IEICE TRANS. COMMUN., VOL.E98–B, NO.11 NOVEMBER 2015

PAPER Special Section on Network Systems for Virtualized Environment

Accelerating the Performance of Software Tunneling Using a Receive Offload-Aware Novel L4 Protocol

Ryota KAWASHIMA[†a)] *and* **Hiroshi MATSUO**[†b)], *Members*

SUMMARY An L2-in-L3 tunneling technology plays an important role in network virtualization based on the concept of Software-Defined Networking (SDN). VXLAN (Virtual eXtensible LAN) and NVGRE (Network Virtualization using Generic Routing Encapsulation) protocols are being widely used in public cloud datacenters. These protocols resolve traditional VLAN problems such as a limitation of the number of virtual networks, however, their network performances are low without dedicated hardware acceleration. Although STT (Stateless Transport Tunneling) achieves far better performance, it has pragmatic problems in that STT packets can be dropped by network middleboxes like stateful firewalls because of modified TCP header semantics. In this paper, we propose yet another layer 4 protocol (Segment-oriented Connection-less Protocol, SCLP) for existing tunneling protocols. Our previous study revealed that the high-performance of STT mainly comes from 2-level software packet pre-reassembly before decapsulation. The SCLP header is designed to take advantage of such processing without modifying existing protocol semantics. We implement a VXLAN over SCLP tunneling and evaluate its performance by comparing with the original VXLAN (over UDP), NVGRE, Geneve, and STT. The results show that the throughput of the proposed method was comparable to STT and almost 70% higher than that of other protocols.

key words: *Software-Defined Networking, network virtualization, Network Function Virtualization, datacenter networks*

1. Introduction

Many functionalities of various network appliances including routers, switches, and firewalls have been migrated to virtual networks with the notion of Network Function Virtualization (NFV) [1]. A cutting-edge network virtualization is mainly based on two approaches, *Hop-by-Hop* and *Edge-Overlay*, however, the former requires a fully OpenFlow [2] ready network environment. The Edge-Overlay approach or Network Virtualization Overlays over Layer 3 (NVO3) [3] introduces L2-in-L3 tunneling between Tunnel End-Points (TEPs) to convey virtual traffic over physical networks. That is, the performance characteristic of the tunneling protocol can affect overall performance of virtual networks. In the NFV concept, high-performance ability of virtual networks is a key requirement for service quality.

VXLAN (Virtual eXtensible Local Area Network) [4] and NVGRE (Network Virtualization using Generic Routing Encapsulation) [5] are representative L2-in-L3 tunneling protocols used in commercial datacenters. In particular, VXLAN has been adopted as RFC 7348 and supported by various platforms. However, VXLAN-based communications with software switches limit the performance of virtual networks compared with *no-encapsulation* communications [6], and the performance of NVGRE is almost the same with VXLAN. STT (Stateless Transport Tunneling) [7] had been proposed to achieve high-performance tunneling by exploiting a hardware offload feature of NIC (TCP Segmentation Offload [8], TSO). The STT protocol has a pseudo-TCP header to disguise STT packets as normal TCP packets as well as to make the STT protocol connection-less and stateless. The use of the pseudo-TCP header, however, causes an additional problem in that middleboxes such as stateful firewalls and load balancers can discard STT packets because of the modified header semantics. Therefore, yet another high-performance and middlebox-transparent tunneling protocol is required.

In our previous work [9], we analyzed the performance of STT and evaluation results revealed that the cause of the high-performance ability of the protocol is 2-level software packet pre-reassembly before decapsulation, rather than TSO feature. In practice, a software-implemented Generic Receive Offload (GRO) [10] feature which reassembles the received packets before SoftIRQ improves the performance of STT considerably. This implicates that the pseudo-TCP header (or the TSO feature) is not necessary to realize high-performance tunneling. Based on the previous results, this paper proposes a novel L4 protocol named SCLP (Segment-oriented Connection-less Protocol) [*]. SCLP is a simple connection-less protocol but is designed to take advantage of the GRO feature unlike UDP, and the performance of existing tunneling can be improved by replacing the original L4 protocol with SCLP. Besides, SCLP does not face the packet discarding problem because header semantics of existing protocols are not changed.

In this paper, we describe the design and implementation of the SCLP protocol as well as how SCLP enables the 2-level pre-reassembly. In addition, we evaluate the performance of SCLP-based tunneling in a CentOS 6.6 platform by comparing with no-encapsulation, original VXLAN, NVGRE, STT, and recently appeared Geneve [12]. The results show that the throughput of the proposed method (VXLAN over SCLP) was comparable to no-encapsulation and STT, and almost 70% higher than that of

Manuscript received March 27, 2015.
Manuscript revised June 26, 2015.

[†]The authors are with the Dept. of Computer Science and Engineering, Nagoya Institute of Technology, Nagoya-shi, 466-8555 Japan.
a) E-mail: kawa1983@ieee.org (Corresponding author)
b) E-mail: matsuo@nitech.ac.jp

DOI: 10.1587/transcom.E98.B.2180

[*]The concept of SCLP is first introduced at IEEE NetSoft 2015 [11].

all the other protocols.

The reminder of this paper is organized as follows. Section 2 discusses related work. Section 3 explains the effect of an L4 protocol type for tunneling. The design of the SCLP protocol is described in Sect. 4 and Sect. 5 provides implementation details of offload features for SCLP. Section 6 gives the evaluation results, and finally, Sect. 7 concludes this study and shows future work.

2. Related Work

The STT [7] protocol had been proposed by Nicira, Inc. (acquired by VMware Inc.) for high-performance L2-in-L3 tunneling. STT can exploit the TCP Segmentation Offload (TSO) feature of NIC by using the pseudo-TCP header. A VMware's report [13] said that the performance of STT was almost equal to a *no-encapsulation* model. However, STT has pragmatic problems in that STT packets can be dropped by network appliances which can interpret the TCP header. Even though explicit flow entries accepting STT packets based on the destination port number are set, middleboxes' TCP state machine function can implicitly discard the packets as illegal TCP packets. Furthermore, its RFC standardization process is no longer active.

There are further network virtualization technologies other than the L2-in-L3 tunneling. We have proposed a non-tunneling edge-overlay model [6] for high-performance virtual networks. This model uses both dynamic Virtual/Physical L2 address translation with OpenFlow [2] protocol and host-based VLAN methods. A performance overhead of this model is negligible, however, this approach cannot be applied to L3 datacenter networks, and therefore, a scalable L2 technology like Ethernet Fabric is required.

DCPortalsNg [14] also modifies MAC address fields of Ethernet frames for network virtualization purpose. DCPortalsNg further overwrites IP address fields in order to identify source/destination VMs on L2-based physical networks while ensuring address space isolation among virtual networks. Considering the coexistence of IPv4 and IPv6 protocols, a special address mapping mechanism is needed (e.g. OpenFlow does not support IPv4–IPv6 address translation).

IP Rewrites [15] had been provided by the first version of Microsoft® Windows® Server 2012. IP Rewrites modify both L2 and L3 address fields for network virtualization too, but there is some difference from DCPortalsNg in that IP Rewrites target L3 datacenter networks. A Provider Address (PA) which is available in physical networks has to be assigned to each VM to support IP-based switching in datacenter networks, which faces complexities of network management because of too many L3 addresses.

OpenVirteX (OVX) [16] is a platform of creating and managing virtual Software-Defined Networks (vSDNs) on physical OpenFlow-based networks. OVX provides various mechanisms of address space and network topology virtualizations to each tenant by internally mapping physical and virtual network elements, such as switches, links and addresses. OVX internally uses the IP address rewriting technique to support L3 address virtualization, therefore, OVX also faces the aforementioned address translation problem.

3. Effect of a L4 Protocol Type for L2-in-L3 Tunneling

In this section, we explain how physical servers (hosts or hypervisors) perform segmentation and reassembly processes for L2-in-L3 tunneling. Basically, offload features perform a packet segmentation process at the sender side and a packet reassembly process at the receiver side. There are two implementation types, *hardware-based* and *software-based*. TCP Segmentation Offload (TSO) and Large Receive Offload (LRO) [17] are hardware-based features, and Generic Segmentation Offload (GSO) [18] and GRO are software-based ones. Modern physical NICs have the hardware-based offload features but supported protocols are generally limited (e.g. TCP over IPv4). Instead, GSO and GRO are provided by the Linux kernel and programmers can implement additional protocol support.

Another important aspect of offload features is that their effects depend on L4 protocol types, *segment-oriented* or *message-oriented*. Segment-oriented protocols (e.g. TCP) have a notion of byte-level *sequence*, and can support a concatenation of consecutive sequences as a large L4 segment before executing the L3/L4 protocol processing at the receiver side. On the other hand, the message-oriented (e.g. UDP) means that consecutive packets in the same flow are independent of one another. Therefore, this type of protocol is not appropriate for the receive offload feature.

Figure 1 shows an example of a sequence of packet encapsulation and segmentation, and Fig. 2 depicts a reverse sequence at a receiver host. Suppose that a VM transmits a large Ethernet frame and the underlying host system encapsulates and divides it into multiple MTU-sized packets. In this example, the pseudo-TCP (pTCP) for STT and UDP for VXLAN are supposed as segment-oriented and message-oriented L4 protocols.

The UDP protocol, which is used by VXLAN and Geneve tunneling, is a representative message-oriented one. When transmitting a large UDP-encapsulated packet whose inner L4 protocol is segment-oriented, the Linux kernel divides the packet into multiple packets using the GSO feature, and each of them has inner L2-L4 headers as shown in the figure. At the receiver side, each UDP packet is decapsulated respectively and the same number of Ethernet frames are forwarded to the destination VM. That is, the greater the size of transmitting Ethernet frames, the greater number of UDP packets have to be received by the receiver host, and therefore, the packet handling process can be performance bottleneck of VM-to-VM communications.

On the other hand, STT tunneling solves this problem by using the pseudo-TCP header. As the original TCP header includes a *sequence number* filed which denotes the byte-level sequence in the flow, the pseudo-TCP header also has a similar field[†] and a *receive offloader* can combine con-

[†]The sequence number of pTCP actually denotes an offset to

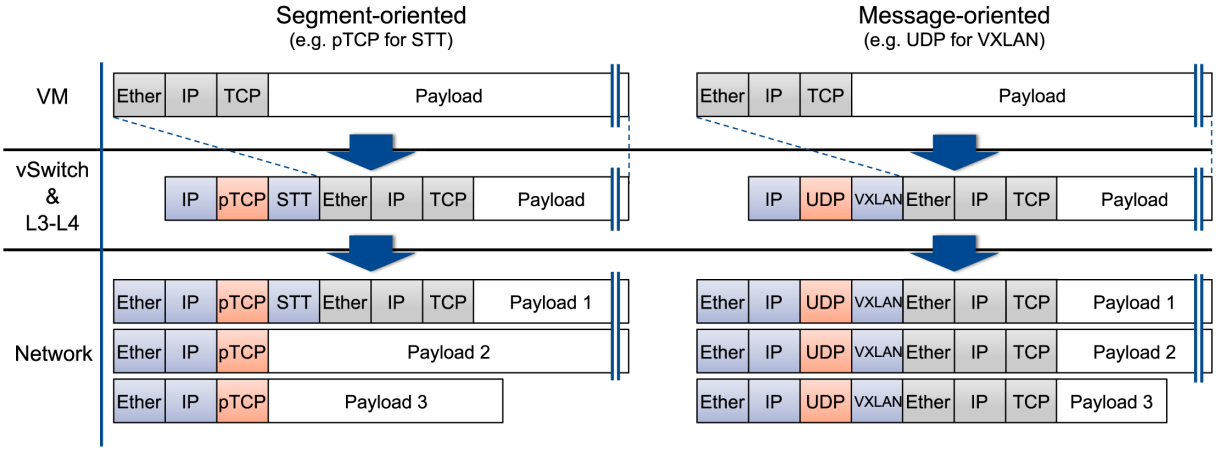

Fig. 1 A sequence of packet encapsulation and segmentation.

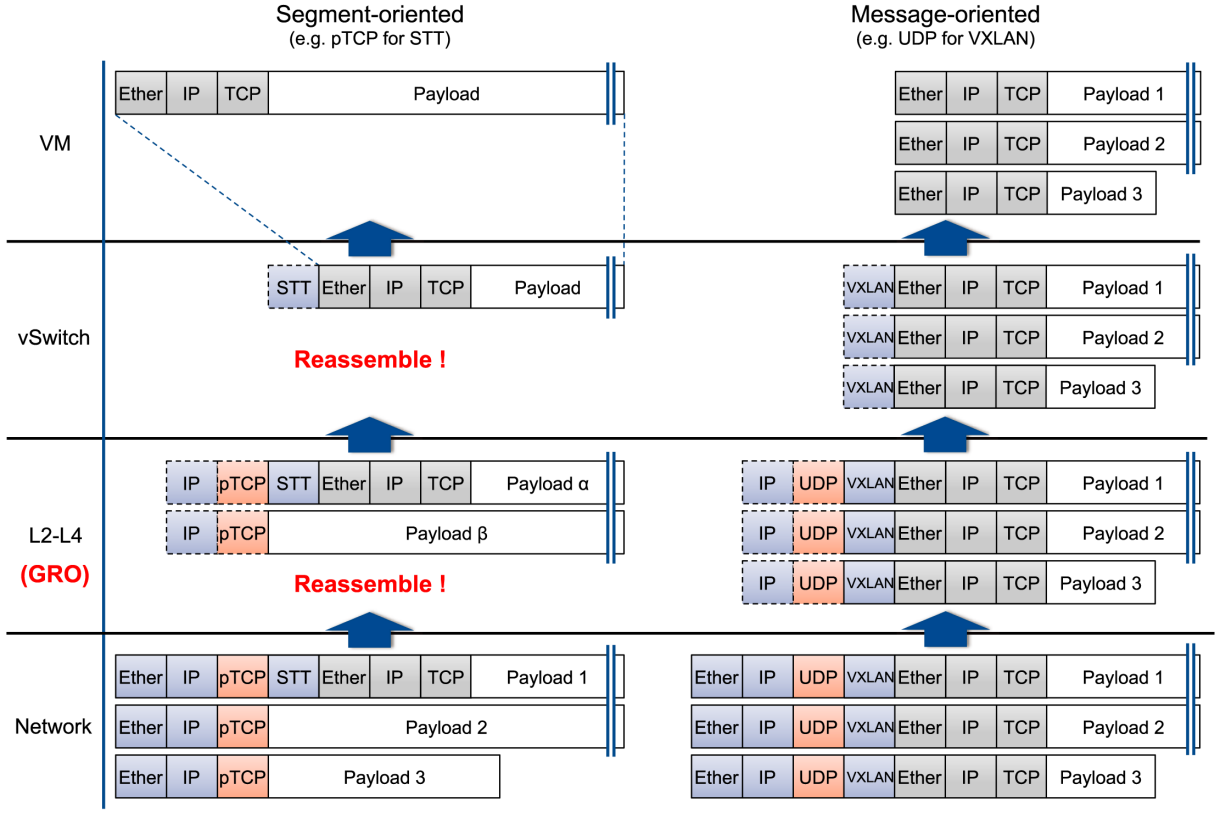

Fig. 2 A sequence of packet decapsulation and reassembly.

secutive L4 segments into a large segment. In the example scenario, the number of divided packets for transmission is the same with the UDP-based encapsulation, but only the first packet contains inner headers. At the receiver side, received packets are merged into larger packets by the GRO feature (first pre-reassembly)[††], and then the virtual switch further combines the packets to restore the original L4 segment for decapsulation (second pre-reassembly). Finally, the original-sized Ethernet frames are passed to the destination VM. Such characteristics of the segment-oriented protocol can drastically reduce the number of both packets and software interruptions, and our previous study [9] has revealed that this 2-level software pre-reassembly is a key aspect of the high-performance ability of STT tunneling.

4. Proposed L4 Protocol (SCLP)

STT achieves high-performance tunneling by taking advan-

the original (before segmentation) large payload.

[††]Current GRO implementation can reassemble up to 16 packets.

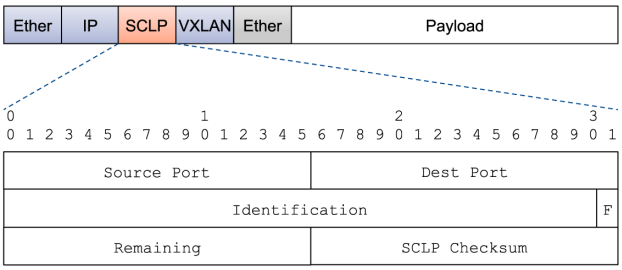

Fig. 3 A header format of the SCLP protocol.

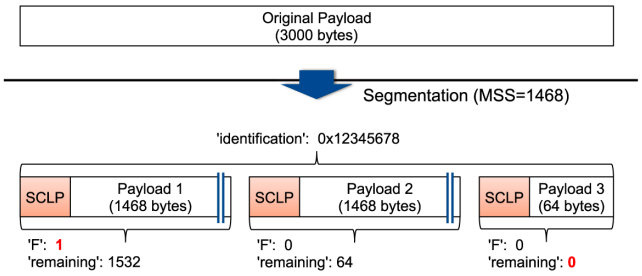

Fig. 4 An example of SCLP segmentation.

tage of the pre-reassembly with GRO as described in the previous section, but use of the preudo-TCP header introduces additional issues in that STT packets have drop-prone characteristics on physical networks with stateful firewalls, load balancers, and Deep Packet Inspection appliances. In this paper, we propose yet another segment-oriented L4 protocol, Segment-oriented Connection-less Protocol (SCLP), for replacing an L4 protocol of existing tunneling protocols like VXLAN.

SCLP itself is a connection-less protocol as indicated by the name and this property is suitable for L2-in-L3 tunneling. The important aspect of SCLP is that it is a segment-oriented protocol unlike UDP. That is, SCLP also supports the notion of byte-level sequence like the pseudo-TCP of STT. SCLP has a simple protocol format and several header fields, such as port numbers, are equivalent to UDP. The difference is that SCLP further provides three fields to make pre-reassembly effective. Figure 3 depicts a header format of the SCLP protocol. SCLP has fixed header length (12 bytes) and each field is explained as follows.

· **Source Port (16 bits)**
  SCLP source port number is determined by a hashing mechanism for ECMP (Equal Cost Multi Path)-based load balancing. The hash value is calculated from inner protocol headers such that every packet of the same flow has a same port number.
· **Dest Port (16 bits)**
  Destination port number indicates the upper layer (tunneling) protocol.
· **Identification (31 bits)**
  This value identifies an original payload data. If multiple SCLP packets have a same value, all of their payloads have been segmented from the same giant payload data at the sender side. This value is used to reassemble the received SCLP segments.
· **(F)irst segment flag (1 bit)**
  This one bit flag is used to declare that the SCLP segment is the first one among all SCLP segments divided from the same payload data.
· **Remaining (16 bits)**
  This 16 bits value represents the remaining payload size used to restore the original payload data. The remaining value decreases to zero with receiving the subsequent SCLP segments. That is, this mechanism can be thought as a reverse (byte-level) sequencing, and therefore, the receive offload feature can be applied to consecutive SCLP segments.
· **SCLP Checksum (16 bits)**
  This field stores a 1's complement sum starting from head of the SCLP header to end of the packet. Unlike the UDP header, the checksum calculation cannot be omitted because SCLP segments except first one do not include inner headers (See Fig. 2).

Here, we explain segmentation and reassembly processes of SCLP with an example scenario. There are two VMs on different physical servers and one VM sends a large Ethernet frame to another using SCLP-based tunneling in this scenario. Note that the size of the frame with a tunnel header is just 3000 bytes and Maximum Segment Size (MSS) in the environment is 1468 bytes (1500 – 20 – 12).

Figure 4 shows a sender side segmentation. There is the original payload data (3000 bytes) for transmission. An SCLP-enabled *segmentation offloader* divides the payload into three segments and each of the three has an SCLP header with the same identification value (0×12345678). First two segments include MSS-sized payload and '*F*' flag of the first segment is set by the definition. The '*remainder*' field of the first one is 1532 bytes (3000 – 1468) and the value of the second one is 64 bytes. The '*remainder*' field of the last segment is 0, which indicates there is no subsequent segment.

Next two figures explain a reassembly process at the receiver side. Figure 5 shows a sequence of the reassembly process when the three SCLP segments are received in order. The SCLP-enabled kernel internally manages reassembly queues with three variables, *id*, *size*, and *offset*. The *id* and the *size* variables store the identification and the total size of the original payload respectively, and the *offset* variable denotes an offset to the next payload. The value of the *id* variable is easily determined by the '*identification*' field within the header. The other variables can be set by the following equations:

$$size = S_{first} + R_{first} \tag{1}$$

$$offset = size - R_{current} \tag{2}$$

where S denotes the size of the segment and R represents the '*remaining*' field value of the segment. Eventually, the reassembly process completes when the values of the *offset* and the *size* are equal.

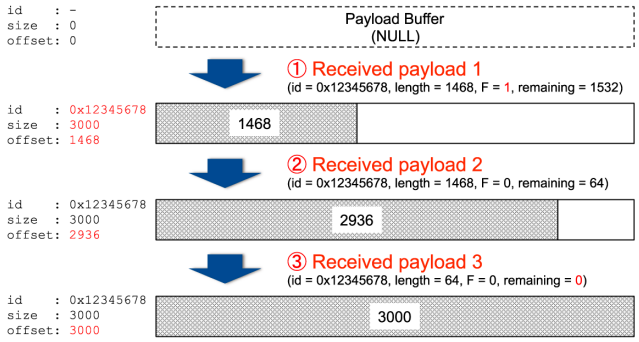

Fig. 5 An example of SCLP reassembly (normal order).

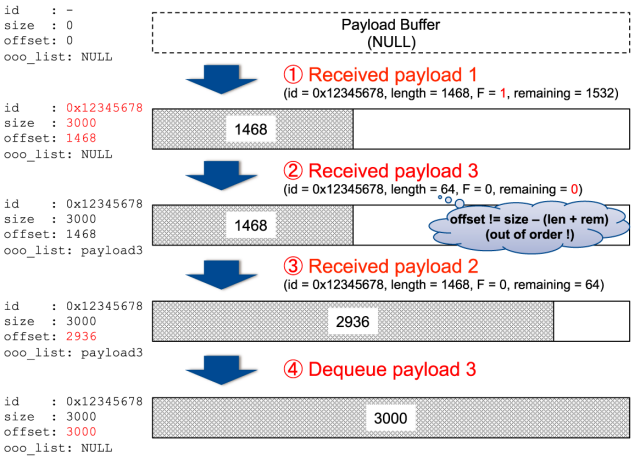

Fig. 6 An example of SCLP reassembly (out of order).

Figure 6 shows a case when the SCLP segments are received in the wrong order. The SCLP-enabled kernel can detect the out of order sequence by the following calculation:

$$offset \neq size - (S_{current} + R_{current}) \quad (3)$$

In such case, the current SCLP segment is enqueued to an out-of-order queue (*ooo_list*) and the reassembly process is suspended.

In terms of interoperability, SCLP requires a new IP protocol number because SCLP is a new L4 protocol over IP. Once the IP protocol number is assigned, network administrators can add a simple flow rule to their middleboxes (e.g. if the IP protocol number of the packet is 234 → ACCEPT). In the mid-term, middlebox vendors will extend their products to interpret the SCLP header fields for more detailed actions. Unlike the STT protocol, SCLP does not pretend to existing L4 protocols having state machine, and therefore, aforementioned middlebox problems can be avoided.

5. Implementation

We have implemented the SCLP protocol and an SCLP-based tunneling mechanism in Linux platform (CentOS 6.6). Generally, high-functional virtual switches play a role of tunnel end-points, but we have used a CVSW (Co-Virtual SWitch) [19] component as the tunnel end-point to ease the development of tunneling protocols. CVSW is implemented as a virtual NIC driver and performs high-functional packet processing, such as the OpenFlow Match/Action and the tunnel encapsulation/decapsulation. Currently, CVSW supports VXLAN, NVGRE, STT, and Geneve tunneling protocols, and we implemented the VXLAN over SCLP tunneling feature into the CVSW component. In addition, we implemented a Linux kernel module which performs both GSO and GRO processing for SCLP segments.

In our implementation, CVSW exists in the VM as a virtual NIC driver, and executes encapsulation and decapsulation processes[†]. A virtio [20] module and Open vSwitch (OVS) [21] bridge the VM and the underlying physical server. The offload module is inserted into the protocol stack of the physical server in order to perform GSO and GRO processing for SCLP segments. The Linux kernel offers a mechanism such that developers can implement both features for custom L4 protocols without modifying existing kernel code, and therefore, we utilized this mechanism for SCLP. Sample code of GSO and GRO implementations are presented as follows:

Program 1: GSO implementation of the SCLP protocol

```
1  struct sk_buff *sclp_gso_segment(
2                            struct sk_buff *skb,
3                            int features)
4  {
5    struct sk_buff *segs;
6    struct sclphdr *sclph;
7    ...
8
9    sclph = sclp_hdr(skb); /* SCLP header */
10   __skb_pull(skb, sizeof(struct sclphdr));
11
12   mss = skb_shinfo(skb)->gso_size;
13   rem = skb->len - mss; /* Remaining size */
14   /* Clear 'F' bit */
15   sclph->id &= htonl(SCLP_ID_MASK);
16   /* Divide the original payload */
17   segs = skb_segment(skb, features);
18
19   skb = segs;  /* First segment */
20   /* Set 'F' bit */
21   sclp_set_first_segment(sclp_hdr(skb));
22
23   while (skb) { /* For each segment */
24     sclph = sclp_hdr(skb);
25     sclph->rem = htons(rem);
26
27     /* Checksum */
28
29     if (rem >= mss) {
30       rem -= mss;
31     } else {
```

[†]CVSW sets up outer headers including Ethernet, IP, and SCLP headers too at the encapsulation process. Likewise, CVSW analyzes entire outer headers at the decapsulation process

```
32        rem = 0; /* This is last */
33      }
34      skb = skb->next;
35    }
36    ...
37  }
```

Program 2: GRO implementation of the SCLP protocol

```
1  struct sk_buff **sclp_gro_receive_impl(
2                          struct sk_buff **head,
3                          struct sk_buff *skb)
4  {
5    struct sk_buff **ppskb;
6    struct sk_buff *pskb;
7    struct sclphdr *sclph;
8    struct sclphdr *sclph2;
9    ...
10
11   skb_gro_pull(skb, sizeof(struct sclphdr));
12   /* Preserve remaining size */
13   rem = sclph->rem;
14
15   /* Look up a segment of the same flow */
16   for (; (pskb=*head); head = &pskb->next) {
17     if (! NAPI_GRO_CB(pskb)->same_flow) {
18       continue;
19     }
20     sclph2 = sclp_hdr(pskb);
21     if (*(u32*)&sclph->source !=
22         *(u32*)&sclph2->source) {
23       NAPI_GRO_CB(pskb)->same_flow = 0;
24       continue;
25     }
26     goto found; /* Same flow ! */
27   }
28   ...
29 found:
30   flush = NAPI_GRO_CB(pskb)->flush;
31   /* Has same id? */
32   flush |= (sclph->id ^ sclph2->id) &
33             htonl(SCLP_ID_MASK);
34
35   mss = skb_shinfo(pskb)->gso_size;
36   flush |= (len - 1) >= mss;
37   /* Is valid offset ? */
38   flush |= (ntohs(sclph->rem) +
39             skb_gro_len(skb)) ^
40             ntohs(sclph2->rem);
41
42   /* Validation check */
43
44   /* skb is concatenated to pskb */
45   pskb = *head;
46   sclph2 = sclp_hdr(pskb);
47   /* Reset remaining size */
48   sclph2->rem = rem;
49   ...
50 }
```

6. Evaluation

In this section, we describe performance evaluation results of the implemented SCLP-based tunneling (VXLAN over SCLP) using 40GbE environment. For comparison, the performance of original VXLAN (CVSW-based and OVS-based), NVGRE, Geneve, STT tunnelings were evaluated, and besides, a no-encapsulation (No-Encap) model that VMs are directly connected to the physical network without tunneling was also evaluated to indicate baseline throughput in the evaluation environment.

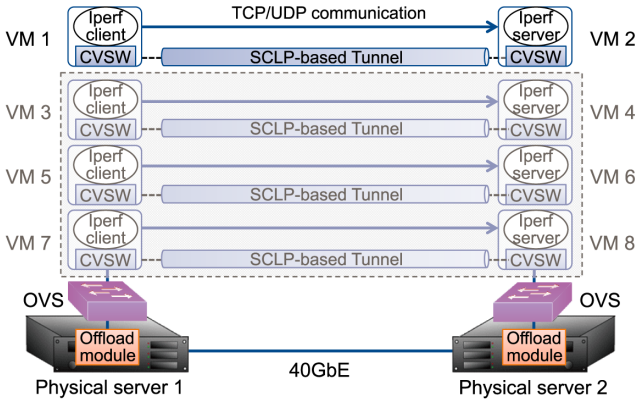

Fig. 7 An evaluation environment.

Table 1 Machine specifications.

Virtual	VM 1 (Sender)	VM 2 (Receiver)
OS	CentOS 6.6 (2.6.32)	CentOS 6.6 (2.6.32)
CPU	1 core	1 core
Memory	2 GBytes	2 GBytes
Virtual NIC	CVSW (virtio-net)	CVSW (virtio-net)
MTU	adjusted	adjusted
Offloading features	TSO, UFO, GSO, GRO, CSUM	TSO, UFO, GSO, GRO, CSUM

Physical	Physical server 1	Physical server 2
OS	CentOS 6.6 (2.6.32)	CentOS 6.6 (2.6.32)
VMM	KVM	KVM
Virtual switch	Open vSwitch 2.3.1	Open vSwitch 2.3.1
CPU	Core i7 (3.60 GHz)	Core i7 (3.40 GHz)
Memory	64 GBytes	32 GBytes
MTU	1500 bytes	1500 bytes
Offloading features	TSO, GSO, GRO, CSUM	TSO, GSO, GRO, CSUM
Network	40GBASE-SR4	40GBASE-SR4

Figure 7 and Table 1 show the evaluation environment and machine specifications respectively. In the experiment, an Iperf [22] client which runs in a VM continuously writes fixed-sized datagram to the socket layer and TCP/UDP packets are sent to the counterpart Iperf server for 60 seconds. The encapsulation/decapsulation processing was performed by CVSW for every tunneling model except OVS-based VXLAN and no-encapsulation. A flow table of CVSW was manually set in advance. Finally, MTU size of each VM was adjusted properly for every tunneling model to prevent IP fragmentation at the encapsulation process.

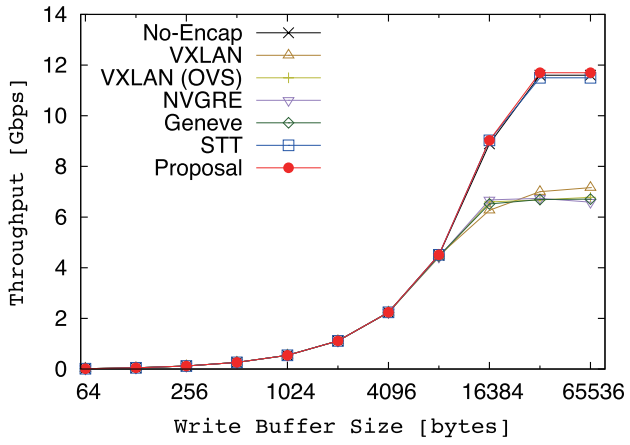

Fig. 8 The evaluation results with TCP communication.

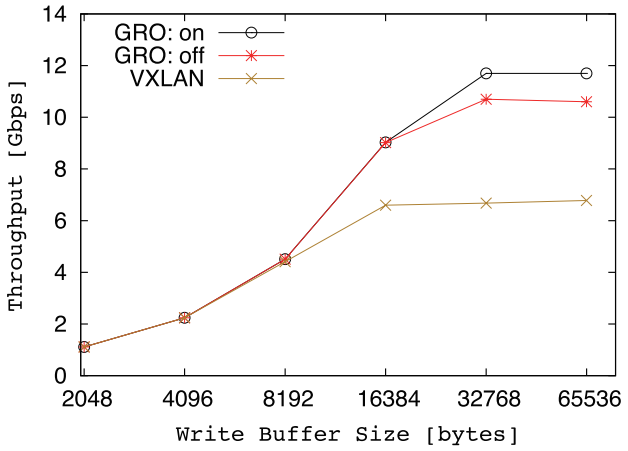

Fig. 9 The performance effect of pre-reassembly for SCLP.

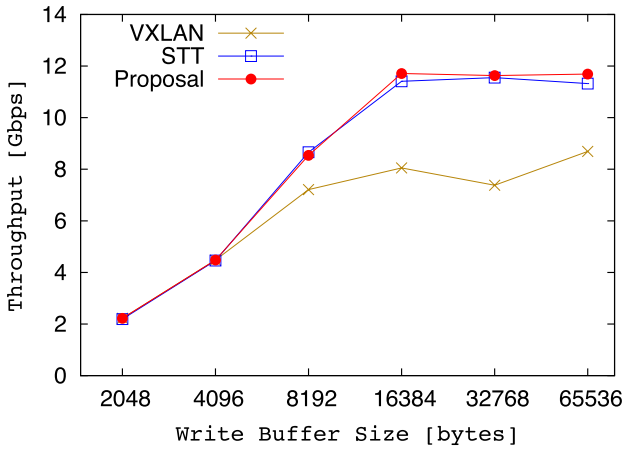

Fig. 10 The total throughput of 2VMs-to-2VMs communications.

6.1 TCP Communication

First, we evaluated the performance of TCP communication between end-to-end VMs with various write buffer sizes of the Iperf client (64–65495 bytes)[†]. Figure 8 shows the actual throughput of every model. The performance of these models were stable throughout the experiment, and the throughput of every model was almost the same when write buffer size was between 64–8192 bytes. However, No-Encap, Proposal and STT models outperformed the others for larger write buffer size and their throughputs finally went up to about 12 Gbps. Considering these models use segment-oriented L4 protocols (TCP, SCLP, and pseudo-TCP), the cause of the performance gap arose from the offload feature. In other words, reducing the number of packets to be software-interrupted can improve the end-to-end throughput drastically.

6.2 Pre-Reassembly Effect for SCLP-Based Tunneling

Next, we evaluated the performance effect of the 2-level pre-reassembly. The effect of first-stage pre-reassembly can be evaluated by disabling the GRO feature, and effect of the second one can be estimated by comparing with the message-oriented tunneling like VXLAN over UDP. Figure 9 shows the throughput of the three cases and the result proves that the pre-reassembly can achieve considerable performance improvement. Note that GRO effect in our previous evaluation [11] on CentOS 6.5 was larger than that in this evaluation, even though the same implementation was used. (The performance gap was about 4 Gbps in the previous experiment).

6.3 Multiple VM-to-VM Communications

We next evaluated aggregated throughput of multiple VMs-to-VMs communications. In this experiment, Receive Side Scaling (RSS) [23] feature of the physical NICs was enabled to distribute received packet processing load. Figiure 10 shows total throughput of 2 flows (VM1–VM2, VM3–VM4) and Fig. 11 shows that of 4 flows (VM1–VM2, VM3–VM4, VM5–VM6, VM7–VM8). The results show that performance of STT and Proposal was almost the same, and VXLAN's performance was obviously below at larger write buffer size. Since original VXLAN requires the greater number of packet processing, total throughput was peaked at lower level. For STT and Proposal, packet processing at receiver side is almost the same manner. In terms of sender side, only STT can take advantage of TSO feature of physical NIC, but as we reported in our previous paper [9], effect of enabling TSO feature on end-to-end throughput was unclear with recent machine power. Therefore, aggregated throughput of both protocols was equivalent.

6.4 UDP Communication

Finally, we measured the performance and packet loss rate of UDP communication. The Iperf client continuously sent UDP packets at 10 Gbps rate (maximum) in this experiment. The results are given in Fig. 12 and Fig. 13. The through-

[†]TCP packet size can be larger/smaller than this size because of TCP's protocol nature.

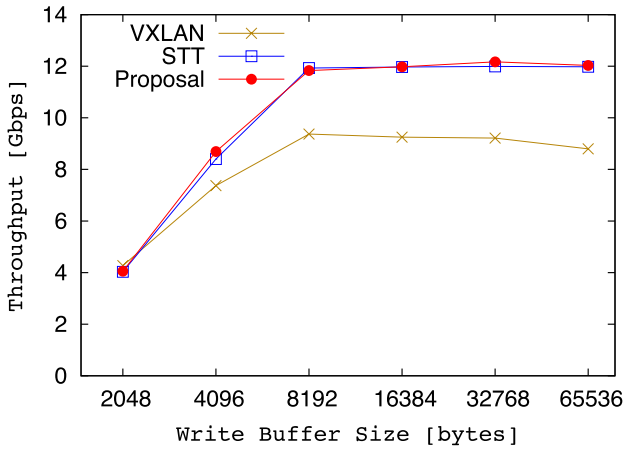

Fig. 11 The total throughput of 4VMs-to-4VMs communications.

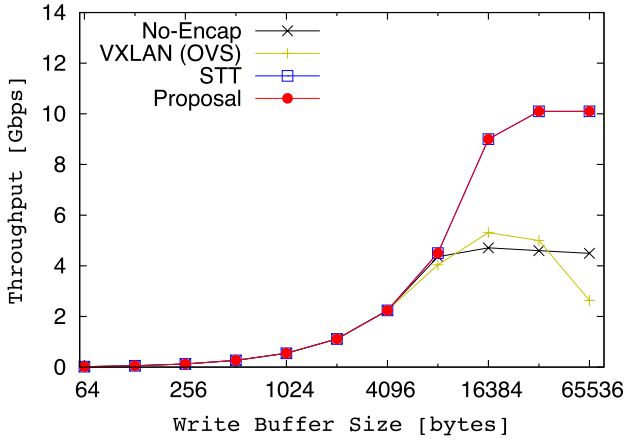

Fig. 12 The evaluation results with UDP communication.

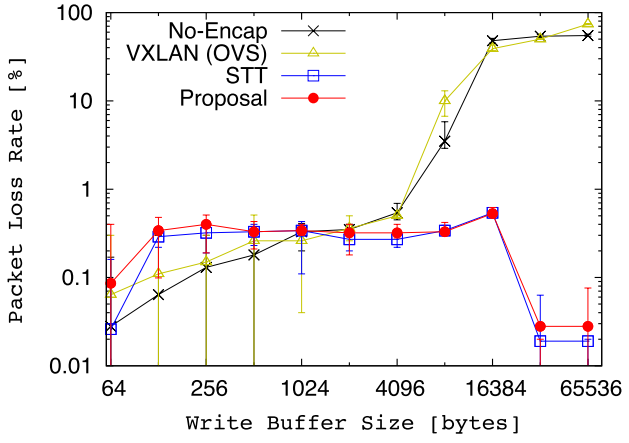

Fig. 13 Packet loss rate of tunneling protocols with UDP.

put of both Proposal and STT models was stable at any packet size and higher than that of No-Encap and VXLAN models with large write buffer sizes. The reason of this performance difference is that received packets of former models are treated as segment-oriented even though the inner L4 protocol is UDP (message-oriented). Therefore, the pre-reassembly effect was applied to these packets. On the other hand, the throughput of No-Encap and VXLAN models peaked under 6 Gbps because the receiver side was not able to handle lots of packets and many of them were eventually dropped as shown in Fig. 13.

6.5 Discussion

The evaluation results have shown that performance of the SCLP-based tunneling is equivalent to STT in various situations. In terms of throughput, both STT and proposed tunneling protocols outperform other protocols when applications use larger write buffer. That is, these two protocols are effective for applications which transmit large data set, and such applications include HTTP/2 [24]-enabled Web servers†, software tunnel gateways like VMware's NSX Edge Gateway [25], virtualized Hadoop clusters [26], and scientific applications for academic clouds.

There is another approach for high-performance overlay-based network virtualization. Header rewriting methods [6], [14], [15], [27] can achieve 'No-Encap'-equivalent throughput, and therefore, there is no clear performance difference between their methods and the proposed method. However, the header rewriting method supports restricted protocols of physical/virtual networks (e.g. L2-based physical networks, TCP/IPv4 virtual network flows).

7. Conclusion

Overlay-based network virtualization has been adopted by various networks including commercial datacenter networks and enterprise networks. Under the concepts of Software-Defined Networking (SDN) and Network Function Virtualization (NFV), further network nodes and appliances will be migrated to virtual networks. Hence, improving the performance of virtual networks is essential for raising service quality.

VXLAN and NVGRE are widely used tunneling protocols but their network performance is lower than that of physical networks. STT is known as the faster tunneling protocol, however, STT faces some practical issue in that various middleboxes discard STT packets because of the modified TCP header semantics.

In this paper, we proposed yet another segment-oriented L4 protocol (Segment-oriented Connection-less Protocol, SCLP) for high-performance software tunneling by exploiting 2-level software pre-reassembly before decapsulation. SCLP can improve the performance of existing tunneling protocols by replacing the original L4 protocol. We implemented SCLP-based tunneling and the offload module in the Linux platform. The evaluation results showed that the throughput of the proposed method (VXLAN over SCLP) exceeded 10 Gbps and was almost

†HTTP/2 uses single TCP connection to transfer entire web content.

70% higher than other mainstream tunneling protocols including original VXLAN.

We are planning to implement VXLAN over SCLP tunneling into Open vSwitch† and develop a hardware-based receive offload feature (e.g. LRO) of the SCLP protocol into programmable network processors for further performance improvement.

Abbreviations

CVSW	Co-Virtual SWitch
ECMP	Equal Cost Multi Path
GRO	Generic Receive Offload
GSO	Generic Segmentation Offload
LRO	Large Receive Offload
MSS	Maximum Segment Size
MTU	Maximum Transfer Unit
NFV	Network Function Virtualization
NVGRE	Network Virtualization using Generic Routing Encapsulation
NVO3	Network Virtualization Overlays over Layer 3
OVS	Open vSwitch
RSS	Receive Side Scaling
SCLP	Segment-oriented Connection-less Protocol
STT	Stateless Transport Tunneling
TEP	Tunnel End-Point
TSO	TCP Segmentation Offload
VXLAN	Virtual eXtensible Local Area Network

References

[1] Network Function Virtualization (NFV), http://www.etsi.org/technologies-clusters/technologies/nfv
[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol.38, no.2, pp.69–74, April 2008.
[3] M. Lasserre, F. Balus, T. Morin, N. Bitar, and Y. Rekhter, "Framework for data center (DC) network virtualization," RFC 7365, 2014.
[4] M. Mahalingam, 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.
[5] 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, 2014.
[6] R. Kawashima and H. Matsuo, "Non-tunneling overlay approach for virtual tenant networks in cloud datacenter," IEICE Trans. Commun., vol.E97-B, no.11, pp.2259–2268, 2014.
[7] B. Davie, Ed. and J. Gross, "A stateless transport tunneling protocol for network virtualization (STT)," Internet draft (expired), 2014.
[8] Offloading the Segmentation of Large TCP Packets, http://msdn.microsoft.com/en-us/library/windows/hardware/ff568840(v=vs.85).aspx
[9] R. Kawashima and H. Matsuo, "Implementation and performance analysis of STT tunneling using vNIC offloading framework (CVSW)," Proc. 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, pp.929–934, Dec. 2014.
[10] JLS2009: Generic receive offload, http://lwn.net/Articles/358910/
[11] R. Kawashima, S. Muramatsu, H. Nakayama, T. Hayashi, and H. Matsuo, "SCLP: Segment-oriented connection-less protocol for high-performance software tunneling in datacenter networks," Proc. 2015 1st IEEE Conference on Network Softwarization (NetSoft), pp.1–8, 2015.
[12] J. Gross, T. Sridhar, P. Garg, C. Wright, I. Ganga, P. Agarwal, K. Duda, D. Dutt, and J. Hudson, "Geneve: Generic network virtualization encapsulation," Internet draft, 2014.
[13] T. Koponen et al., "Network virtualization in multi-tenant datacenters," http://download3.vmware.com/software/vmw-tools/technical_reports/network_virt_in_multi_tenant_dc.pdf, VMware, Technical Report TR-2013-001E, 2013.
[14] H.M.B. Moraes, R.V. Nunes, and D. Guedes, "DCPortalsNg: Efficient isolation of tenant networks in virtualized datacenters," Proc. Thirteenth International Conference on Networks (ICN 2014), pp.230–235, France, Feb. 2014.
[15] J. Savill, Microsoft Virtualization Secrets, Wiley, July, 2012.
[16] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make your virtual SDNs programmable," Proc. Third Workshop on Hot Topics in Software Defined Networking — HotSDN'14, pp.25–30, 2014.
[17] L. Grossman, "Large receive offload implementation in neterion 10 GbE Ethernet driver," Proc. Linux Symposium, vol.1, pp.195–200, Ottawa, CA, July 2005.
[18] Generic Segmentation Offload (GSO), http://www.linuxfoundation.org/collaborate/workgroups/networking/gso
[19] cvsw_net, https://github.com/sdnnit/cvsw_net
[20] R. Russell, "virtio: Towards a De-Facto standard for virtual I/O devices," SIGOPS Oper. Syst. Rev., vol.42, no.5, pp.95–103, 2008.
[21] Open vSwitch, http://openvswitch.org/
[22] Iperf, http://iperf.sourceforge.net/
[23] Microsoft Corporation, "Scalable networking: Eliminating the receive processing bottleneck — Introducing RSS," Technical report, http://download.microsoft.com/download/5/D/6/5D6EAF2B-7DDF-476B-93DC-7CF0072878E6/NDIS_RSS.doc, 2004.
[24] M. Belshe, R. Peon, and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, 2015.
[25] VMware, Inc., "VMware NSX for vSphere (NSX-V) network virtualization design guide," https://www.vmware.com/files/pdf/products/nsx/vmw-nsx-network-virtualization-design-guide.pdf
[26] K. Ye, X. Jiang, Y. He, X. Li, H. Yan, and P. Huang, "vHadoop: A scalable Hadoop virtual cluster platform for MapReduce-based parallel machine learning with performance consideration," Proc. 2012 IEEE International Conference on Cluster Computing Workshops, pp.152–160, 2012.
[27] S. Guenender, K. Barabash, Y. Ben-Itzhak, A. Levin, E. Raichstein, and L. Schour, "NoEncap: Overlay Network Virtualization with no Encapsulation Overheads," Proc. 1st ACM SIGCOMM Symposium on Software Defined Networking Research — SOSR'15, pp.1–7, 2015.

†OVS-based SCLP implementation is now available at GitHub (https://github.com/sdnnit/sclp)

Ryota Kawashima was born in 1983. He received his M.S. degree from Iwate Prefectural University in 2007 and also received Ph.D. degree from The Graduate University for Advanced Studies (SOKENDAI) in 2010. He had worked as a software engineer at ACCESS CO., LTD. and Stratosphere, Inc. He has become an assistant professor at Nagoya Institute of Technology in 2013. His research interest is Software-Defined Networking. He is a member of IEICE, IPSJ, and IEEE.

Hiroshi Matsuo was born in 1960. He received his M.S. in 1985 and also received Ph.D. degree in 1989 from Nagoya Institute of Technology. He became an assistant professor in 1989, lecturer in 1993, associate professor in 1995, and professor in 2003 at Nagoya Institute of Technology. His research interest is distributed cooperative system. He is a member of IEICE, IPSJ, and IEEE.