needs ~20 ms. The measurement overhead of O&R is only 2.19% of Pingmesh.**

Keywords—In-band network telemetry, programmable data plane, monitoring, fine-graining, real time

I. INTRODUCTION

In recent years, as a result of the proliferation of non-elastic services and the adoption of novel paradigms, monitoring networks with a high level of detail is crucial to correctly identify and characterize situations related to faults, performance, and security [1]. In other words, fine-grained and real-time monitoring is an urgent requirement.

In-band network telemetry (INT)[13] has emerged as a promising approach to meet the above-mentioned demands, enabling production packets to directly report their experience inside a network. This enables unprecedented monitoring accuracy and precision but leads to performance degradation if applied indiscriminately using all network traffic. An alternative to avoid this situation is orchestrating telemetry tasks and using only a portion of traffic to monitor the network via INT. However, this brings about a dilemma problem regarding assigning subsets of traffic to perform INT and providing full monitoring coverage while minimizing overheads. Besides, INT can only detect fixed telemetry indicators on the specified paths but cannot adapt to the changing network environment and telemetry requirements. INT cannot detect the network status on other paths to cover the whole network, thereby making it difficult to obtain a global network view. Meanwhile, encapsulating telemetry commands and data into normal data packets leads to high overhead, and the complexity of deployment, operation, and maintenance leads to poor scalability.

Behind these above-mentioned drawbacks, it is the rigidity of the data plane that the fundamental limitation of existing INT-based network monitoring solutions lies in—off-the-shelf switching ASICs can only provide simplistic functions to compute fixed and aggregated counters or mechanically mirror packets to the management plane. Thus, despite the origin of measurement target (such as delay, queue, and packet loss event) is on the data plane, performing sophisticated monitoring logics (such as event detection, data cleaning, and compression) must rely on the remote management plane, which results in low real-time, huge traffic transmission, and computation overhead.

Naturally, performing monitoring logics directly in the origin—the switch data plane itself should be one promising way to achieve fine-granularity, cost efficiency, and real-time monitoring. Fortunately, recent advances on the programmable data plane (PDP) have provided us a new foundation to realize this vision. Programming protocol-Independent packet processors (P4)[8] is a reconfigurable,

Corresponding author:*Wai-xi Liu

ISBN 978-3-903176-39-3© 2021 IFIP

protocol independent and platform independent data plane programming language. P4 makes it possible to reach deep inside the switch to obtain highly accurate network state information.

Range-customizable measurements can be achieved faster and more accurately using INT+PDP [4, 9, 10, 11]. INTOpt [10] assigns probing tasks to suitable flows, but some ports are repeatedly probed. Another INT-based packet-level network monitoring scheme [4] also has several disadvantages. First, its real-time monitoring performance is limited because events are firstly pushed to a database and the network controller then queries the information from it. Second, its monitoring range is limited, and the maximum number of hops is limited to six. In addition to poor real-time performance, excessive overhead, and limited measurement range, INT+PDP-based solutions find it difficult to monitor the global view event, such as throughput of the whole network.

As a new network frame, software-defined networking (SDN) [3] separates the control plane of the network from the data plane, thus enabling flexible control over network traffic. Most importantly, SDN provides a controller that can control the entire network in a centralized manner to obtain a global view.

Thus, we propose an INT+PDP-based fine-grained real-time telemetry scheme by observing and recording on the programmable data plane, and referred as O&R. The key idea is that by using programmability of switch data plane, on which some registers are design to observe the states of packets forwarded by it; moreover, a customized header is added to piggyback on normal data packet to record how does it be forwarded on its routing path. A measurement controller, which is deployed on the SDN controller, reads collected data from the registers to compute the delay, jitter, throughput, packet loss rate, queue state, and route tracing. In summary, the main contributions of this paper are as follows:

(1) An INT+PDP-based fine-grained real-time telemetry scheme through the cooperation of the register on switch and customized header on the packet is proposed. We design a clock offset elimination algorithm to realize the time synchronization of adjacent two switches, based on which we can complete more fine-grained delay and jitter measurement, such as, queuing delay, processing delay, transmission delay, propagation delay on any hop.

(2) A method to measure queue state that includes real-time queue depth and how much flows share the queue is proposed.

(3) Through extensive experimental results for K=4 fat-tree data-center network, we demonstrate the effectiveness of O&R in terms of higher accuracy, better real-time monitoring, and better fine-graining than existing schemes.

II. Related work

Probe packet-based measurement schemes make up the majority of existing network telemetry methods, mainly including traceroute, Swift [9], INTOpt [10], sINT [11], NetSeer [15], OmniMon [16], ML-INT [17], PINT [2], and HyperSight [14].

For congestion control, swift[9] proposed a closed-loop measurement method, which requires the delay information to be transmitted back to the sending host through the CTS response packet, which has a large delay, and this method integrates various delays on the link as a link delay, it can only be found that the link is congested, but the exact location of the congestion cannot be judged. PINT[2] is an INT framework that limits the amount of information added to a packet. OmniMon[16] decomposes network telemetry into partial operations through segmentation, and schedules these partial operations among different entities. But the algorithm tracks every flow in the whole network; at the same time, the controller manages and comprehensively analyzes the whole network resources of all terminal hosts and switches, which will have greater requirements on the operation of switches and controllers, and the cost of equipment update is unrealistic. ML-INT [17] is a flexible multi-layer in band network telemetry system based on P4. In ML-INT, some packets in IP stream are selected to encode INT header, which is biased for accurate real-time measurement and is not conducive to the diagnosis of network performance.

In short, although the network performance can be accurately measured using probe packets, such measurements also have drawbacks, such as the limitation in the time span of the measurement and great difficulty in measuring the whole network. In addition, the header of a probe packet occupies considerable network bandwidth.

Switch-based probing methods mainly include BasisDetect [5], NetPilot [7], and CorrOpt [6]. They record network performance by deploying registers on the switch. The management layer extracts measurement data from the registers as needed. However, such methods also have drawbacks such as an increase in the hardware overhead of the switch upon register addition and latency in measurement data extraction.

III. System design

A. System framework

The first step is to design the P4 program according to the measurement task and deliver it to the P4 switches, during which O&R designs two customized headers (i.e., *loss_t* and *hop_t*). O&R also designs 28 registers for storing measurement data for each P4 switch. As shown in Fig.1, the measurement process is as follows:

(1) The administrator initiates relevant measurement rules via the controller according to the measurement task.

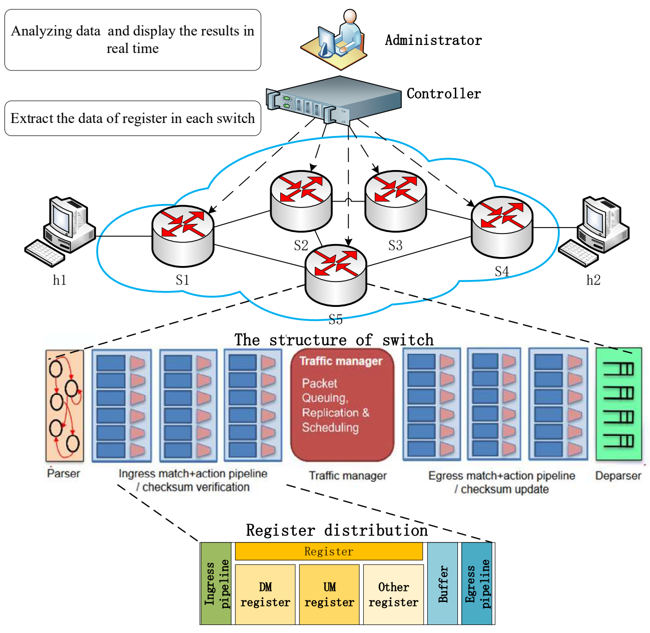
(2) When a packet arrives at the first-hop switch, the ingress pipeline of the switch adds the required customized header (which is used to record the measurement data) to the normal header of this packet according to the measurement task. In addition, the switch stores the measurement data in its corresponding register, and the packet is subsequently forwarded to the next hop along the preset route[18].

(3) When the packet travels through the P4 switch, the switch first parses the packet according to the measurement task. If the packet does not contain the required customized header, then the ingress pipeline of the switch adds the required customized header to the normal header of this packet when receiving this packet. If the packet already has the required customized header, then no header is added. In addition, the switch reads the measurement data recorded in the customized header from the switch on the previous hop and calculates the data with its own measurement data. The calculation result is stored in the register of this switch.

(4) The above procedure repeats until the packet reaches its destination. The final-hop switch removes the added

customized header so that the measurement remains transparent to the sender and the receiver.

(5) Depending on the requirements of the task, the controller reads the data from the required switch registers as well as analyzes and processes such data to obtain the final measurement results for in-band telemetry purposes.



Analyzing data and display the results in real time
Administrator
Controller
Extract the data of register in each switch
h1
S1
S2
S3
S4
S5
h2
The structure of switch
Traffic manager
Packet Queuing, Replication & Scheduling
Parser
Ingress match+action pipeline / checksum verification
Traffic manager
Egress match+action pipeline / checksum update
Deparser
Register distribution
Register
Ingress pipeline
DM register
UM register
Other register
Buffer
Egress pipeline

Fig. 1. Overall system architecture

B. Key measurement schemes

1) Clock offset elimination algorithm

Emerging applications such as interactive augmented reality (AR)/virtual reality(VR) and industrial Internet have higher requirements for deterministic low delay and deterministic low jitter. Clock synchronization between switches is the basis for accurate measurement of delay and jitter, as well as the key to achieving deterministic low latency and deterministic low jitter. The current mainstream time-sensitive networking (TSN) achieves clock synchronization at the link layer, which requires underlying hardware support. We propose a clock offset elimination algorithm to eliminate the clock offset of the switch without any hardware modification.

As shown in Fig.2., a probe packet is sent first, and it carries the start timestamp (*lastswitch_sendtime*) of the data sending by the sender. After receiving the probe packet, the receiver returns a reply packet that carries the receiving timestamp (*thisswitch_rectime*) and sending timestamp (*thisswitch_sendtime*) of the probe packet and the receiving timestamp of the final reply packet being returned to the original switch (*ingress_global_timestamp*). Based on these timestamps, the clock offset of the two switches (△S) is calculated, which is then introduced into the delay calculation so as to achieve clock synchronization. The calculation process are as follows:

$$Delay1 = thisswitch\_rectime - lastswitch\_sendtime + \triangle S \quad (1)$$

$$Delay2 = ingress\_globaltime\_stamp - thisswitch\_sendtime - \triangle S \quad (2)$$

$$Delay1 = Delay2 \quad (3)$$

Based on Eqs. (1), (2), and (3), we can obtain:

$$\triangle S = (ingress\_global\_timestamp - thisswitch\_sendtime - thisswitch\_rectime + lastswitch\_sendtime)/2 \quad (4)$$

In the clock offset elimination algorithm, the switch needs to support loopback forwarding to calculate the loopback time. Therefore, the P4 switch herein has two forwarding behaviors: loopback forwarding and normal IP forwarding. The choice of the forwarding behavior is made by determining whether values have been stored in the switch's synchronous time difference register (*synchronous_diff_time*). If yes, normal IP forwarding is selected; if no, loopback forwarding is selected.

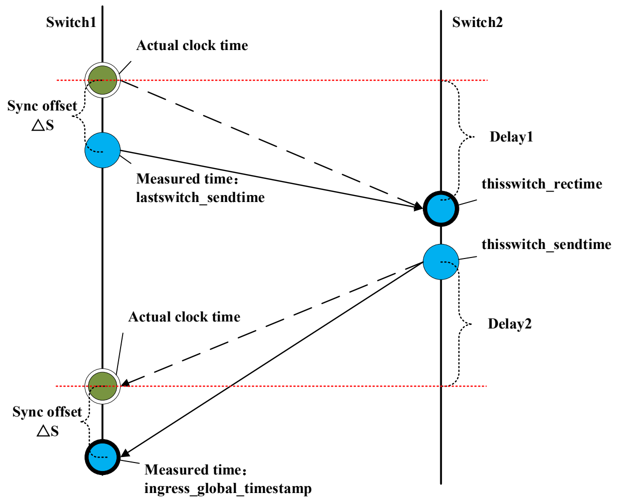


Fig. 2. Clock offset elimination algorithm

As shown in Figure 3, the loopback forwarding steps are as follows:

Step 1: The initial state of the *re_flag* field in the *loss_t* header is set to 2. At the time, normal IP forwarding is performed. The values in the synchronous_diff_time register of the switch are stored into the *last_synchronous_diff* field in the *loss_t* header, and the time when the packet arrives at the switch's egress port is stored in the *thisswitch_sendtime* field in the *loss_t* header. Then, the *re_flag* field in the *loss_t* header is set to 0, and the packet containing this header is sent to the next switch.

Step 2: When the next switch receives the packet in which *re_flag*=0, the switch determines whether *last_synchronous_diff* is 0. If *last_synchronous_diff* is 0, the switch records the timestamp of the packet entering the switch's ingress port in the *thisswitch_sendtime* field in the *loss_t* header, and the timestamp of the packet's arrival at the switch's egress port in the *lastswitch_sendtime* field. Finally, *re_flag* is set to 1, and loopback forwarding is performed.

Step 3: If the value of the *re_flag* field in the packet arriving at the next switch is 1, it means that three timestamps of the packet, namely *thisswitch_sendtime*, *thisswitch_rectime*, and *thisswitch_sendtime* have been recorded. In this case, the switch extracts the *ingress_global_timestamp*. Eq. (4) is used to calculate △S, the synchronization time difference between s1 and s2. Subsequently, *re_flag* is set to 2, and △S is recorded in the synchronous_diff_time register.

The synchronization time difference between the two switches in forward and reverse directions is a pair of opposite numbers. Besides, the register cannot store negative numbers, so we add a synchronous time difference flag, i.e., *last_synchronous_diff_flag*. When the computed synchronous time difference *last_synchronous_diff* is negative, *last_synchronous_diff_flag* is set to 1. *last_synchronous_diff* and *last_synchronous_diff_flag* are stored in the respective locations in the *synchronous_diff_time* and *synchronous_diff_time_flag* registers.

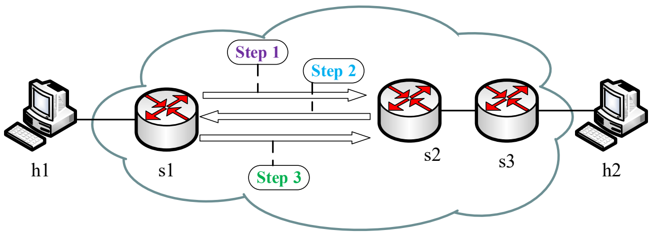


Fig. 3. Schematic of the loopback forwarding

2) Route tracking

To observe the routing paths of packets in the network, we design the route tracing functions, involving three switches—Src router, intermediate router, and sink router. The measurement process is shown in Fig.4.

(1)The controller obtains the IP address of each router port and then sends it to the corresponding router. When h1 sends out a packet, the router receives it and then checks its quintuple. If the port number is 45,000, then src router inserts the *hop_t* header. The receiving port and forwarding port are used as matching ports to obtain the corresponding IP addresses, which are then filled in the header. The packet containing this header is then forwarded.

(2)When the intermediate router receives a packet containing the *hop_t* header, it directly fills in the routing and delay information.

(3)When the sink router receives a packet containing the *hop_t* header and the next hop is the host (i.e., the packet is to be delivered to the destination host h2), the switch extracts the information from the *hop_t* header and temporarily stores it into the register and then removes the header before delivery (as can be seen from the sink router in Fig.4). When the controller completes extracting the switch registers, it clears them.

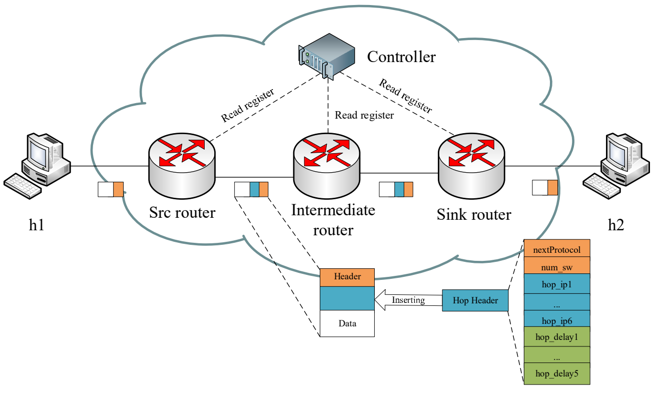


Fig. 4. Route tracking principle

IV. Experiments

A. Experimental setup

We built a network prototype to demonstrate the proposed O&R. The hardware is G4560 CPU and 16-GB memory, running Ubuntu 16.04 OS. We used Mininet to implement the K=4 fat-tree data-center network, as shown in Fig.5, where the default bandwidth each link is 10 Mbps. Specifically, we adopted the P4 software switch (i.e., bmv2) to achieve the capability of forwarding of customized INT packets. The queue comprises 1000 packets. To simulate the communication in a real network, we also generated background traffic in the network. We let each host in the topology send communication requests to other hosts, with any two hosts communicating with each other simultaneously. The communication process is described as follows:

- The communication request process is according to a Poisson process, whereas the source and destination of each flow are selected uniformly at random.. For each host, the parameter τ is independently and randomly generated in (150, 200).
- Some micro-burst flow, congestion, packet loss events are created by setting the bandwidth of a certain link and enabling the host to send packets randomly.
- Flow scheduling mechanism: equal-cost multipath routing (ECMP).

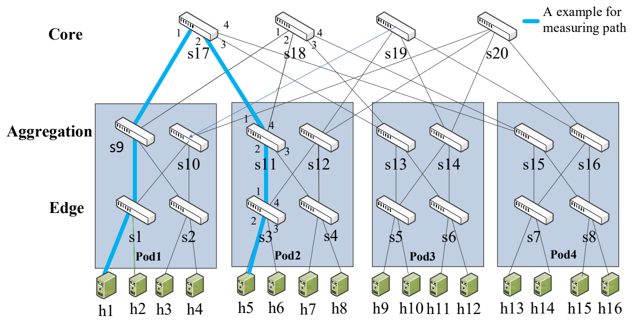


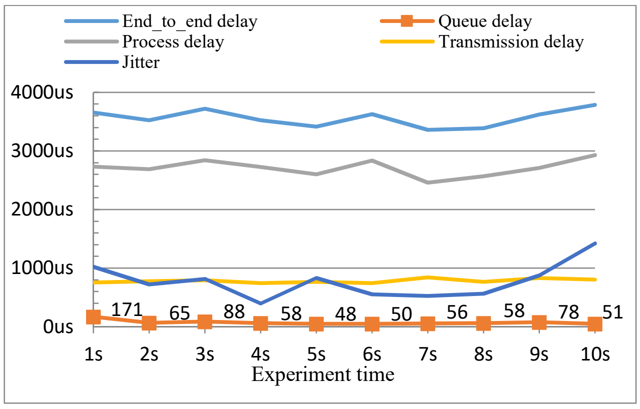
Fig. 5. K=4 fat-tree data-center network

B. Experimental results

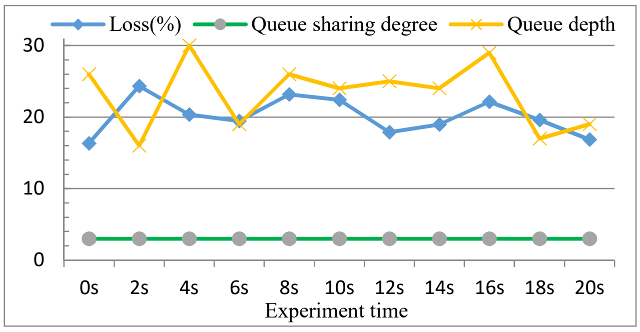
O&R can measure any one-hop link, end-to-end path, or a particular network in the entire network. Fig.6(a) shows the changes in end-to-end delay, queuing delay, transmission delay, processing delay, and jitter over time in 10 s for the link S11-S3.

As shown in Fig.6 (a), end-to-end time delay = processing delay + queuing delay + transmission delay. The processing delay accounts for the majority, which is due to the small queuing delay and transmission delay caused by the large enough bandwidth at the time of measurement. From the ninth to tenth seconds, the end-to-end network delay rises, while jitter changes more dramatically compared to the end-to-end delay. A comprehensive analysis of the changes in end-to-end delay, processing delay, and jitter shows that the dramatic change in jitter may be due to the sudden increase in processing delay of individual packets. The end-to-end delay, processing delay, and jitter are values averaged in a period of time. Therefore, when the processing delay of a packet increases suddenly, a large delay occurs. After this sharp rise, the delay falls to a steady state. In this process, two large jitters occur, so the jitter change is more drastic than the delay change.

Fig. 6(b) shows the real-time change in queue depth, queue sharing degree, and packet loss rate over time on the link S11-S3 within 10 s when there is congestion on the link. Within 1–4 s, the packet loss rate shows a negative correlation with the queue depth. This is because the increase in queue depth during this time leads to further congestion of the network. In this case, the switch has to directly discard a batch of packets to ensure that the queue depth does not exceed the upper limit. This phenomenon does not occur from the fourth to tenth seconds because the packet loss rate and queue depth are relatively stable during the time. The queue-sharing state is always that the switch is simultaneously shared by three queues. This is due to the fact that only three flows are set for measurement.



(a)



(b)

Fig. 6. Network performance measured by O&R

The path tracking experiment was conducted for the path h1->s1->s9->s18->s11->s3->h5. The results of six hops in non-congestion conditions were obtained. Table I lists the time required for each hop and the traveling paths of the packet.

TABLE I. COMPARISON OF ROUTE TRACKING MEASUREMENTS

	Actual path	O&R		Traceroute	
	Link IP	Link IP	Delay per hop	Link IP	Delay per hop
1	h1->s1	h1->s1	688us	10.1.1.1	688us
2	s1->s9	s1->s9	2124us	20.1.9.2	1782us
3	s9->s18	s9->s18	3100us	20.9.17.2	4799us
4	s18->s11	s18->s11	2723us	20.11.17.1	4006us
5	s11->s3	s11->s3	3460us	20.3.11.1	4454us
6	s3->h5	s3->h5	4769us	10.3.5.2	4769us

C. *Performance of measurement schemes*

For the path h1->s1->s9->s18->s11->s3->h5, we compared the performances of different measurement schemes, namely O&R, traditional measurement schemes (ping and traceroute), INT-like (a method similar to the probe packet method of INTOpt [10]), and Pingmesh [12] in terms of the accuracy, real-time performance, and measurement overhead.

1) *Measurement accuracy*

Table II presents the measurement results of end-to-end delay from h1 to h5. O&R witnesses an improvement in measurement accuracy by 39.9%, 45.2%, and 46.3% compared to ping, traceroute, and INT-like. However, the difference between measured values and actual values of O&R is around 1 ms, mainly due to the limited accuracy of the time synchronization algorithm.

The packet loss rate of the h1->h5 path was measured. Considering ECMP is a load-balanced flow scheduling policy, we let s1->s9 and s1->s10 to randomly discard packets, so the packet loss rate is the total packet loss rate of the two links. We set the theoretical packet loss rate of the links to be 20%, 30%, 40%, and 50%. The comparison results of between the measured values and the actual value are shown in Fig.7. In most cases, O&R can achieve near-100% measurement accuracy. Even in cases where the packet loss rate is up to 50%, O&R can achieve near-100% measurement accuracy as well.

TABLE II. COMPARISON OF THE ACCURACY OF DELAY MEASUREMENTS

	Theoretical reference value (ms)	O&R (ms)	Ping (ms)	Traceroute (ms)	INT-like (ms)
1	16.772	17.981	22.1	21.448	21.987
2	13.698	14.989	18.1	20.498	21.777
3	13.437	14.353	21.2	20.498	21.526
4	14.750	15.099	20.6	19.343	21.566
5	12.591	13.024	21.9	25.851	21.585

2) *Real-time measurement performance*

The average time of O&R requiring complete all measurement tasks for a link is 8.5153 ms. During the time, the operation of clearing the registers takes 6 ms due to the limitation by the write speed of the memory in the simulation environment. That is, the measurement time depends mainly on the memory read and write speeds of the P4 switch.

The time required to complete a single measurement task is shown in Table III. O&R is significantly better than the other schemes. As O&R obtains measurement results by parsing the information from the switch statistics, the measurement delay is the time taken to execute the code that extracts and parses the register information. However, all other three schemes proactively send probe packets, and the probe packets eventually need to return to the source host; therefore, the measurement delay is minimized as the time taken to return a probe packet from the destination host to the source host. The route tracking feature of O&R can measure both paths traveled through by a packet and the time incurred by the packet traveling through that path in a short period of time, whereas traditional measurement schemes (traceroute/tracer) take tens of times longer.

V. CONCLUSION

This paper proposes an INT+PDP-based fine-grained and real-time telemetry scheme, observing and recording on programmable data plane. Benefitted by proposed clock offset elimination algorithm, O&R can complete more fine-grained measurement, such as, queuing delay, processing delay, transmission delay, and propagation delay on any hop. O&R can realize the measurement of the queue depth and shared state of queue. Experiment results show that the measurement accuracy of O&R is 46.3% higher than the INT-like method. The measurement delay of O&R is ~1ms when INT-like method needs ~20ms. Furthermore, future studies can be performed to develop the mult-grained telemetry, such as flow level, event level, and behavior level telemetry on programmable data plane.

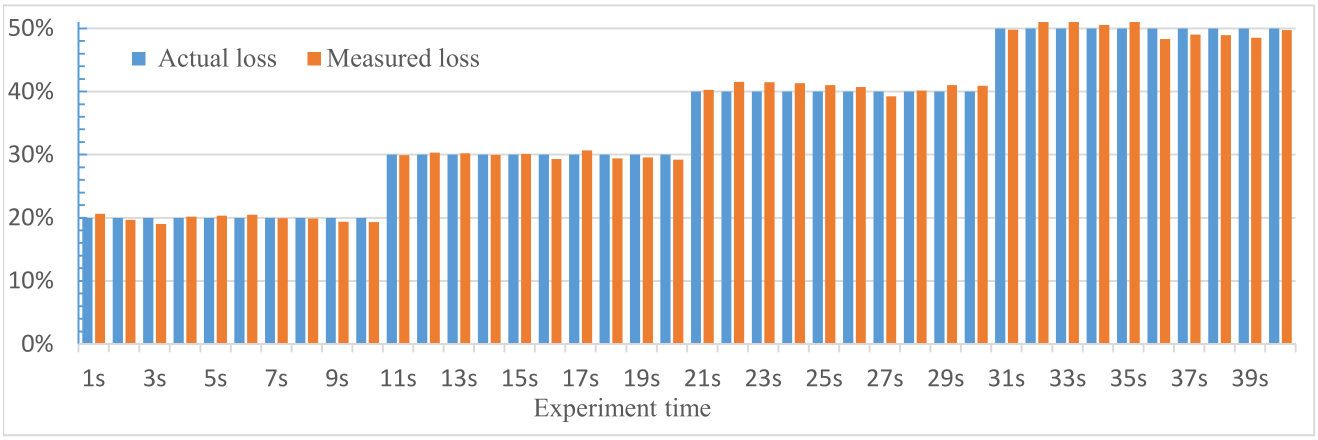


Fig.7. Comparison of packet loss measurements

TABLE III. COMPARISON OF THE REAL-TIME PERFORMANCE OF EACH MEASUREMENT SCHEME

Measurement indicators	O&R (ms)	INT-like (ms)	Pingmesh (ms)	Traceroute (ms)
Transmission delay	0.6184	-	20.78	-
Queuing delay	0.5135	-		-
Processing delay	0.5135	-		-
Synchronous time difference	0.6254	-	-	-
Jitter	1.5660	-	-	-
Packet loss rate	1.1218	-	-	-
Throughput	0.6374	21.69	-	-
Queue depth	0.9468	-	-	-
Queue shareability	1.9525	-	-	-
Route tracking	11.254	-	-	21.53

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (61872102, 61972104, 61802080), the Guangdong Basic and Applied Basic Research Foundation (2021A1515012306, 2018A0303130045), National Key Research and Development Program of China(2018YFB1501201), the International Collaborative Research Program of Guangdong Science and Technology Department(2020A0505100061, 2020A0505100060), the National Social Science Fund of China (15CTQ034), and Project of Guangzhou University (YK2020011), China.

REFERENCES

[1] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In SIGCOMM.

[2] Basat R B , Ramanathan S , Li Y , et al. PINT: Probabilistic In-band Network Telemetry[J]. 2020.

[3] Kirkpatrick K. Software-defined networking[J]. Communications of the ACM, 2013, 56(9): 16-19.

[4] Jonghwan Hyun, Nguyen Van Tu, Jae-Hyoung Yoo, James Won-Ki Hong: Real-time and fine-grained network monitoring using in-band network telemetry. International Journal of Network Management. 29(6) (2019).

[5] Eriksson B , Barford P , Bowden R , et al. BasisDetect: a model-based network event detection framework[C]// Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement 2010, Melbourne, Australia - November 1-3, 2010. ACM, 2010.

[6] Zhuo D, Ghobadi M, Mahajan R, et al. Understanding and mitigating packet corruption in data center networks[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 2017: 362-375.

[7] Wu X , Turner D , Chen C C , et al. NetPilot: Automating Datacenter Network Failure Mitigation[J]. ACM SIGCOMM Computer Communication Review, 2012, 42(4):419-430.

[8] Bosshart P, Daly D, Gibb G, et al. P4: Programming protocol-independent packet processors[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.

[9] Kumar G , Dukkipati N , Jang K , et al. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter[C]// SIGCOMM '20: Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. ACM, 2020.June 2020.

[10] Bhamare D , Kassler A , Vestin J , et al. IntOpt: In-Band Network Telemetry Optimization for NFV Service Chain Monitoring[C]// IEEE International Conference on Communications (ICC2019). IEEE, 2019.

[11] S. Tang, D. Li, "Sel-INT: A Runtime-Programmable Selective In-Band Network Telemetry System," IEEE Transactions on Network and Service Management, 2020, vol. 17, no. 2, pp. 708-721.

[12] C. Guo, L. Yuan, D. Xiang, Y. Dang, et al., 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, London, UK, 2015, pp. 139-152.

[13] Tan L, Su W, Zhang W, et al. In-band network telemetry: A survey[J]. Computer Networks, 2020: 107763.

[14] Zhou Y, Bi J, Yang T, et al. HyperSight: Towards Scalable, High-Coverage, and Dynamic Network Monitoring Queries[J]. IEEE Journal on Selected Areas in Communications, 2020, 38(6): 1147-1160.

[15] Zhou Y , Sun C , Liu H H , et al. Flow Event Telemetry on Programmable Data Plane[C]// SIGCOMM '20: Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. ACM, 2020.

[16] Qun Huang, Haifeng Sun, and Yungang Bao. OmniMon: Re-architecting Network Telemetry with Resource Efficiency and Full Accuracy. In: 2020 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '20, August, 2020.

[17] Bin, Niu, Jiawei, et al. Visualize Your IP-Over-Optical Network in Realtime: A P4-Based Flexible Multilayer In-Band Network Telemetry (ML-INT) System[J]. IEEE Access, 2019, 7:82413-82423.

[18] Liu W, et al. DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks[J]. Journal of Network and Computer Applications, 2021, 177: 102865.