Journal of Computers Vol. 30 No. 5, 2019, pp. 98-110
doi:10.3966/199115992019103005008

Traffic Engineering Algorithms between Different Tunnels: Stateful and Stateless

Jun Zhao[1*], Congxiao Bao[2], Xing Li[1]

[1] Department of Electronic Engineering, Tsinghua University, Haidian District, Beijing, China
zhaojun12@mails.tsinghua.edu.cn, xing@cernet.edu.cn

[2] Information Technology Center, Tsinghua University, Haidian District, Beijing, China
congxiao@cernet.edu.cn

Received 20 November 2017; Revised 19 March 2018; Accepted 3 May 2018

Abstract. More and more networking services create overlay network using physical links between geo-distributed data centers and many kinds of tunnel technologies, to make full use of link resources between data centers. To achieve efficient construction of overlay network and flexible traffic engineering between different tunnels in overlay network, this paper presents the design Software Overlay Router (SOR), which is used to relay packets. This paper also presents a stateful and a stateless algorithms based on OpenFlow protocol to establish a tunnel without the requirement of global IP address in a simple and scalable manner. The stateful algorithm modifies incoming packets according to a database of four tuples mapping relationship, then forwards modified packets based on kernel routing table through established tunnel, and finally reconstructs received packets and transfers to the original destination. While stateless algorithm benefits from using IP transition algorithm without the need of maintaining four tuples mapping relationship, which largely reduces the number of OpenFlow rules need to be updated. The experiment shows the SOR and the algorithms proposed in this paper can effectively improve utilization of network link resources.

Keywords: OpenFlow, OpenVPN, overlay network, software defined networking, traffic engineering

1 Introduction

Software Defined Networking (SDN) [1] is a newly proposed network architecture, which has been widely used to redesign and accelerate traditional network [2]. This is done by decoupling the system that makes decisions about where traffic is sent (control plane) from the underlying systems that forward traffic to the selected destination (data plane) using OpenFlow protocol.

Studies [3] have shown that overlay networks can significantly improve network performance. With the development of SDN technology, more and more applications [4] create overlay network using physical links between geo-distributed data centers and many kinds of tunnel technologies, to make full use of link resources between data centers, in order to provide better service quality.

Thanks to the considerable flexibility offered by SDN, many technologies have been proposed to build overlay network, which can be divided into two categories: header rewriting [5-7] and packet encapsulation [8-9].

In the case of header rewriting [5-7], packet headers are replaced by special short tags. Kannan and Banerjee [5] introduces the notation of Flow-ID, where all packets belong to a flow are identified using this unique Flow-ID, and leverages the capabilities of the SDN to dynamically allow switching devices to route the packets based on the Flow-IDs. Similar to [5], Kawashima [6] proposes a header rewriting method using VLAN-ID to identify packets belong to the same flow. Mina [7] proposes a scalable VPN gateway, which relays packets from internet to a virtual machine (VM) with private IP address located in

* Corresponding Author

data center. The VPN gateway can handle hundreds of tunnels simultaneously leveraging a commodity switch. Header rewriting technology has several disadvantages. First, all the switches along the routing path must recognize the matching field [5-7], e.g., FLOW-ID and VLAN-ID, thus header rewriting is mostly used in a fully controlled intra-data center network. Second, header rewriting cannot relay packets through Network Address Translation (NAT) devices [5-6]. Third, although literature [7] can forward traffic through NAT, it requires a special hardware to achieve that.

However, in the case of packet encapsulation [8-9], the whole Ethernet frame is encapsulated into another IP packet. Yang et al. [8] design a tunnel broker, which can build overlay network based on OpenVPN tunnel to avoid the requirement of global IP address. Rodriguez et al. [9] present the design of an open source software overlay router (OOR) using Locator/ID Separation Protocol (LISP) as tunnel protocol, and claims that LISP does not reduce efficiency given that LISP encapsulation typically adds 36 bytes (IPv4) or 56 bytes (IPv6) extra length. Virtual Extensible LAN (VXLAN) is a traditional technology used to build overlay network. Packet encapsulation technology also has some disadvantages. First, traditional encapsulation method VXLAN is based on UDP protocol, which has performance issues due to that Internet Service Provider (ISP) usually limits the transmission rate of UDP protocol. Second, although literature [8] can establish OpenVPN tunnel, the forwarding rules should be manually configured, which has scalability issues. Third, although OOR [9] can build tunnel through NAT, it needs some modifications on NAT devices in the middle of the path.

So for the ease of building overlay network, three aspects should be taken into consideration. First, the maximum transfer rates of one path achieved by different kinds of tunnels may differ largely among each other. Overlay router should be able to build different kinds of tunnels and establish specific tunnel according to link measurement results. Second, due to security consideration, many hosts are connected to internet through NAT devices. Overlay router should be able to build tunnels through NAT without the requirement of global IP address. Third, overlay routers may build several different tunnels with each other simultaneously, so an algorithm, which is able to flexibly relay packets through different tunnels is urgently needed.

We claim that the technologies mentioned above [5-9] cannot fully meet the above three requirements. So in this paper, we propose a Software Overlay Router (SOR). To make full use of link bandwidth resources, SOR can build at least 3 kinds of tunnels, such as Generic Routing Encapsulation (GRE), VXLAN and VPN (TCP protocol based). In order to establish tunnel through NAT, the SOR behind NAT device initially sends requests to other SORs to build VPN tunnel. A stateful algorithm and a stateless algorithm have been proposed to forward packets through different kinds of tunnels. Table 1 summaries the advantages and disadvantages of technologies mentioned above.

Table 1. Differences between related works and SOR

Compared work	NAT support	Dynamic routing	Bandwidth utilization	Deployment scenario
[5]	No	Yes	Normal	Intra-datacenter
[6]	No	Yes	Normal	Intra-datacenter
[7]	No	Yes	Normal	Intra-datacenter
VXLAN	Yes	Yes	Low	Internet[1]
[8]	Yes	No	High	Internet
[9]	Yes[2]	Yes	Normal	Internet
SOR	Yes	Yes	High	Internet

This paper extends our previous work [10]. Our contributions are four folds. First, we design a Software Overlay Router (SOR), which is able to establish several different kinds of tunnels through NAT with other SORs simultaneously and increase utilization of link resources. Second, we introduce a stateful traffic engineering algorithm, which is based on OpenFlow protocol to establish a tunnel without the requirement of global IP address in a simple and scalable manner. The stateful algorithm modifies incoming packets according to a database of four tuples mapping relationship, then forwards modified packets based on kernel routing table through established tunnel, and finally reconstructs received packets and transfers to the original destination. Third, we devise a stateless traffic engineering algorithm,

[1] The word *Internet* means that there is no limitation in deployment scenario.

[2] In order to support traversing through NAT, OOR needs some modifications on NAT devices.

which benefits from using IP transition algorithm without the need of maintaining four tuples mapping relationship, thus largely reduces the number of OpenFlow rules need to be updated. Fourth, an extensive evaluation of SOR proves that SOR can achieve high throughput and effectively improve utilization of network link resources.

This paper is organized as follows. Section 2 exposes the design of Software Overlay Router (SOR) and implementation related issues. Section 3 contains the details of stateful traffic engineering algorithm proposed in this paper, while stateless one is explained in section 4. The performance evaluation is presented in section 5. In section 6, some useful use cases are discussed, and finally section 7 draws the conclusion of this paper.

2 Design of Software Overlay Router

In order to solve problems mentioned above, we implement Software Overlay Router, shown in Fig. 1, to create tunnels between hosts distributed all over the world based on the mechanism proposed in this paper. For brevity, we firstly summarize the notations that will be used in Table 2. Our mechanism is still based on packet encapsulation technology, which has no limitations in deployment scenarios.

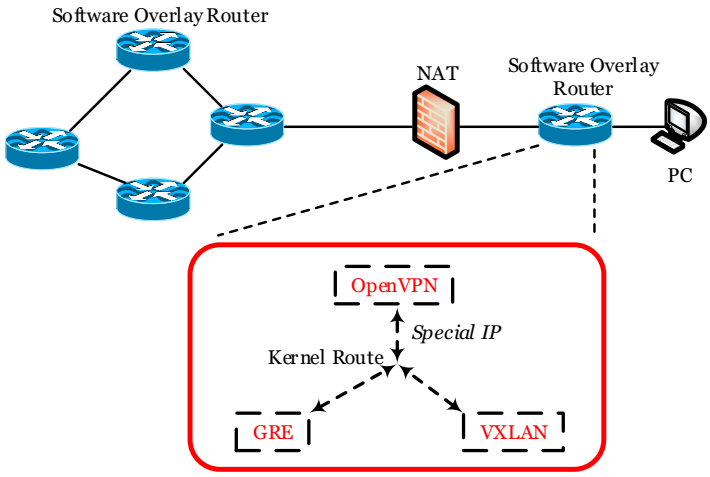


Fig. 1. Architecture of Software Overlay Router

Table 2. Notations Used in This Paper

Notation	Description
SIP_T	the special IPv4 allocated for tunnel T ($SIP_{vxlan}$for VXLAN and SIP_{vpn} for VPN)
$SIPv6_T$	the special IPv6 allocated for tunnel T ($SIPv6_{vxlan}$ for VXLAN and $SIPv6_{vpn}$ for VPN
LOCAL	refer to the OpenFlow local port
NORMAL	process the packet using traditional Layer 2 or Layer 3 processing
D_{ip}	the destination IP address of forwarding packet
D_{port}	the destination port of forwarding packet
S_{ip}	the source IP address of forwarding packet
S_{port}	the source port of forwarding packet
$H_{T,s}$	the source end host of tunnel T
$H_{T,d}$	the destination end host of tunnel T
$PORT_{T,s}$	the global unique destination port allocated for modified packet in host $H_{T,s}$
IP_H	the IP address of host H
$PORT_H$	the global unique source port allocated for NAT in host H
IFVIS	newly defined OpenFlow action used translate IPv4 packet to IPv6 format with Prefix and Length as input
ISVIF	newly defined OpenFlow action used translate IPv6 packet to IPv4 format with Prefix and Length as input
LEN(ip)	the function used to calculate length of Prefix with *ip* as input

2.1 Tunnel Protocols Chosen

GRE can encapsulate a wide variety of network layer protocols inside virtual point-to-point links over an Internet Protocol network. VXLAN uses a VLAN-like encapsulation technique to encapsulate MAC-based layer 2 Ethernet frames within layer 4 UDP packets, using 4789 as the default destination UDP port number. Neither of them can make full use of the link bandwidth resources.

This paper introduces TCP based OpenVPN tunnel, based on the following two intuitions. First, TCP protocol has a traffic congestion control algorithm, which can make full use of high bandwidth networks. Second, OpenVPN supports that host with a private IP address actively establishes tunnel with host in global network. In a nutshell, overlay router should have the ability of establishing at least three different tunnels, such as GRE, VXLAN and OpenVPN.

2.2 Architecture of Software Overlay Router

Based on the concept of SDN, SOR locates in the data plane, focusing on traffic transmission between different tunnels. Open source software Virtual Switch, usually called Open vSwitch, is currently widely used in traffic engineering system, which supports forwarding traffic between GRE and VXLAN tunnels by installing OpenFlow rules. However, OpenVPN tunnel cannot be controlled by OpenFlow protocol, making it difficult to control traffic transmission between GRE and OpenVPN, VXLAN and OpenVPN using OpenFlow protocol.

In order to solve above problem, this paper proposes the SOR architecture shown in Fig. 1. As discussed above, each SOR establishes GRE, VXLAN and OpenVPN tunnels with other SORs. SOR achieves traffic scheduling between different tunnels through pre-configured Linux kernel routing table rules.

Simply using pre-configured Linux kernel routing table to achieve traffic engineering between different tunnels has scalability issues. First, there are billions of IP addresses, all of which can be destination address of IP packets. What is more, there are even 750,000 IP blocks based on the prefixes advertised by BGP router, which is extremely large. Second, packets' destination addresses are dynamically changing all the time, and even for the same destination address, packets may be transferred through different tunnels at different time. Dynamically updating Linux kernel routing table definitely increases the complexity of SOR design.

2.3 Concept of Special IP

In order to solve the problem of Linux kernel routing table scalability, this paper introduces the concept of Special IP. SOR pre-allocates an IP block for each type of tunnel, named as Special IP, which can be either a specific IP address, or an IP block represented by CIDR.

If SOR needs to create three types of tunnels, it needs pre-allocate three Special IPs, and pre-configure in the kernel routing table. In a nutshell, it only needs to configure three static routing rules in the kernel, which is a perfect solution to the problem of hundreds of thousands of IP address prefixes.

When traffic needs to be forwarded through certain tunnel, it only needs to modify the destination IP address of the packets to the Special IP allocated for that tunnel, which excellently solves the problem of dynamically changing destination IP addresses.

2.4 Implementation of SOR

We have successfully implemented SOR in Raspberry PI 3 Model B and Amazon AWS EC2 virtual machine to demonstrate the largely deployment feasibility of SOR.

SOR consists of a variety of open source software. Table 3 lists the software used to build SOR and their corresponding versions. Open vSwitch software is used to establish GRE and VXLAN tunnels, while OpenVPN is used to build TCP based VPN tunnel. RYU is used as the central controller of SOR, and forwards traffic among tunnels by installing specific OpenFlow rules.

Table 3. Open source software used in SOR

No.	Open Source Software	Version
1	Open vSwitch	2.6.0
2	OpenVPN	2.3.2
3	RYU	4.6

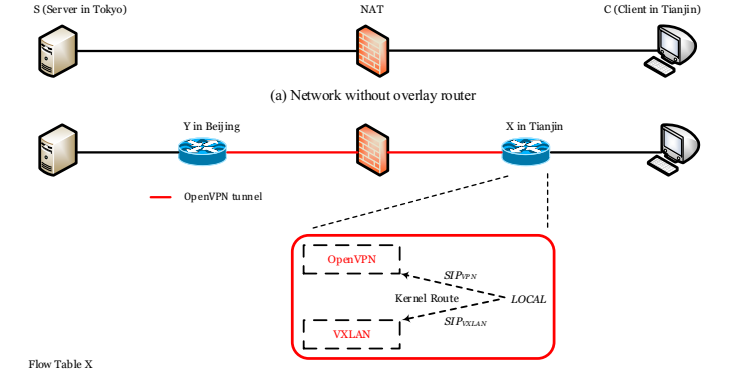
On the choice of Special IP, in order to avoid conflicts with the commonly used private IP address block (10/8, 176.16/12, 192.168/16), SOR uses preserved IP address block 100.64/10 as Special IP allocation pool. For example, SOR pre-allocates 100.64.0./24, 100.64.1/24, 100.64.2/24 for GRE, VXLAN, and OpenVPN respectively.

3 Stateful Mechanism

In this section, we give a detailed explanation of stateful algorithm.

3.1 An Example of Mechanism

In this subsection, we give an example to illustrate the procedure to forward traffic by building OpenVPN tunnel through NAT. Fig. 2(a) shows the network topology without Software Overlay Router, in which, client (C) behind NAT directly communicates with remote server (S) through underlay network. In Fig. 2(b), traffic between client and server is forwarded through overlay path C-X-Y-S with Software Overlay Router. When packet with IP_S as destination IP address, $PORT_S$ as destination port, is sent from C, the flow entries are shown in Fig. 2(b).



(a) Network without overlay router

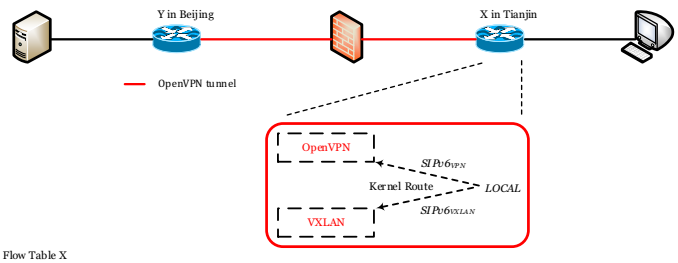
Flow Table X

Matching Field	Action
DST IP = IP_S; DST Port = $PORT_S$; SRC IP = IP_C; SRC Port = $PORT_C$	Set DST IP = SIP_{VPN}; Set DST Port = $PORT_X$; Output: *LOCAL*

Flow Table Y

Matching Field	Action
DST IP = SIP_{VPN}; DST Port = $PORT_X$; SRC IP = IP_C; SRC Port = $PORT_C$	Set DST IP = IP_S; Set DST Port = $PORT_S$; Output: *LOCAL*
DST IP = IP_S; DST Port = $PORT_S$; SRC IP = IP_C; SRC Port = $PORT_C$	Set SRC IP = IP_Y; Set SRC Port = $PORT_Y$; Output: *NORMAL*

(b) Network with overlay router and stateful algorithm



Flow Table X

Matching Field	Action
DST IP = IP_S; DST Port = $PORT_S$; SRC IP = IP_C; SRC Port = $PORT_C$	IFVIS [Prefix = $SIPv6_{VPN}$, Length = $LEN(SIPv6_{VPN})$]; Output: *LOCAL*

Flow Table Y

Matching Field	Action
DST IPv6 = $IPv6$ [Prefix = $SIPv6_{VPN}$, Length = LEN($SIPv6_{VPN}$)]	ISVIF [Prefix = $SIPv6_{VPN}$, Length = $LEN(SIPv6_{VPN})$]; Output: *LOCAL*
DST IP = IP_S; DST Port = $PORT_S$; SRC IP = IP_C; SRC Port = $PORT_C$	Set SRC IP = IP_Y; Set SRC Port = $PORT_Y$; Output: *NORMAL*

(c) Network with overlay router and stateless algorithm

Fig. 2. An example of SOR

We allocate two Special IPs for OpenVPN and VXLAN tunnels separately (i.e., SIP_{vpn} and SIP_{vxlan}), and statically configure the Linux kernel routing table for the two tunnels in X. In SOR X, packets are forwarded through OpenVPN tunnel after modifying destination IP address to SIP_{vpn} and destination port to $PORT_X$. Based on the matching fields, SOR Y resets the destination IP address and port to original ones (i.e., IP_S and $PORT_S$) before forwarding collaboratively.

3.2 Detail of Mechanism

The core idea of Algorithm 1 is to reconstruct the modified packets at the end host of tunnel by maintaining the packets' four tuples mapping relationship in the controller.

Algorithm 1. Stateful Mechanism

Input: To be forwarded packet with D_{ip} as destination IP address and D_{port} as destination port
Output: Installing new flow entries on tunnel end hosts $H_{T,S}$ and $H_{T,d}$

1.
2. Install one flow entry on tunnel end host $H_{T,S}$, where the matching field is original destination IP address and port, the action is configured as follows:
   (1) set destination IP address to SIP_T
   (2) set destination port to $PORT_{T,S}$
   (3) set output port to *LOCAL*
3. Install one flow entry on tunnel end host $H_{T,d}$, where the matching field is Special IP address for tunnel *T* and port, the action is configured as follows:
   (1) set destination IP address to D_{ip}
   (2) set destination port to D_{port}
   (3) set output port to *LOCAL*
4. Install one flow entry on tunnel end host $H_{T,d}$, where the matching field is original destination IP address and port, the action is configured as follows:
   (1) set source IP address to IP_Y
   (2) set source port to $PORT_Y$
   (3) set output port to *NORMAL*

When host C wants to communicate with host S through tunnel *T*, it is only necessary to modify the destination IP address of packet to the Special IP SIP_T assigned by host $H_{T,S}$ for tunnel *T* and the destination port to the globally unique port $PORT_{T,S}$ of host $H_{T,S}$. In this way, the modified packet will be forwarded through tunnel *T* after searching the Linux kernel routing table. According to the packet's four tuples mapping relationship maintained by controller, host $H_{T,d}$ will reconstruct the modified packet, and transfer the packet to the original destination after NAT process based on OpenFlow rule (Line 4).

Because the Alg.1 needs to maintain a 4 tuple mapping relationship of TCP/UDP packets in the controller, it is also referred as a stateful traffic engineering algorithm. If the number of flows need to be updated is *N*, then the total number of OpenFlow rules need to be installed can be defined as Eq. (1).

$$N_{stateful} = 3 \times N. \quad \textbf{(1)}$$

Unlike physical switch, which stores flow entries in Ternary Content Addressable Memory (TCAM) limited in size, SOR keeps updated OpenFlow rules in memory, making it easier to handle much larger number of packets simultaneously.

4 Stateless Mechanism

In this section, the detailed explanation of stateless algorithm will be given.

4.1 IP Translating Actions

In OpenFlow protocol, a flow rule has 2 parts: matching field and action. We implement two new actions named *IFVIS* and *ISVIF*, which respectively mean translating packet header from IPv4 to IPv6 and from

IPv6 to IPv4, using stateless translating algorithm [11]. We take a simple case as an example shown in Fig. 3.

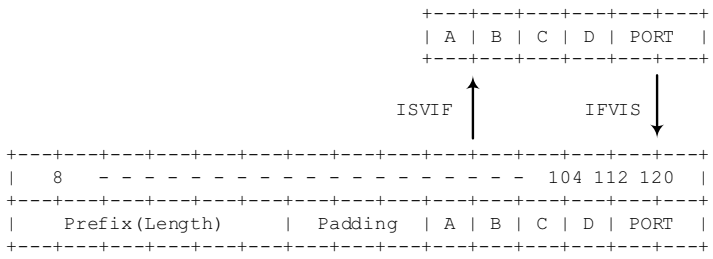


Fig. 3. An example of IFVIS and ISVIF actions

For ease of implementation, both *IFVIS* and *ISVIF* take IPv6 prefix and prefix length as input parameters. *IFVIS* action encapsulates 4 bytes IPv4 address (i.e., A.B.C.D) and 2 bytes port (i.e., PORT) into the last 6 bytes of IPv6 address, and adds IPv6 prefix (i.e., Prefix) to the first. If the Length is less than 80 bits, then spare space will be padded with 0. While *ISVIF* simply extracts IPv4 address and port information from the last 6 bytes, which is much simpler.

4.2 An Example of Mechanism

Similar with example shown in Fig. 2(b), in Fig. 2(c) traffic between client and server is forwarded through overlay path C-X-Y-S with IP_S as destination IP address, $PORT_S$ as destination port.

Algorithm 2. Stateless Mechanism

Input: To be forwarded packet with D_{ip} as destination IP address and P_{port} as destination port
Output: Installing new flow entries on tunnel end hosts $H_{T,S}$ and $H_{T,d}$

1. Install one flow entry on tunnel end host $H_{T,S}$, where the matching field is original destination IP address and port, the action is configured as follows:
   (1) *IFVIS* action with $SIPv6_T$ Prefix and Length as parameters
2. Install one flow entry on tunnel end host $H_{T,d}$, where the matching field is Special IPv6 address for tunnel *T* and port, the action is configured as follows:
   (1) *ISVIF* action with IPv6 Prefix and Length as parameters
3. Install one flow entry on tunnel end host $H_{T,d}$, where the matching field is original destination IP address and port, the action is configured as follows:
   (1) set source IP address to IP_Y
   (2) set source port to $PORT_Y$
   (3) set output port to *NORMAL*

On the contrary, we allocate two Special IPv6 addresses for OpenVPN and VXLAN tunnels separately (i.e., $SIPv6_{vpn}$ and $SIPv6_{vxlan}$), and statically configure the Linux kernel routing table for these two tunnels in X. In SOR X, packets are forwarded through OpenVPN tunnel after translating original IPv4 packets to IPv6 packets using *IFVIS* action, based on the previously configured IPv6 kernel routing table rules. SOR Y reconstructs the original destination IPv4 address and port (i.e., IP_S and $PORT_S$) using *ISVIF* action before forwarding.

4.3 Analysis of Mechanism

Unlike stateful algorithm, stateless mechanism aims to release controller from maintaining four tuples mapping relationship through using *IFVIS* and *ISVIF* actions. Due to that IPv6 space is much larger than IPv4 address, we can simply reserve the mapping relationship by coding it into IPv6 header as shown in Fig. 3 (Line 1 in Alg.2). Host $H_{T,d}$ will rebuild original IPv4 header by examining the last 6 bytes (Line 2 in Alg.2), before doing NAT processing (Line 3 in Alg.2).

As long as the destination IPv6 addresses of all flows forwarded through tunnel T belong to IPv6 address space with Prefix and Length in SOR Y, we only need install a single OpenFlow rule, no matter

how many traffic flows are to be forwarded, which significantly reduces the number of OpenFlow rules need to be updated. If the number of flows need to be forwarded is N, then the total number of OpenFlow rules need to be installed can be defined as Eq. (2).

$$N_{stateless} = N + 2. \tag{2}$$

$N_{stateless}$ is only a third of $N_{stateful}$. As the number of N grows larger, the reduced number of OpenFlow rules is significantly considerable.

4.4 Distributed Controllers

With centralized controller, when a flow enters SOR X, the time it will take to construct overlay path for transferring flow traffic equals the largest RTT between SORs and controller, which may be several hundred milliseconds. In the meanwhile, installing 3 OpenFlow rules for each flow traffic introduces additional costs. What is worse in stateful algorithm is that the controller needs to maintain four tuples mapping relationship, which adds extra loads.

Unlike stateful algorithm, in stateless algorithm controller does not need to maintain global unique port $PORT_{T,S}$, making it easy to deploy distributed controllers. What is more, the OpenFlow rules installed in SOR X and SOR Y have no relationship except the same $SIPv6_T$, which is statically allocated previously. So we can deploy distributed controllers in geo-distributed SORs locally, that is each controller controls SOR in the same virtual machine, and the controllers can work cooperatively perfectly without the need to exchange information with each other.

Fig. 4 shows the time model consumed by stateful and stateless algorithms when updating OpenFlow rules. In stateful algorithm, most time is spent when propagating **Packet_In** message from SOR to centralized controller and send **Flow_Mod** message in response, which is defined as Eq. (3).

$$T_{stateful} = T_{CAL} + T_{X,C} + mx\{T_{C,Y}, T_{C,X}\}. \tag{3}$$

The time spent by stateless algorithm can be expressed as Eq. (4).

$$T_{stateless} = T_{UP} + 2 \times T_{CAL}. \tag{4}$$

$$T_{UP} = T_{X,CX} + T_{CX,X} + T_{Y,CY} + T_{CY,Y}. \tag{5}$$

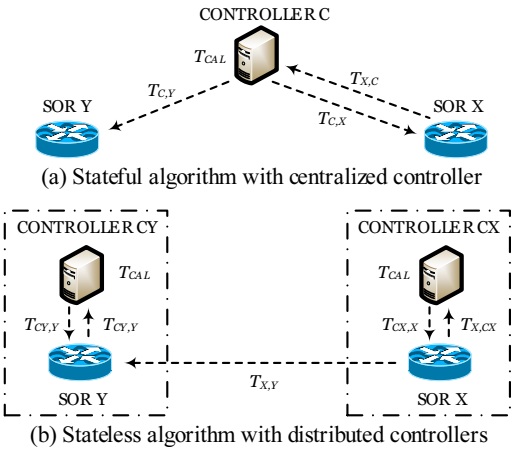


Fig. 4. Time to update OpenFlow rules

T_{CAL} represents the time spent by stateful and stateless algorithms when calculating appropriate OpenFlow rules. T_{CAL} in stateful and stateless algorithms are approximately equal with each other, while controllers in stateless algorithm are collocated with SORs in the same servers, making the four parameters (i.e., $T_{X,CX}$, $T_{Y,CY}$, $T_{Y,CY}$ and $T_{CY,Y}$) less than 1 millisecond measured by ping command, comparing with $T_{C,X}$ or $T_{C,Y}$, which may be several hundred milliseconds. The detailed comparison will be explained in the evaluation section.

5 Performance Evaluation

In this section, we investigate the performance gained by using SOR and above mechanism to build Overlay Network.

5.1 Kernel Route Lookup

To find out how Linux kernel performance route lookup is, we can accurately benchmark the **fib_lookup()** function in Linux kernel. The measurement is done in a virtual machine with eight 1.6 GHz vCPUs, 64 GB RAM and Linux kernel 4.11.0-14.

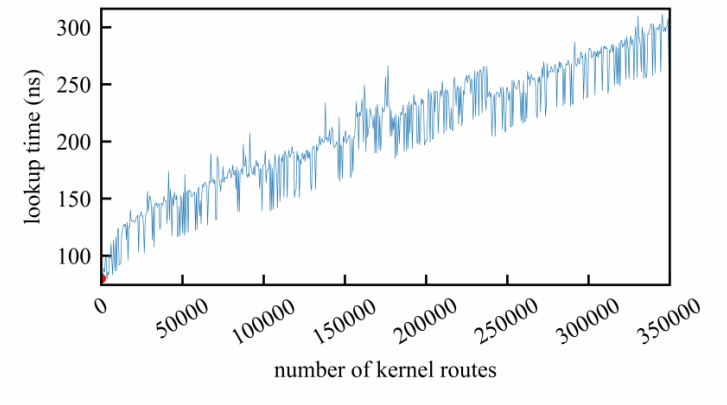


Fig. 5. Kernel route lookup performance

Fig. 5 shows the kernel route lookup performance. The lookup time increases linearly as the number of kernel routes grows. As there are billions of IP addresses, the lookup time may be several micro seconds, which cannot be neglected. The stateful and stateless algorithms only need install several static kernel routes, making the lookup time less than 80 ns represented by red point (left bottom corner of Fig. 5).

5.2 Throughput

Here we focus on the throughput of SOR's data plane. The SOR is deployed in two Intel Core PCs (3.2 GHz CPU and 4 GB RAM), both of which are connected through a Gigabit Ethernet link. Traffic is generated using iPerf tool (https://iperf.fr/), and we monitor both input and output rates. As Fig. 6 shows, SOR scales close to the link capacity with a maximum throughput of 850 Mbps.

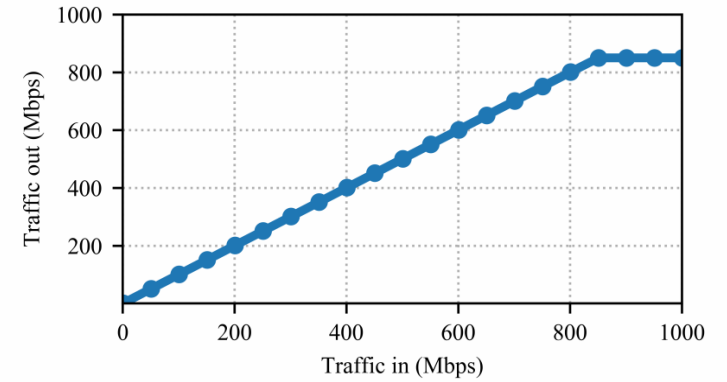


Fig. 6. Throughput

5.3 Real World Experiment

In the emulation environment shown in Fig. 2, two SORs (represented by X and Y, Pentium Dual-Core E6700 CPU 3.20GHz, RAM 8GB, and Ubuntu 14.04 64bit System) are deployed, supporting establish VXLAN tunnel using UDP protocol and OpenVPN tunnel using TCP protocol. SOR X, located in Tianjin, plays a role as the source end host of OpenVPN tunnel, through which the client connects to the Internet. While SOR Y, located in Beijing, forwards packets received from OpenVPN tunnel to the original destination acting as an overlay router.

In the emulation, we use iPerf tool to measure link bandwidth and packet loss rate between client and server, and compare results measured under network with SOR and without SOR environment. SOR Y is located in CERNET data center in Beijing, with 1000Mbps upload bandwidth, so link between SOR Y and Server in Tokyo will not become the bottleneck of this evaluation. On the contrary, the outbound link bandwidth of SOR X (or Client) in Tianjin is only 100Mbps rate, which may have a great impact on the measurement results.

Fig. 7 shows the link bandwidth between Client and SOR Y during one day time measured using UDP and TCP protocols separately, similar with results between Client and Server, which are not shown in this paper. From Fig. 7 we can see that TCP protocol can consume total link bandwidth achieving 90Mbps traffic transferring rate, while bandwidth measured by UDP protocol is only about 16Mbps. The reason behind this is that UDP protocol is often used by multi-player online gaming and video conferencing system, for which 16Mbps is totally enough. Due to the reason mentioned above, ISP usually limits the speed of UDP protocol to more properly schedule bandwidth resource allocation among different protocols.

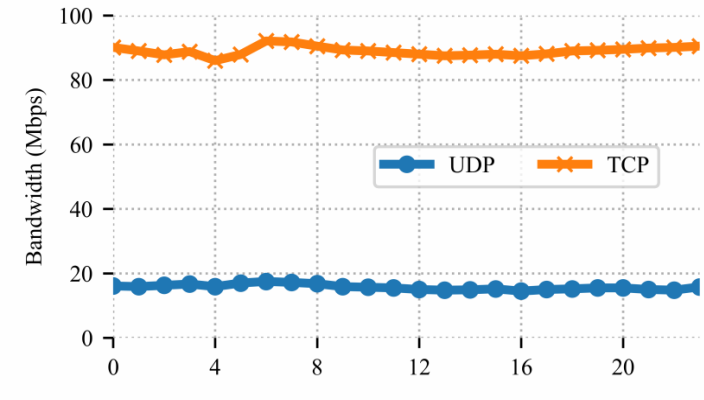


Fig. 7. Bandwidth between Tianjin and Beijing

Fig. 8 compares the link bandwidth between Client and Server achieved under network with SOR and without SOR environment. The metrics are collected by iperf tool using UDP protocol. The bandwidth achieved by iperf in network without SOR is quite similar with Fig. 7, which proves that bandwidth between SOR Y and Server is not the performance bottleneck from another perspective. However, bandwidth under overlay network reaches 70Mbps, a litter lower than 90Mbps. This is because that except original packet with IP header destined for Server, OpenVPN tunnel will encapsulate original packet as payload of TCP header destined for SOR Y, which increases the total length of packet forwarding along overlay network path, thus decreases actual bandwidth achieved by iperf tool. It is still much larger than bandwidth achieved without SOR, though OpenVPN tunnel definitely adds extra costs.

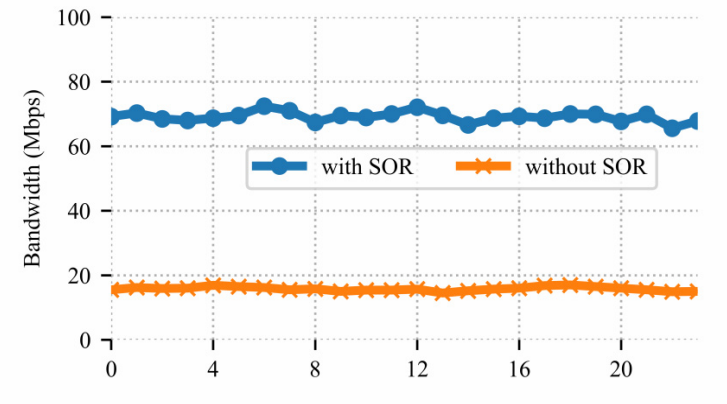


Fig. 8. Bandwidth between Tianjin and Tokyo with and without SOR

The packet loss rate under circumstances with SOR and without SOR is depicted in Fig. 9, which is calculated by sending stream at 32Mbps rate. As can be seen from measurement results, the packet loss rate is constantly zero under network with SOR, because that bandwidth is approximately 70Mbps, much larger than the sending rate. While in network without SOR, the packet loss is around 56%, due to that available bandwidth is only 16Mbps, much lower than 32Mbps.

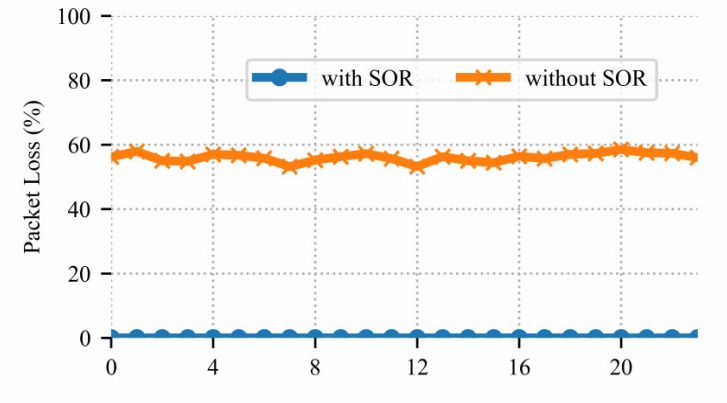


Fig. 9. Packet loss between Tianjin and Tokyo with and without SOR

The controller uses the stateful algorithm in the above experiment. Due to space limitation, the results gained by stateless algorithm are omitted, which are similar to the ones measured using stateful algorithm.

Fig. 10 shows the number of OpenFlow rules installed when using stateful and stateless algorithms separately. From Fig. 10, we can see that the number of rules installed by stateless algorithm is only a third of the number installed by stateful algorithm, which proves the accuracy of Equations (1) and (2).

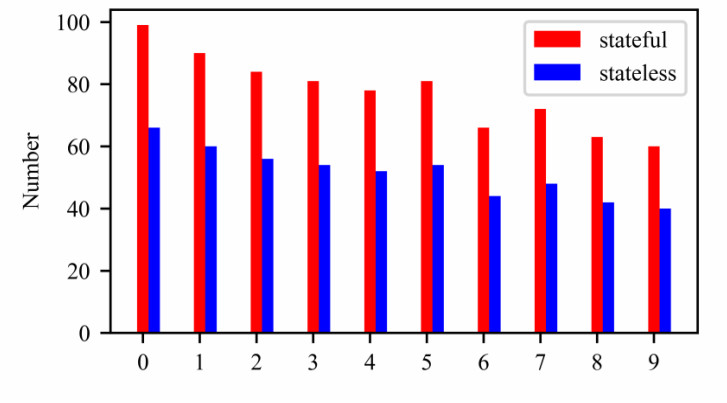


Fig. 10. Number of OpenFlow rules installed

When using stateful algorithm with centralized controller, the largest RTT between controller and SOR is about 58ms, while the latency experienced by stateless algorithm with distributed controllers is merely 1ms, shown in Fig. 11. In QoS system, flows must be transferred without experiencing packets loss. In order to achieve this, several Megabytes size of packets must be buffered in SOR locally, due to several hundred milliseconds latency before the overlay path can be established, which leads SOR consuming much more memory. However, the situation is improved much in stateless algorithm.

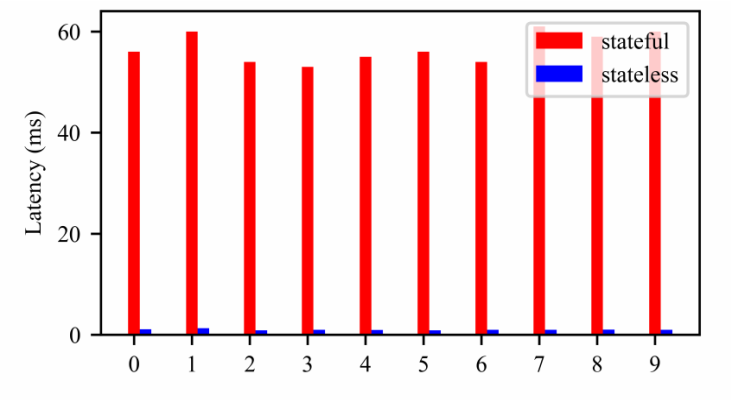


Fig. 11. Time taken to build overlay path

6 Discussion

The mechanism explained in this paper can be used to build a tunnel using OpenVPN through NAT. In the meantime, although native OpenVPN tunnel does not support OpenFlow protocol, this paper proposes the concept of Special IP to achieve controlling traffic traversing through OpenVPN tunnel without modifying OpenVPN source code in a scalable manner, which further reduces the development and maintenance costs.

QoS guarantee is one of the most important use cases. Companies always lease proprietary network link from ISP to achieve efficient access to the internet, which usually costs a lot of money. With the help of mechanism proposed in this paper, we can build multiple tunnels to the servers located geo-distributed in the world simultaneously, to make full use of link bandwidth resources, based on the intuition that the closer the distance between two servers is, the higher the bandwidth will be [12]. We need implement traffic classification mechanism in the controller to classify traffic into different SLA groups, and forward packets through different tunnels based on the tunnel link performance metrics collected by measurement tools in the background. At the same time, we can cluster traffic in the same SLA group into different clusters based on destination IP address, and forward packets through tunnel, whose server is nearest to original destination of packets, by modifying packets' destination IP address to Special IP corresponding to that specific tunnel. We can use sPing [13] to debug the SDN based overlay network. Through this way, companies can benefit from much more better quality of services.

7 Conclusion

This paper presents the design of Software Overlay Router (SOR), which is used to build overlay network using SDN technology. This paper also presents a stateful and a stateless algorithms based on OpenFlow protocol to establish a tunnel without the requirement of global IP address in a simple and scalable manner. The stateful algorithm modifies incoming packets according to a database of four tuples mapping relationship, then forwards modified packets based on Linux kernel routing table through established tunnel, and finally reconstructs received packets and transfers to the original destination. While stateless algorithm benefits from using IP transition algorithm without the need of maintaining four tuples mapping relationship, which largely reduces the number of OpenFlow rules need to be updated and the latency taken to establish overlay path. Finally we implement the two algorithms in SOR, totally based on Open Source software, which reduces the development and maintenance costs.

To evaluate performance gained by using SOR, this paper conducted several experiments by constructing a testing environment across several autonomous systems (AS). By measuring link bandwidth and packet loss rate under network with SOR and without SOR, we come to a conclusion that the algorithms presented in this paper can largely improve the utilization of network link resources. By comparing the two algorithms, it is concluded that stateless algorithm can greatly reduce the number of OpenFlow rules needed to be installed, the latency elapsed before the overlay path established and the memory usage of SORs.

The limitation of SOR is that it leverages OpenVPN to build TCP based VPN tunnel, which is CPU intensive. So the further study is to integrate Stateless Transport Tunneling (STT) [14] protocol into SOR. STT utilizes the capabilities of the network interface card to improve performance, the magic behind this is in its TCP like header. Through this way, we can further reduce the CPU usage of SOR without decreasing the performance.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38(2)(2008) 69-74.

[2] H.-H. Cho, C.-F. Lai, T. K. Shih, H.-C. Chao, Integration of SDR and SDN for 5G, IEEE Access 2(2014) 1196-1204.

[3] C. Cai, F. Cai, X. Sun, CRONets: Cloud-Routed Overlay Networks, in: Proc. 2016 International Conference on Distributed Computing Systems, 2016.

[4] Y. Liu, D. Niu, B. Li, Delay-optimized video traffic routing in software-defined inter-datacenter networks, IEEE Transactions on Multimedia 18(5)(2016) 865-878.

[5] K. Kannan, S. Banerjee, Scissors: dealing with header redundancies in data centers through SDN, in: Proc. 2012 International Conference on Network and Service Management. International Federation for Information Processing, 2012.

[6] R. Kawashima, H. Matsuo, Non-tunneling Edge-Overlay model using OpenFlow for cloud datacenter networks, in: Proc. 2013 IEEE International Conference on Cloud Computing Technology and Science, 2013.

[7] M.T. Arashloo, P. Shirshov, R. Gandhi, A scalable VPN gateway for multi-tenant cloud services, ACM SIGCOMM Computer Communication Review 48(1)(2018) 49-55.

[8] S. Yang, Q. Zhang, X. Li, A tunnel broker based IPv6 access system for a small scale network with IPv4 upstream, in: Proc. 2016 Information Technology, Networking, Electronic and Automation Control Conference, 2016.

[9] A. Rodriguez, J. Paillisse, F. Coras, Programmable overlays via openoverlayrouter. IEEE Communications Magazine 55(6)(2017) 32-38.

[10] J. Zhao, C. Bao, X. Li, Building a tunnel through NAT using OpenFlow, in: Proc. 2018 Proceedings of ACM Ubiquitous Information Management and Communication, 2018.

[11] G. Han, C. Bao, X. Li, S. Liu, IPv6 transition for the other billions, in: Proc. 2015 International Conference on Computer Communication and Networks, 2015.

[12] D. Ren, Y. Xu, S.H. Chan, Beyond 1mbps global overlay live streaming: the case of proxy helpers, ACM Transactions on Multimedia Computing Communications & Applications 11(2)(2015) 1-22.

[13] F.H. Tseng, K.D. Chang, S.C. Liao, H.C. Chao, V.C. Leung, sPing: a user-centred debugging mechanism for software defined networks, IET Networks 6(2)(2017) 39-46.

[14] Nicira Networks Inc., A stateless transport tunneling protocol for network virtualization (STT). <https://datatracker.ietf.org/doc/draft-davie-stt/>, 2016 (accessed 18.03.10).