Non-Tunneling Edge-Overlay Model using OpenFlow for Cloud Datacenter Networks

Ryota Kawashima
Dept. of Computer Science and Engineering
Nagoya Institute of Technology
Aichi, Japan
e-mail: kawa1983@nitech.ac.jp

Hiroshi Matsuo
Dept. of Computer Science
Nagoya Institute of Technology
Aichi, Japan
e-mail: matsuo@nitech.ac.jp

***Abstract*—In current SDN paradigm, an edge-overlay (distributed tunneling) model using L2-in-L3 tunneling protocols, such as VXLAN, has attracted attentions for multi-tenant datacenter networks. The edge-overlay model can establish rapid-deployment of virtual networks onto existing traditional network facilities, ensure flexible IP/MAC address allocation to VMs, and extend the number of virtual networks regardless of the VLAN ID limitation. However, such model has performance and incompatibility problems on the traditional network environment.**

For L2 datacenter networks, this paper proposes a pure software approach that uses OpenFlow virtual switches to realize yet another edge-overlay without IP tunneling. Our model leverages a header rewriting method as well as a host-based VLAN ID usage to ensure address space isolation and scalability of the number of virtual networks. In our model, any special hardware equipments like OpenFlow hardware switch are not required and only software-based virtual switches and the controller are used.

In this paper, we evaluate the performance of the proposed model comparing with the tunneling model using GRE or VXLAN protocol. Our model showed better performance and less CPU usage. In addition, qualitative evaluations of the model are also conducted from a broader perspective.

***Index Terms*—SDN, OpenFlow, edge-overlay, tunneling, datacenter, VXLAN, VLAN, virtual switch**

I. Introduction

Virtual networks are required to realize logically independent VM-to-VM networks for each user (tenant). Current datacenter networks have to support a lot of virtual networks (tenant networks) sharing same physical equipments. Network virtualization can achieve this by separating network traffics of each tenant network using a network ID like VLAN ID (VID). However, various problems came out from traditional VLAN-based virtualization. For example, the number of tenant networks is limited to 4094 because of VID-range, and physical switches have to learn many MAC addresses of VMs.

An edge-overlay (distributed tunneling) model based on L2-in-L3 (IP) tunneling protocol has been adopted by commercial datacenter networks to address above drawbacks without special hardware equipments, and especially VXLAN[1], NVGRE[2], and STT[3] have been used as the tunneling protocol. These protocols encapsulate a whole Ethernet frame of the VM into an IP packet with a tunneling header. This enables each tenant network to have its own IP/MAC address space (address space isolation) and the number of MAC addresses that physical switches learn can be drastically reduced. In addition, each tunneling header has at least 24 bit ID space, and therefore the scalability of the number of tenant networks is ensured.

Such overlay model can induce performance degradation caused by IP fragmentation for tunnel encapsulation. Since tunnels are transparent to VMs, VMs transmit MTU-sized packets without considering encapsulating (outer) headers and traditional Path MTU Discovery is not effective. That is, the number of *physical* frames can be double by IP fragmentation at encapsulation. It is possible to directly adjust MTU size of VM or use *jumbo frames*, however, this causes less manageability and compatibility problems on existing network environment. NVGRE and STT try to address this problem, however, several issues still remain (See section V).

This paper proposes OpenFlow[4] based yet another edge-overlay model for L2 datacenter networks that does not rely on IP encapsulation. Our model leverages frame header rewriting at OpenFlow-enabled virtual switches to prevent the fragmentation and MAC learning problems, and also provides a host-based VID usage to scale the number of tenant networks on the datacenter. In practice, the src/dest MAC addresses of the transmitting frames of the VM are replaced with the physical servers' ones by the virtual switch, and the modified destination address is restored to the original one at the receiver side. Besides, VID is used to distinguish the destination VM rather than the tenant network itself, which implies that VID is unique only within a host. Such host-based VID usage enables unlimited scalability of the number of tenant networks on the datacenter network. Note that our model does not rely on any special hardware equipments such as OpenFlow hardware switch, therefore, our model can be deployed on traditional network environment.

In this paper, the details of the proposed model and performance evaluation comparing with GRE[5] and VXLAN tunnels are described. As a result, the proposed model showed better performance and less CPU usage. In addition, qualitative analysis of the model is presented from a broader perspective.

The rest of the paper is organized as follows. Section II gives the basic of the existing edge-overlay model and section III explains the mechanism of the proposed model. The performance evaluation result is presented in section IV and we discuss the broader aspects of our approach in section V. Section VII concludes this study and gives future work.

II. Edge-Overlay (Distributed Tunneling) Model

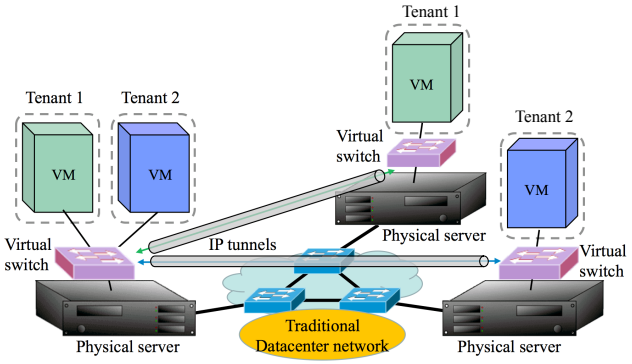

Fig. 1. A representative edge-overlay model with tunnels

Figure 1 illustrates a representative edge-overlay architecture for traditional datacenter networks. Each physical server hosts a software-implemented virtual switch, such as Open vSwitch[6], in order to bridge the virtual NIC (vNIC) to the physical NIC (pNIC). Some virtual switches can have tunnel ports that encapsulates/decapsulates the forwarding frames using tunneling protocols like VXLAN, and generally, each tunnel has a determined source/destination IP address pair. By placing tunnels among the virtual switches, virtual overlay networks can be constructed over the physical network without depending on the physical topology.

As described above, tunnel encapsulation can cause the additional IP fragmentation because VMs do not know that their frames are to be encapsulated. Figure 2 depicts an example of this fragmentation. An application running on the VM sends a large packet that exceeds (VM's) MTU size, then IP layer within the VM divides it (this is not a problem) into two frames. The divided frames are passed down to the underlying virtual switch and it encapsulates each frame with the corresponding tunnel protocol. Then, the former frame is further divided into two frames because the original frame and additional headers certainly exceed (server's) MTU size. That is, the number of *physical* frames increases because of the encapsulation. One other problem of tunneling is that a significant number of tunnels have to be managed as increasing the physical servers on the datacenter network.[1]

III. Proposed Model

Our proposed model is designed to resolve aforementioned issues and its details are described in this section.

There is also a virtual switch on a physical server in this model, but all the virtual switches must support OpenFlow protocol for dynamic frame header rewriting and VID handling. Besides, an OpenFlow controller also needs to be deployed to manage the virtual switches and their flow entries. The characteristics of the proposed model are as follows:

[1]The number of tunnels (unidirectional) is calculated by $N*(N-1)$ where N is the number of physical servers.

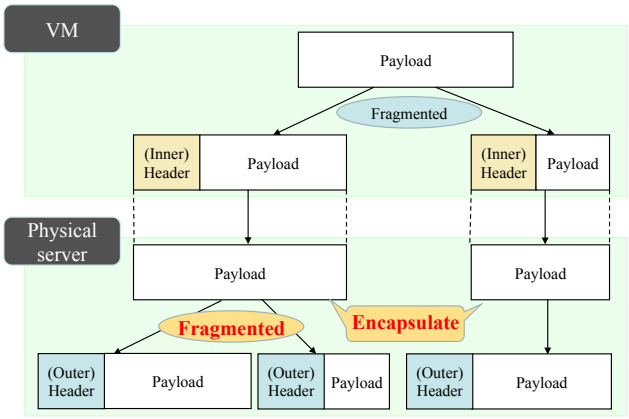

Fig. 2. An example of packet fragmentation caused by tunnel encapsulation

- A frame rewriting method is applied to VMs' frames to ensure end-to-end reachability with hiding their MAC addresses from physical switches
- A host-based VLAN ID usage ensures provisioning of a lot of tenant networks onto the datacenter network
- These features are realized by software-implemented switch and controller that support standard OpenFlow 1.0 protocol without any special hardware equipment

A. Frame Header Rewriting

In order to deliver frames from VMs to other physical hosts, our model uses the frame header rewriting method instead of using tunneling protocols. Figure 3 shows the transition of the frame structure during VM-to-VM frame transmission.

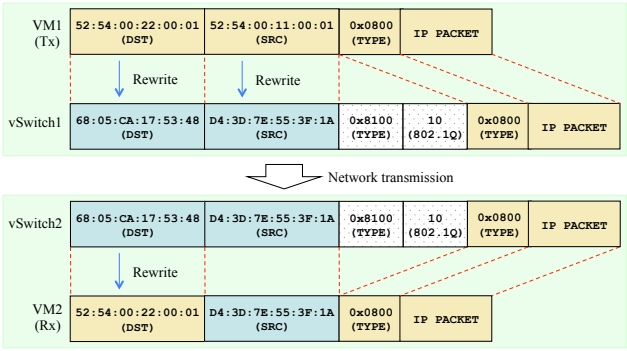

Fig. 3. The transition of the frame structures between end-to-end VMs VM2 can know the valid MAC address of VM1 by the ARP (reply) message.

Suppose that VM1 (52:54:00:11:00:01) linked to vSwitch1 on Server1 (D4:3D:7E:55:3F:1A) sends an Ethernet frame to VM2 (52:54:00:22:00:01) linked to vSwitch2 on Server2 (68:05:CA:17:53:48). Naturally, the destination MAC address is 52:54:00:22:00:01 and the source address is 52:54:00:11:00:01. But these addresses are replaced with the addresses of Server1 and Server2 by vSwitch1 for two main reasons. The first is delivering the frame to Server2 just like traditional physical host-to-host L2 frame transmission. The other reason is that physical switches on the path do not need

to learn MAC addresses of the VMs for FDB (Forwarding Database) and this results in drastic decrease of the number of MAC addresses in the FDB. In addition, a VLAN tag (802.1Q) is inserted into the frame by vSwitch1 and the VID value denotes the destination VM on Server2. At receiving, vSwitch2 inspects the VLAN tag first and rewrites the destination MAC address again to VM2's one. Then, the VLAN-stripped frame is finally delivered to VM2. Note that VM2 does not seem to know the valid VM1's MAC address at first sight, however, VM2 can hold the VM1's address in its ARP table by receiving ARP reply messages from VM1. To hold valid peer MAC addresses is important in considering the VM migration because the MAC and IP addresses of the VM should not be changed after the migration (migration transparency).

With the proposed model, aforementioned IP fragmentation does not occur even though the VM transmits maximum sized frame since there is no additional outer headers to the frame.

B. Host-based VLAN ID Usage

Generally, VLAN is used to logically separate virtual networks on the physical network and each virtual network is identified by the VID value. That is, each VM belonging to the same virtual network has a same VID. This traditional VLAN usage works well in small networks that hold the limited number of tenants, however, further approaches need to be considered for large-scale networks that have to support a lot of tenants because of VID space limitation (4094). Current tunneling protocols addresses this problem by providing larger ID space at least 24 bit instead of using the VID value.

In the proposed model, VID is just only used to determine the destination MAC address at the virtual switch on the receiver host. Each virtual switch has flow entires to map VID value to the destination VM, for example, if the received frame has VID 10 then it is forwarded to VM2 with MAC address 52:54:00:22:00:01. Such VID-VM mappings are managed by an OpenFlow controller, and flow entires are preliminary set to the virtual switch when a VM instance starts up. While, sender side virtual switches also have flow entires to determine inserting VID value based on the destination VM. These entries are set when a first ARP request message is passed to the virtual switch. Then, the request is escalated to the controller as an `OFPT_PACKET_IN` OpenFlow message. In our model, the controller is supposed to know all the information about VM, such as virtual network (tenant), IP/MAC addresses, and physical server (+VID), by working together with an IaaS controller like OpenStack[7]. Therefore, the OpenFlow controller can determine the VID value and set proper flow entries based on the information, and ensure one VM can communicate with VMs of the same tenant only.

This approach extends the scalability of the number of virtual networks because each physical server has its own VID space, and this VID space is enough considering current server machine can run over 100 VMs at most. Theoretically, the number of virtual networks can be increased unlimitedly by adding physical servers to the physical network.

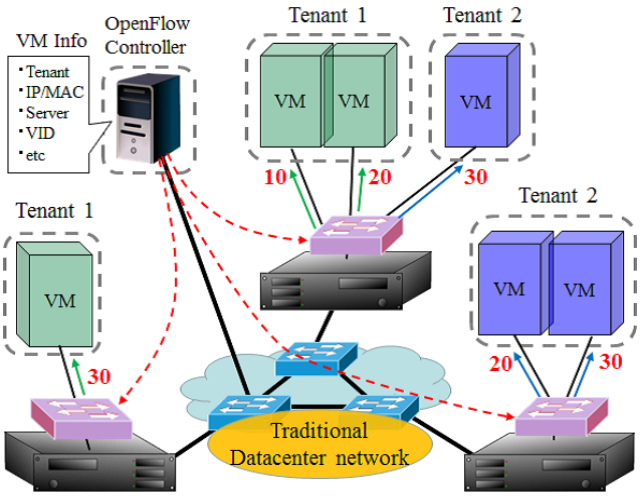

Fig. 4. An example of VID usage under the proposed model

Figure 4 shows the VID usage example in the proposed model. It should be noted that the center server runs three VMs and each VM corresponds to different VID value even though the two VMs on the left belong to the same tenant (Tenant 1) network. VMs on different physical servers can communicate with each other using different VID spaces as long as they belong to the same tenant. For example, the VM on the left server sends a frame to the center server with VID=10, and the leftmost VM on the center server will receive it. Likewise, the leftmost VM also sends a frame to the source VM with VID=30 and the VM on the left server will receive it.

C. Flow Entries

```
in_port=1,dl_vlan=10 actions=mod_dl_dst:52:54:00:11:
22:33,strip_vlan,output:2
in_port=1,dl_vlan=20 actions=mod_dl_dst:52:54:00:44:
55:66,strip_vlan,output:3 ...
```

Fig. 5. An example flow entries for incoming frames (p1:trunk, p2,3:VM)

```
in_port=2,dl_dst=52:54:00:12:34:56 actions=mod_vlan_
vid:50,mod_dl_dst:68:05:CA:17:53:48,mod_dl_src:D4:3D:
7E:55:3F:1A,output:1
in_port=2,dl_dst=52:54:00:AA:BB:CC actions=mod_vl
an_vid:30,mod_dl_dst:7C:C3:A1:86:9A:4B,mod_dl_src:
D4:3D:7E:55:3F:1A,output:1 ...
```

Fig. 6. An example flow entries for outgoing frames (p1:trunk, p2:VM)

Here, example flow entries of a virtual switch (Open vSwitch) are presented. Figure 5 expresses flow entries for incoming frames. Every frames come from a trunk port (e.g. eth0) and a corresponding flow entry is determined

based on the VID value (`dl_vlan`). Each flow entry has `actions` directive that includes destination address rewriting, VLAN tag stripping, and forwarding. For example, a received frame via switch port 1 having VID=20 is passed to port 3 with 52:54:00:44:55:66 destination address without VLAN tag. Figure 6 shows flow entries for outgoing frames. The corresponding flow entry is determined by the source VM and the destination MAC address. The `actions` directive has a VLAN tag insertion and MAC addresses rewriting instructions.

IV. Performance Evaluation

This section gives the performance evaluation results of the proposed model. The performance was measured using Iperf, and GRE and VXLAN were also used for comparison.

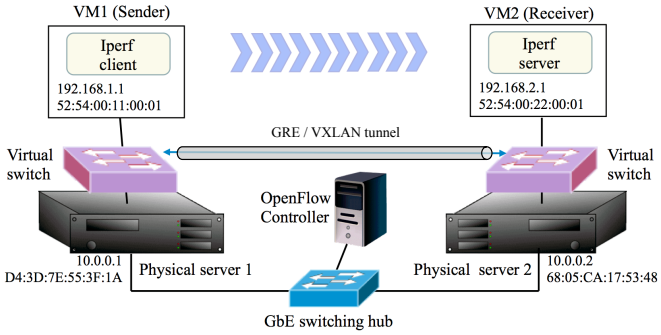

Fig. 7. Experiment environment

TABLE I
Machine specifications

VM 1 (Sender)	
OS	LinuxBean (3.2.0)
CPU	1 core
Memory	1 GByte
vNIC	virtio-net

VM 2 (Receiver)	
OS	LinuxBean (3.2.0)
CPU	1 core
Memory	1 GByte
vNIC	virtio-net

Physical server 1	
OS	CentOS6.4 (2.6.32)
VMM	KVM
vSwitch	Open vSwitch 1.10
CPU	Core i7 (3.60 GHz)
Memory	64 GBytes
Network	1000BASE-T

Physical server 2	
OS	CentOS6.4 (2.6.32)
VMM	KVM
vSwitch	Open vSwitch 1.10
CPU	Core i7 (3.40 GHz)
Memory	32 GBytes
Network	1000BASE-T

TABLE II
Frame processing patterns

	Overhead	VLAN	IP tunnel	Rewriting
Optimal	0 [bytes]	-	-	-
Proposed	4 [bytes]	✓	-	✓
GRE	38 [bytes]	-	✓	-
VXLAN	50 [bytes]	-	✓	-

Figure 7 and table I show the experimental environment. In the experiment, the Iperf client continuously sent UDP/TCP packets for a minute to the Iperf server. The virtual switch on Physical server 1 inserts a VLAN tag and rewrites both MAC addresses of the header in the case of the proposed model. For GRE and VXLAN tunneling, the virtual switches provide tunneling ports between 10.0.0.1 and 10.0.0.2, and encapsulates/decapsulates the frames. In addition, the performance with "optimal" condition (see table II) was also measured to provide upper bound performance under the environment. Note that flow entries of the virtual switches were preliminarily set by the controller before the measurements.

A. UDP Performance Results

The performance results for UDP are presented in figure 8–11. In figure 8 and 9, the x-axis is data chunk size of the Iperf client (excluding IP header) and the y-axis is a bandwidth measured by the Iperf server. The results show that there was no performance differences (and the bandwidth was low) with small data chunk size (less than about 1500 bytes). This was because the packet processing (including interruption and system-call invocation) was relatively heavy. Next, the results of GRE and VXLAN were below the optimal and proposed models in larger packet size. In figure 9, there were some performance drop points on the graph. The performance of VXLAN drastically decreased when data chunk size exceeded 2902 bytes, likewise 2914 bytes for GRE, and 2952 bytes for every models. Apparently, these performance degradations were triggered by the IP fragmentation. As proof of this, 2902 bytes data chunk was divided into two VM's frames (1514[2] and 1464[3] bytes respectively) and 1464 + VXLAN(8) + UDP(8) + IP(20) just equals to the MTU size (1500). And this calculation can apply equally to GRE. Since optimal and proposed model do not encapsulate frames, the number of fragmented packets was less than tunneling models. Consequently, the performance of the proposed model was fairly good compared to the tunneling model with larger packet size.

Figure 10 and 11 express packet loss rate and CPU usage of VM2. For smaller data chunk size, the results of every models were high because there were many physical frames on the network and pNIC has limited frame buffer. For larger data chunk size, tunneling models drastically worse than the others. This can be thought that the differences of the total number of fragmented packets caused the differences of the packet processing load of the kernel.

B. TCP Performance Results

Figure 12 and 13 present throughput and CPU usage results using TCP. Similar with the UDP throughput result, the performance of GRE and VXLAN models were below the optimal and the proposed models when data chunk size becomes large. However, the CPU usage result vastly differs from the result of UDP because of the stream-oriented TCP characteristic. For example, when the Iperf client continuously sends 1024 bytes data chunks, TCP protocol tries to segment the continuous data chunks into 1460 bytes segments in order to maximize each segment size upto MSS (Maximum Segment Size). That is, many VM's frames have MTU-sized payload and this causes further IP fragmentation by tunneling protocols.

[2] 1472 + 8(UDP) + 20(IP) + 14(Ethernet) = 1514
[3] 1430 + 20(IP) + 14(Ethernet) = 1464

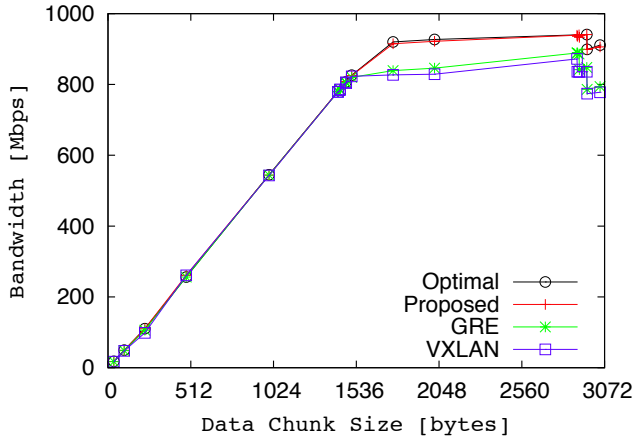

Fig. 8. UDP : Bandwidth

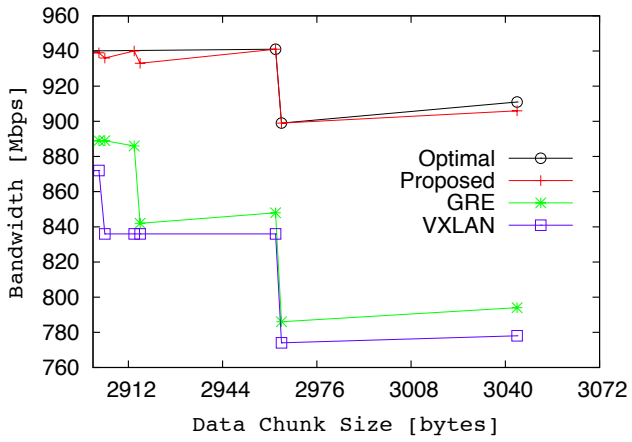

Fig. 9. UDP : Bandwidth (focusing on around 3000 bytes)

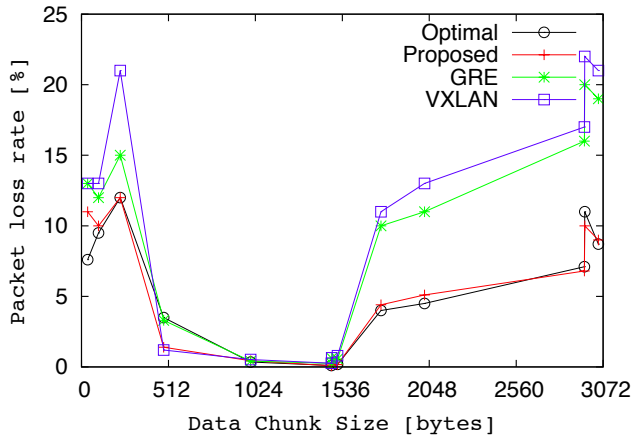

Fig. 10. UDP : Packet loss rate

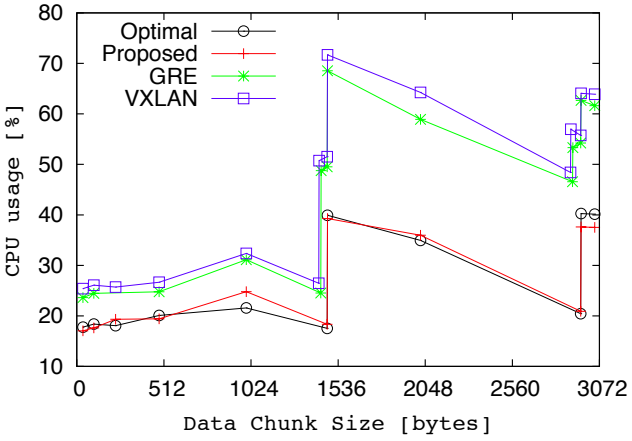

Fig. 11. UDP : CPU usage of the receiver VM

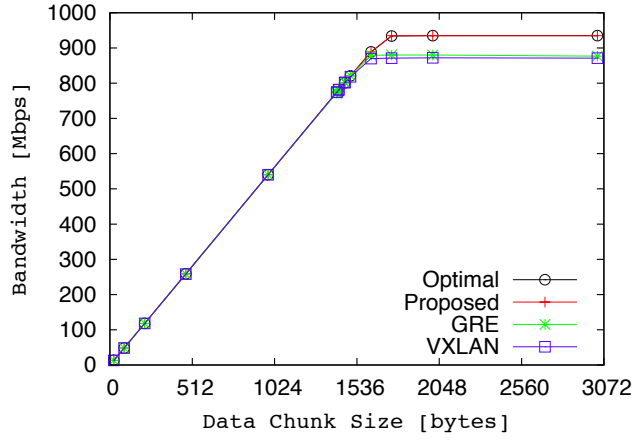

Fig. 12. TCP : Bandwidth

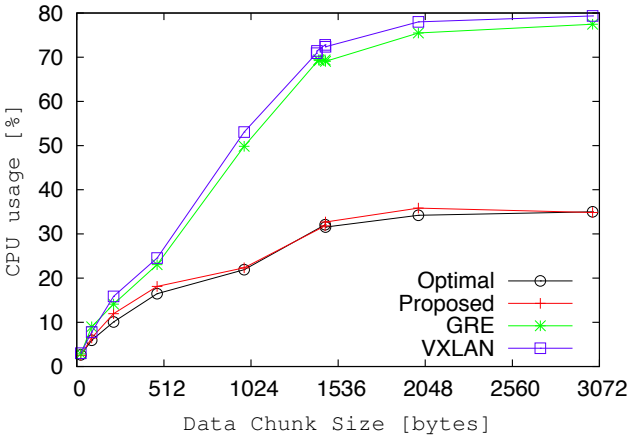

Fig. 13. TCP : CPU usage of the receiver VM

V. Discussion

Performance impact on actual datacenter networks

Benson et al.[8] studied about characteristics of practical datacenters traffics and their results showed that about 40% of packets were over 1400 bytes. Suppose that all these packets are MTU-sized packets, our model can reduce about 30% packets comparing with the tunneling model.

MTU size adjustment / Jumbo frames

It is possible to reduce the MTU size of the VM in order to prevent the IP fragmentation. However, this MTU adjustment increases the operational cost of the datacenter networks if tens of thousands of VMs are running. Instead, we can also use jumbo frame mechanism that enables whole frame encapsulation without the IP fragmentation, but every network equipments must support same extended MTU-size.

Existing tunneling protocols

NVGRE avoids the fragmentation by returning ICMP messages (type=3, code=4), however, this does not work when the VM filters them. STT uses hardware offloading to improve the performance. Since such offloading mechanisms have compatibility problems, some datacenter operators disable the mechanism for stability. Besides, STT packets can be treated as invalid by security appliances because of fake-TCP header.

Broadcast frame handling

In our model, broadcast frames need to be treated by the OpenFlow controller to deliver them to specified VMs. The controller can create copies of the frame and modify their destination MAC addresses from broadcast to individual ones because it knows the physical location of the target VMs.

VM Migration

When a VM is migrated to another host, virtual switch settings also need to be updated. The OpenFlow controller deletes the old flow entries of related switches triggering `OFPT_PORT_STATUS` with `OFPPR_DELETE` message. Likewise, new flow entries can be set into the switches when the status message with `OFPPR_ADD` is received.

VI. Related Work

Matias et al.[9] proposed L2 network virtualization using OpenFlow and L2PNV (Layer 2 Prefix-based Network Virtualization)[10]. In their model, L2 subnetworks can be created by prefix-based MAC addresses, however, OpenFlow switches have to be modified to support MAC subnetting mechanism. Besides, MAC address of physical servers need to be reassigned for the model.

Scissors[11] can reduce header redundancies by replacing the header information to a Flow-ID using OpenFlow. Flow-ID is used for end-to-end routing instead of the header, and the controller coordinates hardware switches on the path. This approach can be applicable to encapsulated flows using tunneling protocols and reduce outer header information. But scissors require a special hardware for the switches to handle Flow-ID and header trimming processing.

HostVLAN[12] realized MAC address based network virtualization by mapping MAC addresses and Logical Network IDs (LNID). HostVLAN identifies the LNID from the source address of the received frame by searching the mapping table and then forwards it to the destination VM through a tenant filtering module. Like our model, HostVLAN targets L2 network and addresses the limitation of the number of tenant networks. However, MAC addresses of VMs are not hidden from the physical switches in HostVLAN model, and existing virtual switches have to be modified to support filtering functions.

VII. Conclusion

This paper proposed a pure software approach for yet another edge-overlay model for L2 datacenter networks, and this model leverages OpenFlow instead of L2-in-L3 tunneling. In practice, two approaches, the frame header rewriting and the host-based VID usage were proposed. The frame rewriting ensures the address space isolation of each virtual network and reduces the number of MAC addresses physical switches have to learn. The alternative VID usage enables unlimited scalability of the number of virtual networks that share the same physical network resources.

A performance evaluation and discussions about several aspects of the proposed model were also conducted in this paper. The results showed that the our model indicated better performance and less CPU usage with a certain packet size and superior compatibility with existing L2 networks.

Further experiments are necessary to evaluate load performance of the model under high bandwidth network and multiple VMs' traffics. In addition, our model is planned to be extended to support IP/MPLS-based networks and inter-datacenter networks in future work.

References

[1] M. Mahalingam, D. Dutt, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", Internet draft, 2013.

[2] M. Sridharan, A. Greenberg, N. Venkataramiah, Y. Wang, K. Duda, I. Ganga, G. Lin, M. Pearson, P. Thaler, and C. Tumuluri, "NVGRE: Network Virtualization using Generic Routing Encapsulation", Internet draft, 2013.

[3] B. Davie, Ed. and J. Gross, "A Stateless Transport Tunneling Protocol for Network Virtualization (STT)", Internet draft, 2013.

[4] N. McKeown, T. Andershnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turneron, and H. Balakris, "OpenFlow: Enabling Innovation in Campus Networks", ACM Computer Communication Review, Vol. 38, Issue 2, pp. 69-74, April 2008.

[5] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, 2000.

[6] Open vSwitch, http://openvswitch.org/.

[7] OpenStack, http://www.openstack.org/.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild", Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, pp.267–280, 2010.

[9] J. Matias, B. Tornero, A. Mendiola, E. Jacob, and N. Toledo, "Implementing Layer 2 Network Virtualization using OpenFlow: Challenges and Solutions", 2012 European Workshop on Software Defined Networking (EWSDN), pp.30–35, 2012.

[10] J. Matias, E. Jacob, N. Toledo, and J. Astorga, "Towards Neutrality in Access Networks: A NANDO Deployment with OpenFlow", International Conference on Access Networks (ACCESS), pp.7–12, 2011.

[11] K. Kannan and S. Banerjee, "Scissors: Dealing with Header Redundancies in Data Centers through SDN", 8th International Conference on Network and Service Management (CNSM), pp.295–301, 2012.

[12] K. Onoue, N. Matsuoka, and J. Tanaka, "Host-based Logical Isolation Technology for Scalable Cloud Networks", IPSJ Transactions on Advanced Computing Systems, Vol.4, No.4, pp.180–190, 2011.