Elastic Edge-Overlay Methods Using OpenFlow for Cloud Networks

Amer Aljaedi, C. Edward Chow and Jia Rao

Abstract The virtualization of cloud network requires flexible and effective techniques to accommodate the rapid changes in the network configurations and updates. OpenFlow protocol has attracted attentions for cloud networks since it facilitates managing and sharing the network resources, and it can be utilized to create an overlay abstraction on top of the network infrastructure for the flow setup in the cloud. However, the traditional reactive flow setup of OpenFlow introduces higher flow latency and overhead on the network controller. This paper discusses the issues of the reactive flow setup and presents two optimized overlay network virtualization methods that leverage OpenFlow to control and forward the tenants' traffic. The proposed methods enable tenants to use their own MAC/IP addresses in the cloud. We have implemented and evaluated the proposed overlay methods, and the experimental results show that our methods have less flow latency than the traditional reactive approach, and higher performance than the popular overlay tunneling protocols such as VXLAN, STT, and NVGRE.

Keywords OpenFlow · Overlay · Cloud network · Flow rules · Reactive rules · Proactive rules · Flow setup · Virtual network

1 Introduction

Network virtualization is one of the key components for the multi-tenancy services in the cloud that enables the cloud provider to satisfy the customer requirements. As thousands of tenants subscribe to the cloud services on a daily basis, the cloud has to handle a massive number of network configurations and updates. Therefore, the network virtualization in the cloud requires effective techniques for sharing the network infrastructure and maintaining an efficient

A. Aljaedi(✉) · C.E. Chow · J. Rao
Department of Computer Science, University of Colorado, Colorado Springs, USA
e-mail: {aaljaedi,cchow,jrao}@uccs.edu

S. Latifi (ed.), *Information Technology New Generations*,
Advances in Intelligent Systems and Computing 448,
DOI: 10.1007/978-3-319-32467-8_3

isolation of the tenants' traffic. It should support VMs migration to arbitrary locations in the cloud while it allows the cloud tenants to use their own addressing scheme and configure their virtual subnets.

The traditional network virtualization techniques such as VLAN cannot accommodate large multi-tenant datacenter due to its VLAN ID, 12 bits, which supports only 4096 virtual networks. Therefore, the cloud industry has adopted tunneling (L2-in-L3) overlay protocols such as VXLAN, NVGRE, and STT to address the VLAN limitations. These tunneling protocols encapsulate the whole Ethernet frame of VM in an IP packet in order to transmit the VM frame to its destination through a tunnel in the cloud physical network. This encapsulation technique hides the MAC/IP addresses of the tenant virtual network (TVN) from the cloud physical network, and it maps the tenants' virtual addressing schemes to the cloud physical topology. These tunneling protocols can accommodate millions of tenants in the cloud since the virtual network ID (VNID in the outer headers) is 24 bits in VXLAN and NVGRE, and 64 bits in STT (Context ID field). On the other hand, these tunneling protocols have introduced manageability and compatibility issues in the traditional networks.

Usually, the VM fragments the packetinto standard MTU-size without considering the additional tunneling headers since the tunneling process is transparent to VMs. Consequently, the frame is fragmented again after the tunneling encapsulation [1], which affects the network performance [2]. Furthermore, VXLAN and NVGRE depend on a multicast-enabled network for forwarding the tenants' traffic, which adds more complexity on troubleshooting the network problems, besides the underlying network has to handle a large number of multicast trees [3]. The STT protocol was designed to utilize the standard offloading capabilities in the network interface cards (NIC) to improve performance. However, since it uses a TCP-like header in L4 of the outer headers (i.e., it does not engage in the usual TCP 3-way handshake), it is treated as an invalid packet by the traditional network security appliances. The NVGRE cannot utilize ECMP-based load balancing since it uses GRE protocol for encapsulation, which does not have a standard transport layer (TCP/UDP) header.

This paper presents two flexible edge-overlay methods for network virtualization in the cloud datacenter. Both methods apply the same principles and leverage OpenFlow to rewrite the addresses of the VM frame before transmitting the frame through the cloud physical network. Our methods allow the tenants to use their own MAC/IP addresses and forward the tenants' traffic without relying on multicasting or IP encapsulation.Consequently, they eliminate the limitations of the tunneling protocols. The first edge-overlay method can be utilized by the cloud provider that forward traffic based on layer two (i.e., L2 datacenter network fabric), while the second method is edge-overlay for cloud datacenters that rely on layer three network infrastructure. Also,performance evaluation of the proposed edge-overlay methods, compared to VXLAN, STT, and GRE, is included in this paper.

The rest of this paper is organized as follows. Section 2 provides an overview of OpenFlow, and it discusses the flow rules for traffic forwarding in the cloud. Section 3 presents the edge-overlay methods for L2 and L3 fabrics of the datacenter networks. Section 4 elaborates on the implementation of the proposed methods. Section 5 shows the experimental results of the proposed OpenFlow-based overlay with detailed analysis, and Section 6 surveys the related work. Finally, Section 7 concludes this research and highlights the future work.

2 OpenFlow

Under the current SDN paradigm, OpenFlow protocol has gained broad support throughout the networking industry as you can easily find OpenFlow-enabled commodity switches with reasonable prices in the networking market. OpenFlow allows more flexibility in managing, programming, and dynamically controlling the whole network devices by a logically centralized controller. The network operators/applications can specify a global network policy via the controller, and then the controller translates this high-level policy into low-level instructions that add/delete/modify flow entries in the flow tables of the related switches in the network. It has been utilized in the large datacenters such as Google B4 [4] and VMWare/Nicira NVP [5] to control and forward the network traffic efficiently.

The OpenFlow controller can install the forwarding flow rules in switches using either reactive or proactive flow setup. In the reactive approach, when the ingress switch receives a packet, it performs lookup for a match in its flow table (i.e., based on headers of the received packet) to forward that packet to one or

```
in_port=1, ip, nw_dst=0.0.0.2
actions=mod_dl_dst:68:54:A1:05:53:48,
mod_nw_dst:172.16.0.1, output:2
in_port=1, ip, nw_dst=0.0.0.3
actions=mod_dl_dst:52:54:00:af:87:2a,
mod_nw_dst:10.12.11.22, output:3
in_port=1, ip, nw_dst=0.0.0.4
actions=mod_dl_dst:42:34:01:ab:87:a1,
mod_nw_dst:192.168.0.1, output:4
```

Fig. 2 Flow rules for incoming packets to OVS in host 1(L2network fabric).

```
in_port=1, ip, dl_dst=00:00:00:00:00:02
actions=mod_dl_dst:68:54:A1:05:53:48,
mod_nw_dst:172.16.0.1, output:2
in_port=1, ip, dl_dst=00:00:00:00:00:03
actions=mod_dl_dst:52:54:00:af:87:2a,
mod_nw_dst:10.12.11.22, output:3
in_port=1, ip, dl_dst=00:00:00:00:00:04
actions=mod_dl_dst:42:34:01:ab:87:a1,
mod_nw_dst:192.168.0.1, output:4
```

Fig. 3 Flow rules for incoming packets to OVS in host 1 (L3 network fabric).

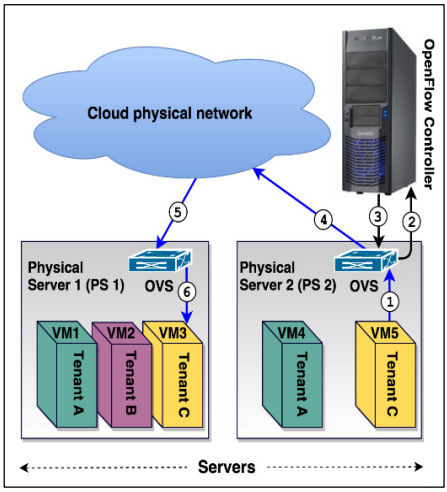

Fig. 1 Overview of the proposed OpenFlow edge-overlay.

more egress port(s). If it does not find the matching flow rule, it will drop the packet or send it to the controller via OFPT_PACKET_IN message for forwarding decision. Typically, there is a flow entry in flow table called table *miss*, which specifies how to process unmatched packets. After receiving PACKET_IN message, the controller installs the flow rules on the related switches along the flow path by sending OFPT_FLOW_MOD message to these switches.

This reactive flow setup provides a fine-grained flow visibility for the network controller, and it saves the switch memory since the reactive flow rules have an idle and hard timeout for their expiration. However, this traditional OpenFlow reactive approach increases the workload on the controller and the flow latency in the network [6,7]. For example, for each bi-directional flow setup, there will be: $2N_{FLOW_MOD} + 2_{PACKET_IN} + 2_{PACKET_OUT}$transmitted control messages between the controller and N number of switches along the flow path. Consequently, the flow latency is increased along with the increase of network diameter since each switch in the flow path has to process OFPT_FLOW_MOD message and install the flow rules.

Our proposed OpenFlow edge-overlay methods emphasize limiting the reactive control messages by using hybrid flow setup. Here, whenever VM is migrated to another location in the cloud and connected to a virtual switch, which is running on the physical host server, the controller proactively installs flow rules for the incoming packets to that VM in the virtual switch. Thus, the reactive flow setup is used only for the outgoing traffic in order to instruct the virtual switch to rewrite the headers of the outgoing packet before transmitting that packet to its destination through the cloud physical network (see Section 3).This hybrid flow setup reduces the control messages to $2_{PACKET\_IN} + 2_{FLOW\_MOD+PACKET\_OUT}$ for each bi-directional flow. Also, the flow latency is reduced in this approach since only the ingress virtual switch processes the FLOW_MOD message for each new flow. Note, the OpenFlow controller knows the location of every VM in the cloud (i.e., virtual switch ID, and the port number where the VM is connected to) as follows:

- When the cloud controller (e.g., OpenStack) configures and adds a new VM to the virtual switch, the switch sends an OFPT_PORT_STATUS message to notify the OpenFlow controller of the change. Also, when VM is migrated to another host, it sends gratuitous ARP, which is intercepted and sent by the virtual switch to the OpenFlow controller.Hence, the controller can obtain MAC/IP and location of themigrated VM(i.e., Sender Protocol Address (SPA) field in ARP has the IP address of the ARP sender).
- In addition, the OpenFlow controller can utilize the Neutron plug-in of OpenStack to obtain information about the VMs and their locations directly.

3 Edge-Overlay

This section presents the proposed edge-overlay for cloud infrastructure that uses L2 fabric (see Section 3.1) and elaborates on the rewriting techniques of the packets' headers using OpenFlow. It also shows how the same overlay principles

can be applied to the cloud that relies on L3 fabric (see Section 3.2). Our proposed overlay methods only require that the cloud physical switches should be OpenFlow-enabled, which is easily attainable nowadays as OpenFlow has been widely supported in the networking industry. This requirement is necessary to control the MAC-learning among the intermediate L2 switches and prevent the MAC table explosion problem as OpenFlow can help to reduce the forwarding tables by using hierarchical addressing scheme [8].

3.1 Overlay for L2 Fabric

In virtualized datacenter, the first switch that receives the VM packet is the virtual switch (e.g., OVS [9]) in the physical host. There we can rewrite the MAC/IP addresses of the packet in order to send it to its destination through the cloud physical network as shown in Figure 1. Here is the workflow:

1. VM5 in physical server 2 sends a packet to VM3 in physical server 1 (both VMs belong to Tenant C).
2. The virtual switch receives the outgoing packet to VM3. As the virtual switch does not know the destination, it sends `OFPT_PACKET_IN` to the controller.
3. As the controller keeps tracks of the location, MAC/IP addresses, and VNID of each VM in the cloud, it replies to the virtual switch and installs the reactive flow rule for forwarding the packet. The flow rule instructs the virtual switch to replace the destination MAC address of the packet with MAC address of the physical server 1 that hosts VM3 and replace the destination IP address with Virtual IP (VIP) address. This VIP address is used to tell the virtual switch in the destination physical server 1 how to forward the received packet (see Section 3.3).
4. After rewriting the destination MAC and IP addresses, the virtual switch transmits the packet to the physical network through the trunk port.
5. When the virtual switch in the physical server 1 receives the incoming packet, it knows that the packet should be forwarded to VM3 based on the VIP in the destination IP field.
6. The virtual switch in physical server 1 replaces the destination MAC and IP addresses in the received packet with the original MAC and IP of the VM3, and then it forwards the packet to the destination VM3. Figure 4 shows the addresses of the transmitted packet during VM-to-VM communication as described in the six steps above.

3.2 Overlay for L3 Fabric

As some cloud datacenters forward traffic based on L3, we also provide another edge-overlay for L3 datacenter network fabric. This overlay method applies the same 1, 2, 4, and 6 steps above-mentioned when forwarding the tenants' traffic. The difference here is in the steps 3 and 5. In step 3, the controller replies to the

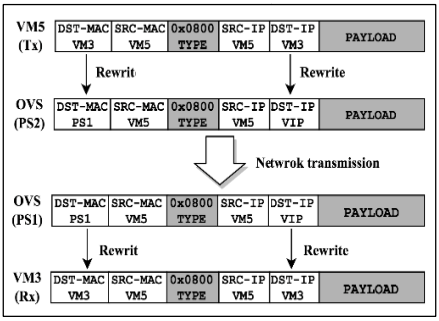

Fig. 4 Rewriting the packet headers during transmission (overlay for L2 fabric).

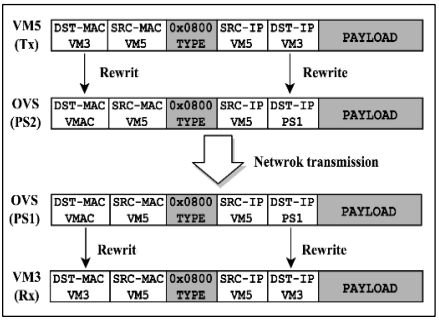

Fig. 5 Rewriting the packet headers during transmission (overlay L3 fabric).

virtual switch and installs flow rule to replace the destination IP address of the packet with IP address of the physical server 1 that hosts VM3 and replace the destination MAC address with Virtual MAC (VMAC) address. The VMAC is similar to VIP. It is just used to inform the virtual switch in the destination physical server 1 how to forward the received packet (see Section 3.3). In step 5, the virtual switch in the destination, where the target VM3 is connected, knows that the received packet should be forwarded to VM3 based on the VMAC in the destination MAC field. Figure 5 shows the addresses of the transmitted packet during VM-to-VM communication with edge-overlay for L3 datacenter network fabric.

3.3 Virtual Addresses

As it is highlighted earlier, the OpenFlow switch is relatively simple because the forwarding decision is defined by a controller, rather than by switch firmware. Whenever it receives a new packet, it checks the flow table(s) to find the corresponding action of the matching flow entry. The action field in the flow entry specifies how to forward the received packet. If it does not find a matching entry in its table(s), it will forward the packet to the controller. Our proposed methods were designed to reduce the number of the exchanged control messages between the virtual edge switches and the controller, which consequently reduces the workload on the controller for forwarding the tenants' traffic. We achieved this by using hybrid flow setup. In the hybrid approach, the controller proactively installs flow rules for the incoming traffic to the VMs, which are connected to the virtual switch. Thus, the virtual switch only sends the reactive `OFPT_PACKET_IN` message to the controller for the outgoing traffic. Here, when the virtual switch receives an incoming packet via its trunk port, it can forward the received packet to destination VM based on VIP or VMAC as described below:

Virtual IP (VIP). VIP is a simplified methodto instruct the virtual switch in the destination on how to forward the received packet from the trunk port. For example, in step 3 of Figure 1, the controller knows that the destination VM3 is connected to port 4 of OVS in physical server 1. Consequently, it installs flow rule in the OVS of physical server 2 that rewrites the original destination IP address of the outgoing packet with VIP = "0.0.0.4". We used only the first byte of the VIPto encode the switch port number where the destination VM is connected to. Now, when the OVS in physical server 1 receives that packet, it replaces the destination MAC and IP fields in the incoming packet with the original MAC and IP of VM3, and then it forwards the packet through port 4. Thus, the header rewriting is transparent to VMs in the cloud. Figure 2 shows an example of flow rules for incoming packets into OVS of the physical server 1, which are installed proactively by the controller when VMs were connected to the virtual switch. In this example, the port number one of the OVS is a trunk port while VM1, VM2, and VM3 are connected to ports two, three, and four respectively. When the OVS receives a packetwith VIP = "0.0.0.4" in the destination IP field, the third flow rule in Figure 2 instructs the OVS to change the destination MAC address to "42:34:01:ab:87:a1", IP address to "192.168.0.1", and forward the packet through port 4 where the destination VM3 is connected. Note, in the hybrid flow setup, the controller deletes the proactive flow rules for the disconnected VMs to save the switch memory and keep the flow tables updated.

Virtual MAC Address (VMAC). VMAC is used for the second proposed edge-overlay in L3 fabric instead of VIP. Since the destination IP is used for forwarding purposes in the cloud physical network, which relies on layer three network infrastructure, we used the destination MAC field to tell the virtual switch in the destination how to forward the received packet from the trunk port. In our implementation, only the first byte of the VMAC is used as the port number where the destination VM is connectedto the virtual switch. Figure 3 shows an example of the flow rules for incoming packets into OVS of physical server 1, which can be installed proactively with the deployment of edge-overlay for L3 fabric. For example, when the OVS receives a packet with VMAC = "00:00:00:00:00:02" in the destination MAC field, the first flow rule for incoming traffic in Figure 3 instructs the OVS to change the destination MAC address to "68:54:A1:05:53:48", IP address to "172.16.0.1", and forward the packet through port 2 where the destination VM1 is connected to.

3.4 ARP Processing

When the OVS receives an ARP request from VM, it sends `OFPT_PACKET_IN` message to the controller. As mentioned earlier, the controller knows the MAC/IP addresses of all VMs in the cloud, so it replies with the flow rule that instructs the OVS to create an ARP reply (i.e., the ARP reply is created by using OVS-specific actions which are `move`, `load`, and`mod_dl_src` [10]) and send it to the same port where the ARP request came from.

4 Implementation

The proposed OpenFlow edge-overlay methods are implemented as Floodlight applications [11]. Both methods utilize `StaticFlowPusher` and `Forwarding` modules in the Floodlight controller to implement the aforementioned hybrid flow setup. Also, The `IOFMessageListener` module of the Floodlight is used to create the VM profile, which maps each VM to its location in the cloud (i.e., switch ID and port number where the VM is connected to) and the MAC/IP addresses of the physicalhost server. Our methods use the `StaticFlowPusher`module to proactively install and update the flow rules for incoming traffic into the virtual edge-switches, which is OVS in our experiments.These proactive flow rules are installed based on the information that is collected from `TopologyManager` and `DeviceManager` modules in Floodlight.

In our implementation, we modified the `Forwarding` module of the Floodlight to reactively install flow rules for the outgoing traffic in the virtual ingress switch only, which sends the `OFPT_PACKET_IN`message, and the ARP request flood is prevented as ARP is handled directly by the controller. Also, the `LinkDiscoveryManager` module in the Floodlight controller was utilized to notify our network applications of the changes in the network topology (i.e., added/removed switches) and the status of the network links.

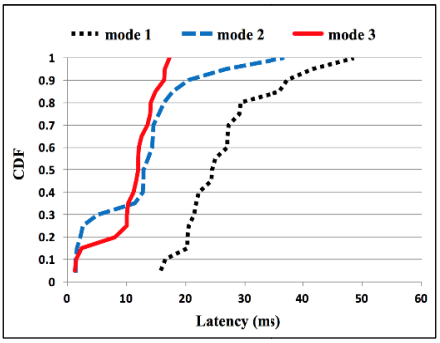

Fig. 6 Latency variation in a tree topology.

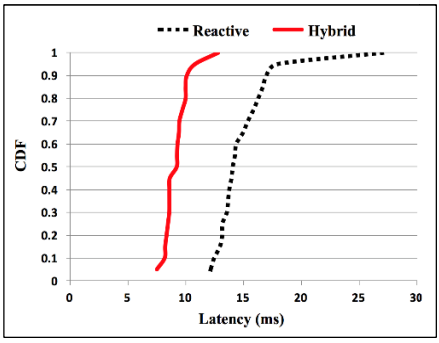

Fig. 7 Latency variation in a fatigued path.

5 Evaluation

This section presents and discusses the performance evaluation results of the proposed edge-overlay methods. We tested our overlay methods through a series of experiments. The first set of the experiments shows how the proposed hybrid flow setup helps to reduce the flow latency compared with the traditional reactive flow setup. The second set of experiments compares the network throughput of our edge-overlay to the tunneling protocols VXLAN, GRE, and STT.

5.1 Flow Latency

This set of experiments was conducted to investigate the impact of the traditional reactive flow setup on the flow latency and validate the enhancement using the hybrid flow setup. All the flow latency experiments were conducted in the same experimental environment. We used the Mininet emulator to configure and run different types of SDN topologies with using software OpenFlow switches. Mininet integrates Open vSwitch in its framework, which is a popular OpenFlow-compliant switch. Note, Mininet can create many OVS instances and configure them to run in kernel space. In our experiments, the network emulator was running in a server that is equipped with four CPUs, each Intel(R) Xeon(R) E5530 @ 2.40GHz, and 16 GB of RAM. Another server with the same above-mentioned resources was used to run the Floodlight controller.

In the first experiment, 243 hosts and 121 switches were emulated by Mininet to create a tree network topology with fanout 3 and depth 5. The emulated hosts generate traffic randomly using *Iperf* and create synthetic workload on the network. We chose two hosts in the network that are 9 hops apart and used ping responses to measure the latency between them in three modes. The first mode is the reactive flow setup for all flows in the network, including the flows from the two selected hosts. The second mode is same as the first mode except that the flow rules for the path between the selected two hosts are installed proactively in the intermediate switches. In the third mode, we deployed the hybrid approach and used our implemented applications on the controller. Here, the flow tables of the intermediate switches were populated proactively for traffic forwarding between the edge switches, and the flow rules for incoming traffic are installed proactively in the edge switches as described in Section 3.3. Every run was repeated 20 times, and the results are plotted in Figure 6. The ping response latency between the selected hosts is significantly high in mode one, compared to mode two and three. Also, the standard deviation of the ping responses' latencies in mode one was 8.4, while it was 4.6 in mode three. The maximum latency in mode one was 48.6 *ms,* whereas it was approximately 64% less in mode three. In this experiment, we did not configure any transmission or propagation delays between the network nodes in the emulator in order to focus only on the flow setup latencies in the reactive and hybrid approaches. As shown in Figure 6, the reactive flow setup introduces higher latency because all switches in the flow path have to process `OFPT_FLOW_MOD` message before forwarding the traffic.

The second experiment was aiming to investigate the impact of processing additional flow rules while there is traffic in transit, and testing the flow latency for this scenario in the reactive and hybrid approaches. Therefore, a linear topology with two edge switches and 12 intermediate switches was created in Mininet. We configured the controller to push randomly and continuously additional flow rules in the intermediate switches, and the ping responses were used to measure the flow latency between hosts, which are connected to edge switches. Each run was repeated 20 times. As you can see in Figure 7, generally the flow latency is lower in the hybrid approach, and the standard deviation of the responses' latencies in the hybrid setup is approximately 62% less than the reactive approach.

5.2 Network Throughput

The second set of experiments was designed to assess the performance of the proposed edge-overlay methods in an emulated cloud environment. Therefore, we used three physical servers and Kernel-based Virtual Machine(s) (KVM). Two servers were used for virtualization, server one and server two were hosting VM1 and VM2 respectively, and the third server was used as a controller. Figure 8 shows the machine specifications of the virtualized environment. OVS, version 2.4, was used as the virtual switch in server one and server two, and it was configured with OVS Linux-kernel modules, which are included in the OVS distribution. All the three servers were connected via Gigabit Ethernet switch. In this experiment, *Iperf* version 2.0.8 was used to send/receive UDP and TCP packets for 25 seconds in each run. The *Iperf* client was running in VM1 while the *Iperf* server in VM2. All the conducted experiments share the same environment.

VM 1 (Sender)	
OS	Fedora 22 (4.0.4)
CPU	1 core
Memory	2 GB
vNIC	virtio-net
Offloading	TSO, GSO, GRO

Physical Server 1	
OS	CentOS 7 (3.10.0)
CPU	8 × i7-2600 CPUs (3.40GHz)
Memory	8 GB
VMM	KVM
vSwitch	Open vSwitch 2.4
NIC	1000BASE-T
Offloading	TSO, GSO, GRO

VM 2 (Receiver)	
OS	Fedora 22 (4.0.4)
CPU	1 core
Memory	2 GB
vNIC	virtio-net
Offloading	TSO, GSO,GRO

Physical Server 2	
OS	CentOS 7 (3.10.0)
CPU	8 × i7-2600 CPUs (3.40GHz)
Memory	8 GB
VMM	KVM
vSwitch	Open vSwitch 2.4
NIC	1000BASE-T
Offloading	TSO, GSO, GRO

Fig. 8 Machine specifications.

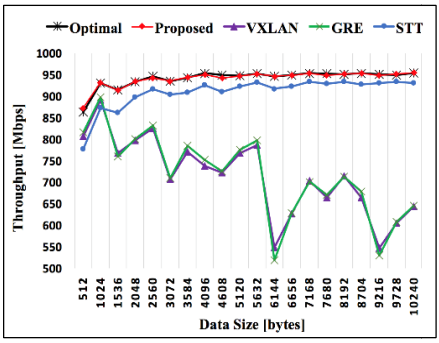

Fig. 9 UDP throughput.

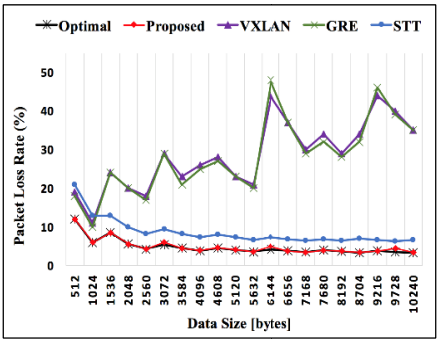

Fig. 10 UDP loss rate.

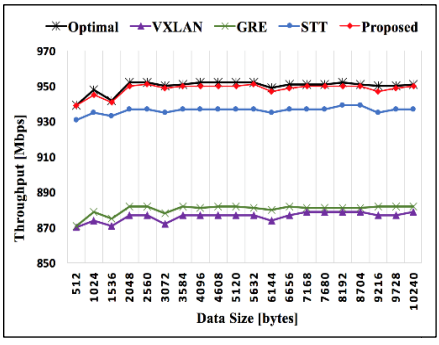

Fig. 11 TCP throughput.

The network throughput of the proposed edge-overlay methods are similar. Therefore, we present the performance results of the edge-overlay for L3 fabric and omit the results for the other overlay method due to the space limit.

The results of UDP throughput in the proposed edge-overlay are plotted in Figure 9 with the UDP throughput in VXLAN, STT, and GRE for comparison. Besides, the optimal throughput for VM to VM communication is included as the upper performance bound in the experimental environment. Here, the VMs were bridged to the physical network (i.e., no headers rewriting or tunneling encapsulation). The x-axis is the size of the transmitted bytes in UDP, excluding headers, by *Iperf* client. The y-axis is the throughput measured by the *Iperf* server. As you see in Figure 9, the STT performance was close to the optimal, while the proposed method almost matched the optimal throughput. The overhead of the outer headers in STT does not affect its performance as it utilizes the offloading capabilities of NIC. In our method, there are no outer headers, so there are no any additional fragmentations, and with the support of OVS-kernel module, the performance was high. Contrarily, the throughput of VXLAN and GRE was far below the optimal, especially for large data chunks in UDP. The VXLAN specification recommends setting MTU size to a value that can accommodate the outer headers and avoid fragmentation. In our experiment, we kept the default MTU (i.e., 1500 bytes) to test all protocols under the same conditions and obtain a fair comparison. Comparatively, Figure 10 shows the packet loss rate, which is high in VXLAN and GRE. Whereas TCP retransmits the missingfragments, UDP drops the whole packet when it misses fragments, especially when processes send packets rapidly as it has a limited frame buffer. In our experiments, the *Iperf* client in the sender side was configured to consume all the available bandwidth. Thus, Figure 10 shows the packet loss rate increases gradually with VXLAN, notably when data chunk is multiple of 1500 bytes. The TCP in our overlay method can use the offloading capabilities of NIC same as STT. Figure 11 shows the throughput of TCP in the proposed method, which is even higher than STT and very close to the optimal throughput.

6 Related Work

NetLord [12] architecture encapsulates the VM frame with additional MAC/IP headers. The outer MAC headers have the addresses of the edge switches where the sender and receiver VMs are connected, and the outer destination IP has the tenant ID. Our solution does not require any additional headers as it rewrites the addresses in the original headers and restores them in the destination. VL2 [13] uses L3 fabric for forwarding traffic in Clos topology, and it uses IP-in-IP encapsulation for network virtualization. It relies on IP multicasting to handle the virtual network broadcasts. Kawashima et al. [2] proposed non-tunneling edge-overlay model for the cloud network. His model utilizes OpenFlow to rewrite the MAC addresses (source and destination) of the VM frame and replace them with the MAC addresses of the physical servers in the cloud. It uses VLAN tag as VM

identifier in the host. However, his model does not hide the addresses of the physical servers in the cloud from the tenants, which exposes the cloud network infrastructure to threats. Chen et al. [14] proposed a scalable L2 architecture that spans multiple datacenters across diverse geographical locations. Their architecture only serves the cloud provider that relies on L2 fabric datacenter networks. Guenender et al. [19] published non-encapsulation overlay technique for network virtualization, which replaces the addresses in VM packet with the addresses of the edge-switches, and they used the source TCP port number to identify the VM in the destination host. Their research focused only in TCP, while our overlay methods consider both TCP and UDP protocols.

7 Conclusion

In this paper, we have presented two OpenFlow-based overlay methods for the cloud networks. The first method is designed to serve L2 fabric datacenter network, while the second method can be used for L3 cloud networkinfrastructure. Our methods utilize OpenFlow for rewriting VM packet headers in order to forward the tenants' traffic through the cloud physical network with maintaining sufficient isolation. Both methods do not use any additional encapsulation headers, and they can handle broadcast traffic such as ARP with less overhead. We are planning to extend our design to forward traffic between multiple datacenters in different locations as future work.

References

1. Kawashima, R., Matsuo, H.: Performance evaluation of non-tunneling edge-overlay model on 40GbE environment. In: IEEE 3rd Symposium on Network Cloud Computing and Applications, pp. 68–74. IEEE Xplore (2014)
2. Kawashima, R., Matsuo, H.: Non-tunneling edge-overlay model using openflow for cloud datacenter networks. In: IEEE 5th International Conference on Cloud Computing Technology and Science, pp. 176–181. IEEE Xplore (2013)
3. Nakagawa, Y., Hyoudou, K., Shimizu, T.: A management method of IP multicast in overlay networks using OpenFlow. In: First Workshop on Hot Topics in Software Defined Networks, pp. 91–96. ACM, New York (2012)
4. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A.: B4: experience with a globally-deployed software defined wan. In: ACM SIGCOMM 2013, pp. 3–14. ACM, New York (2013)
5. Koponen, T., Amidon, K., Balland, P., Casado, M., Chanda, A., Fulton, B., Ganichev, I., Gross, J., Gude, N., Ingram, P., Jackson, E., Lambeth, A., Lenglet, R., Li, S., Padmanabhan, A., Pettit, J., Pfaff, B. Ramanathan, R., Shenker, S., Shieh, A., Stribling, J., Thakkar, P., Wendlandt, D., Yip, A., Zhang, R.: Network virtualization in multi-tenant datacenters. In: 11th USENIX NSDI, pp. 203–216. ACM, Berkeley (2014)

6. Curtis, A., Mogul, J., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S.: DevoFlow: scaling flow management for high-performance networks. In: ACM SIGCOMM 2011, pp. 254–265. ACM, New York (2011)
7. Bu, K.: Gotta tell you switches only once: toward bandwidth-efficient flow setup for SDN. In: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 492–497. IEEE Xplore (2015)
8. Chen, C., Liu, C., Liu, P., Loo, B., Ding, L.: A scalable multi-datacenter layer-2 network architecture. In: 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Article No. 8. ACM, New York (2015)
9. Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., Casado, M.: The design and implementation of Open vSwitch. In: 12th USENIX NSDI, pp. 117–130. ACM, Berkeley (2015)
10. OVS ARP Responder – Theory and Practice. http://assafmuller.com/2014/05/21/ovs-arp-responder-theory-and-practice/
11. Floodlight Project. https://floodlight.atlassian.net/wiki/display/floodlightcontroller/For+Developers
12. Mudigonda, J., Yalagandula, P., Mogul, J., Stiekes, B., Pouffary, Y.: Netlord: a scalable multi-tenant network architecture for virtualized datacenters. In: ACM SIGCOMM 2011, pp. 62–73. ACM, New York (2011)
13. Greenberg, A., Hamilton, J., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D., Patel, P., Sengupta, S.: VL2: a scalable and flexible data center network. In: ACM SIGCOMM 2009, pp. 51–62. ACM, New York (2009)
14. Guenender, S., Barabash, K., Ben-Itzhak, Y., Levin, A., Raichstein, E., Schour, L.: NoEncap: overlay network virtualization with no encapsulation overheads. In: 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Article No. 9. ACM, New York (2015)