VSE: Virtual Switch Extension for Adaptive CPU Core Assignment in softirq

Shin Muramatsu
Nagoya Institute of Technology
Nagoya, Aichi, 466-8555, Japan
muramatsu@matlab.nitech.ac.jp

Ryota Kawashima
Nagoya Institute of Technology
Nagoya, Aichi, 466-8555, Japan
kawa1983@nitech.ac.jp

Shoichi Saito
Nagoya Institute of Technology
Nagoya, Aichi, 466-8555, Japan
shoichi@nitech.ac.jp

Hiroshi Matsuo
Nagoya Institute of Technology
Nagoya, Aichi, 466-8555, Japan
matsuo@nitech.ac.jp

***Abstract*—An Edge-Overlay model constructing virtual networks using both virtual switches and IP tunnels is promising in cloud datacenter networks. But software-implemented virtual switches can cause performance problems because the packet processing load is concentrated on a particular CPU core. Although multi queue functions like Receive Side Scaling (RSS) can distribute the load onto multiple CPU cores, there are still problems to be solved such as IRQ core collision of heavy traffic flows as well as competitive resource use between physical and virtual for packet processing. In this paper, we propose a software packet processing unit named VSE (Virtual Switch Extension) to address these problems by adaptively determining softirq cores based on both CPU load and VM-running information. Furthermore, the behavior of VSE can be managed by OpenFlow controllers. Our performance evaluation results showed that throughput of our approach was higher than an existing RSS-based model as packet processing load increased. In addition, we show that our method prevented performance of high-loaded flows from being degraded by priority-based CPU core selection.**

***Keywords*-Software-Defined Networking, Network Virtualization, OpenFlow, Virtual Switch, RSS**

I. INTRODUCTION

In public cloud datacenters, virtual networks have to be provided for each tenant to realize multi-tenant services. Generally, there are many virtual network equipments, such as virtual machines (VMs), virtual routers and virtual firewalls, and they are logically connected one another. Each virtual network shares same physical network resources, therefore, a mechanism that logically separates traffic of virtual networks is required. Recently, an Edge-Overlay model (or NVO3[1]) constructing virtual networks using both virtual switches and IP tunnels is promising in cloud datacenter networks. In this model, virtual switches establish IP tunnels, such as VXLAN[2], NVGRE[3], and STT[4], among them and each virtual network traffic can be identified by a tunnel ID included in the tunnel header.

Virtual switches have an impact on actual performance of virtual networks in that their software-based packet processing can be a performance bottleneck. In addition, state of the art virtual switches support high-level functionality including OpenFlow[5], QoS control, and security, which makes worse the performance issue. Furthermore, the notion of Network Functions Virtualization (NFV)[6] that realizes various types of virtual network appliances has gained wide attentions. As a result, further performance improvement of virtual networks is required.

Again, fundamental functionality of virtual switches is implemented in software, and therefore, they occupy CPU resources of the underlying physical server when they do heavy packet processing such as encryption and tunneling. Thus, degradation of VMs' CPU resources results in poor virtual network performance. To address the problem, an additional mechanism that adaptively distributes packet processing load based on CPU resource usage is required.

Many hardware-assisted techniques have been proposed to improve the performance of virtual networks. sNICh[7] allows hardware offloading of OpenFlow processing onto physical NICs. Tanyingyong et al.[8] have proposed a method that caches flow entries of the virtual switch at FlowDirector[9] implemented in Intel® NICs, in order to decrease the load of look-up processing of virtual switches. Receive Side Scaling (RSS)[10] and FlowDirector provide multi-queue functions that alter CPU cores to be interrupted by the hardware NIC based on the received packet headers. Although such high-end technologies have been adopted by many vendor products, vendor-specific technologies can cause vendor lock-in problems.

This paper proposes yet another software packet processing component, Virtual Switch-extension (VSE), to improve performance of virtual networks by adaptively selecting CPU cores for received packet processing using existing hardware equipments. VSE resides in between the virtual switch and the physical NIC, and has its own OpenFlow-based flow table to distribute the flow handling load to adequate CPU cores. In addition, the behavior of VSE can be managed by a unified

controller using an OpenFlow-based protocol. In this paper, we describe architectural design of VSE and how to control its behavior by the controller system. Besides, its implementation details within the device driver of the physical NIC are also explained. Performance evaluation results showed that throughput of our approach was higher than that of a simple RSS-based model as the packet processing load increases. In addition, we show that performance degradation of higher-priority flows can be prevented by the proposed method when the physical server is high load.

The rest of this paper is organized as follows. Section II gives related work. Section III describes overall architecture of the proposed method, and implementation details is described in Section IV. Section V shows the performance evaluation results, and finally, Section VI concludes the paper and gives future work.

II. Related work

sNICh[7] manages a flow table in the physical NIC to reduce the OpenFlow-related packet processing load of virtual switches. However, such hardware-based offloading approach bypasses the processing of high-functional virtual switches, and therefore, applying this method to Edge-Overlay model is difficult.

Sira et al.[11][12] have proposed SR-IOV[13] based approach to improve performance of virtual routers. By using this technology, received packets are directly passed from the physical NIC to the virtual NIC, therefore, the switching overhead between kernel and user space can be reduced. Since this technology also bypasses virtual switches, there remains the same problem to be considered.

Tanyingyong et al.[8] uses FlowDirector[9] supported by Intel NICs to accelerate look up processes of the flow table and reduce the CPU loads of the physical server. However, such vendor specific technologies cannot be applied to data-center networks where various hardware appliances are used. Furthermore, FlowDirector can cause out-of-order packets as reported in [14].

Receive Side Scaling (RSS)[10] and Receive Packet Steering (RPS)[15] distribute the packet processing load onto multiple CPU cores by using hash values calculated from received packet headers. Since hash values are deterministically calculated, heavy-cost flows can be assigned to the same core. Moreover, if VMs are running on CPU cores where intensive packet processing is being performed, performance degradation of virtual network can occur as shown in Section V.

[16] and [17] have been built on top of Intel's Data Plane Development Kit (DPDK)[18]. DPDK adopts Polling Mode Driver (PMD) to exclude interruption overheads and moves all packet processing function to user space. However, the provided functions such as look up processing of virtual switches and packet transmission require fixed CPU core assignment. Thereby, this technology has many challenges in adaptive load distribution based on machine load fluctuation.

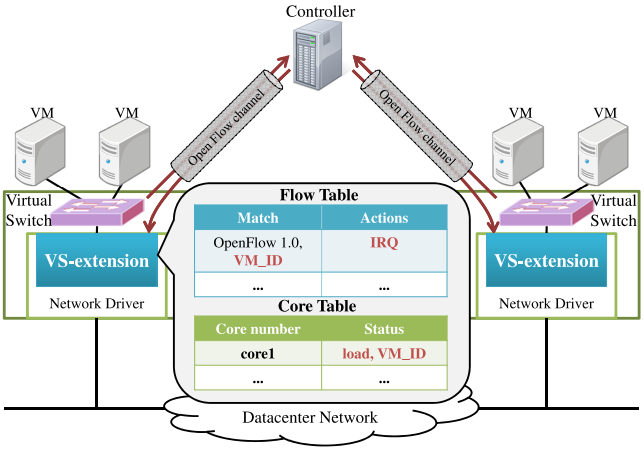

Fig. 1: Overview architecture of the proposed system

III. Architecture of proposed method

As described in Section II, the intensive packet processing load can occur with existing approaches. In this section, we describe architectural overview of the proposed method designed for adaptive load balancing in virtual switches and how unified OpenFlow controllers manage the behavior of our system. Figure 1 depicts overview architecture of the proposed system. Virtual Switch-extension (VSE) resides in NIC driver and controls Soft IRQ cores using its own flow and core tables. Furthermore, the controller manages flow tables of both virtual switches and VSEs using OpenFlow protocol with vendor extensions.

A. Received packet processing in Linux

First, we explain general procedure of packet reception in Linux kernel. The hardware NIC first interrupts the device driver (Hard IRQ) when packet reception. After that, the driver issues Soft IRQs to the same CPU core and the kernel executes protocol stack processing, such as TCP/IP, IPsec, and VXLAN[2]. Since Soft IRQs are to be processed on the same CPU core where Hard IRQs occurred, the packet processing load can concentrate on the single core. In this paper, we propose VSE that adaptively determines Soft IRQ cores for incoming flows based on the current CPU load.

B. Flow Table/Core Table

As illustrated in Fig. 1, VSE has two types of tables, a flow table and a core table. The flow table has OpenFlow 1.0 based match entries for classification of incoming flows. In addition, VSE's flow table can also support encapsulated packet header matching (e.g. VXLAN tunnneling). The core table stores statistical load information of each CPU core as well as VM-core binding relations. Unlike OpenFlow, VSE determines Soft IRQ cores for each flow in the action directive. If the matching process failed, protocol stack processing is to be performed on the same core straightforwardly. When an incoming flow matches an entry of the flow table, VSE references the core table to decide a Soft IRQ target core based

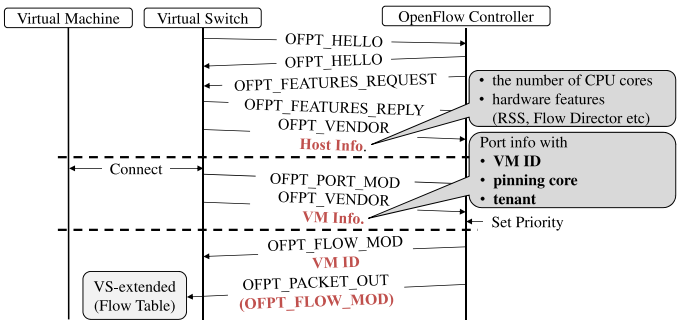

Fig. 2: A protocol between a vswitch and an OpenFlow controller

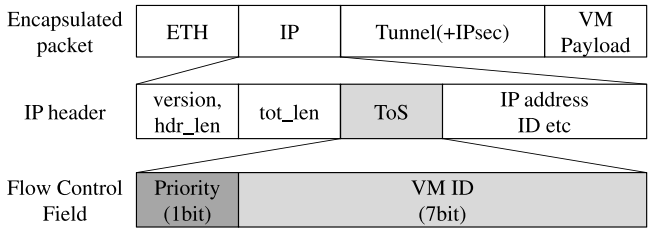

Fig. 3: A flow control field structure in ToS field

on the CPU load and VM-running information. In this way, VSE can adaptively distribute the packet processing load.

C. Flow table configuration

Next, we describe how OpenFlow controllers configure flow tables of VSEs using the existing OpenFlow protocol framework. When a virtual switch establishes an OpenFlow session to the controller, the switch notifies how many CPU cores are equipped with the physical server and what hardware-acceleration features such as RSS and FlowDirector are supported using OpenFlow's vendor extension message (`OFPT_VENDOR`). When a VM starts up, the virtual switch sends an `OFPT_PORT_MOD` message to the controller. In our approach, the virtual switch additionally sends VM-related information such as `VM_ID`, tenant ID, CPU core where the VM is running on. The controller decides whether the proposed method is enabled and how the flow table of VSE should be configured based on these information. Flow tables of VSEs are configured using an `OFPT_FLOW_MOD` message structure like the table of OpenFlow switches, however, VSE does not have direct connection with the controller. The controller therefore encapsulates the message with OpenFlow's `OFPT_PACKET_OUT` message and the virtual switch acts as a proxy to forward the `OFPT_FLOW_MOD` message to the VSE.

D. Flow Control Field

Most incoming flows can be identified by an address—port pair in the packet headers, however, special care is required for certain type of flows that includes IP fragmentation or IPsec. The proposed method provides an optional solution to identify these flows by re-defining Type of Service (ToS) field in the IP header or IEEE 802.1Q VLAN tag as Flow Control Field (FCF). Figure 3 and 4 illustrates packet structure of FCF. If original ToS field usage is required in datacenter

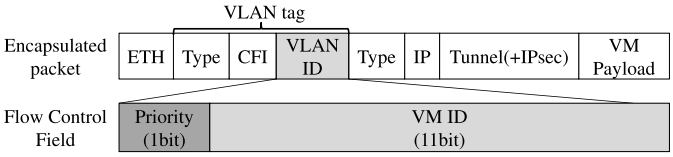

Fig. 4: A flow control field structure in VLAN tag

TABLE I: Additional rules

vSwitch	`Actions:set_flow_control_field`
VS-extend	`Match:VM_ID, Actions:IRQ`

networks, the VLAN tag is used as FCF instead. VSE uses VM ID to distinguish the destination VM running on the same physical server. For example, existing OpenFlow-based flow matching cannot identify destination VMs for VXLAN over IPsec because VM's IP header is encapsulated and encrypted. FCF enables outer IP header to include the destination VM identifier and VSE can use this field to determine appropriate CPU cores. Moreover, the same benefits can be applied for IP fragmented packets. Since IP tunneling can cause IP fragmentation at the encapsulation process, VM ID in FCF is an only key to know the destination VM for receive-side VSEs. In our model, the use of FCF is an option and controllers can insert flow entries including standardized matching fields only. As shown in Fig. 3 and 4, FCF has not only VM ID, but priority bit frag. When the frag is set, the incoming flows are preferentially assigned to lower load CPU cores.

E. Received packet processing with VSE

Here, we summarize packet processing flows in VSE-enabled systems as follows:

1) The physical NIC notifies packet reception to the corresponding network driver as Hard IRQ.
2) VSE performs flow matching based on its flow table.
3-a) If matching succeeds, VSE decides a CPU core for Soft IRQ based on the flow and core tables.
3-b) If matching fails, VSE simply assigns the current CPU core for Soft IRQ.
4) The driver delivers the packet to the kernel layer with processing Soft IRQ.

F. Appended Action rules

In order to realize VSE's functionality, existing OpenFlow's Action rules have to be extended. We show the appended rules in table I. To support FCF setting, `OFPAT_SET_FCF` action is newly supported between virtual switches and controllers. On the other hand, `OFPAT_SET_IRQ_CORE` action is added for VSE.

IV. Implementation

We have implemented VSE's functionality on a Mellanox ConnectX®-3 driver[19]. In this section, implementation details of VSE focusing on how VSE controls Soft IRQ core selection is described. Figure 5 shows a flowchart of received packet processing with VSE. Our implementation is based

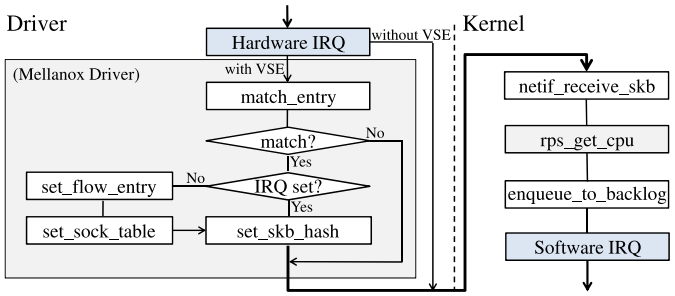

Fig. 5: A flowchart of frame processing in Mellanox Driver

on Receive Packet Steering (RPS) feature provided by Linux kernel. RPS is a software implementation of RSS. If RPS function is enabled, `rps_get_cpu` function selects a Soft IRQ core based on a `sock_flow_table` that contains `<hash:core>` relation pairs after the kernel received the packet from the driver by `netif_receive_skb` function. Then, `enqueue_to_backlog` function executes a softirq to the core. Like RSS, RPS also simply determines CPU cores to interrupt based on hash values (`skb->rxhash`) calculated from received packet headers. Meanwhile, the proposed method adaptively determines Soft IRQ cores based on the CPU load. First, every incoming flows are matched with the VSE's flow table in `match_entry` function. If matching fails, VSE simply assigns the current CPU core for Soft IRQ. If matching succeeds, VSE executes a softirq to the core specified in the entry. In the case of practical CPU core number is not given, `set_flow_entry` function references the core table to determine the softirq core. For example, if the core table has flow entries given in table II, core 2 is selected because its load is lower and VM is not running on the core. Then, VSE sets `IRQ:2, rxhash:skb->rxhash` to the entry. Existing methods simply assign Soft IRQ cores based on the hash values, whereas our proposed method considers fluctuating CPU loads. When all cores are high-load, VSE does not alter Soft IRQ cores. Next, `set_sock_table` function sets `skb->rxhash` and the corresponding core number to `sock_flow_table` provided by Linux kernel. The kernel is supposed to select VSE's intended softirq core because `sock_flow_table` retains the relation between the hash value and the core number. For subsequent packets of the same flow, `set_skb_hash` function overwrites the `skb->rxhash` value such that the packet is processed on the same core included in the entry. When the entry becomes useless, an aging timer of the VSE deletes the entry. Note that our VSE implementation does not require any kernel modification because entire VSE's functionality is implemented in the Ethernet driver.

V. Evaluation

In this section, we evaluate performance of the proposed method for packet processing by comparing with the existing RSS model. First, we evaluated actual throughput of VM-to-VM communication with heavy packet processing, 256-bit AES block cipher and VXLAN tunneling used as substitute for

TABLE II: Statuses of Core Table and Flow Table

Core Table

core	status
0	`load:10, VM pinning, VM_ID:1`
1	`load:50, VM pinning, VM_ID:2`
2	`load:20`
3	`load:50`

Flow Table

Match	Actions
`VM_ID=1`	`IRQ:3,rxhash:45678`
`VM_ID=2`	`IRQ:-,rxhash:-`

VXLAN over IPsec as preliminary evaluation. In this experiment, two types of heavy workload of CPU cores, two flows are simultaneously processed on a same core and each flow is handled on VM's running core, were evaluated. Next, effect of the adaptive load balancing by the proposed method was evaluated. Finally, we show the priority-based load distribution results. Figure 6 and Table III give the experimental environment. The throughput of VM-to-VM communication between different physical servers was measured using Iperf[20] with various packet sizes (64, 1400 and 8192 bytes) .

A. Preliminary evaluation

In this experiment, an Iperf client in VM1 and VM3 continuously sends UDP packets to the Iperf server in the counterpart VM for 80 seconds, and the average throughput during 60 seconds of the Iperf server was evaluated. Table IV shows performance results of two cases, 'two flow-collision' and 'PM/VM flow-collision'. The 'two-flow-collision' shows that two independent flows are handled on the same CPU core. The 'PM/VM-flow-collision' shows that a flow is processed by both the virtual switch and the destination VM on the same core. Throughput of 'non-flows collision' was 1509.9 Mbps when packet size was 1400 bytes, by contrast, that of 'flows collision' decreased. Furthermore, when packet size was 8192 bytes, VM-to-VM communication was not established for 'two flow-collision' case, even though increase in throughput can been seen for 'non-flows collision' case. Besides, the

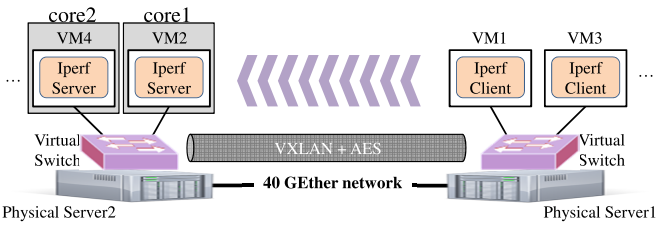

Fig. 6: Experimental environment

TABLE III: Machine specifications

	Physical Server1	Physical Server2	VM
OS	CentOS 6.5(2.6.32)		Ubuntu 12.04(3.5.0)
CPU	Core i5(4 core)	Core i7(4 core)	1 core(Pinning)
Memory	16 GB		2GB
Buffer	4 MB		
Network	40Gbit Ethernet		-

TABLE IV: Preliminary evaluation [Mbps]

packet size	64bytes	1400bytes	8192bytes
flows collision	39.5	954.9	-
non-flows collision	39.4	1509.9	1702.4
flow-VM collision	19.7	617.3	4.0
non flow-VM collision	19.8	754.6	880.1

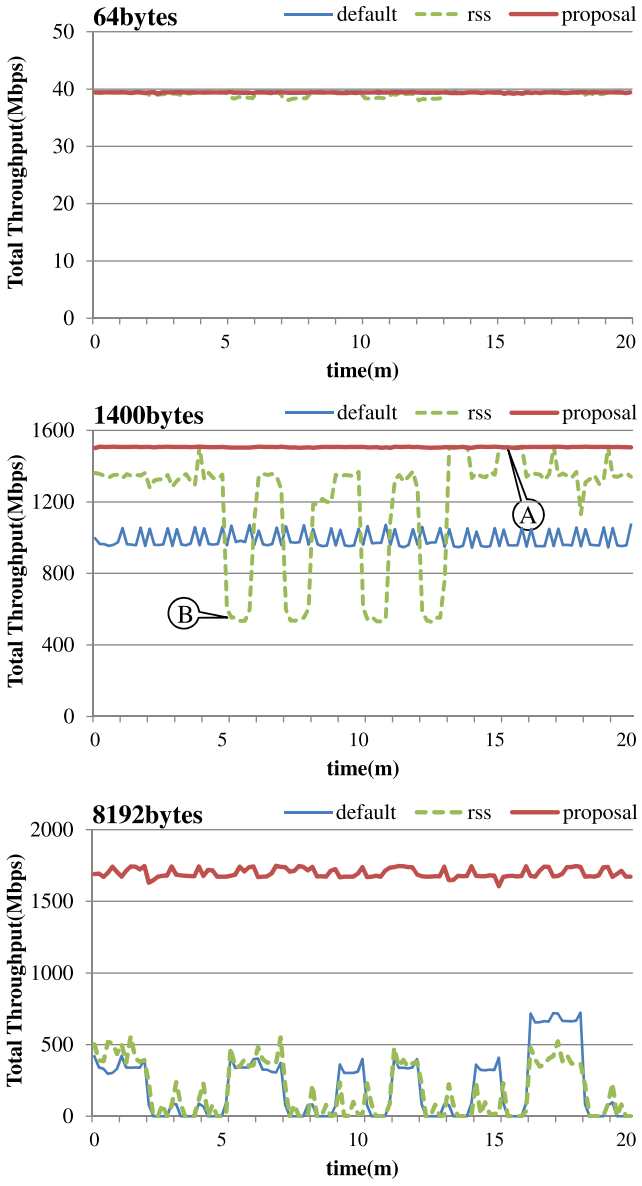

Fig. 7: Total throughput in two VM-to-VM communications

throughput of 'flow-VM collision' degraded when packet sizes were 1400 and 8192 bytes. This was because the virtual switch occupied CPU resources for heavy packet processing. These results proved that both types of collision can degrade the end-to-end performance requiring heavy packet processing.

B. Two VM-to-VM communications

Next, we evaluated total throughput of two VM-to-VM communications. In this experiment, the Iperf client sequentially created UDP flows 20 times in order to vary rxhash values at the receiver side, and each flow continues 1 minute. We modified the Iperf client to assign a source port because the underlying kernel selects different port number each time. In our experiments, same addresses and port numbers were used for every model 'default', 'rss' and 'proposal'. The 'default' model does not use RSS or similar technologies. The 'rss' model uses RSS function at the physical NIC. The 'proposal' also supports RSS as well as VSE feature implemented in NIC driver (Mellanox 2.2.0). Note that flow entries of VSEs and virtual switches were set in advance (VM1's flow:core2, VM2's flow:core3).

TABLE V: The total number of Soft IRQs per CPU core

64bytes	core0	core1	core2	core3
default	0	0	0	165,159,804
rss	39,936,135	53,710,644	41,317,702	28,924,990
proposal	0	0	82,634,610	82,634,450
1400bytes				
default	0	0	0	109,134,696
rss	35,263,529	48,655,872	41,435,459	29,002,358
proposal	0	0	82,873,766	82,875,742
8192bytes				
default	0	0	0	184,126,030
rss	197,691,348	5,391,984	6,221,397	10,844,689
proposal	0	0	171,580,236	171,738,991

1) Total throughput of 2 flows:

Figure 7 expresses a relationship between elapsed time and the total bandwidth of VM-to-VM links. When the packet size was 64 bytes, throughput was the same for each model. On the other hand, when the packet size was 1400 bytes, throughput of 'default' peaked at 1000 Mbps and the result of 'rss' varied in each flow. The point 'A' in 1400 bytes' graph shows that RSS optimally distributed the packet processing load by accident. By contrast, the point 'B' indicates that RSS caused flow collision on VM2's running core, therefore the throughput of 'rss' was lower than that of 'default'. Furthermore, when the packet size was 8192 bytes, the throughput of 'proposal' was obviously higher than that of the other models.

2) The total number of Soft IRQs:

Table V shows that the 'rss' model frequently handled Soft IRQs on VMs' running cores, and moreover core0 handled a significant number of interruptions compared with the others when the packet size was 8192 bytes. This was because all fragmented packets except the first one had the same hash value and naturally they ware interrupted to the same core (core0). In contrast, the 'proposal' model processed softirq on core2 and core3 only, therefore, the flow-VM collision was not caused and throughput did not decrease.

3) Packet loss rate at the receiver:

Next, we show the packet loss rate at the receiver VMs in Table VI. As you can see, the packet loss rate of 'default' and 'rss' models were apparently higher when packet sizes were 1400 and 8192 bytes. In these models, concentration of packet processing on a specific CPU core caused frequent buffer overflow at the physical NIC.

C. Throughput of priority-based flows

Finally, we evaluated throughput of priority-based flows with the priority bit in FCF field for three VM-pairs. In

TABLE VI: Packet loss of VMs[%]

	64bytes		1400bytes		8192bytes	
	VM1	VM2	VM1	VM2	VM1	VM2
default	0.1	0.1	34.5	34.5	51.4	52.4
rss	0.8	0.9	15.8	17.0	95.3	43.5
proposal	0.1	0.1	0.2	0.2	14.8	14.7

TABLE VII: Throughput of priority-based flows

	64bytes			1400bytes		
	VM1	VM2	VM3	VM1	VM2	VM3
rss	19.4	19.5	19.3	481.8	465.6	406.3
proposal	19.7	19.7	19.7	745.3	635.0	646.4

	8192bytes		
	VM1	VM2	VM3
rss	3.4	192.8	20.9
proposal	844.8	466.3	181.4

the proposed method, VM1's flow was set as a high priority flow, and was processed on core3 while the other flows were processed on the other cores where the destination VMs were running on. Table VII gives average throughput of receiver VMs during 20 minutes. From the result, throughput of VM1 was higher when the packet size were 1400 and 8192 bytes respectively. These results indicate that the higher priority flow is preferentially processed on low-load CPU core to prevent its performance from being decreased.

D. Discussion

As shown in the evaluation results, our method can improve performance of virtual networks by distributing heavy packet processing load on adequate CPU cores. This is beneficial in SDN-enabled virtual networks based on high-functional virtualization edges (e.g. an edge-overlay model). In addition, simple hash-based load-balancing methods such as RSS and RPS are not suitable for certain flows that require heavy packet processing. The proposed priority-based approach using FCF is effective for encapsulated or fragmented packets because VMID can identify the destination VMs correctly. This enables OpenFlow controllers to differentiate tenants by their priorities, which is useful for some datacenter systems adopting grading system.

VI. Conclusion

Network virtualization based on IP tunneling has been getting attention lately for multi-tenant datacenter networks. In such network, intelligent packet processing is often performed by software-implemented virtual switches, and therefore, the performance of the virtual switches is important for the efficiency of overall virtual networks.

This paper has presented VSE that is a software component for adaptively distributing the packet processing load of virtual switches without specific hardware-assistance. VSE matches incoming flows using its own OpenFlow-based flow table, and determines an appropriate CPU core for Soft IRQ on-the-fly. In addition, the flow table of VSE can be managed by a unified controller using the OpenFlow message format. The performance evaluation result showed that the packet processing load was properly distributed in the proposed method, moreover, it confirmed that performance of higher-priority flows were higher than non-priority flows by preferentially assigning low-load CPU core. We are planning to implement the southbound protocol between VSEs and controllers. Besides, dynamic alternation of CPU core selection based on CPU load change has to be implemented in future work.

ACKNOWLEDGMENTS

This work was supported in part by MEXT KAKENHI Grant Number 25400113.

References

[1] D. Black, J. Hudson, L. Kreeger, M. Lasserre, and T. Narten, *An Architecture for Overlay Networks(NVO3)*, Aug. Internet Draft. 2013.
[2] P.Agarwal, D.Dutt, K.Duda, P.Agarwal, L.Kreeger, T.Sridhar, M.Bursell, and C.Wright, "Virtual eXtensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks," RFC 7348, 2014.
[3] M. Sridharan, A. Greenberg, N. Venkataramiah, Y. Wang, K. Duda, I. Ganga, G. Lin, M. Pearson, P. Thaler, and C. Tumuluri, "Network virtualization using generic routing encapsulation," Internet Draft. 2014.
[4] B. Davie and J. Gross, "A stateless transport tunneling protocol for network virtualization (STT)," Internet draft 2014.
[5] 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, no. 2, pp. 69–74, April 2008.
[6] *Network Functions Virtualization*, SDN and OpenFlow World Congress, http://portal.etsi.org/nfv/nfv_white_paper.pdf, Oct. 2012.
[7] K. Ram, J. Mudigonda, A. Cox, S. Rixner, P. Ranganathan, and J. Santos, "sNICh: Efficient last hop networking in the data center," *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (SNCS 2010)*, pp. 1–12, Oct. 2010.
[8] V. Tanyingyong, M. Hidell, and P. Sjodin, "Using hardware classification to improve pc-based openflow switching," *Proc. IEEE 12th International Conference on High Performance Switching and Routing (HPSR 2011)*, pp. 215–221, July 2011.
[9] "Flowdirector," www.kernel.org/doc/Documentation/networking/ixgbe.txt.
[10] "Design considerations for efficient network applications with intel multi-core processor-based systems on linux," http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/multi-core-processor-based-linux-paper.pdf.
[11] M. Rathore, M. Hidell, and P. Sjodin, "PC-based router virtualization with hardware support," *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, pp. 573–580, March 2012.
[12] M. Rathore, H. Markus, and S. Peter, "KVM vs. LXC: comparing performance and isolation of hardware-assisted virtual routers," *American Journal of Networks and Communications*, no. 2, pp. 88–96, Aug. 2013.
[13] "Single Root I/O Virtualization," http://www.pcisig.com/specifications/iov/single_root.
[14] W. Wu, P. DeMar, and M. Crawford, "Why Does Flow Director Cause Packet Reordering?" *IEEE Commun.Lett.*, vol. 15, pp. 253–255, 2011.
[15] "Receive packet steering," https://www.kernel.org/doc/Documentation/networking/scaling.txt.
[16] H. Masutani, Y. Nakajima, T. Kinoshita, T. Hibi, H. Takahashi, K. Obana, K. Shimano, and M. Fukui, "Requirements and design of flexible NFV network infrastructure node leveraging SDN/openFlow," *Proc. 2014 International Conference on Optical Network Design and Modeling*, pp. 258–263, May 2014.
[17] G. Pongracz, L. Molnar, and Z. Kis, "Removing roadblocks from SDN: OpenFlow software switch performance on intel DPDK," *Proc. 2013 Second European Workshop on Software Defined Networks (EWSDN)*, pp. 62–67, Oct. 2013.
[18] "DPDK: Data plane development kit," http://dpdk.org/.
[19] "ConnectX EN 10 and 40 gigabit linux driver," http://www.mellanox.com/page/products_dyn?product_family=27.
[20] "Iperf," http://iperf.sourceforge.net/.