SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

February 2017, Vol. 60 022304:1–022304:13
doi: 10.1007/s11432-015-1057-7

A tool for tracing network data plane via SDN/OpenFlow

Yangyang WANG[1,3], Jun BI[1,2,3*] & Keyao ZHANG[1,2]

[1]*Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing* 100084*, China;*
[2]*Department of Computer Science, Tsinghua University, Beijing* 100084*, China;*
[3]*Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing* 100084*, China*

Received December 30, 2015; accepted April 25, 2016; published online November 9, 2016

Abstract SDN provides an approach to create desired network forwarding plane by programming applications. For a large-scale SDN network comprised of multiple domains and running multiple controller applications, it is difficult to measure and diagnose the problems of flow tables in data plane. Tracing the forwarding path of SDN is one of effective way for data plane state measurement. Previously proposed methods for debugging SDN were applied to a single administrative domain. There is less effort to trace the flow entries of the data plane in large-scale multi-domain SDN networks. In this paper, we propose a method of software defined data plane tracing in large-scale multi-domain SDN networks. Our method can trace forwarding paths, and get the matched flow entries and other customized trace information. We present the designs compatible with OpenFlow 1.0 and 1.3 switches. The performance and deployment effect are evaluated by simulation test and analysis. It shows that our method has better performance than traditional IP traceroute, and its deployment at about 20% of AS nodes can enable 70% of AS paths to be traceable.

Keywords software defined networking, data plane, network measurement, traceroute, Internet routing

Citation Wang Y Y, Bi J, Zhang K Y. A tool for tracing network data plane via SDN/OpenFlow. Sci China Inf Sci, 2017, 60(2): 022304, doi: 10.1007/s11432-015-1057-7

1 Introduction

In today's IP infrastructure, Internet Control Message Protocol (ICMP) is widely used for network data plane measurement and troubleshooting in large IP networks and the Internet. By `ping` and `traceroute` tools, we can perceive end-to-end forwarding paths performance to diagnose where routing problems possibly happen. Also, network researchers and operators use `traceroute` to discover the IP interfaces and topology of a network [1]. However, these diagnoses and measurements usually are error-prone and limited [2–4] because of the closed IP protocol stack in network devices. We have to do some inference measurements in constrained situations before protocol innovations are widely deployed on practical networks.

In recent years, software defined networking (SDN) [5] emerges, and OpenFlow is a standard instance [6]. SDN breaks the closure of today's network architecture. It decouples the control plane and data

* Corresponding author (email: junbi@tsinghua.edu.cn)

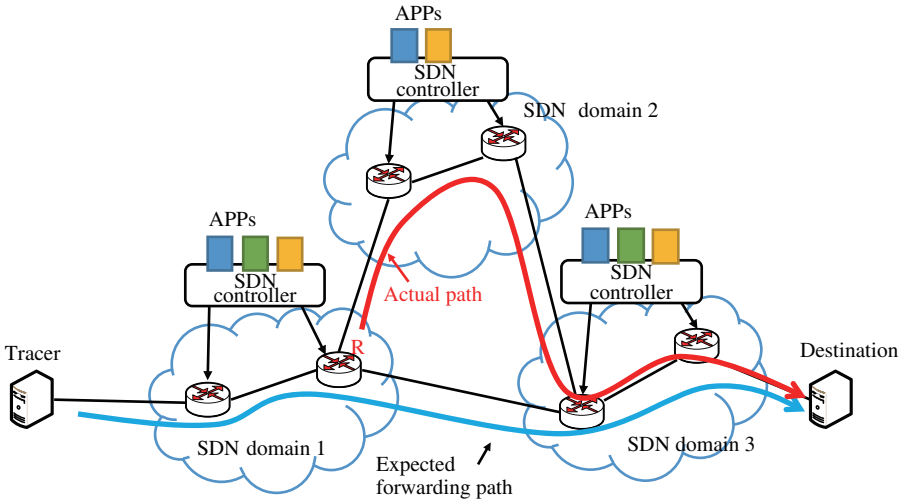

Figure 1 (Color online) An example of forwarding state conflict in SDN networks.

plane of networks, and improves network programmability. The flexibility and programmability of SDN has been applied to local networks, such as campus networks, data centers, and wide-area networks [7], to support innovations in networking applications, such as routing optimization [8], and new network architectures [9,10]. When a small SDN network is comprised of a few of switches, controller applications and simple forwarding rules, we can find the causes of problems in networks by manually checking flow tables using `ping` or other proposed SDN debugging tools used in a single administrative SDN network. But for a large-scale SDN network, such as the networks of large Internet service providers (ISPs), or an inter-domain SDN network composed of multiple SDN domains, it is difficult to locate the problems of data plane in a large-scale and dynamic environment.

Tracing forwarding plane with `traceroute`-like tools is an effective way for troubleshooting in large-scale networks. Moreover, the open paradigm of SDN can conquer the traceroute limitations against traditional IP protocol stack. Thus, applying SDN-based data plane tracing mechanism to SDN networks and the Internet has the following demands and benefits.

(1) Network state visibility of multi-domain SDN networks. For a small SDN network, it is easier to test network performance, or detect failure problems by manually checking network states. Some researches have developed tools and systems for automatical test and troubleshooting. But they are mostly applied to the SDN networks in a single administrative domain. In the case of networks that are running multiple controller applications and across multiple domains, it will take higher cost to record and manage the state of data plane by delegating data plane tracing to control plane. Moreover, an operator of one SDN domain may have no permission to obtain the forwarding plane states of other administrative domains. Traceroute-like tools for SDN networks can provide visibility of the data plane in large multi-domain SDN networks.

(2) Network state visibility in real time. An SDN network may run multiple SDN applications on controllers. When a packet is being forwarded, it may match a flow entry generated by other applications at the same time, which causes the conflict of forwarding states. Figure 1 presents an example. This example shows a multi-domain SDN network. There are multiple applications (APPs) running on SDN controllers. Each application computes its own routing paths for different policy routing or traffic engineering requirements. A packet from tracer to destination will take the path (SDN domain 1→SDN domain 3) as the expected forwarding path generated by the same APP (represented by the second block on the left on the controllers of SDN domain 1 and 3). But, the packet actually matches the flow entry installed by the other APP (represented by the first block on the left on the controllers of SDN domain 1, 2 and 3) on the router R at a certain moment. As a result, the actual forwarding path is switched to the SDN domain 2 at the router R. It is necessary for network troubleshooting to trace the actual matched flow entries in real time.

(3) User-defined rich trace information and control. The functionality of traceroute is restricted to the implementation of IP and ICMP protocol. The traditional traceroute can only obtain the IP addresses of routers' incoming interfaces along forwarding paths. For other network information, for example, the AS number where a router belongs, it needs additional measurement and inference from other data sources [11, 12]. SDN provides an open approach to innovate network functionality. Network operators can define rich trace information, such as the identifications (IDs) of the switches and ports along a forwarding path, the AS number and name of the ISP where a switch is located. SDN can also apply fine-grained control over the flow entries that can be traced.

To get the above benefits, in this paper, we propose a software defined traceroute mechanism, named `sTrace`. It aims to provide flexible data path tracing mechanism to improve data plane visibility for large SDN networks. It can trace SDN forwarding paths and the matched flow entries at each hop in real time, and return rich trace information, including switch and port IDs, matched flow entries, AS numbers, and other user-defined information as required. We present the design based on OpenFlow 1.3 and 1.0 standards. Because sTrace does not rely on priori knowledge of the entire network topology, it can adapt to dynamic topology, and can be deployed incrementally by ISPs.

2 Background and related work

The commonly used tool for detecting IP forwarding hops is `traceroute`. It makes use of existing mechanism defined by ICMP. The working process of `traceroute` is well known [13]. One host sends out UDP packets with a large and unused port number to a destination. The immediate routers will decrease the IP TTL field by 1 and forward packets to the next hop to the destination. If the TTL value reaches zero, the immediate router will discard this UDP packet and return ICMP Time Exceeded error message to the probe host. By sending IP packets of increasing TTL value from 1, the probe host will receive returned ICMP error messages from the sequence of router interfaces starting from first hop to the destination. When probe packets reach the destination, ICMP port unreachable messages will be returned.

However, `traceroute` just provides a basic path tracing approach in IP layer. For SDN networks, we expect to extend the basic traceroute function to provide flexible path tracing mechanism with rich trace information, including not only IP addresses of interfaces. Some researches have made efforts for SDN data plane measurement and debugging. The study `ndb` [14] proposes a debugging tool for SDN networks. Inspired by the software debugger tool `gdb`, the goal of `ndb` is to support `gdb`-like debugger actions such as breakpoint, backtrace, single-step to examine network events related to a network error. The key mechanism of `ndb` is to create and collect postcards. A postcard contains a truncated copy of original packet's header, the matched flow entry, switch ID, and output port. A collector of `ndb` stores postcards and constructs backtrace for breakpoint-marked packets. The design of `ndb` provides more information than `traceroute` for a purpose of powerful debugger tool. When `ndb` deals with debugging, it needs to modify each flow entry to create postcards for each packet, and store these postcards. This will take much cost in memory and packet processing. It is suitable for a small SDN networks with an administrative domain.

The work [15] proposes a SDN traceroute tool that probes actual packet forwarding path without changing the flow entries being measured. The probe packets will visit each switch on the path to destination. Its method has two steps. First step is to collect network topology and assign a color value for each switch to distinguish neighboring siwtch. And then, it installs proper flow entries to switches to trigger `PACKET_IN` messages to the controller, where the controller deals with these `PACKET_IN` messages to obtain the in-port and out-port information of probe packets traversing a switch. This method does not determine the matched table entries in data plane. And, it needs to know beforehand the entire network topology for color assignment, which make it not suitable in the environment of large and dynamic networks.

Some other previous researches also provide approaches to debugging and troubleshooting network errors. NetSight [16] stores the packet historical records by modifying flow table entries in the SDN. The work OFf [17] has some similar features for tracing network state, but OFf can trace controller program

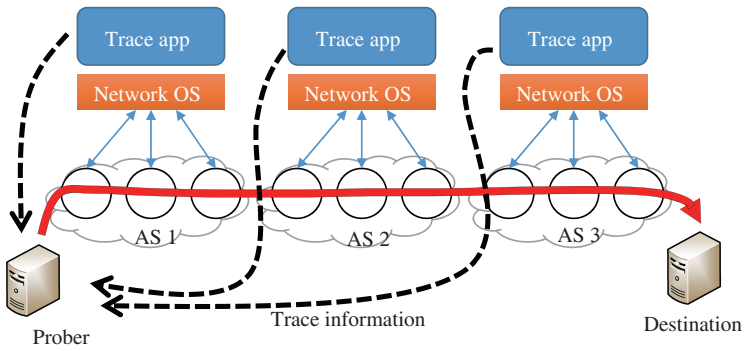

Figure 2 (Color online) The conceptual model of software defined traceroute.

states that induce observed network behavior. OFRewind [18] allows recording and playing back of SDN control plane traffic. Monocle [19] provides a rule-level data plane monitoring method. It creates a proxy between an SDN controller and switches, which collects global flow tables of each switch and automatically generates test packets and observes the actual behaviors of this network treating these test packets. The work [20] designs a test framework FlowTest to integrate other test tools and plans for testing stateful data plane. The work NICE [21] focuses on automating the testing of controller applications by model checking with symbolic execution of event handlers. It does not measure the forwarding plane.

Some researches are focused on validate correctness of SDN control. For example, FlowChecker [22] creates a tool to identify the misconfiguration within a single flow table. The work [23] presents a system for testing SDN control policy violation by simulation-based causal inference. HSA [24] proposes methods to check network configurations and identify network errors such as reachability failures and forwarding loops by analysis of packet header space. NetPlumber [25] and VeriFlow [26] are to verify correctness properties violation on the system by real-time analysis over flow rules and header fields. Libra [27] collects the states of SDN data plane to verify the forwarding correctness in a large networks. Using MapReduce, Libra has a higher performance. However, frequent state updates may cause computation cost and affect the correctness of results.

In summary, these previous studies mostly focus on the test and troubleshooting in a single administrative SDN domain. There is little effort to measure data plane in a large multi-domain SDN networks. Our work aims to provide data path tracing mechanism to improve data plane visibility for large SDN networks.

3 Design

In this section, we introduce our sTrace mechanism. Firstly, we propose an ideal design for traceroute-like functions in SDN. And then, We propose the function implementations for sTrace using OpenFlow 1.3 and 1.0. After that, we discuss possible support for reverse traceroute function with SDN and incremental deployment on the Internet.

3.1 Concept model and design architecture

The concept model in Figure 2 describes a concept of traceoute-like process. To enable data plane to be traceable, network operators need to install Trace applications on their SDN controllers. The Trace controller will modify the flow entries to be installed, or install additional entries to the data plane to enable tracing functions. In data plane, when an OpenFlow switch that supports ideal traceroute model receives a packet, it looks up the flow table for a matched entry. The actions of matching entry may decrease the TTL value by 1, and determine whether the TTL value is zero. If the TTL value is not zero, this packet will be forwarded to next SDN switch. If the TTL value reaches zero, this packet will be issued to a controller through a `PACKET_IN` message, with associated extra information of the matched flow entry, and the switch ID, input port, timestamp when the switch receives this packet. When the traceroute

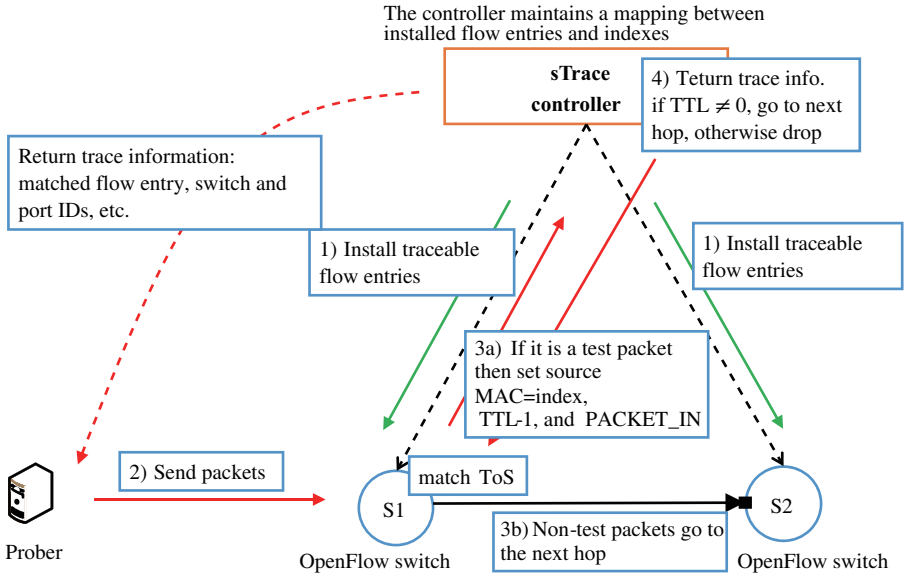

Figure 3 (Color online) The sTrace process using OpenFlow 1.3 switches.

application on the controller receives this message, it could return a user-defined trace information to the probe host that originally issues this test packet. That trace information may include not only the matched flow entry, switch id, etc., as well as other network operation information such as domain AS number, which can be defined as required.

The design architecture of sTrace is similar to the above conceptual model. The traceroute-like function of sTrace is driven by two modules: a sTrace server on SDN controllers, and a prober running on a host. The sTrace controller provides traceroute-like support in SDN networks. Similar to `ndb`, the sTrace controller monitors flow entry modification `FLOW_MOD` messages, and installs new entries or modifies some required actions such as decreasing TTL value, etc., to enable data plane tracing functionality. The sTrace controller also processes `PACKET_IN` messages, and returns rich and user-defined trace information. It can control traceable flow entries by SDN scheme. The sTrace prober is used to issue test packets, receive and parse the returned trace information. We will present the details of design for OpenFlow 1.3 and OpenFlow 1.0 in the following sections.

Different from the conceptual model, in our design, we use the IP type of service (ToS) field to identify test traffic and TTL value to control test range. Traditional IP layer uses TTL-based method to trace forwarding path. The TTL field of IP header is designed to avoid forwarding loop. TTL-based tracing methods return trace information when the TTL value reaches zero. It has two problems when using TTL-based approach in SDN networks: (1) It has to send packets with specific TTL values for measuring each hop. This way will produce lots of test traffic. (2) When a test packet does not match an entry in a flow table, it will trigger a `PACKET_IN` message to request the controller to install missing flow entries. This test behavior will change the original settings in data plane. It is difficult for the controller to distinguish test traffic with product traffic based only on TTL values. If the controller can identify test traffic, it could control (i.e. enable/disable) the reaction to the missing flow entry of test packets. For example, the controller can inform the prober of flow entry miss, and disable missing flow entry installation for test packets. To identify test traffic, we set the ToS field in IP header with an unused particular value, such as bits 00011000 according to RFC 2474 [28]. And the TTL field is used to control test range of hops. By this way, we can trace a range of hops with a single test packet.

3.2 Design for OpenFlow 1.3

OpenFlow 1.3 switches support multi-level flow tables and TTL decrement in data plane. To identify test packets, we set the IP ToS field of test packets with a particular value (such as ToS bits 00011000). The basic process of sTrace with OpenFlow 1.3 switches is shown in Figure 3. In Figure 3, S1 and

S2 represent OpenFlow switches. The sTrace server on the controller maintains a mapping between an increasing index number and a flow entry. For a flow entry modification (i.e., `FLOW_MOD`) message M, sTrace controller will extract and keep a record of the original flow entry f that is to be installed to the switch, and assign f an index number i, and then update i with $i+1$. After that, sTrace will modify f into a new traceable flow entry f' and install it to flow table 1. It moves each of the original actions of f to an action set O using the instruction Write-Actions. And then, it appends f with a list of actions of setting the metadata to i and the original output port of f, and sets an instruction Goto-Table(2). By this way, we get a new flow entry f' from f. In flow table 2, sTrace will install a flow entry g that uses a particular ToS value to match test packets (such as ToS=00011000). And g has a list of actions that set source MAC address to the metadata, decrease TTL, and output packets to the controller, and append an instruction Clear-Actions to clear all the original actions in the action set O. Installing flow entries f' and g corresponds to the Step 1) "install traceable flow entries" in the Figure 3. The sTrace controller also installs a table-miss flow entry in flow table 2 for non-test traffic. It has no match fields (all fields omitted) and has the lowest priority (0).

After the installation of traceable flow entries, the prober may issue test/non-test packets, as Step 2) shown in Figure 3. (1) If a test packet does not match any flow entry in flow table 1, it will trigger a `PACKET_IN` message to the controller. When the sTrace controller receives the packet and finds it is a test packet based on ToS value, it will discard this packet, and return a "flow entry missed" trace information to the prober. The controller does not install a flow entry to the switch. This way can avoid the unnecessary missed flow entry installation for test packets. (2) If a test packet matches the flow entry f' in flow table 1, f' will append this packet with an action set O containing the original actions of the flow entry f, and a metadata containing the index number i and the original output port of f. Then, the instruction Goto-Table(2) forwards the test packet to flow table 2. In flow table 2, the test packet will match the flow entry g with ToS field (i.e. "match ToS" in Figure 3). And then, the flow entry g will apply its actions to the test packet. It will set the source MAC address field of the test packet to the metadata, decrease TTL, and then trigger a `PACKET_IN` output to the controller, as Step 3a) shown in Figure 3. The action set O will be cleared by the instruction Clear-Actions of flow entry g. When the sTrace controller receives the test packet, it can extract the index number of matched flow entry and its out-port from the metadata in the source MAC address field, and locate the matched flow entry in the mapping table. It can also get switch and in-port IDs from the `PACKET_IN` message. Based on these data and other user-defined information such as AS number, the sTrace controller creates trace information and returns it to the prober. And then, the sTrace controller will discard the packet if this test packet has a zero TTL. Otherwise, the controller forwards the packet of non-zero TTL value to the switch out-port through `PACKET_OUT` message. Note that, the TTL field is used as a control over the range of test hops. At each hop in a test range, test packets will trigger `PACKET_IN` messages to the controller. (3) For a packet of non-test traffic, it does not match the ToS value of test packet, and falls into the table-miss entry. This entry has no actions, and the original actions in the action set O appended to the packet will be executed, which forward this packet to the next switch, as Step 3b) shown in Figure 3.

In the design for OpenFlow 1.3 switches, we install a modified flow entry f' for each original flow entry f in flow table 1, and two additional flow entry in flow table 2. It introduces few extra flow entries.

3.3 Design for OpenFlow 1.0

OpenFlow 1.0 does not support multi-level flow tables and TTL value change in data plane. Therefore, TTL decrement has to resort to sTrace controllers. We also use the IP ToS field of particular value (such as ToS bits 00011000) to identify a test packet. The process of OpenFlow 1.0 design is similar to that of OpenFlow 1.3. The basic process of OpenFlow 1.0 design is shown in Figure 4. Different from OpenFlow 1.3 installing only a modified flow entry f' for an original flow entry f, the design for OpenFlow 1.0 needs to install both the original flow entry f and a new flow entry f', which will double the number of flow entries in data plane.

For a flow entry modification i.e., `FLOW_MOD` message M, sTrace controller will extract and keep a record

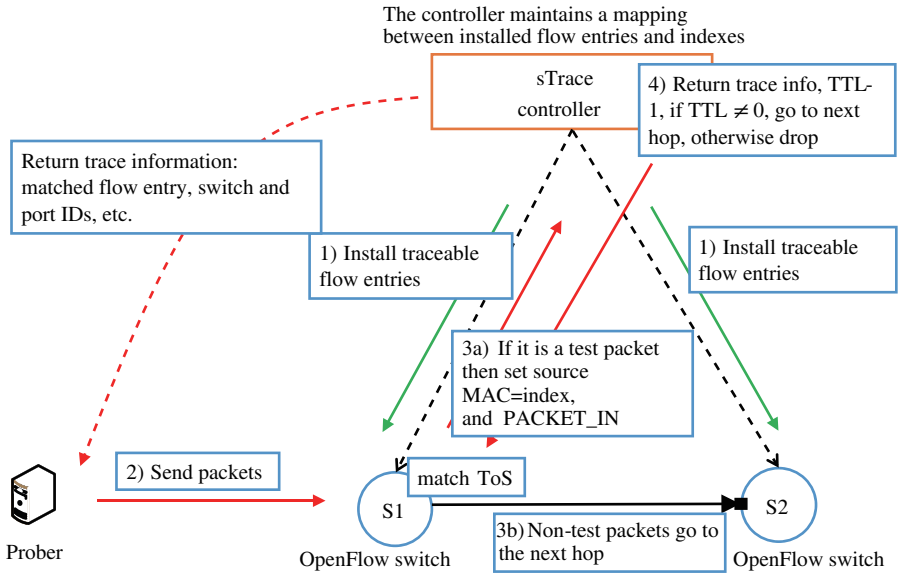

Figure 4 (Color online) The sTrace process using OpenFlow 1.0 switches.

of the original flow entry f to be installed to a switch, assign it an index number i and update i with $i+1$. After that, the sTrace controller will install the original flow entry f and a new flow entry f' to the switch, i.e. the Step 1 as shown in Figure 4. In Figure 4, S1 and S2 represent OpenFlow switches. The match fields of f' include all match fields of f, and an additional ToS field (ToS=00011000) to match test packets. The actions of f' will set source MAC address to the flow entry index number i and the output port of f, and output packets to the controller. The non-test packets will match flow entry f and go to the next switch as Step 3b) shown in Figure 4. The test packet matching the flow entry f' will be modified in source MAC address and forwarded to the controller through `PACKET_IN` message, as Step 3a) shown in Figure 4. The sTrace controller could get in-port and switch IDs from the `PACKET_IN` message, and the index number and out-port of the matched flow entry from the source MAC address of the test packet. Based on these data and other user-defined information, the sTrace controller creates trace information and returns it to the prober. And then, the sTrace controller decreases the TTL value. If the TTL is not zero, the packet will be sent to the switch out-port through `PACKET_OUT` message. Otherwise, the sTrace controller discards the packet of zero TTL value. This way controls the hop range of test by TTL values. For a test packet that does not match any flow entry in the flow table, it will trigger a `PACKET_IN` message to the controller. And the controller will discard this packet, and return a "flow entry missed" trace information to the prober.

3.4 Operation on hybrid networks

In fact, a large SDN networks composed of multiple SDN domains would use many different switches of various implementations of OpenFlow. sTrace will work on a hybrid network environment. The least support for data plane functions is based on OpenFlow 1.0. This situation needs to insert a new flow entry to support traceroute-like functions for each original flow entry, which will double the memory cost of flow entries in data plane. The feature of multi-level flow tables in higher version of OpenFlow can save more flow entries than single-level flow tables. In the situation of hybrid SDN devices, we could prefer to deploy traceroute test flow entries in the switches supporting multi-level flow tables to save data plane entries. For the switches supporting only single-level flow tables, we can adaptively install the necessary test flow entries for the test traffic with ToS tag.

3.5 Reverse traceroute

In many cases, we hope to discover what forwarding path it will take from a remote end host to local

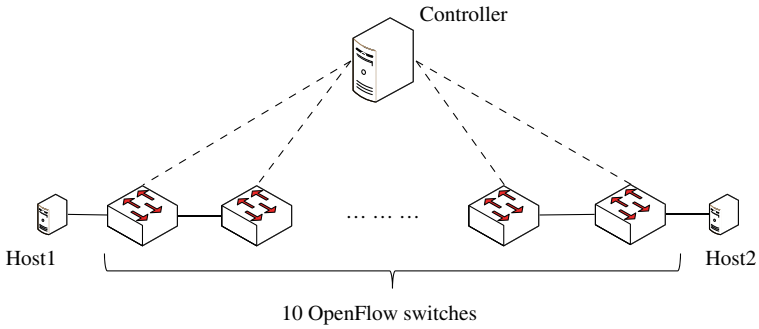

Figure 5 (Color online) The simulation environment is composed of a Floodlight controller and ten OVS OpenFlow switches in Mininet.

host. This is called reverse traceroute [29]. If we have a probe at remote side, it is easy to get the reverse forwarding path. But there is no probe in most cases at remote side. Although the effort on reverse traceroute has been done with IP options, it is constrained by many conditions on the Internet. SDN's programmability may provide potential approach for reverse traceroute. For example, we can set the IP ToS field to a particular value k as a tag identifying a test packet for preparing reverse traceroute. For a test packet of ToS tag k from a source host h to a destination d, when the packet's TTL value reaches zero, or this packet arrives at the last hop on the forwarding path, the immediate sTrace controller will issue a test packet p from d to h in terms of the reverse trace tag value k, and change the tag to a reverse trace tag $k+1$ in this newly issued test packet p. In the process of forwarding the packet p to h, the immediate controllers take a process similar to traceroute. The difference from forward traceroute is that the sTrace controllers will check this reverse trace tag $k+1$, and return a trace information to the destination host h, not to the source host d.

4 Evaluation

4.1 Performance of sTrace

Because of lack of the real environment of multi-domain SDN networks, we evaluate the performance cost sTrace process by simulation. We installed a Floodlight controller on a host, and created a SDN network using Open vSwitch (OVS) software OpenFlow switches in Mininet. We implemented a simple function prototype of sTrace server on Floodlight controller for OpenFlow 1.0 and OpenFlow 1.3. Our simulation has two hosts (host1 and host2), and a linear topology of ten OVS OpenFlow switches between the two hosts, shown in Figure 5. sTrace will install proper test flow entries to OpenFlow 1.3 and 1.0 OVS switches for traceroute from host1 to host2. And then, we use `ping` to issue test packet from host1 to host2. Each test packet will be processed in the way of traceroute, such as decreasing TTL value, setting source MAC address field, etc., as described in Section 3. We calculate the latency time between sending out a test packet and receiving a returned trace information of the last hop switch. The result is a average time over 500 times of tests. For comparison, we simulate the normal packet forwarding with general ICMP packets that do not trigger traceroute process. We also change the number of switches from 2 to 10 with a step interval 2.

The results are shown in Figure 6. In Figure 6, the case "`OF1.0 normal`" uses a single-level flow table in each switch, and just forwards packets without tracing operations. The host1 sends out a general ICMP request packet to the destination host2, and then gets an ICMP reply from host2. The curve of "`OF1.0 normal`" shows the round trip time values of different number of immediate switch hops. Similar to "`OF1.0 normal`", the curve "`OF1.3 normal`" also represents the general ICMP request/reply round trip time values. But it uses multi-level flow table on each switch. The TTL of a test packet is decreased by 1 at each switch hop. We can see that the two test cases have almost the same values. This indicates the multi-level flow table introduces little processing cost. "`OF1.0 trace`" and "`OF1.3`

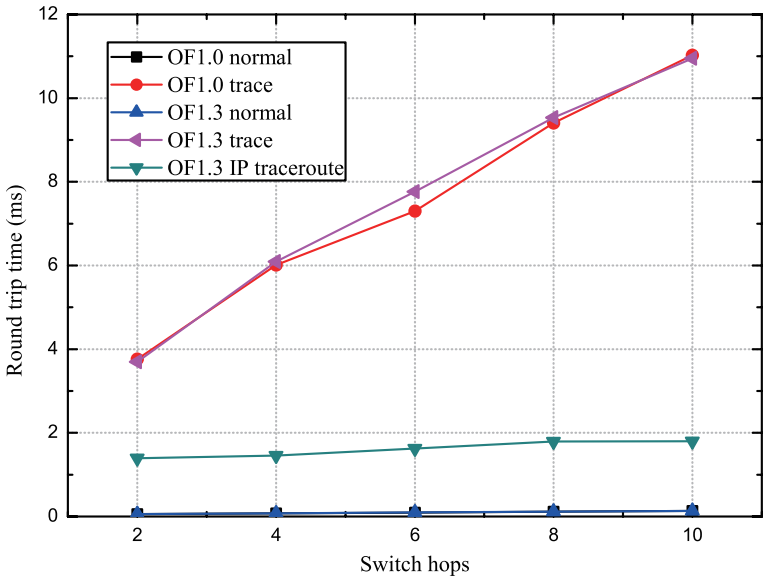

Figure 6 (Color online) A comparison of round trip time of normal packet forwarding and sTrace test packet processing.

`trace`" respectively mean using one test packet in tracing the entire forwarding path for OpenFlow 1.0 and 1.3 switches. In the case of "`OF1.3 trace`", for each test, host1 sends out an ICMP request test packet that sets a particular ToS filed value and sets the TTL value as the number of switch hops (i.e., 2,4,...,10). At each switch hop, the test packet will be processed by the flow entry actions, including decreasing TTL, described in Section 3, and then triggers a `PACKET_IN` message to the controller. After that, the controller processes the `PACKET_IN` message, returns trace information to host1 and directs the test packet to the correct nexthop by OpenFlow `PACKET_OUT` action. When the TTL value reaches zero, i.e., the trace arrives at the last switch hop, the controller will return the final reply of trace information. We compute the time between host1 sending out a test packet and receiving the final relay as a round trip time. The curve '`OF1.3 trace`" shows the resulting average round trip time of testing different number of hops. The case "`OF1.0 trace`" has the same testing process to "`OF1.3 trace`". We can see that their results have a notable increasing RTT value because the test packet goes back and forth many times between switches and the controller along the path from host1 and host2. (There is a notable difference between "`OF1.3 trace`" and "`OF1.0 trace`" when there are 6 switch hops. This difference is caused by the accidental jitters in the system of testing environment). The result of "`OF1.3 IP traceroute`" uses OpenFlow 1.3 to simulate the traditional IP traceroute where only the last switch hop triggers a `PACKET_IN` message when the TTL of a test packet reaches zero. It shows a less average round trip time as expected.

In the following, we analyze the cost to trace a path with n switch hops between host1 and host2. We denote host1 as hop(0), and host2 as hop($n+1$), and the immediate switches are referred as hop(1), hop(2),..., hop(n). We use $P(x)$ to denote the time cost processing test packet on switch or host x. For example, $P^{\mathrm{OF1.3}}(i)$ means the time cost of processing test packet on OpenFlow 1.3 switch hop(i), ($i = 1, 2, \ldots, n$). To simplify the following formalization, we assume $P(i)$ of each switch hop(i) is almost equal and denoted as P. The P(host2) means the time cost of ICMP reply on host2. Similarly, we use $L(i)$ to denote the propagation time cost on the link between hop($i-1$) and hop(i), ($i = 1, 2, \ldots, n+1$), and use $H(i)$ to denote the propagation time cost between the switch hop(i) and the controller. And we assume $L(i) \simeq L$ and $H(i) \simeq H$ for each $L(i)$ and $H(i)$. In this way, the round trip time $T_{\mathrm{OF1.3}}(n)$ of "`OF1.3 normal`" traversing n switches can be approximately calculated as

$$T_{\mathrm{OF1.3}}(n) = 2\sum_{i=1}^{n+1} L_i + 2\sum_{i=1}^{n} P^{\mathrm{OF1.3}}(s_i) + P(\mathrm{host2}) \simeq 2(n+1)L + 2nP^{\mathrm{OF1.3}} + P(\mathrm{host2}). \quad (1)$$

In the above formula, OF1.3 is a short representation of OF1.3 normal. Similarly, the round trip time

$T_{\mathrm{OF1.0}}(n)$ of "`OF1.0 normal`" with n switches can be approximately calculated as

$$T_{\mathrm{OF1.0}}(n) = 2\sum_{i=1}^{n+1} L_i + 2\sum_{i=1}^{n} P^{\mathrm{OF1.0}}(s_i) + P(\mathrm{host2}) \simeq 2(n+1)L + 2nP^{\mathrm{OF1.0}} + P(\mathrm{host2}). \quad (2)$$

From the results shown in Figure 6, we can see that $T_{\mathrm{OF1.3}}$ and $T_{\mathrm{OF1.0}}$ are very little. It means the time cost L, P and P(host2) are also little.

The round trip time $T_{\mathrm{OF1.3t}}(n)$ and $T_{\mathrm{OF1.0t}}(n)$ of "`OF1.3 trace`" and "`OF1.0 trace`" with n switches can be approximately calculated as

$$\begin{aligned} T_{\mathrm{OF1.3t}}(n) &= \sum_{i=1}^{n}(2L_i + 2P^{\mathrm{OF1.3t}}(s_i) + 2H_i + P^{\mathrm{OF1.3t}}(\mathrm{controller})) \\ &\simeq 2nL + 2nP^{\mathrm{OF1.3t}} + 2nH + nP^{\mathrm{OF1.3t}}(\mathrm{controller}), \quad (3) \\ T_{\mathrm{OF1.0t}}(n) &= \sum_{i=1}^{n}(2L_i + 2P^{\mathrm{OF1.0t}}(s_i) + 2H_i + P^{\mathrm{OF1.0t}}(\mathrm{controller})) \\ &\simeq 2nL + 2nP^{\mathrm{OF1.0t}} + 2nH + nP^{\mathrm{OF1.0t}}(\mathrm{controller}). \quad (4) \end{aligned}$$

Here, $P^{\mathrm{OF1.3t}}$(controller) represents the process cost on the controller which runs OpenFlow 1.3 protocol and directs test packets to the next switch or creates an ICMP reply packet and returns it to host1. The $P^{\mathrm{OF1.0t}}$(controller) has the similar meaning.

The round trip time $T_{\mathrm{ipt}}(n)$ of "`OF1.3 IP traceroute`" with n switches can be approximately calculated as

$$\begin{aligned} T_{\mathrm{ipt}}(n) &= \sum_{i=1}^{n}(2L_i + 2P^{\mathrm{OF1.3}}(s_i)) + 2H_n + P^{\mathrm{OF1.3}}(\mathrm{controller}) \\ &\simeq 2nL + 2nP^{\mathrm{OF1.3}} + 2H + P^{\mathrm{OF1.3}}(\mathrm{controller}). \quad (5) \end{aligned}$$

From the comparison between $T_{\mathrm{ipt}}(n)$ and $T_{\mathrm{OF1.3}}(n)$, we can see that the difference of "`OF1.3 IP traceroute`" and "`OF1.3 normal`" in Figure 6 is mainly caused by $2H + P^{\mathrm{OF1.3}}$(controller), which include the communication delay between the switch and the controller, and the process cost on the controller which creates an ICMP reply packet and returns it to host1. We also can find that $T_{\mathrm{OF1.3t}}(n)$ grows up with n because of the item $2nH$.

To trace every hop along a path with n switch hop, the case "`OF1.3 trace`" in Figure 6 only issue one test packet that will traverse each switch hop of the path, and the time cost is $T_{\mathrm{OF1.3t}}(n)$. But the case "`IP traceroute`" in Figure 6 needs to issue n test packets with different TTL values. Its time cost for testing the entire path is the sum of the time cost of each test packet, i.e. $\sum_{k=1}^{n} T_{\mathrm{ipt}}(k)$. It is larger than $T_{\mathrm{OF1.3t}}(n)$. This implies sTrace has a better performance than traditional IP traceroute.

Actually, the above formulation is simplified and approximate. For example, the P(controller) may have different process cost. The controller directs test packets to the next switch at the immediate switch hop, while it constructs and returns an ICMP reply at the last hop. The cost of the latter is more than that of the former. For this reason, $T_{\mathrm{OF1.3t}}(2)$ is larger than the difference $T_{\mathrm{OF1.3t}}(4) - T_{\mathrm{OF1.3t}}(2)$. The P(controller) of $T_{\mathrm{OF1.3t}}(2)$ contains the cost of the controller creating ICMP reply packets at the hop(2), but the P(controller) of $T_{\mathrm{OF1.3t}}(4) - T_{\mathrm{OF1.3t}}(2)$ contains only the cost of the controller directing test packet to next switch at the immediate switch hop(2) and hop(3).

4.2 Deployment effect on the Internet

We evaluate the deployment effect by Internet AS topology. The approach is to analyze the parts of AS path that can be traceable when more and more ASes deploy SDN-based traceroute-like mechanism. We define the following metrics used in the evaluation. Suppose that a given set of AS nodes that deploy sTrace is denoted as A, and a given set of AS paths is denoted as P. For a path $p \in P$, its traceable part can be evaluated by $t(p)/\mathrm{len}(p)$, where $t(p)$ is the number of AS nodes in path p that are also in

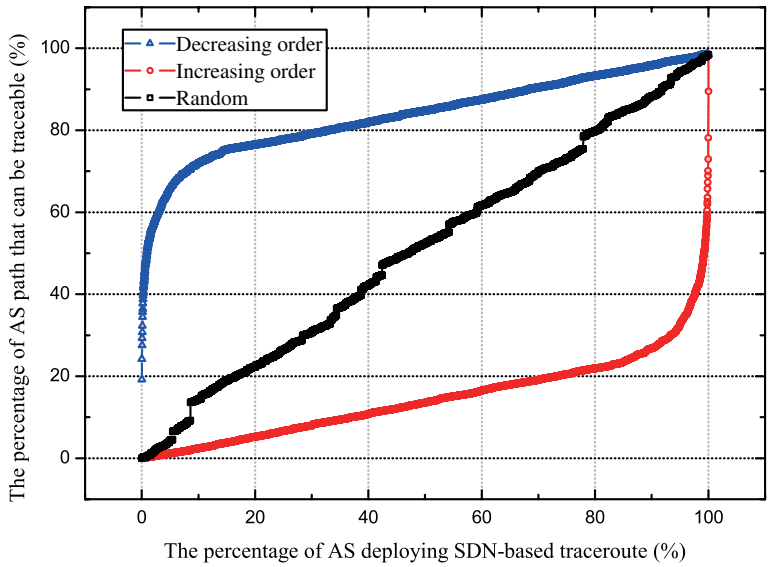

Figure 7 (Color online) The impact of AS deploying SDN-based traceroute on AS path traceability.

the set A (i.e., the ASes deploying sTrace). The len(p) is the total number of AS nodes in path p. The total traceable parts can be averaged over all paths, and defined as $(\sum_{p\in P} t(p)/\text{len}(p))/N$, where N is the total number of paths in P.

The BGP AS paths used in the evaluation are collected from the public BGP data [30], including RouteViews, PCH and RIPE RIS. The AS topology is collected from CAIDA [31]. Firstly, we rank AS nodes based on AS cone size [31]. The AS nodes with small AS cone usually are the customer networks connected to the Internet, while the AS nodes with large AS cone are the provider networks located in the transit core of the Internet. We take three deployment strategies: 1) deployment in increasing order of AS cone size, from edge customer networks to transit core; 2) deployment in decreasing order of AS cone size, from transit core to edge customer networks; 3) random selection that selects AS nodes in random order. The results are shown in Figure 7. We can see that the deployment strategy from core to edge (i.e, in decreasing order) can make most number of AS paths (more than 70% of total AS paths) traceable with only a small number of AS nodes (about 20% of total AS nodes) deploying sTrace. The random strategy shows a linear growth with the number of deployed AS nodes.

5 Discussion

5.1 Depolyment on the Internet

The tool sTrace can be used to trace the data plane of large-scale SDN networks. However, before SDN networks are widely adopted on the Internet, it can also be deployed on the Internet locally to improve the traceability of the Internet data plane. This is another aim of this tool. An ISP can deploy SDN switches or routers in its networks, and install the desired test flow entries according to IP routing tables by sTrace controllers. This can create a traceable plane in one ISP or between multiple ISPs. Also, we can deploy sTrace at Internet eXchange Points (IXPs) or SDXs [32], such that we can trace the IXPs connectivity and traffic policy on the Internet. In the case that a routing path contains some domains which may still be traditional IP networks without SDN, sTrace can trace only the domains that support SDN along the path. To work out the whole path in this case, we can combine traditional Traceroute tool with sTrace.

5.2 Scalability and limitations

A sTrace controller just enables the function of tracing the SDN domain under its control. In a multi-

domain SDN network, there are several SDN controllers. These sTrace controllers do not need to exchange measurement information and cooperate with each other. Because each sTrace controller ensures the local controlled domain can be traceable, the paths across the multi-domain SDN network can be traced completely if every SDN domain deploys sTrace. This feature will benefit the scalability and incremental deployment of sTrace. The sTrace has also limitations on capability. The sTrace mechanism for OpenFlow 1.0 switches will install an extra flow entry for each original entry to be traced. This will lead to table expansion. But OpenFlow standard is developing, and OpenFlow 1.3 will get adopted widely. The sTrace for OpenFlow 1.3 switches only modifies the original entries to be measured and installs two more extra flow table entry. It will not take much flow table space. This may improve its scalability. The design of sTrace is suited to trace single-level flow rules. It sets a tag in the source MAC address field to record the index of a matched flow entry. The MAC address field has limited space, which cannot store many index tags. Therefore, sTrace cannot trace a pipeline of flow entries of multi-level flow tables. Solution to trace a pipeline rule is in our future work.

6 Conclusion

In this paper, we propose a software defined traceroute-like tool named sTrace. It can trace the matched flow entries in multi-domain SDN networks in real time, and return rich user-defined trace information to tracers. We present the implementation based on OpenFlow 1.0 and 1.3 specifications. By simulation with Mininet and Open vSwitch, we find that the processing cost in data plane for traceroute packet is very little, and the overhead become growing when processing probe packet on controllers. We analyze the performance of sTrace by formalization. It shows sTrace has a better performance than traditional IP traceroute. We also evaluate the effect of incremental deployment of sTrace on the Internet with different deployment strategies. The results show that deployment at about 20% of AS nodes can enable 70% of AS paths to be traceable. In summary, sTrace provides a mechanism of tracing the data plane states in large-scale multi-domain SDN networks. In future work, we will enhance the capability and real deployment of tracing SDN networks.

Acknowledgements This work was supported by National High Technology Research and Development Program of China (Grant No. 2013AA013505), National Natural Science Foundation of China (Grant Nos. 61472213, 61303194)

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Motamedi R, Rejaie R, Willinger W. A survey of techniques for Internet topology discovery. IEEE Commun Surv Tutor, 2015, 17: 1044–1065
2 Keys K. Internet-scale IP alias resolution techniques. ACM SIGCOMM Comp Commun Rev, 2010, 40: 50–55
3 Marchetta P, Persico V, Pescapé A, et al. Don't trust traceroute (completely). In: Proceedings of the Workshop on Student Workhop. Santa Barbara: ACM, 2013. 5–8
4 Keys K, Hyun Y, Luckie M, et al. Internet-scale IPv4 alias resolution with MIDAR. IEEE ACM Trans Netw, 2013, 21: 383–399
5 Nunes B, Mendonca M, Nguyen X N, et al. A survey of software-defined networking: past, present, and future of programmable networks. IEEE Commun Surv Tutor, 2014, 16: 1617–1634
6 McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comp Commun Rev, 2008, 38: 69–74
7 Jain S, Kumar A, Mandal S, et al. B4: experience with a globally-deployed software defined WAN. ACM SIGCOMM Comp Commun Rev, 2013, 43: 3–14
8 Xu M W, Li Q, Yang Y, et al. Self-healing routing: failure, modeling and analysis. Sci China Inf Sci, 2011, 54: 609–622
9 Wu J P, Ren G, Li X. Building a next generation Internet with source address validation architecture. Sci China Ser F-Inf Sci, 2008, 51: 1681–1691
10 Li X, Bao C X. Address switching: reforming the architecture and traffic of Internet. Sci China Ser F-Inf Sci, 2009, 52: 1203–1216
11 Mao Z M, Rexford J, Wang J, et al. Towards an accurate AS-level traceroute tool. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. Karlsruhe: ACM, 2003.

365–378
12 Zhang Y, Oliveira R, Wang Y, et al. A framework to quantify the pitfalls of using traceroute in AS-level topology measurement. IEEE J Sel Areas Commun, 2011, 29: 1822–1836
13 Luckie M, Hyun Y, Huffaker B. Traceroute probe method and forward IP path inference. In: Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement. Seattle: ACM, 2008. 311–324
14 Handigol N, Heller B, Jeyakumar V, et al. Where is the debugger for my software-defined network? In: Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks. Helsinki: ACM, 2012. 55–60
15 Agarwal K, Rozner E, Dixon C, et al. SDN traceroute: tracing SDN forwarding without changing network behavior. In: Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking. Chicago: ACM, 2014. 145–150
16 Handigol N, Heller B, Jeyakumar V, et al. I know what your packet did last hop: using packet histories to troubleshoot networks. In: Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation. Seattle: USENIX Association, 2014. 71–85
17 Durairajan R, Sommers J, Barford P. OFf: bugspray for openflow. In: Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking. Chicago: ACM, 2014. 225–226
18 Wundsam A, Levin D, Seetharaman S, et al. OFRewind: enabling record and replay troubleshooting for networks. In: Proceedings of the USENIX Annual Technical Conference. Portland: USENIX Association, 2011
19 Peresini P, Kuzniar M, Kostic D. Monocle: dynamic, fine-grained data plane monitoring. In: Proceedings of the 11th International Conference on Emerging Networking EXperiments and Technologies. Heidelberg: ACM, 2015. In press, doi: http://dx.doi.org/10.1145/2716281.2836117
20 Fayaz S K, Sekar V. Testing stateful and dynamic data planes with FlowTest. In: Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking. Chicago: ACM, 2014. 79–84
21 Canini M, Venzano D, Peresini P, et al. A NICE way to test OpenFlow applications. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation. San Jose: USENIX Association, 2012. 127–140
22 Al-Shaer E, Al-Haj S. FlowChecker: configuration analysis and verification of federated OpenFlow infrastructures. In: Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration. Chicago: ACM, 2010. 37–44
23 Scott R C, Wundsam A, Zarifis K, et al. What, Where, and When: Software Fault Localization for sdn. Technical Report UCB/EECS-2012-178. 2012
24 Kazemian P, Varghese G, McKeown N. Header space analysis: static checking for networks. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation. San Jose: USENIX Association, 2012. 113–126
25 Kazemian P, Chan M, Zeng H, et al. Real time network policy checking using header space analysis. In: Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation. Lombard: USENIX Association, 2013. 99–111
26 Khurshid A, Zhou W, Caesar M, et al. Veriflow: verifying network-wide invariants in real time. ACM SIGCOMM Comp Commun Rev, 2012, 42: 467–472
27 Zeng H, Zhang S, Ye F, et al. Libra: divide and conquer to verify forwarding tables in huge networks. In: Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation. Seattle: USENIX Association, 2014. 87–99
28 Nichols K, Black D L, Blake S, et al. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474. https://www.ietf.org/rfc/rfc2474.txt. 1998
29 Katz-Bassett E, Madhyastha H V, Adhikari V K, et al. Reverse traceroute. In: Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation. San Jose: USENIX Association, 2010. 219–234
30 Gregori E, Improta A, Lenzini L, et al. On the incompleteness of the AS-level graph: a novel methodology for BGP route collector placement. In: Proceedings of the 12th ACM SIGCOMM Internet Measurement Conference. Boston: ACM, 2012. 253–264
31 Luckie M, Huffaker B, Dhamdhere A, et al. AS relationships, customer cones, and validation. In: Proceedings of the Internet Measurement Conference. Barcelona: ACM, 2013. 243–256
32 Gupta A, Vanbever L, Shahbaz M, et al. Sdx: a software defined Internet exchange. In: Proceedings of the ACM Conference on SIGCOMM. Chicago: ACM, 2014. 551–562