Network Virtualization with OpenFlow for Large-Scale Datacenter Networks

Amer Aljaedi, C. Edward Chow, Abdelhamid Elgzil, Naif Alamri and Ismail Bahkali
Department of Computer Science, University of Colorado, Colorado Springs, CO 80918, USA

Abstract
Network virtualization is one of the key components for the multi-tenancy services in the datacenter environment, where overlay layer 2-in-layer 3 tunneling protocols have gained wide traction. These overlay tunneling protocols assist in overcoming the scalability challenges associated with the traditional network virtualization primitives such as VLAN, and they enable tenants to use their own IP/MAC addresses while ensuring traffic isolation. However, the tunneling protocols have introduced compatibility and performance issues. This paper discusses the issues of the tunneling protocols and proposes a scalable edge-overlay solution for network virtualization in multi-tenant datacenters. The proposed solution leverages OpenFlow to control and forward the tenants' traffic without using additional encapsulation. Also, it allows the tenants to use their IP/MAC addressing scheme. We have implemented and evaluated the proposed solution, and the results from the experiments demonstrate that our solution provides higher performance than the tunneling protocols.

Key words:
OpenFlow; network virtualization; software-defined networking; datacenter networks; flow rules; virtual addresses; flow setup

1. Introduction

The increasing demand for computing resources as utility services has driven the growth of large datacenters, which form the backbone of cloud computing infrastructures. The evolution of datacenter virtualization has enabled the cloud operators to efficiently utilize the resources of their datacenters in serving a large number of clients and deploying a wide range of online applications, which is also reflected in their revenues growth [1]. While the virtualization in the datacenters has become mature in providing a sufficient abstraction layer for the computing resources, the virtualization of the network infrastructure is still behind [2]. A multi-tenant datacenter is considered one of the most challenging networking environments since network configurations change rapidly to accelerate application deployment, optimise traffic routing, and satisfy tenants' requirements. The network architecture for large-scale, multi-tenant datacenters should meet the following fundamental requirements to accommodate a large number of clients and effectively manage the shared networking environment:

Mobility of virtual machine*:* The datacenter network architecture should enable Virtual Machine (VM) live migration to any physical server in the datacenter without changing the addresses of the migrated VM. VM migration can be essential for rebalancing the workload on the host servers and optimising the provision of computing resources. In such case, the migrated VM should keep its IP/MAC addresses to preserve the running applications' state at the VM. Therefore, the VM addresses should be independent of its location in the datacenter network.

Decoupling of physical and virtual networks: The address spaces of the tenants' virtual networks should be independent of the physical addressing scheme of the host servers in the datacenter for two reasons. First, changing the physical network topology (i.e., adding/removing switches) should not interfere with the configurations of the tenant virtual networks. Even changing the IP/MAC addresses of host server should not affect VMs that are hosted on the server. Second, the tenant address spaces may overlap as they are isolated and managed independently. This fundamental requirement allows the tenants to configure the addresses of their VMs conveniently as they like.

Scaling of virtual network segments: Datacenters are growing in size as a result of the increasing demand for computing resources under the pay-as-you-go business model. In such large-scale networks, proper planning is critical for achieving the benefits of resource sharing and accommodating the increasing number of tenants. Therefore, scaling the virtual network segments has become important as a considerable number of tenants subscribe to the datacenter services on a daily basis. Any adopted network virtualization technique in a large multi-tenant datacenter should be able to serve millions of tenants [3].

Routing/forwarding tables of the switches: Although some modular switches have large tables to handle thousands of forwarding entries, it is still limited for large-scale datacenters [4]. If the traffic forwarding in the datacenter is determined based on the addresses of VMs, then each switch in the underlying physical network needs to handle a large number of forwarding entries in its table, which significantly consumes the switch resources (i.e., TCAM memory). Therefore, the forwarding scheme that will be deployed should assist in reducing the forwarding tables for the entire network fabric (i.e., Rack, Aggregate, and Core switches in the network topology) while enabling any VMs to communicate with one another when access is granted.

Manuscript received September 5, 2017
Manuscript revised September 20, 2017

Satisfying the above requirements for large-scale datacenter networks raises the importance of utilizing a scalable network virtualization technique that can accommodate a large number of tenants while reducing the traffic forwarding/routing overhead on the core network. Note that datacenter networks are often structured and managed as a single logical network fabric that interconnects all the datacenter resources together through multi-rooted tree-like topology with multiple core switches [5, 6]. Typically, datacenter networks have multiple paths between each end pair of nodes to mitigate link failures and load balance the traffic (e.g., using ECMP). Therefore, the traditional Layer-2 (L2) switching is not suitable for large virtualized datacenters due to the following reasons: First, relying on the flat L2 addressing for forwarding traffic will result in large and overlapping MAC-learning tables among the intermediate L2 switches since the virtualized infrastructure further increases the density of L2 addresses. Second, L2 switching forwards traffic through spanning tree(s) to control forwarding loops in the network (i.e., the nature of L2 broadcast traffic), which consequently restricts multi-path routing as redundant links that are not present in the tree will not be used to carry traffic [7]. This would result in wasting considerable proportion of network capacity in load balancing the network traffic.

On the other hand, the hierarchical routing based on Layer-3 (L3) addresses is easier to manage, but it also introduces limitations when deployed in datacenters. First, IP adopts the location-aware addressing scheme, so the VM mobility is bounded to its IP prefix (e.g., subnet). If the IP address of the VM is determined by the location of its host server or the Top-of-Rack switch, it will be difficult to perform VM live migration across IP prefix boundaries [8, 9] as the IP address of the migrated VM will be changed according to its new location; inevitably invalidating existing network sessions. Second, in such restricted routing schemes, the tenant has limited choices for the network configurations of his VMs.

Since using the traditional routing/forwarding in modern datacenters would limit the network scalability, flexibility, and manageability, the networking industry has been seeking ways to virtualize the datacenter network infrastructure and segregate the configuration of the tenant virtual network from the datacenter physical network. Thus, there has been a growing interest in overlay (L2-in-L3) tunneling protocols such as VXLAN (Virtual eXtensible Local Area Network, RFC 7348) [10], NVGRE (Network Virtualization using Generic Routing Encapsulation, RFC 7637) [11], and STT (Stateless Transport Tunneling) [12], which are well-known tunneling protocols for network virtualization in large datacenters. However, these tunneling protocols have introduced an additional overhead on the underlying network and some performance issues that we highlighted in the next section as motivations for this research.

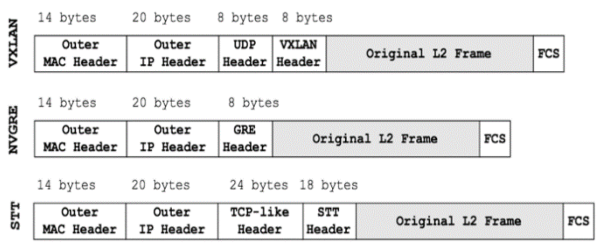


Fig. 1 The encapsulation formats for VXLAN, NVGRE and STT.

The rest of the paper is structured as follows. The design of our solution, which avoids the limitations associated with overlay tunneling protocols, is described in Section 3. Section 4 introduces the implementation details. Section 5 presents the evaluation results of our solution, compared to the tunneling protocols, and we discuss the merits and broader aspects of our approach in Section 6. Section 7 elaborates on the related work. Finally, Section 8 concludes this research.

2. Background and Motivations

Since the traditional network virtualization primitives such as VLAN suffers scalability problems, including being limited to 4096 virtual networks due to its 12-bit VLAN ID, configuration overhead, and forwarding traffic through spanning tree [6], the datacenter networking industry has alternatively adopted overlay (L2-in-L3) tunneling protocols. These protocols are based on NVO3 (Network Virtualization Over Layer 3) framework (RFC 7365), which defines a reference model capable of isolating not only virtual networks from each other, but also separating them from the underlying physical network. These tunneling protocols encapsulate the whole Ethernet frame sent from the VM in an IP packet in order to transmit the VM frame to its destination over the physical datacenter network (i.e., the tunnel endpoints, which can be virtual switches in the host servers, perform the frame encapsulation and decapsulation). After the encapsulation, the new packet has outer (encapsulation) headers, which carry the addresses of the physical hosts where the sender/receiver VMs are located. Thus, the addresses in the outer headers belong to the datacenter physical network, while the addresses in the inner headers (i.e., in the payload of the encapsulated packet) belong to the tenant virtual network (i.e., virtual addresses).

The encapsulation formats of the tunneling protocols are shown in Figure 1. When the encapsulated VM frame reaches the physical server that is hosting the destination VM, the virtual switch will remove the encapsulation headers and forward the frame to its destination based on inner header and the virtual network identifier, which specifies the virtual network the packet belongs to.

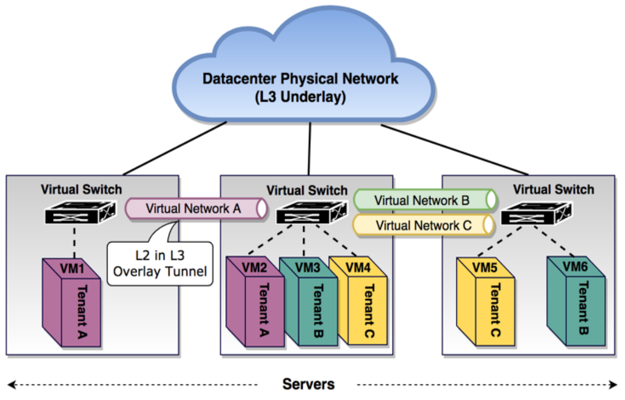


Fig. 2 Network virtualization via overlay tunnelling protocols.

Unlike VLAN, these tunneling protocols support a large number of virtual networks. The Virtual Network Identifier (VNI) field in VXLAN header is 24-bit, so it supports over 16 million virtual networks. Similarly, the Virtual Subnet Identifier (VSID) in NVGRE is 24-bit, while the context ID field in STT header is 64-bit. Note that this overlay tunneling technique extends the tenant virtual networks across the physical host servers as shown in Figure 2, while it reduces the routing information in the underlying physical network (i.e., the VMs addresses are hidden from the physical network). The datacenter underlying network is unaware of the overlay tunneling process as it only routes packets between the host servers based on the outer encapsulation headers (e.g., destination IP). Also, this tunneling process is transparent to the tenants' virtual networks, and effectively allows the tenants to have their own IP/MAC address spaces.

2.1 Issues of Tunneling Protocols

While the overlay tunneling protocols discussed above can reduce the need for switches in the network to learn all addresses (i.e., VM addresses), they introduce another overhead on the network. This section highlights the overhead and the limitations of utilizing the aforementioned tunneling protocols for forwarding and isolating the tenants' traffic in datacenter networks.

Multicast overhead: VXLAN and NVGRE depend on multicast-enabled networks for forwarding the tenants' traffic (i.e., broadcast and unknown unicast destination), which adds more complexity for troubleshooting the network problems, after all, the underlying network has to handle a large number of multicast trees [16]. Using IP multicast as an approach for destination discovery will form a large number of multicast trees in the datacenter network, where each multicast group requires a state to be held in the network layer. In addition, the edge of tunnels (e.g., VXLAN Tunnel End Point (VTEP) in the physical server) has to handle a considerable number of multicast messages. Nakagawa et al. [16] studied the multicast traffic in overlay networks, and highlighted that relying on IGMP as a dynamic registration protocol for managing the multicast traffic in a multi-tenant datacenter will result in millions of membership reports every second. Therefore, datacenter network vendors start encouraging the usage of a centralized control plane to manage the tunnels instead of relying on multicasting for destination discovery [2, 17].

Fragmentation: VXLAN and NVGRE encapsulate the VM Ethernet frame into UDP and GRE respectively. Such encapsulation causes further fragmentations that are processed by the end-hosts (i.e., most network cards support TCP offloading). Usually, the VM fragments the packet into standard MTU-size without considering the additional encapsulation headers since the tunneling process is transparent to VMs. Consequently, the frame is fragmented again after the tunneling encapsulation, which affects the network performance [18]. As this is a well-known issue associated with the encapsulation protocols [20], the VXLAN (RFC 7348) and NVGRE (RFC 7637) specifications recommend setting the MTU (Maximum Transmission Unit) size to a value that can accommodate the outer encapsulation headers and avoid fragmentation (e.g., reducing the standard MTU-size at the NICv of VM). Consequently, the encapsulation headers impose additional overhead in the traffic tunneling process.

Compatibility: The Stateless Transport Tunneling (STT) protocol for network virtualization utilizes the standard offloading capabilities in the network interface cards (i.e., TSO) 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.

Load balancing: NVGRE uses GRE protocol for encapsulation. As it does not have a standard transport layer (TCP/UDP) header, it cannot provide ECMP hash naturally for flow-level granularity distribution among multiple paths. Therefore, it cannot utilize the standard ECMP-based load balancing. The updated NVGRE (RFC 7637) specifications suggested using customized ECMP that its hash is calculated based on the outer IP fields and the entire Key field (32 bits) in GRE header. The Key field is composed of two parts: The first 24 bits are assigned to the Virtual Subnet ID (VSID), and the second part is an 8-bit value, named *FlowID* in the NVGRE specification, which can be used to provide per-flow entropy for flows in the same VSID. However, this requires a special load balancing technique that understands the NVGRE header format, which should be installed in the core network. Supporting such a special load balancing technique in the industry is limited.

In this paper, we propose a fixable edge-overlay solution for network virtualization in multi-tenant datacenters. Our network virtualization technique leverages OpenFlow/SDN (Software-Defined Networking) to isolate the tenants' traffic and rewrite the addresses of the VM frames before transmitting them through the datacenter physical network; instead of encapsulating every VM frame. Thus, it eliminates the limitations of tunneling protocols. The proposed technique employs a centralized SDN controller for mapping the tenant virtual networks to the physical network and making decisions on traffic forwarding in the datacenter network. The network state and intelligence are (logically) centralized in SDN, and in turn facilities the traffic control and forwarding for dynamic networking environments, where the configurations of the end nodes (i.e., virtual switches in host servers) frequently change as users/applications come and go.

3. Network System Design

Before describing the design of our proposed edge-overlay solution, we first outline the OpenFlow, which is a leading SDN protocol in the networking industry and has been used in large production datacenters [2, 21]. Under the SDN paradigm, network operators/applications can specify high-level network policies, which are automatically translated into low-level rules/instructions and installed in network switches by a logically centralized controller. Thus, the SDN controller serves traffic routing applications in SDN-enabled datacenters, besides observing and controlling the network forwarding state.

The controller communicates with the switches via OpenFlow, a standardized SDN protocol [13], which allows the controller to instruct the switches and control the network forwarding states in either a proactive or reactive mode. In the latter, when an ingress OpenFlow switch receives a new flow, it performs lookup for a matching flow entry in its flow table (e.g., based on the headers of the received packet and the ingress port) to forward that packet to one or more egress port(s). If it does not find the matching flow entry, it will forward the first packet of the flow (or just its headers) to the controller via `OFPT_PACKET_IN` message in seeking instructions on how to handle the packets of such flow. Typically, there is a special flow entry in the switch table called *table-miss*, which specifies how to process packets unmatched by other flow entries in the table. The controller, upon receiving the `OFPT_PACKET_IN` message, will instruct the ingress switch and other related switches that reside in the packet's path on how to handle the packets of such flow by adding a new flow entry in the switches' tables using `OFPT_FLOW_MOD` message. These forwarding instructions are cached by the switches for some period of time to handle upcoming packets at line rate.

This reactive mode, also known as a reactive flow setup, provides a fine-grained flow visibility and control for the SDN controller since the controller decides the path for each flow in this operational mode. On the other hand, the flow rules can be installed proactively in the switches when the routing application in the controller has computed all the traffic routes in the network. The proactive mode can reduce the overhead on the controller, but it basically resembles the traditional networking, where the traffic is forwarded based on the destination addresses since the specific flows are not known in advance. Consequently, it limits the controller capability of observing and controlling the traffic dynamically in the network. Unlike the reactive flow setup; which usually installs *micro-flow* rules (i.e., exact-match flow rules) [22], the proactive approach installs *mega-flow* rules for traffic forwarding. The *mega-flow* rule only matches one or more fields of the packet headers with other fields being wildcarded. Note that many *micro-flow* rules can be covered by one *mega-flow* rule as the latter allows rules aggregation by wildcard matching. For instance, flow rules that forward TCP packets from host *A*, *port:66*, to different destination ports in host *B* are *micro-flow* rules of the *mega-flow* rule that forwards any packet from host *A* to host *B*. Assume that $H=\{h_1, h_2, \dots, h_i\}$ is the set of match fields for the packet headers that are present in the flow rule f_1 . The flow rule f_2 is considered a strict subset of f_1 when the condition in Eqn. (1) is satisfied, where $h_i^1 \in f_1$, $h_i^2 \in f_2$, and $i \neq j$.

$$f_2 \subset f_1 \ if \ (\exists\, j \mid h_j^2 \subset h_j^1) \wedge [\forall\, i \mid h_i^2 \subseteq h_i^1] \qquad (1)$$

First, note that the wildcard match field (i.e., means any value) in a flow rule is considered a superset of any defined values in the corresponding field of the other flow rules. Also, some match fields can be partially wildcarded (i.e., the address is associated with a bitmask to specify which bits are wildcard). Therefore, if at least one matching field j in f_2 is a strict subset of the corresponding field in f_1 while the other fields i in f_2 are equal (or subsets) of the corresponding fields in f_1, then $f_2 \subset f_1$ because every packet that matches f_2 will also match f_1. Therefore, *mega-flow* rules can be used to reduce the number of flow entries in the switch's table, but they cripple many SDN applications that depend on the reactive, fine-grained flow rule installation for monitoring or correction operations such as traffic engineering [24] and intrusion detection/prevention [23].

3.1 Hybrid Flow Setup

As discussed above, the reactive flow setup empowers the SDN controller to observe and control the network traffic, but it increases the workload on the controller (i.e., the number of exchanged control messages between the controller and network switches). For example, for each bi-directional flow setup, there will be: $2N_{\texttt{FLOW\_MOD}}$ + $2_{\texttt{PACKET\_IN}}$ + $2_{\texttt{PACKET\_OUT}}$ transmitted control messages between the controller and N number of switches along the flow path

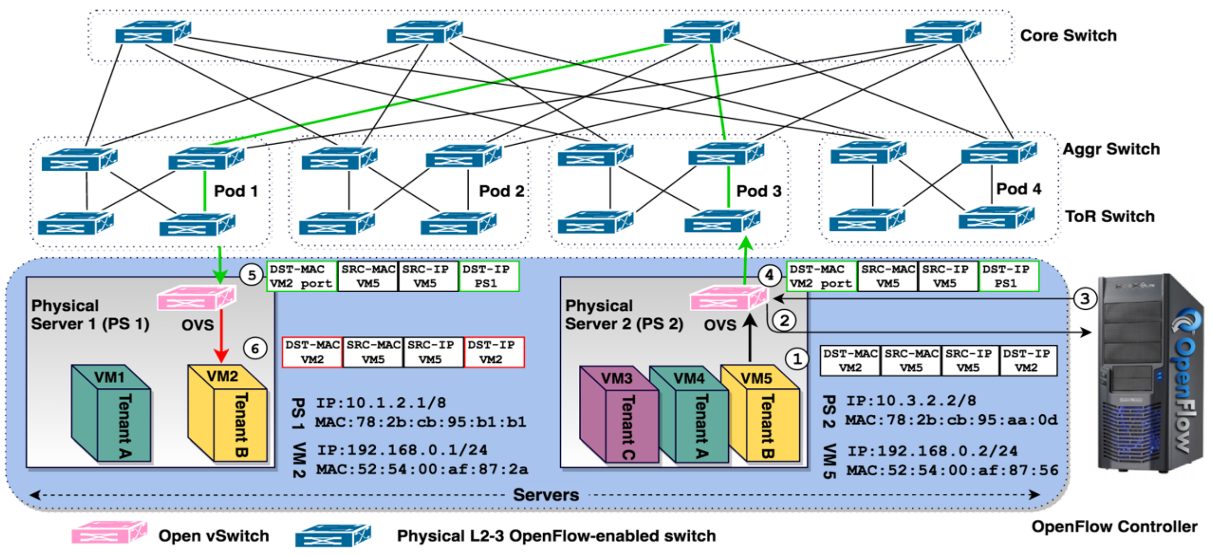

Core Switch
Aggr Switch
ToR Switch
Pod 1
Pod 2
Pod 3
Pod 4
Physical Server 1 (PS 1)
OVS
VM1
Tenant A
VM2
Tenant B
Physical Server 2 (PS 2)
VM3
Tenant C
VM4
Tenant A
VM5
Tenant B
DST-MAC VM2 port | SRC-MAC VM5 | SRC-IP VM5 | DST-IP PS1
DST-MAC VM2 | SRC-MAC VM5 | SRC-IP VM5 | DST-IP VM2
PS 1 IP:10.1.2.1/8 MAC:78:2b:cb:95:b1:b1
VM 2 IP:192.168.0.1/24 MAC:52:54:00:af:87:2a
PS 2 IP:10.3.2.2/8 MAC:78:2b:cb:95:aa:0d
VM 5 IP:192.168.0.2/24 MAC:52:54:00:af:87:56
Servers
Open vSwitch
Physical L2-3 OpenFlow-enabled switch
OpenFlow Controller
OpenFlow

Fig. 3 Overview of the proposed OpenFlow based edge-overlay.

Consequently, the flow latency is increased along with the increase of network diameter as we have presented in our previous study [31], since each switch in the flow path has to process the `OFPT_FLOW_MOD` message and install the flow rules. Therefore, our proposed OpenFlow edge-overlay technique emphasizes limiting the reactive control messages while also reducing the entries in the switches' tables. We achieved this by using hybrid flow setup. In the hybrid approach, whenever a VM is migrated to another location in the virtualized datacenter and connected to a virtual switch, which is running on the physical host server, the controller proactively installs flow rules in the virtual switch for the incoming packets to that VM. 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 packets before transmitting them to their destinations through the datacenter physical network (see Section 3.2.1 for packet headers rewriting).

Moreover, since the physical topology of the datacenter network changes occasionally, but far less frequently than the tenants' virtual networks, the controller can proactively install flow rules in the intermediate switches of the datacenter network for routing the traffic (i.e., in virtualized datacenter, the first and last switches in the flow path are the virtual switches that are running in the host servers). These routing rules are installed based on routing application in the controller, and they are only updated when the datacenter physical topology has changed. Thus, hybrid flow setup reduces the control messages to $2_{\texttt{PACKET\_IN}} + 2_{\texttt{FLOW\_MOD+PACKET\_OUT}}$ for each bi-directional flow. Also, the flow latency is reduced in this approach since only the ingress virtual switch processes the `OFPT_FLOW_MOD` message for each new flow. Note, the OpenFlow controller knows the location of every VM in the datacenter (i.e., virtual switch DPID, and the port number where the VM is connected) as follows:

- When the datacenter 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 server, 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 the migrated 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.

3.2 OpenFlow Based Edge-Overlay

This section presents the proposed edge-overlay for network virtualization in a multi-tenant datacenter, and elaborates on the rewriting techniques of the packets' headers using OpenFlow. Also, the fundamental forwarding components in the multi-tenant datacenter are highlighted in our discussion.

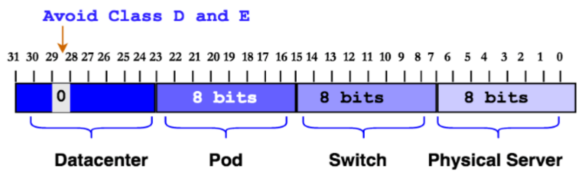


Fig. 4 Hierarchical addressing scheme.

Figure 3 presents an example for the deployment of our solution. Our proposed edge-overlay technique only requires that the datacenter physical switches should be OpenFlow-enabled, which is easily attainable as OpenFlow has been widely supported in the networking industry. Our solution builds upon two observations. First, using Layer-3 instead of Layer-2 of the packet headers for forwarding traffic is commonly used in datacenter networks [24, 26]. Second, unlike the traditional routers, OpenFlow-enabled devices forward packets without changing any fields of the packet headers (e.g., source/destination MAC addresses and TTL) unless explicitly instructed by the controller to do so.

3.2.1 Rewriting the Packet Headers

In our edge-overlay solution, we utilize OpenFlow to stretch the virtual network segments across the physical servers in the datacenter instead of relying on the packet encapsulation. Note that in virtualized datacenter, the first and last switches that receive the VM packets are the virtual switches in the physical host servers, e.g., OVS (Open vSwitch) [28]. Hence, we can use OpenFlow to rewrite the MAC/IP addresses of the tenant's packet before sending it to its destination through the physical datacenter network, and restore the original addresses when this packet reaches the virtual switch at the other end of the communication, where the destination VM is connected. Here is the workflow as shown in Figure 3:

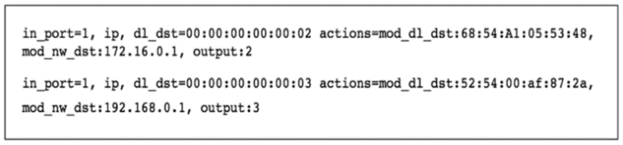
(1) VM5 in physical server 2 sends a packet to VM2 in physical server 1 (both VMs belong to Tenant B).
(2) The virtual switch in physical server 2 receives the outgoing packet to VM2. As the virtual switch does not know where the destination is located in the datacenter network, it sends `OFPT_PACKET_IN` message to the controller.
(3) As the controller keeps tracks of the location, MAC/IP addresses, and VNID of each VM in the datacenter, 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 IP address of the VM packet with IP address of the physical server 1 that hosts VM2 and replace the destination MAC address with Virtual MAC (VMAC) address. This VMAC address is used to tell the virtual switch in the destination physical server 1 how to forward the received packet (see Section 3.2.2 for more details).
(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 VM2 based on the VMAC in the destination MAC 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 VM2, and then it forwards the packet to the destination VM2.

3.2.2 Addressing Scheme

As it is highlighted earlier, OpenFlow has been adopted by a long list of vendors, and using Layer-3 (L3) instead of L2 for forwarding traffic in the datacenter is common. Therefore, our solution can efficiently reduce the forwarding tables in the intermediate switches (i.e., *Core*, *Aggregate*, and *ToR* switches) of the datacenter physical network. In a fat-tree, which is the prevailing topology in datacenter networks, you can easily aggregate the forwarding entries in the switch table and reduce them to the number of output ports. For example, in a fat-tree topology with $k = 4$ such as the topology in Figure 3 (i.e., k is the switch port density in a fat-tree topology), the *Core* switch can have only four forwarding entries in its table, which is equal to the number of pods in the topology. Note that the *Core* switch receives/sends packets from/to one of the *Aggregate* switches in each pod, so it can forward the traffic based on a specific byte in the destination IP address according to a hierarchical addressing scheme as discussed below, which assists in reducing the forwarding entries in the switches' tables.

Traffic forwarding in datacenter networks: The proposed solution utilizes L3 for forwarding VMs traffic in the datacenter network. As discussed in Section 3.2.1, when VM sent a packet to another VM that is located in a different physical server, the destination IP address is replaced with the IP address of the physical server where the destination VM is running. Therefore, the proposed solution can exploit the characteristics of the fat-tree topology and leverage L3 hierarchical addressing scheme (see Figure 4). The fourth byte of the IP can be used to identify the datacenter D_i while the third byte for the pod P_D located in D_i. The second byte is assigned to the ToR switch (i.e., the physical access switch) S_p that is located in pod P_D, and the first byte for the physical server that is connected to S_p. For example, the IP address 10.1.2.1 is assigned to physical server 1 that is connected to ToR switch number 2, and the ToR switch 2 is located in the pod number 1 within datacenter number 10.

This addressing scheme helps to reduce the size of the forwarding tables by aggregating the forwarding entries in a hierarchical fashion since it assigns certain bits of the IP as a location identifier in the multi-rooted tree with multiple core switches, which in turn supports the load balancing techniques such as ECMP. Second, this scheme scales up to 16M physical host servers and 65K *ToR* switches in each datacenter. Note, the proposed addressing scheme is adjustable as the datacenter operators can assign more bits for the hosts or switches by reducing the bits of the datacenter or pod.



```
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:192.168.0.1, output:3
```

Fig. 5 An example of flow rules for incoming packets to OVS in physical server 1 (port 1: trunk, port 2 and 3 for VMs).

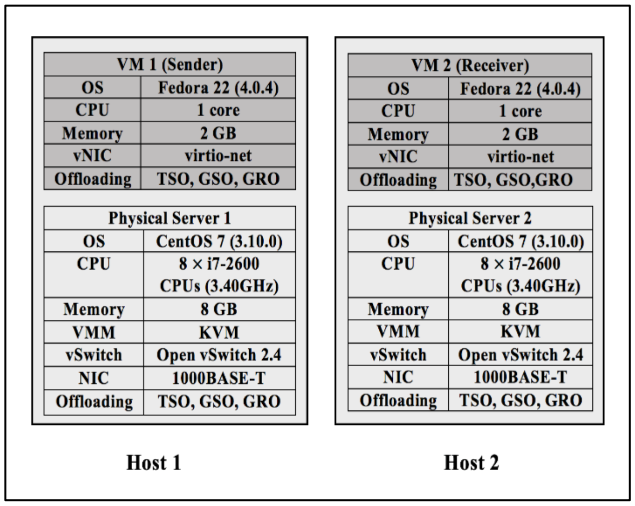
Virtual MAC address (VMAC): VMAC is a simplified method to instruct the virtual switch in the destination on how to forward the received packet from the trunk port. For example, in step 3 of Figure 3, the controller knows that the destination VM2 is connected to port 3 of OVS in physical server 1. Consequently, it installs a flow rule in the OVS of physical server 2 that rewrites the original destination MAC address of the outgoing frame with VMAC = “**00.00.00. 00.00.03**”. We used only the first byte of the VMAC to encode the switch port number where the destination VM is connected. 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 addresses of VM2, and then it forwards the packet out of port 3. Thus, the header rewriting is transparent to VMs in the datacenter. Figure 5 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 connected to the virtual switch as discussed earlier in Section 3.1 (i.e., in the hybrid approach, the virtual switch does not need to consult the controller for the incoming traffic). In this example, the port number one of the OVS is a trunk port while VM1 and VM2 are connected to ports two and three respectively. When the OVS receives a packet with VMAC = “**00.00.00.00.00.03**” in the destination MAC field, the second flow rule in Figure 5 instructs the OVS to change the destination MAC address to **52:54:00:af:87:2a**, IP address to **192.168.0.1**, and forward the packet out of port 3 where the destination VM2 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.

3.2.3 ARP Processing

The VMs in each virtual network need to learn the destination MAC addresses for their communications. In traditional networks, this would be achieved by the ARP (Address Resolution Protocol) broadcasts. In case the host does not have the MAC address that is associated with a certain IP address in its ARP cache, it will send ARP broadcast, which usually controlled by STP (Spanning Tree Protocol) to prevent loops and broadcast storm. However, in large datacenter networks where multipath routing is used for traffic load balancing, STP is discouraged as discussed in Section 1 since it eliminates forwarding over redundant links, and thus imposes restrictions on traffic engineering. In OpenFlow-enabled networks, ARP broadcasts can be processed differently. Since the network controller knows the MAC/IP addresses of each VM in the datacenter, the controller can reply to the ARP requests on behalf of the target VM. When the VM broadcasts an ARP request, the virtual switch, where the VM is connected, will intercept and send the ARP request to the controller via `OFPT_PACKET_IN` message. Then, the controller will generate an ARP reply and send it to the virtual switch via `OFPT_PACKET_OUT` message. This message instructs the switch to forward the ARP reply out of the same port that has received the ARP request, so the VM will receive the ARP reply.

4. Implementation

The proposed OpenFlow based edge-overlay solution is implemented as SDN application running on the Floodlight controller [27], which is an open source OpenFlow controller enhanced and supported by a large community of contributors from academia and industry.



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

Host 1

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

Host 2

Fig. 6 Machine specifications.

It is also compatible with a large number of OpenFlow physical and virtual switches. Our application utilizes both `StaticFlowPusher` and the reactive `Forwarding` modules in the Floodlight controller to implement the aforementioned hybrid flow setup.

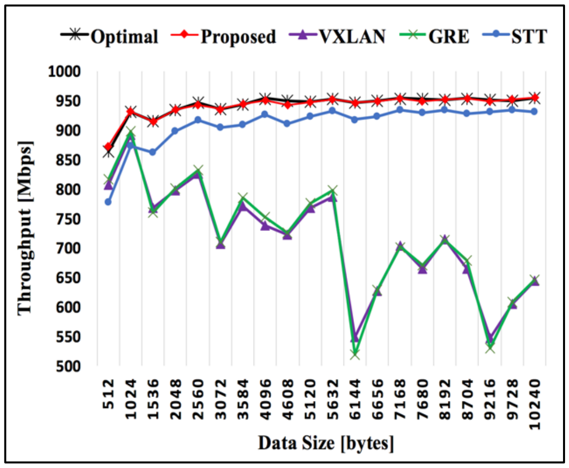


Fig. 7 UDP throughput.

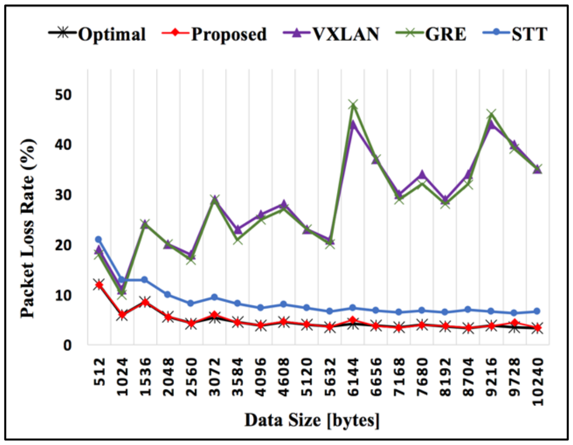


Fig. 8 UDP loss rate.

The `StaticFlowPusher` module is used to proactively install and update the flow rules for incoming traffic into the virtual edge switches, which is Open vSwitch (OVS) in our experiments discussed below (i.e., the incoming traffic in our discussion denotes the traffic received through the trunk port in the virtual switch). These proactive flow rules are installed based on the information that is collected from `TopologyManager` and `DeviceManager` modules in the Floodlight controller. Also, the `IOFMessageListener` module of the Floodlight is used to create the VM profile, which maps each VM to its location in the datacenter (i.e., switch ID and the port number, where the VM is connected) and the MAC/IP addresses of the physical host server. This mapping information is used to supply the `Forwarding` module with the required data during the reactive flow setup for the outgoing traffic from the virtual switches.

In our implementation, we modified the `Forwarding` module of the Floodlight to reactively install flow rules for the outgoing traffic only in the virtual ingress switch when the switch sends an `OFPT_PACKET_IN` message seeking forwarding instructions. Also, we eliminated the ARP request flooding methods that are embedded in the `Forwarding` module as ARP is handled directly by the modified controller, which generates ARP reply using OVS-specific instructions for ARP processing. The `LinkDiscoveryManager` module in the Floodlight controller was utilized to notify our SDN application of the changes in the network topology (i.e., added/removed links between switches).

5. Experimental Evaluation

This section presents and discusses the performance evaluation results of the proposed OpenFlow edge-overlay solution, and it compares the solution performance to the tunneling protocols VXLAN, GRE, and STT.

5.1 Experimental Environment

This set of experiments was designed to assess the performance of the proposed solution 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 an OpenFlow controller. Figure 6 shows the machine specifications of the virtualized environment. The OVS, version 2.4, was used as the virtual switch (i.e., OpenFlow-compliant switch) in server one and server two, and it was configured with OVS Linux-kernel modules, which are included in the OVS distribution. All three servers were connected via Gigabit Ethernet switch. In the experiments, *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.

5.2 Performance Results

The performance results of UDP throughput in the proposed edge-overlay are plotted in Figure 7 with the UDP throughput in VXLAN, STT, and GRE for comparison. Also, the optimal network 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 in the performance plot is the size of the transmitted bytes in UDP packet (excluding headers) by *Iperf* client. The y-axis is the throughput measured by the *Iperf* server.

As it is shown in Figure 7, the STT performance was close to the optimal, while the proposed technique 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 (i.e., TSO).

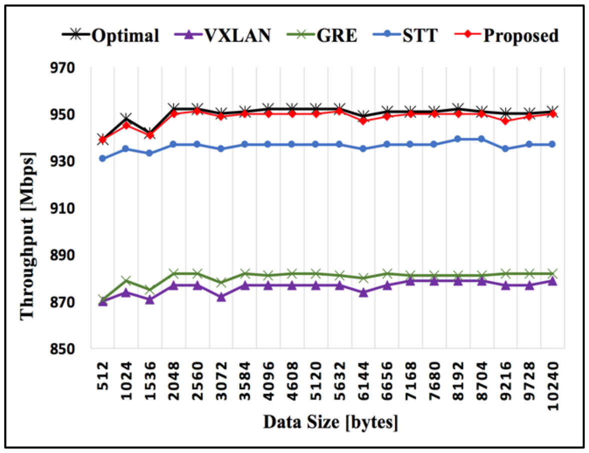


Fig. 9 TCP throughput.

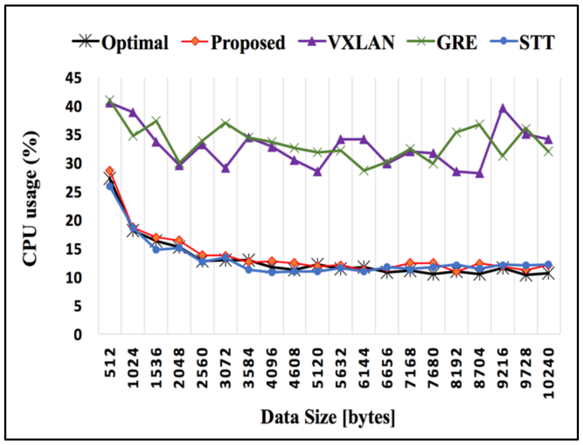


Fig. 10 TCP – CPU usage of the receiver VM.

In our approach, there are no outer headers, so there are no additional fragmentations, and with the support of OVS-kernel module, the performance was high. Contrarily, the throughput of VXLAN and GRE were 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 8 shows the packet loss rate, which is high in VXLAN and GRE. Whereas TCP retransmits the missing fragments, 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 8 shows the packet loss rate increases gradually with VXLAN, notably when data chunk is multiple of 1500 bytes.

The TCP in our solution can use the offloading capabilities of NIC same as STT. Figure 9 shows the throughput of TCP in the proposed technique, which is even higher than STT and very close to the optimal throughput. Also, note that VXLAN and GRE have better throughput in Figure 9 than Figure 7 because of the TCP characteristics, but both have higher CPU usage, as shown in Figure 10, due to the interruption, which is associated with the number of fragments. On the other hand, our solution has less CPU usage similar to STT because both can utilize the standard TCP offloading in the NIC.

6. Discussion

Flow visibility: The design of the proposed solution allows the network controller to obtain a fine-grained flow visibility with less overhead that is achieved by the hybrid flow setup. Since the first packet of the outgoing flow will be sent to the controller, the controller can easily monitor VMs activities in the network. Therefore, our solution is compatible with SDN-based traffic engineering techniques [24] and security applications [23].

Compatibility with network security tools: Unlike STT, which has difficulty traversing stateful firewalls because its encapsulation uses *pseudo*-TCP header, TCP traffic in our solution does not encounter such problem since it does not add any fake headers or even modify L4 fields.

OpenFlow control messages: As reducing the number of the exchanged control messages helps to reduce the overhead on the centralized control plane, our proposed solution only consult the controller for outgoing traffic from the edge switches, but not for the incoming traffic. While this reduces the control messages as discussed in Section 3.1, it also reduces the number of forwarding entries in the switch table, compared to the pure proactive flow setup. For further scalability, distributed controller systems, such as *ONOS* [29] and *OpenDaylight* [30], can be easily integrated with our proposed virtualization technique since we do not restrict the architecture of the controller.

Virtual network segments: The headers rewriting method in our solution provides unlimited scalability regarding the number of supported virtual networks, compared to the tunneling protocols (see Table 1). Also, the proposed hierarchical addressing scheme can accommodate millions of physical hosts and thousands of switches in each datacenter.

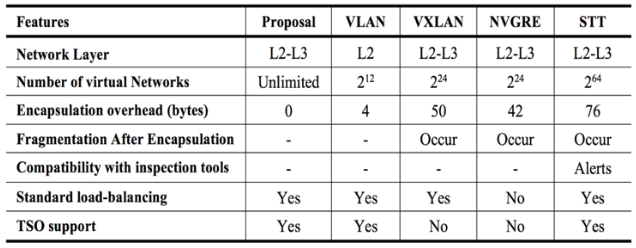
Dual-stack network: The network virtualization principles in the proposed solution can also be applied to IPv6, which has been included in the OpenFlow specification since version 1.2, and it is supported by OVS.

MTU size: Although adjusting the MTU size of VM or enabling jumbo frame in the datacenter network can mitigate the impact of IP fragmentation, they introduce other issues. Reducing the MTU size could affect the network throughput of the VM and increase the operational cost. On the other hand, using jumbo frame requires that every forwarding element in the network must support the extended MTU size. In such case, the datacenter gateways would process a large volume of traffic and fragment every packet to the standard MTU-size when the VMs communicate with users on the Internet.

7. Related Work

Mudigonda et al. [15] presented *NetLord,* which is a Linux kernel module that encapsulates the VM frame with additional MAC/IP headers. The outer MAC headers have the addresses of the end-hosts (i.e., or the software switch in the end-host) where the sender and receiver VMs are located, so the VM frame can be transferred over the L2 network fabric without exposing the VM addresses to the underlying network. In *NetLord,* the outer IP addresses are used to identify the tenant virtual network and the VM, to which the frame belongs, at the destination host. This solution relies on packet encapsulation with additional headers like the tunneling protocols, which fundamentally increases the packet header space and causes packet fragmentation. Kawashima et al. [19] proposed a non-tunneling overlay model for L2 cloud networks. Their model utilizes OpenFlow to rewrite the MAC addresses of the VM frame and replace them with the MAC addresses of the physical servers, where VMs are located, in the cloud. Their module utilizes VLAN tag as VM identifier in the host server. However, their technique does not hide the MAC addresses of the physical servers in the cloud from the tenants, which could expose the cloud network infrastructure to threats such as cloud cartography [32]. Koponen et al. [2] presented *NVP* (Network Virtualization Platform), which uses the STT tunneling protocol; discussed above, for network virtualization in datacenters. Chen et al. [8] proposed *WL2*, multi-tenant network architecture for datacenters that forward traffic based on L2 addresses. However, similar to *NVP, WL2* installs the flow rules proactively, and by default drops packets that do not match any of the installed rules. As discussed in Section 3, the proactive installation of the forwarding rules can reduce the overhead on the controller, but it does not support many of the SDN applications. Also, Guenender et al. [14] published non-encapsulation technique for network virtualization, which replaces the addresses of the VM packet with the addresses of the edge switches.

Table 1: Comparison of network virtualization solutions.



Features	Proposal	VLAN	VXLAN	NVGRE	STT
Network Layer	L2-L3	L2	L2-L3	L2-L3	L2-L3
Number of virtual Networks	Unlimited	2^{12}	2^{24}	2^{24}	2^{64}
Encapsulation overhead (bytes)	0	4	50	42	76
Fragmentation After Encapsulation	-	-	Occur	Occur	Occur
Compatibility with inspection tools	-	-	-	-	Alerts
Standard load-balancing	Yes	Yes	Yes	No	Yes
TSO support	Yes	Yes	No	No	Yes

They used the source TCP port number to encode a certain value that identifies the VM at the destination host. Their method focused only on TCP, and is not generic (e.g., does not support ICMP). Unlike previous works, our proposed edge-overlay is a generic network virtualization technique that supports a wide range of SDN applications running on the SDN controller.

8. Conclusion

In this paper, we discussed the limitations of the overlay tunneling protocols that used in production datacenter networks, and presented an OpenFlow based edge-overlay solution for network virtualization in multi-tenant datacenter networks. Our solution utilizes OpenFlow features to reactively install flow rules at the edge switches for the initiated flows while minimizing the number of control messages. It adopts packet headers rewriting at the ingress edge switch in order to forward the packet over the physical network, and the original addresses are restored at the egress edge switch. Thus, it eliminates the limitations associated with encapsulation protocols and enables fine-grained traffic control. We also evaluated the performance of our network virtualization technique, and the evaluation results show that our solution outperforms the overlay tunneling protocols.

References

[1] Synergy Research Group, “2016 Review Shows $148 billion Cloud Market Growing at 25% Annually,” [online]. https://www.srgresearch.com/articles/2016-review-shows-148-billion-cloud-market-growing-25-annually (accessed August 3, 2017).

[2] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, “Network virtualization in multi-tenant datacenters,” in Proc. of the 11th USENIX NSDI’14, pp. 203-216, 2014

[3] K. Atkinson, G. Wong, and R. Ricci, “Operational Experiences with Disk Imaging in a Multi-Tenant Datacenter,” in Proc. of the 11th USENIX NSDI’14, pp. 217-228, 2104.

[4] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo, “Explicit path control in commodity data

centers: Design and applications," in Proc. of the 12th USENIX NSDI'15, pp. 15-28, 2015.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in Proc. of the ACM SIGCOMM, pp. 63-74, 2008.

[6] R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in Proc. of the ACM SIGCOMM, 2009.

[7] Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network,", in Proc. of the ACM SIGCOMM, 2009.

[8] Chen, C. Liu, P. Liu, B. Loo, and L. Ding, "A scalable multi-datacenter layer-2 network architecture," in Proc. of the 1st SIGCOMM SOSR'15, 2015.

[9] Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machine," in Proc. of the 2nd USENIX NSDI, 2005.

[10] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, C. Wright. Virtual eXtensible Local Area Network (VXLAN): A framework for overlaying virtualized layer 2 Networks over layer 3 networks, RFC 7348, 2014.

[11] P. Garg and Y. Wang. NVGRE: Network Virtualization Using Generic Routing Encapsulation, RFC 7637, 2015.

[12] B. Davie and J. Gross. A Stateless Transport Tunneling Protocol for Network Virtualization (STT). [online]. https://tools.ietf.org/html/draft-davie-stt-08

[13] Open Networking Foundation (ONF). OpenFlow Switch Specification Version 1.5.

[14] S. Guenender, K. Barabash, Y. Ben-Itzhak, A. Levin, E. Raichstein, and L. Schour, "NoEncap: Overlay Network Virtualization with no Encapsulation Overheads,". in Proc. of the 1st SIGCOMM SOSR '15, 2015.

[15] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "Netlord: a scalable multi-tenant network architecture for virtualized datacenters," in Proc. of the ACM SIGCOMM, 2011.

[16] Y. Nakagawa, K. Hyoudou and T. Shimizu, "A management method of IP multicast in overlay networks using OpenFlow," in Proc. of the 1st SIGCOMM HotSDN'12, 2012.

[17] Reference Design: VMware NSX for vSphere (NSX-V), Network Virtualization Design Guide. [online]. https://communities.vmware.com/docs/DOC-27683

[18] R. Kawashima and H. Matsuo, "Performance Evaluation of Non-Tunneling Edge-Overlay Model on 40GbE Environment," in Proc. of the IEEE 3rd Symposium on Network Cloud Computing and Applications (NCCA '14), 2014.

[19] R. Kawashima and H. Matsuo, "Non-Tunneling Edge-Overlay Model using OpenFlow for Cloud Datacenter Networks," in Proc. of IEEE Cloud Computing Technology and Science (CloudCom), 2013.

[20] R. Bonica, C. Pignataro, and J. Touch. A widely deployed solution to the Generic Routing Encapsulation (GRE) fragmentation problem, RFC 7588, 2015.

[21] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined WAN," in Porc. of the ACM SIGCOMM, 2013.

[22] Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "UMON: Flexible and Fine Grained Traffic Monitoring in Open vSwitch," in Proc. of the 11th ACM CoNEXT '15, 2015.

[23] N. Huang, C. Wang, and I. Liao, "An OpenFlow-based Collaborative Intrusion Prevention System for Cloud Networking," in Proc. of the IEEE International Conference on Communication Software and Networks (ICCSN), 2015.

[24] M. Al-Fares, S. Radhakrishnan,B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in Proc. of the 7th USENIX NSDI'10, 2010.

[25] F. Tso, G. Hamilton, R. Weber, C. Perkins, and D. Pezaros, "Longer is better: Exploiting path diversity in data center networks," in Proc.of the IEEE 33rd International Conference on Distributed Computing Systems (ICDCS), 2013.

[26] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo, "Explicit path control in commodity data centers: Design and applications," in Proc. of the 12th USENIX NSDI'15, 2015.

[27] Floodlight Controller. [online]. http://www.projectfloodlight.org/floodlight/

[28] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in the Proc. of the 12th USENIX NSDI'15, 2015.

[29] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: towards an open, distributed SDN OS," in Proc. of the 3rd SIGCOMM HotSDN'14, 2014.

[30] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in Proc. of the IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014.

[31] Aljaedi, C. E. Chow, and J. Rao, "Elastic edge-overlay methods using OpenFlow for cloud networks," in Proc. of the 13th International Conference on Information Technology: New Generations (ITNG), vol. 448, Springer, pp 25-37, 2016.

[32] Z. Xu, H. Wang, and Z. Wu, "A measurement Study on Co-residence Threat inside the Cloud," in the Proc. of the 24th USENIX Security Symposium, 2015.