Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

FlowTrace: measuring round-trip time and tracing path in software-defined networking with low communication overhead[*]

Shuo WANG[†‡1,2], Jiao ZHANG[†1], Tao HUANG[1,2,3], Jiang LIU[1,2], Yun-jie LIU[1,3], F. Richard YU[4]

([1]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China*)
([2]*Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory, Shijiazhuang 050081, China*)
([3]*Beijing Advanced Innovation Center for Future Internet Technology, Beijing University of Technology, Beijing 100124, China*)
([4]*Department of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada*)
[†]E-mail: shuowang@bupt.edu.cn; jiaozhang@bupt.edu.cn

Received May 23, 2016; Revision accepted Sept. 14, 2016; Crosschecked Nov. 8, 2016

Abstract: In today's networks, load balancing and priority queues in switches are used to support various quality-of-service (QoS) features and provide preferential treatment to certain types of traffic. Traditionally, network operators use 'traceroute' and 'ping' to troubleshoot load balancing and QoS problems. However, these tools are not supported by the common OpenFlow-based switches in software-defined networking (SDN). In addition, traceroute and ping have potential problems. Because load balancing mechanisms balance flows to different paths, it is impossible for these tools to send a single type of probe packet to find the forwarding paths of flows and measure latencies. Therefore, tracing flows' real forwarding paths is needed before measuring their latencies, and path tracing and latency measurement should be jointly considered. To this end, FlowTrace is proposed to find arbitrary flow paths and measure flow latencies in OpenFlow networks. FlowTrace collects all flow entries and calculates flow paths according to the collected flow entries. However, polling flow entries from switches will induce high overhead in the control plane of SDN. Therefore, a passive flow table collecting method with zero control plane overhead is proposed to address this problem. After finding flows' real forwarding paths, FlowTrace uses a new measurement method to measure the latencies of different flows. Results of experiments conducted in Mininet indicate that FlowTrace can correctly find flow paths and accurately measure the latencies of flows in different priority classes.

Key words: Software-defined networking; Network monitoring; Traceroute
http://dx.doi.org/10.1631/FITEE.1601280 **CLC number:** TP393

1 Introduction

Software-defined networking (SDN) separates the control plane of switches from their data plane (McKeown *et al.*, 2008), and uses a centralized controller to control all switches. Specifically, network control policies running on a controller are translated into low-level flow entries. Therefore,

[‡] Corresponding author

[*] Project supported by the National High-Tech R&D Program (863) of China (No. 2015AA016101), the National Basic Research Program (973) of China (No. 2012CB315801-1), the Beijing Nova Program, China (No. Z151100000315078), and the National Natural Science Foundation of China (No. 61302089)

ORCID: Shuo WANG, http://orcid.org/0000-0002-6350-6362 forwarding behaviors can be explicitly expressed through centralized control policies. This enables operators to efficiently schedule flows and design more flexible quality-of-service (QoS) policies.

To provide QoS to applications, different types of traffic are treated differently in networks. For example, load balancing mechanisms (Al-Fares *et al.*, 2010; Curtis *et al.*, 2011) balance flows to multiple paths to guarantee QoS. Usually, long flows (more than 10 MB) are forwarded to the path with the maximum bandwidth (Al-Fares *et al.*, 2010). In addition, some packet scheduling mechanisms (Alizadeh *et al.*, 2013; Bai *et al.*, 2015) use the priority queues in switches to provide preferential treatment to flows according to their priorities. As a result, flows are balanced to different paths and scheduled in various priority queues.

However, it is hard to find out whether flows meet their QoS demands, when network troubles occur. First, network troubles are caused by updates (Reitblatt *et al.*, 2012; Katta *et al.*, 2013; Perešíni *et al.*, 2013). In SDN, policy updating usually takes several steps to install all the flow entries. Therefore, if flow entries are updated in a wrong order, network troubles will occur. On the other hand, network troubles can be caused by application bugs. A bug occurs when controller applications are unable to change their policies to adapt to different network variations. These variations include network topologies, users' demands, security concerns, and network loads. For example, in Hedera (Al-Fares *et al.*, 2010), when network loads change, Hedera needs to reschedule flows between multiple paths and update flow entries to optimize the network bandwidth utilization. Therefore, if any one of these variations is not handled correctly by applications, network troubles may occur, and the QoS of flows cannot be guaranteed.

To find network troubles and application bugs, many troubleshooting mechanisms have been proposed (Table 1). However, these mechanisms all have some limitations in helping operators find QoS configuration problems.

First, the mechanisms proposed in Wundsam *et al.* (2011), Handigol *et al.* (2012), and Scott *et al.* (2014) are offline debugging tools, and cannot be used to detect troubles in real networks. They can be used only by application programmers to fix controller application bugs when programmers develop applications. For example, OFRewind (Wundsam *et al.*, 2011) and SDN Troubleshooting System (STS) (Scott *et al.*, 2014) enable programmers to record and replay traffic to find troubles. Network Debugger (NDB) (Handigol *et al.*, 2012) is a prototype network debugger, which provides breakpoints and packet back-traces for programmers to rebuild the sequence of events leading to forwarding errors.

Second, some real-time debugging tools, such as VeriFlow (Khurshid *et al.*, 2012) and NetPlumber (Kazemian *et al.*, 2013), are used mainly to automatically check a network configuration's correctness. They can check limited network troubles, such as black holes, loops, and reachability. Thus, they lack the ability to provide operators with flow forwarding paths.

Third, Agarwal *et al.* (2014), Chowdhury *et al.* (2014), Zhang *et al.* (2014), Guo *et al.* (2015), and Yu *et al.* (2015) separately considered path tracing and latency measurement. In fact, existing path tracing tools cannot measure path latencies, and flow measurement tools are blind to the real paths of flows. Therefore, we want to provide network operators with a simple tool to help them find the real forwarding paths of flows and measure flow-based latencies in their paths.

Table 1 Comparison of FlowTrace with several prior mechanisms

Mechanism	Online	Measuring latency	Path visibility	Overhead
OFRewind (Wundsam *et al.*, 2011), NDB (Handigol *et al.*, 2012), and STS (Scott *et al.*, 2014)	No	No	No	NA
VeriFlow (Khurshid *et al.*, 2012) and NetPlumber (Kazemian *et al.*, 2013)	Yes	No	No	High
PathletTracer (Zhang *et al.*, 2014) and SDN traceroute (Agarwal *et al.*, 2014)	Yes	No	Yes	Medium
SLAM (Yu *et al.*, 2015), PingMesh (Guo *et al.*, 2015), and OpenNetMon (Chowdhury *et al.*, 2014)	Yes	Yes	No	Medium
FlowTrace	Yes	Yes	Yes	Low

NA: not available

We introduce FlowTrace, a network path tracing and latency measurement mechanism in SDN. To implement flow-based measurement, FlowTrace combines path tracing and latency measurement by making full use of the benefits of SDN.

To accurately trace a path, FlowTrace needs to monitor all the flow tables (http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf) through a centralized controller. However, monitoring all the flow tables will cause high overhead to the control plane of SDN. Accordingly, we propose a passive flow table collecting method to reduce the control plane overhead. After obtaining all the flow tables, we design a path tracing algorithm with low overhead, which simulates forwarding behaviors of physical switches to find flow paths.

In addition, an accurate latency measurement approach is proposed to measure the latency in the flow level. After obtaining real flow paths, FlowTrace will send crafted probe packets to measure the delays of flows when applications need to find out whether their QoS is guaranteed.

FlowTrace has been implemented in Floodlight controller (https://github.com/floodlight/floodlight) and its performance has been evaluated in Mininet (http://mininet.org/). The evaluation results show that the passive flow table collecting method has good performance and can greatly reduce the control plane overhead. By removing packet processing delays of switches, our path tracing algorithm is faster than SDN traceroute (Agarwal *et al.*, 2014). More importantly, FlowTrace can measure the latency as accurately as ping.

The three contributions of this paper are listed as follows:

1. A passive flow table collecting method with zero control plane overhead is proposed.

2. A path tracing algorithm is presented, which supports network operators in finding arbitrary forwarding paths.

3. A latency measurement method is introduced in OpenFlow-based networks.

2 Related work

Tracing path and measuring latency are two main functions of FlowTrace. There is much related work, and we summarize only some which are closely related to our approach.

2.1 Path tracing in SDN

Path tracing mechanisms in SDN-enabled networks fall into mainly two categories (Agarwal *et al.*, 2014), i.e., model-driven and active probe approaches. Both methods have their advantages and disadvantages. For example, model-driven approaches are faster, but active probe approaches are more accurate.

PathletTracer (Zhang *et al.*, 2014) and SDN traceroute (Agarwal *et al.*, 2014) both use data plane probe packets to find flow paths. PathletTracer reuses fields in the IP header to record path IDs, and looks up flow paths according to path IDs received by the destination hosts. This approach makes sure that PathletTracer finds flow forwarding paths without any ambiguity. However, it needs to know all the paths between source hosts and destination hosts in advance, which makes it hard to apply in real datacenter topologies (Clos, 1953; Al-Fares *et al.*, 2008; Greenberg *et al.*, 2009; Liu *et al.*, 2011; Qi *et al.*, 2014; Ding *et al.*, 2015). This is because these topologies usually contain a lot of different paths, and the number of path IDs needed is not enough for thousands of concurrent flows.

SDN traceroute uses the existing flow entries in switches to trace a path. It sends a probe packet from the controller to a switch, and this probe packet is forwarded to the next hop switch according to the existing flow entries. The controller then records the next hop to determine the flow path. This approach works well when flow paths change frequently. However, in huge networks, it will impose large traffic to the control plane due to the large number of probe packets. In addition, both PathletTracer and SDN traceroute are unable to measure delays and this is important for network operators to find link congestions and QoS problems.

2.2 Delay measurement in SDN

NetFlow (http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html) and sFlow (http://www.sflow.org/about/index.php) use agents on a switch to collect flows. Then agents send information about flows to the collector. After that, the collector analyzes the information to obtain the delays. Because NetFlow and sFlow rely on agents on switches to send information, they need special hardware switches.

Phemius (Phemius and Bouet, 2013) uses probe packets sent by the controller to estimate the latency of each link. It is a simple method, but the accuracy of this method is affected mainly by the period of sending probe packets.

Similar to Phemius, SLAM (Software-defined LAtency Monitor) (Yu *et al.*, 2015) sends probe packets from the controller to estimate latencies. Besides the latency of a single link, it can measure the latency of a flow path. However, it needs to measure the latency between the controller and switches, and this latency has a large influence on the final measurement results.

Jarschel *et al.* (2013) inserted temporary flow rules into switches to send all or a sample of the packets to the controller to calculate the latency. It will consume a large control plane bandwidth, when there are a lot of flows in networks. Although Chowdhury *et al.* (2014) and Su *et al.* (2014) proposed a new polling scheme to reduce the control plane traffic, it still needs to poll flow entries periodically. Yu C *et al.* (2013) and Yu M *et al.* (2013) designed new measurement tools that are used to measure the bandwidth instead of latency.

3 Objectives

Today, the majority of network operators use 'traceroute' and 'ping' to find network bugs. By displaying flow paths on a terminal, traceroute provides network operators and end-users with an easy way to determine whether packets are forwarded correctly. However, traceroute cannot work in OpenFlow-based networks (McKeown *et al.*, 2008). In traditional networks, traceroute sends a sequence of packets to a destination host; then each router decrements the time-to-live (TTL) value; finally, routers return ICMP error messages (ICMP Time Exceeded) when TTL reaches zero (https://tools.ietf.org/html/rfc792). However, switches using the OpenFlow protocol (http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf) cannot return the ICMP error messages when the TTL value reaches zero, because OpenFlow-based switches can apply actions only according to the OpenFlow action list.

On the other hand, traceroute has some potential problems, and it may display wrong flow paths that do not match the real paths of flows when network policies are complex. For example, in Fig. 1a, there are two paths between the source host (H1) and the destination host (H2), and packets of different flows are randomly balanced between the two paths by S1. As a result, ICMP packets sent by traceroute and the packets sent by the flow are most likely to be balanced into two different paths, and traceroute cannot trace the real path of the flow. Since the SDN controller has all the forwarding entries, this problem can be easily solved in an SDN-based network. For example, in Fig. 1b, the SDN controller installs flow entries for all the flows, and then it can directly and easily find the path of the flow by analyzing flow entries without any confusion.

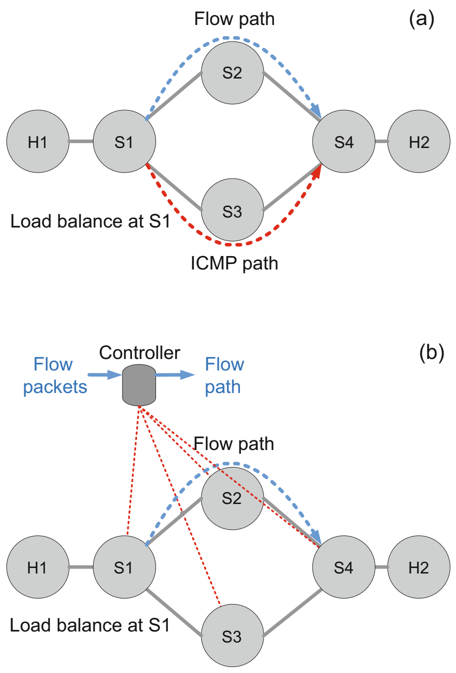

Fig. 1 Switches using equal-cost multi-path (ECMP) routing to balance traffic to multiple links: (a) ICMP packets being balanced to a different path; (b) tracing path using the SDN approach

4 Design

We elaborate the architecture of FlowTrace in Fig. 2. There are three modules in FlowTrace: a collector, a path calculator, and a latency monitor. The collector collects flow entries and then builds the virtual flow tables. Thus, the virtual flow tables have the same flow entries of physical switches. According to the virtual flow tables, the path calculator simulates the forwarding behaviors of switches to calculate paths. After knowing flow paths, the latency monitor will insert temporary flow entries along the flow paths when users need to measure the latencies of flows.

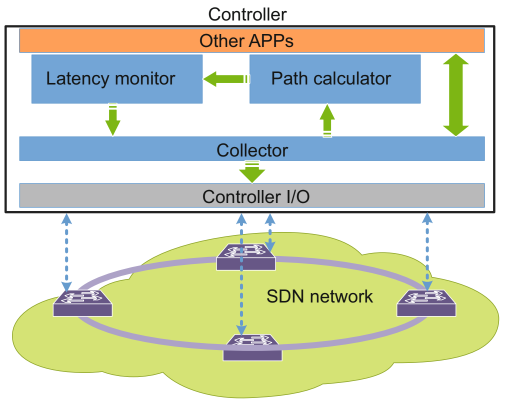

Fig. 2 FlowTrace architecture

4.1 Collector

To collect flow entries, a simple idea is that the controller periodically polls flow entries from switches and updates the virtual flow tables. However, this simple method will induce large traffic to the control plane and increase overhead. In addition, if the updating rate of flow entries is larger than the polling rate, virtual flow tables will no longer have the same entries as the flow tables in switches. Instead of using the polling method, the FlowTrace collector uses a passive collecting method to collect flow tables with zero cost, and the virtual flow tables can be updated as soon as flow entries change.

In our design, the collector is a layer between the controller I/O (input/output) and upper layers, and it monitors the sending of FLOW_MOD and the receiving of FLOW_REMOVED messages. We fully use the fact that the controller controls switch flow tables, and switches are unable to add any flow entry without the controller's instructions. Flow tables can be modified in two situations. First, the controller sends FLOW_MOD messages to modify flow tables, such as adding or deleting entries. By monitoring every message sent from the controller, the collector knows which entry is added or modified. Second, switches send FLOW_REMOVED messages to notify the controller which flow entry is removed. In the standard OpenFlow protocol, there is an optional bit (OFPF_SEND_FLOW_REM) in a FLOW_MOD message. When the OFPF_SEND_FLOW_REM bit is set, the switch must send a FLOW_REMOVED message to the controller when a flow entry is removed. Thus, the collector rewrites FLOW_MOD messages to set this bit, and monitors FLOW_REMOVED messages to determine which flow is removed.

4.2 Path calculator

Inspired by Kazemian *et al.* (2013), the path calculator imitates physical switches to obtain the tracing path. When network operators query a flow path from FlowTrace, they need to specify the packet header fields of the flow that they want to trace and identify the flow's injection switch and port. Then the path calculator looks up which flow entry matches the operator's requests in the virtual flow table of the first attachment switch, and simulates the forwarding process of switches according to the actions of the matched entries.

Before introducing how to calculate a flow's forwarding path, we show that there are some potential relations between the flow entries of adjacent switches along a packet's forwarding path, and our algorithm leverages these relations. For instance, Fig. 3 shows a network with four switches. A flow entry in OpenFlow usually has several match fields and an action list. We use 'X-X-X-X-X-X' to represent the match fields of an entry. As we know an entry's match fields and its action list, we can determine the header fields of egress packets. In our algorithm, we use the red and green packets, shown in Fig. 3, to denote the header fields of egress packets

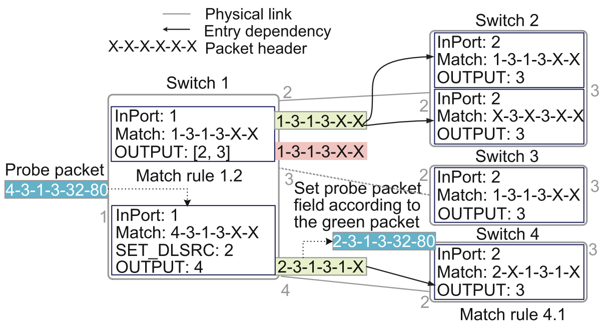

Fig. 3 Relationships between switch 1 and the other three switches. Arrows represent entries' relations; dotted lines between switches represent link failures; dotted arrows represent the processing of a probe packet. Red and green packets represent the header fields of egress packets. References to color refer to the online version of this figure

(also called 'color packets'). Egress packets are calculated by applying a flow entry's actions to a flow entry's match fields. We define that two entries have a relation only if the intersection (Kazemian *et al.*, 2013) between the first entry's color packets and the second entry's match fields is not empty. For example, there is a relation between entry 1 in flow table 1 (rule 1.2) and entry 1 in flow table 4 (rule 4.1), because their intersection (2-3-1-3-1-X) is not empty.

Algorithm 1 shows how FlowTrace finds the flow path of a flow. The collector first builds flow relations. Then the path calculator finds flow paths according to flow entry relations. Each entry in our algorithm has two color packet lists (i.e., an egress packet list and a blocked packet list). The egress packet list recodes the green packets whose egress links work well, while the blocked packet list recodes red packets whose egress links have failures. First, we obtain the first attachment switch of the probePacket (defined by a user's request) and find the highest priority entry which matches probePacket (lines 1–5). Second, we put the matched entry and probePacket in a matching entry search-queue (line 6). Third, we look up entries in the search-queue, and obtain a nextHopPacket (lines 15–17). If entries in the search-queue have relations with entries in the next hop switches, we match nextHopPacket with those entries (lines 18–20). Finally, we can know all the entries that match the probePacket, thus building the flow path.

Algorithm 1 Calculating path

```
1: procedure CalculatePath(probPacket)
2: Entries ← table of the first attachment switch
3: sort(Entries)
4: for entry in Entries do
5:     if entry matches probePacket then
6:         searchqueue.put(entry, probPacket)
7:         break
8:     end if
9: end for
10: while searchqueue $\neq \varnothing$ do
11:     (matchedEntry, packet) = searchqueue.pop()
12:     if packet has been searched before in the same InPort then
13:         continue
14:     end if
15:     for $p$ in matchedEntry.EnabledOutPackets do
16:         nextHopPacket = $p$ & packet
17:         sort($p$.nextHopEntries)
18:         for entry in $p$.nextHopEntries do
19:             if entry matches nextHopPacket then
20:                 searchqueue.put(entry, nextHopPacket)
21:                 break
22:             end if
23:         end for
24:     end for
25: end while
26: BuildPathAccordingMatchedEntries()
27: end procedure
```

More specifically, in Fig. 3, a probe packet (4-3-1-3-32-80) is sent to port 1 of switch 1, and we find it matches rule 1.2. One action of rule 1.2 is to output a packet to port 4. Now, we just replace the header fields of the probe packet with green packet's header fields of rule 1.2 and obtain a nextHopPacket (2-3-1-3-32-80). Instead of comparing the nextHopPacket with all entries in switch 4, we compare it only with rule 4.1 which has relations with rule 1.2.

4.3 Latency monitor

To measure delays along a flow path, FlowTrace inserts each switch with one default flow entry, and inserts some temporary flow entries when needed. The first entry (rule 1 in Table 2) is the default entry of each switch. Each default entry will swap the source MAC, IP address with the destination MAC, and the IP address of packets, and output packets to their ingress port. F represents a field, such as VLAN (http://www.ieee802.org/1/pages/802.1Q.html) and TOS (https://tools.ietf.org/html/rfc1349), in the packet header. We use TOS in our implementation. FlowTrace will assign each switch a unique F value. The temporary entries are also shown in Table 2. These

Table 2 Flow tables of three switches connecting in line (left: switch 1; right: switch 2)

Rule	Priority	Entries	Rule	Priority	Entries
1	65 536	Match: F=1; swap: src and dst; output: InPort	1	65 536	Match: F=2; swap: src and dst; output: InPort
2	2	Match: a to b; output: 2	2	2	Match: a to b; output: 2
3	2	Match: b to a; output: 1	3	2	Match: b to a; output: 1

entries are used to forward probe packets.

We use Table 2 to illustrate the steps to measure latencies between hosts A and B. We first send a probe packet from host A whose F field equals the F value of S1. The probe packet carries a sending timestamp in its payload, and then it will reach S1 and match the default entry (rule 1 of S1). This entry sets our probe packet's source as B, sets its destination as A, and outputs it from the ingress port. Then the probe packet will return to host A, and we calculate the round-trip time (RTT) by reading the timestamp from the payload. After the first round measurement, we send a new probe packet with a new timestamp, and the F filed of this new probe packet is set to the F value of S2. The new probe packet is forwarded by S1 according to the second entry (rule 2 of S1). Then this packet reaches S2, and it will also match the default entry and return. In this way, we can obtain each hop delay round by round.

Note that temporary entries are not always needed. If we want to measure an active flow that is sending data, these flow-related forwarding entries can be used as our temporary entries, and we do not have to install any other entry. Therefore, when users are going to measure the latencies of active flows, FlowTrace does not need to install any additional flow entry.

To achieve flow-level latency measurement, except for the F field, the probe packets have the same packet header fields as the flows that we want to measure. The F field is set to zero for normal packets, and is set to a none-zero value for probe packets. Therefore, the probe packets will be treated as normal packets by switches and experience the same delays when queuing. This ensures that FlowTrace has a high measurement accuracy.

5 Analysis

We now present an in-principle approach to analyze the performance of FlowTrace and show how to improve its performance according to our model. If we use a queuing model, flows arrive at a switch according to a Poisson process of λ, having living time $T \sim F(t)$ ($F(t)$ is the cumulative distribution function (CDF)), $E(t) = \int_0^\infty t\mathrm{d}F(t) = 1/\mu$, $\mathrm{Var}(t) = 1/\sigma$, and the load $\rho = \lambda/\mu$. Thus, the average number of flows in networks is

$$\widetilde{l} = \rho + \frac{\rho^2 + \lambda^2\sigma^2}{2(1-\sigma)}. \quad (1)$$

5.1 Control plane traffic

Periodical polling (Chowdhury *et al.*, 2014) is a popular method for collecting flow entries. Compared to the periodical polling method, FlowTrace imposes zero control plane traffic, because FlowTrace monitors only the messages between the controller and switches and does not generate any traffic.

We can assume that there are $\widetilde{l}$ active entries in one switch and that there are n switches. Let l_{reply} denote the length of the reply message of a single flow entry. For a switch with $\widetilde{l}$ entries, the total traffic n_{traffic} generated by replying to a request is

$$n_{\text{traffic}} = \widetilde{l} \cdot l_{\text{reply}} \cdot n. \quad (2)$$

Therefore, given a network with n switches, the traffic for replying to a request is a quadratic function of λ. In a real environment, a datacenter with 6000 servers and 300 ToRs (top of rack switchs, each ToR has approximately 120 flow entries) (Tavakoli *et al.*, 2009), if the polling interval is 0.1 s, it will produce a traffic rate of 30 MB/s.

5.2 Overhead of collector

The collector needs to maintain virtual flow tables and build flow relations for the path calculator. When a new flow entry of a switch is added or deleted, the collector will search all adjacent switches to build relations. Let s_{adj} denote the number of adjacent switches of one switch. Then the newly added flow entry needs to compare with $s_{\text{adj}} \cdot \widetilde{l}$ flow entries to build relations. If the controller installs p entries along a flow forwarding path and one 'comparison' operation takes t_{cost} to complete, the total overhead of flow table updating will be

$$\text{update}_{\text{once}} = s_{\text{adj}} \cdot \widetilde{l} \cdot p \cdot t_{\text{cost}}. \quad (3)$$

If the collector updates the flow table as soon as a new flow arrives, the average overhead is $\lambda \cdot \text{update}_{\text{once}}$. Actually, the collector does not need to update flow tables when a flow entry is added. Because operators do not trace every flow path, the flow table updating process can be delayed to the time when a request occurs. As shown in Algorithm 2, the collector uses a queue to store messages. When a new Flow_MOD message arrives, the controller puts the message in the queue; when a FLOW_REMOVED message arrives, the controller just needs to remove the related FLOW_MOD message in the message queue.

We assume that the average request interval is δ_r, and δ_r is larger than some flows' duration. Therefore, during the interval, there will be several flows finished, and we do not handle messages related to these flows.

For example, in Fig. 4, between request 1 and request 2, three new flows arrive. Flow 1 and flow 2 finish before request 2 arrives. Therefore, when request 2 arrives, we have to handle messages only related to flow 3.

Now, as we have to handle only $(1 - F(\delta_\mathrm{r})) \cdot \lambda$ flows, the overhead of the collector will be

$$\mathrm{update}_\mathrm{delayed} = (1 - F(\delta_\mathrm{r})) \cdot \lambda \cdot \mathrm{update}_\mathrm{once}. \quad (4)$$

Therefore, if a user's request rate is very small, the overhead of the collector is near zero.

Algorithm 2 Updating flow entries

```
 1: procedure MessageMonitor(message)
 2: if message is FLOW_MOD then
 3:     messagequeue.put(message.match, message)
 4: end if
 5: if message is FLOW_REMOVED then
 6:     messagequeue.remove(message.match)
 7: end if
 8: end procedure
 9: procedure Update // called when there is a request
10: for message in messagequeue do
11:     BuildRelation(message)
12: end for
13: end procedure
```

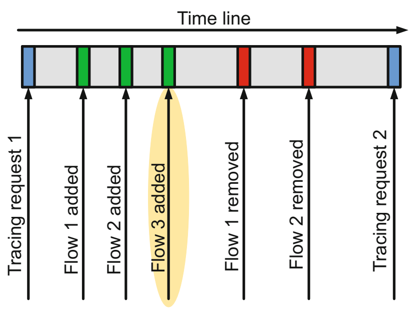

Fig. 4 Three flows arriving between two tracing path requests

5.3 Overhead of measurement

The latency monitor will insert in each switch a default flow entry to swap the source and destination of packets. The number of temporary entries depends on the users' requests. We also assume that the average request interval of measurement is δ_r. Thus, there are $1/\delta_\mathrm{r}$ measurement requests per second. If all measurement requests want to measure the latency of active flows, we will not install any temporary entry, and the number of measurement entries is n. If all measurement requests want to measure the latency of inactive flows, the upper bound of the number of measurement entries is $n + 2p/\delta_\mathrm{r}$. Therefore, the number of measurement entries is

$$n \leq \mathrm{entry}_\mathrm{measurement} \leq n + 2 \cdot \frac{p}{\delta_\mathrm{r}}. \quad (5)$$

However, network operators usually measure the latency of active flows. Thus, the number of measurement flow entries in networks is very close to n most of the time.

The payload length of probe packets is 28 bytes, the IP and TCP header length is usually 20 bytes, and the total length of a probe packet is under 100 bytes. If the measurement interval is r_m s, the maximum measurement traffic is $2(1/r_\mathrm{m}) \cdot p \cdot 100 = 200p/r_\mathrm{m}$ (bytes/s) for a flow with p hops. Therefore, the traffic of measurement is

$$\mathrm{traffic}_\mathrm{measurement} = \frac{200p}{\delta_\mathrm{r} \cdot r_\mathrm{m}} \text{ (bytes/s)}. \quad (6)$$

6 Evaluation

In this section, we evaluate the performance and functionality of FlowTrace in Mininet (http://mininet.org/). FlowTrace is implemented as a module for the Floodlight controller (http://www.projectfloodlight.org/). First, we show the performance of the path tracing function. Second, we evaluate the measurement accuracy of the latency monitor. Finally, we use two simple scenarios to show how path tracing helps improve the measurement accuracy.

6.1 Evaluation of path tracing

6.1.1 Control plane overhead

To assess how FlowTrace reduces the control plane overhead, we compare the passive collecting scheme with periodical polling methods. In this experiment, the periodical polling methods collect all the flow tables per second. Our passive collecting method only passively monitors the modifications of flow entries. We build a tree topology (depath=3, fanout=4) with 21 switches. First, we let the controller install some flow entries to each switch, and the number of flow entries in each switch varies from 100 to 900. Then we record the traffic generated in the control plane during polling these flow entries. Thus, the main metric is the traffic that is induced when the controller collects all the flow entries from switches. In addition, we show the traffic of periodical polling if there is only one switch in the network.

As shown in Fig. 5, the periodical polling method (the third bar) produces a large control plane traffic that is nearly linearly related to the number of flow entries (about 13 000 kb/s for 21 switches, 700 bits/s per flow entry). In contrast, the passive collecting mechanism maintains an average low throughput (35 kb/s) which is too small to see in the figure. Therefore, the periodical polling method which consumes more control plane bandwidth will increase the overhead of the controller (Phemius and Thales, 2013) when it is deployed in datacenter networks, campus networks, or other large-scale networks. In contrast, our passive collecting algorithm has no relation to the number of flow entries and produces near-zero control plane traffic.

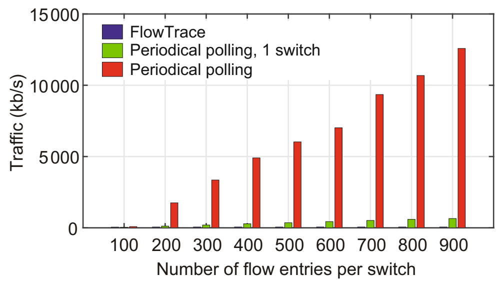

Fig. 5 Control plane traffic by varying the number of flow entries in each switch

6.1.2 Average lookup latency

In this experiment, we show the response time of the path tracing function. We compare FlowTrace with linear search and SDN traceroute (Agarwal *et al.*, 2014) in a linear topology with five switches. FlowTrace uses our proposed relation lookup algorithm, while linear search looks up flow entries one by one. SDN traceroute is implemented in Floodlight, and it sends probe packets to trace flow paths. We generate five flows, and each flow is forwarded by a different number of switches varying from 1 to 5. Then we record the time spent on obtaining the flow paths of five flows. In addition, we insert 1000 flow entries per switch to evaluate the performance of the relational table lookup algorithm when there are many flow entries.

Fig. 6 shows the average response latencies of obtaining a network path by varying the path length from one to five hops. The results show that the response latency of FlowTrace is almost a standard line (about 0.7 ms), and the response latency of the SDN traceroute is nearly linearly related to the number of switch hops, which increases approximately 2 ms per hop. However, different from our initial idea, the relational lookup algorithm has a similar performance to the linear search algorithm when the network size is small. This is because the modern Java program can process thousands of loops in less than 1 ms. We also find that the load of the controller influences the performance of FlowTrace. For example, a high load may increase the response latency to 4 ms. We find a way to improve the performance of lookup, which may have a good performance in a real network consisting of thousands of switches.

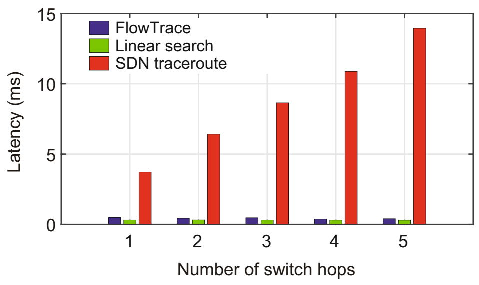

Fig. 6 Response latencies of different methods by varying the number of switch hops

6.2 Evaluation of latency monitor

In this subsection, we evaluate the measurement accuracy of FlowTrace, and compare FlowTrace with the following schemes:

1. ping (the default ping program in the Linux system): To let switches return ICMP messages, we attach a host to each switch, and let the host return ICMP for its attached switch. Therefore, we can use ping to measure link delays in SDN.

2. SLAM: we implement SLAM (Yu *et al.*, 2015) in a Floodlight controller, and SLAM measures packet latencies by sending probe packets from the controller.

6.2.1 Impact of link delays

To evaluate how the accuracy of FlowTrace is influenced by the delays of links, we measure packet latencies with three levels of delays (low, medium, and high). For low-level delays, we use a 1-Gb/s link between two switches with zero traffic, and we measure the latencies using different mechanisms. For medium-level delays, we limit the link bandwidth to 100 Mb/s, and generate a 100-Mb/s flow in the link. For high-level delays, we limit the link bandwidth to 10 Mb/s, and generate a 10-Mb/s flow in the link. Each simulation lasts 300 s. We measure the latency at 0.5 s intervals from the flow start time to its finish time, and plot measurement results against time. In addition, we plot the latency CDF of different mechanisms.

The evaluation results of the link with low-level delays are shown in Fig. 7. Compared to the ping scheme, FlowTrace has almost the same measurement results. In Fig. 7a, the measured latencies of FlowTrace are very close to those of ping. However, SLAM has about 0.5 ms more latency than ping. This is caused mainly by the latency between the controller and switches. In fact, when the link delay is small, the latency between the controller and switches is greater than the link delay and dominates the result. In Fig. 7b, we can see that FlowTrace and ping are very stable and that they have no tails in the CDF, while SLAM has a small tail. This also indicates that the control plane latency has a great impact on the measurement results when the link delay is small.

Fig. 8 shows the evaluation results of the link with medium-level delays. First, from Fig. 8a, we can see that the measured latencies of all schemes have about 5 ms variation at each time point. For example, at 100 s, the latency varies from 5 ms to 9 ms. This indicates that the flow rate of the generated flow is very unstable during its lifetime in the bandwidth limited link. Second, the value of latency is nearly linearly proportional to time in Fig. 8a. This is because the queue length increases with time, when the flow rate is a little greater than the bandwidth. Third, compared to Fig. 7, SLAM has better performance. This is because compared to the flow latency, the control plane latency is marginal and can be ignored in this experiment. Finally, all schemes have very long tails in Fig. 8b. However, FlowTrace still has a similar accuracy to ping.

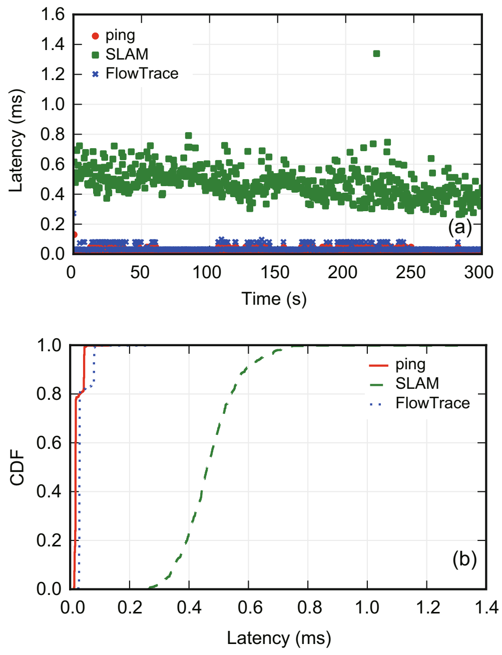

Fig. 7 Latencies in the low-level delay link (a) and the corresponding CDF (b)

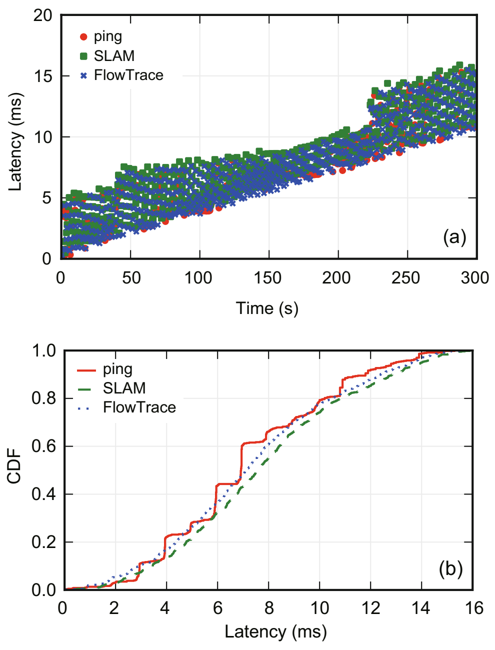

Fig. 8 Latencies in the medium-level delay link (a) and the corresponding CDF (b)

Fig. 9 shows the evaluation results of the link with high-level delays. In Fig. 9a, the latencies of all schemes are about 110 ms at 300 s, and the slope of the curve is greater than that in Fig. 8a. This is because the bandwidth is smaller in this experiment and thus the queue length increases more quickly. Also, the three schemes have similar performance. However, there is one outstanding point at about 260 s in the SLAM scheme. This is caused mainly by the delay variation of the control plane.

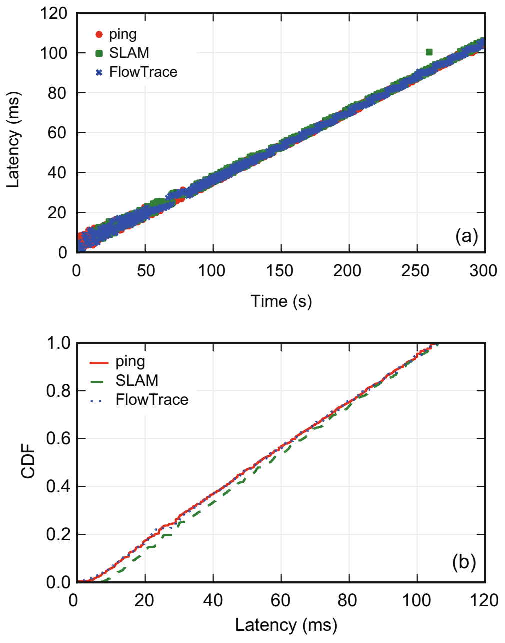

Fig. 9 Latencies in the high-level delay link (a) and the corresponding CDF (b)

6.2.2 Impact of traffic variation

We evaluate the performance of FlowTrace when traffic changes in the network. We generate five flows at 5 s intervals at the same link with 1-Gb/s bandwidth, and each flow continues for 30 s without rate limitation. The measurement results are shown in Fig. 10. The latency of the link begins to increase after 10 s, and stops increasing at about 30 s. This is because multiple flows complete bandwidth at the same link. After 30 s, the latency begins to decrease at 5 s intervals due to the finishing of flows. During this process, we can see that the curve of FlowTrace matches the curve of ping very well. This indicates that FlowTrace can measure the latency variations of flows very well.

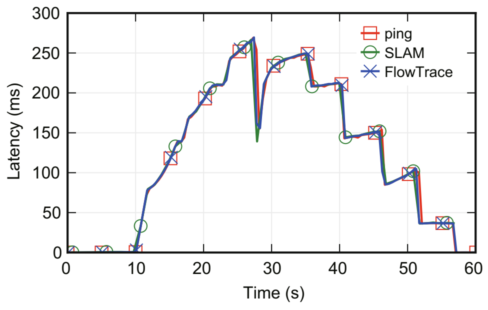

Fig. 10 Five flows generated at 5 s intervals at the same link

6.2.3 Impact of control plane traffic

We repeat the previous traffic variation experiment but induce traffic to the control plane. Our control plane bandwidth is 1 Gb/s, and we generate a flow with 20-Mb/s traffic between the controller and switches. Fig. 11 shows the measurement results. Compared to Fig. 10, the measurement results of SLAM match poorly those of ping and FlowTrace from 0 to 10 s. In addition, the measurement results have some deviations at the point where the flow finishes, such as 35, 45, and 55 s. Because SLAM directly sends measurement probes from the controller, the control plane traffic has more influence on the accuracy of SLAM. In contrast, the measurement results of FlowTrace match very well those of ping, and FlowTrace can accurately measure the latency.

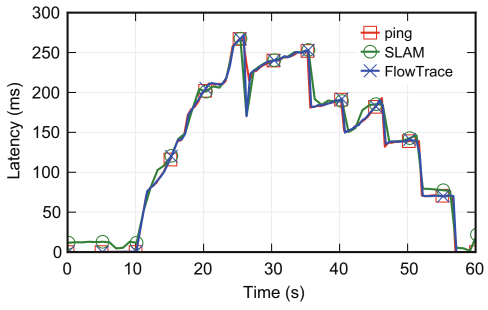

Fig. 11 The control plane having 20-Mb/s traffic when five flows are generated at 5 s intervals at the same link

6.2.4 Impact of throughput

In this experiment, we generate a flow with various throughputs and the boxplot of measurement results is shown in Fig. 12. With the increase of throughput, the variation of latency becomes large. FlowTrace has a similar average value to ping, while the average value of SLAM is a little larger than that of ping. This small difference is induced mainly by the control plane delay.

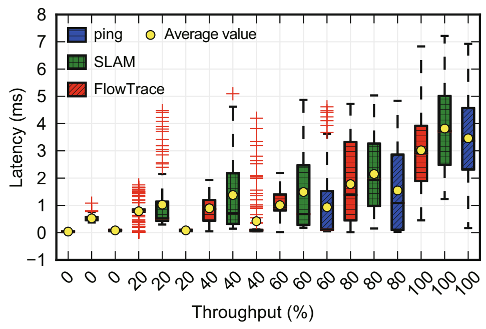

Fig. 12 Latency measurement results by varying throughput

6.2.5 Impact of hop

We build a linear topology with 10 switches, and set the link delay to 1 s. Fig. 13 shows the measurement results when varying the number of switch hops that the flow passes. We can see that the latency is increased by almost 1 ms per hop. The measurement results of FlowTrace are the same as those of ping, while SLAM has an additional 0.2 ms latency compared to ping. This indicates that FlowTrace has very good measurement accuracy when the flows pass through many switches.

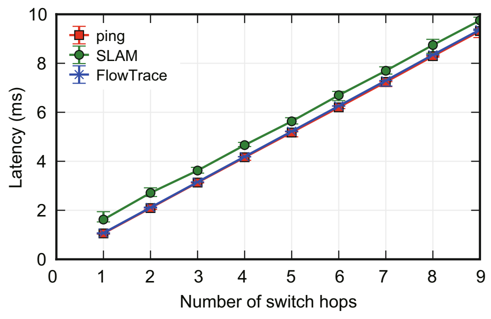

Fig. 13 Latency measurement results by varying the number of switch hops

6.2.6 Measurement overhead

In this experiment, to show that FlowTrace incurs zero measurement traffic on the control plane, we simultaneously measure the latencies of different numbers of flows. The number of flows varies from 1 to 10 000, and the results are shown in Fig. 14. In Fig. 14a, FlowTrace has the best performance, and the average rate of the measurement traffic on the control plane is less than 1 kb/s. Note that the average rate of the measurement traffic of SLAM is about 4.2 Mb/s for 10 000 flows. Therefore, compared to SLAM, when there are a large number of measurement tasks, FlowTrace greatly reduces the measurement cost and has very low overhead. In Fig. 14b, all schemes have similar performance, and the small difference in performance is caused mainly by the difference of probe packet size. In addition, as the number of flows increases, the rate of the measurement traffic on the data plane increases.

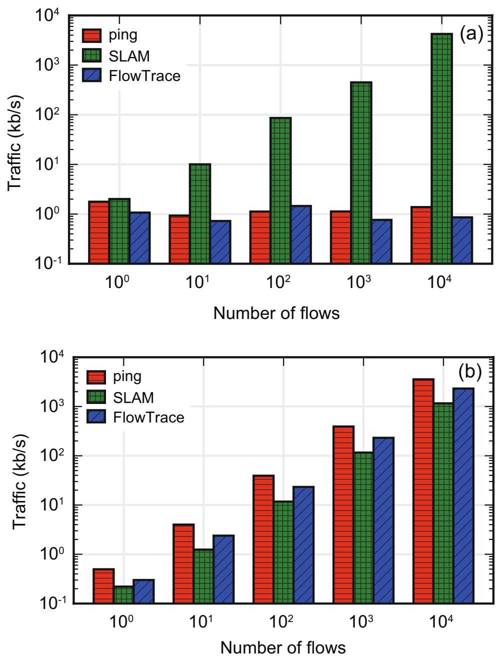

Fig. 14 The control plane traffic (a) and data plane traffic (b)

6.3 User cases

In this subsection we use two simple scenarios to show how the path tracing function helps improve the measurement performance.

6.3.1 Impact of load balancing

Load balancing is used in most datacenter networks to provide large bandwidth for applications. Therefore, two flows are usually forwarded in different paths even if they start from the same source and are going to the same destination. In this experiment, we build a topology that has two hosts and two switches. Each switch is connected to one host, and the two switches are connected by two links with 0 and 1 ms delays, respectively. At the beginning, we generate a flow that is forwarded in link 1 with 1 ms delay. After detecting the congestion, the controller will balance the flow to another link with 0 ms delay.

We plot the measurement results in Fig. 15. We can see that ping, SLAM, and FlowTrace successfully measure the 1 ms delay on link 1 before 10 s. However, when the controller balances the flow to another link after 10 s, the measurement results of ping and SLAM are still at about 1 ms, and only FlowTrace gets about 0.1 ms latency of the flow on the new path. This is because ping and SLAM do not know the real flow path and send only one type of probe packet.

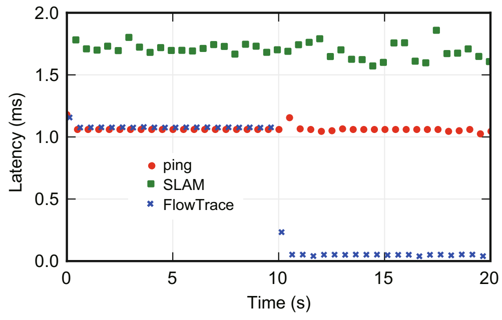

Fig. 15 Measurement results in two paths when the flow is balanced at 10 s

6.3.2 Impact of queueing

Priority queues are used to guarantee the bandwidth of flows with different priorities. Usually, short flows have deadlines, such as web search (Alizadeh *et al.*, 2013), and thus these flows have a higher priority than long flows. In this experiment, we generate two flows with rates of 1 Gb/s and 200 Mb/s respectively, and let them pass through the same link. The link has two priority queues: Q0 and Q1. We limit the upper rate of each queue to 500 Mb/s, and the two flows are forwarded in the two queues.

The measurement results are shown in Fig. 16. First, we can see that the latency of Q1 is about 140 ms and that the latency of Q0 is about 0.1 ms. This is because the rate of the first flow is 1 Gb/s, which is greater than the maximum queue rate, while the rate of the second flow is less than the rate of Q0. Second, ping and SLAM can measure only the latency of one queue, because they send only one type of probe packets and Q1 is the default queue for all the flows. FlowTrace sends two different packets to measure flow-level latencies, which can accurately measure the latencies in different queues.

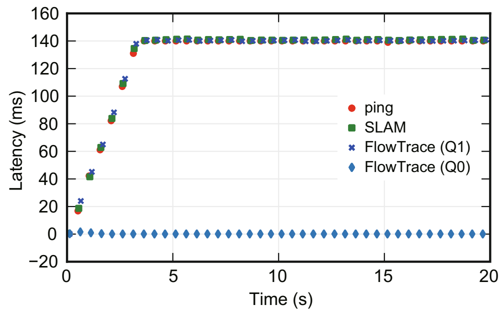

Fig. 16 Measurement results in two queues

7 Conclusions

We have introduced FlowTrace as a real-time network path tracing and latency measurement tool for SDN. Unlike earlier works that use data plane probe packets to obtain a path, FlowTrace calculates a path with zero cost and has a shorter response time by using a relational table query algorithm. By installing temporary measurement rules in switches, FlowTrace enables users to measure their packets' real-time transit delays. Furthermore, FlowTrace uses the OpenFlow protocol, which means that FlowTrace can be directly deployed in a real network without modifying physical switches. Our fundamental design idea is to provide network operators a handy path tracing and measurement tool to enable them to more conveniently manage the network. This design principle is also the purpose of the SDN technology.

References

Agarwal, K., Rozner, E., Dixon, C., *et al.*, 2014. SDN traceroute: tracing SDN forwarding without changing network behavior. Proc. 3rd Workshop on Hot Topics in Software Defined Networking, p.145-150. http://dx.doi.org/10.1145/2620728.2620756

Al-Fares, M., Loukissas, A., Vahdat, A., 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.*, **38**(4):63-74. http://dx.doi.org/10.1145/1402958.1402967

Al-Fares, M., Radhakrishnan, S., Raghavan, B., *et al.*, 2010. Hedera: dynamic flow scheduling for data center networks. Proc. 7th USENIX Conf. on Networked Systems Design and Implementation, p.19.

Alizadeh, M., Yang, S., Sharif, M., *et al.*, 2013. pFabric: minimal near-optimal datacenter transport. *ACM SIGCOMM Comput. Commun. Rev.*, **43**(4):435-446. http://dx.doi.org/10.1145/2534169.2486031

Bai, W., Chen, L., Chen, K., *et al.*, 2015. Information-agnostic flow scheduling for commodity data centers. Proc. 12th USENIX Conf. on Networked Systems Design and Implementation, p.455-468.

Chowdhury, S.R., Bari, M.F., Ahmed, R., *et al.*, 2014. PayLess: a low cost network monitoring framework for software defined networks. Proc. Network Operations and Management Symp., p.1-9. http://dx.doi.org/10.1109/noms.2014.6838227

Clos, C., 1953. A study of non-blocking switching networks. *Bell Syst. Tech. J.*, **32**(2):406-424. http://dx.doi.org/10.1002/j.1538-7305.1953.tb01433.x

Curtis, A.R., Kim, W., Yalagandula, P., 2011. Mahout: low-overhead datacenter traffic management using end-host-based elephant detection. Proc. IEEE INFOCOM, p.1629-1637. http://dx.doi.org/10.1109/infcom.2011.5934956

Ding, J., Huang, T., Liu, J., *et al.*, 2015. Virtual network embedding based on real-time topological attributes. *Front. Inform. Technol. Electron. Eng.*, **16**(2):109-118. http://dx.doi.org/10.1631/fitee.1400147

Greenberg, A., Hamilton, J.R., Jain, N., *et al.*, 2009. VL2: a scalable and flexible data center network. *ACM SIGCOMM Comput. Commun. Rev.*, **39**(4):51-62. http://dx.doi.org/10.1145/1592568.1592576

Guo, C., Yuan, L., Xiang, D., *et al.*, 2015. Pingmesh: a large-scale system for data center network latency measurement and analysis. *ACM SIGCOMM Comput. Commun. Rev.*, **45**(4):139-152. http://dx.doi.org/10.1145/2785956.2787496

Handigol, N., Heller, B., Jeyakumar, V., *et al.*, 2012. Where is the debugger for my software-defined network? Proc. 1st Workshop on Hot Topics in Software Defined Networks, p.55-60. http://dx.doi.org/10.1145/2342441.2342453

Jarschel, M., Zinner, T., Hohn, T., *et al.*, 2013. On the accuracy of leveraging SDN for passive network measurements. Proc. Telecommunication Networks and Applications Conf., p.41-46. http://dx.doi.org/10.1109/atnac.2013.6705354

Katta, N.P., Rexford, J., Walker, D., 2013. Incremental consistent updates. Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, p.49-54. http://dx.doi.org/10.1145/2491185.2491191

Kazemian, P., Chang, M., Zeng, H., *et al.*, 2013. Real time network policy checking using header space analysis. Proc. 10th USENIX Conf. on Networked Systems Design and Implementation, p.99-112.

Khurshid, A., Zhou, W., Caesar, M., *et al.*, 2012. VeriFlow: verifying network-wide invariants in real time. Proc. 1st Workshop on Hot Topics in Software Defined Networks, p.49-54. http://dx.doi.org/10.1145/2342441.2342452

Liu, J., Huang, T., Chen, J., *et al.*, 2011. A new algorithm based on the proximity principle for the virtual network embedding problem. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **12**(11):910-918. http://dx.doi.org/10.1631/jzus.c1100003

McKeown, N., Anderson, T., Balakrishnan, H., *et al.*, 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.*, **38**(2):69-74. http://dx.doi.org/10.1145/1355734.1355746

Perešíni, P., Kuzniar, M., Vasić, N., *et al.*, 2013. OF.CPP: consistent packet processing for OpenFlow. Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, p.97-102. http://dx.doi.org/10.1145/2491185.2491205

Phemius, K., Bouet, M., 2013. Monitoring latency with OpenFlow. Proc. 9th Int. Conf. on Network and Service Management, p.122-125. http://dx.doi.org/10.1109/cnsm.2013.6727820

Phemius, K., Thales, B.M., 2013. OpenFlow: why latency does matter. Proc. IFIP/IEEE Int. Symp. on Integrated Network Management, p.680-683.

Qi, H., Shiraz, M., Liu, J., *et al.*, 2014. Data center network architecture in cloud computing: review, taxonomy, and open research issues. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **15**(9):776-793. http://dx.doi.org/10.1631/jzus.c1400013

Reitblatt, M., Foster, N., Rexford, J., *et al.*, 2012. Abstractions for network update. *ACM SIGCOMM Comput. Commun. Rev.*, **42**(4):323-334. http://dx.doi.org/10.1145/2377677.2377748

Scott, C., Wundsam, A., Raghavan, B., *et al.*, 2014. Troubleshooting blackbox SDN control software with minimal causal sequences. Proc. ACM SIGCOMM Conf., p.1-12.

Su, Z., Wang, T., Xia, Y., *et al.*, 2014. FlowCover: low-cost flow monitoring scheme in software defined networks. Proc. IEEE Global Communications Conf., p.1956-1961. http://dx.doi.org/10.1109/glocom.2014.7037094

Tavakoli, A., Casado, M., Koponen, T., *et al.*, 2009. Applying NOX to the datacenter. Proc. 8th ACM Workshop on Hot Topics in Networks, p.1-6.

Wundsam, A., Levin, D., Seetharaman, S., *et al.*, 2011. OFRewind: enabling record and replay troubleshooting for networks. Proc. USENIX Annual Technical Conf., p.29.

Yu, C., Lumezanu, C., Zhang, Y., *et al.*, 2013. FlowSense: monitoring network utilization with zero measurement cost. Proc. 14th Int. Conf. on Passive and Active Measurement, p.31-41. http://dx.doi.org/10.1007/978-3-642-36516-4_4

Yu, C., Lumezanu, C., Sharma, A., *et al.*, 2015. Software-defined latency monitoring in data center networks. Proc. 16th Int. Conf. on Passive and Active Measurement, p.360-372. http://dx.doi.org/10.1007/978-3-319-15509-8_27

Yu, M., Jose, L., Miao, R., 2013. Software defined traffic measurement with OpenSketch. Proc. 10th USENIX Conf. on Networked Systems Design and Implementation, p.29-42.

Zhang, H., Lumezanu, C., Rhee, J., *et al.*, 2014. Enabling layer 2 pathlet tracing through context encoding in software-defined networking. Proc. 3rd Workshop on Hot Topics in Software Defined Networking, p.169-174. http://dx.doi.org/10.1145/2620728.2620742