Protocol-Independent FIB Architecture for Network Overlays

ABSTRACT

We introduce a new forwarding information base architecture into the stacked layering model for network overlays. In recent data center networks, network overlay built upon tunneling protocols becomes an essential technology for virtualized environments. However, the tunneling stacks network layers twice in the host OS, so that processing to transmit packets increases and throughput will degrade. First, this paper shows the measurement result of the degradation on a Linux kernel, in which throughputs in 5 tunneling protocols degrade by over 30%. Then, we describe the proposed architecture that enables the shortcut for the second protocol processing for network overlays. In the evaluation with a dummy interface and a modified Intel 10-Gbps NIC driver, transmitting throughput is improved in 5 tunneling protocols and the throughput of the Linux kernel is approximately doubled in particular protocols.

Categories and Subject Descriptors

D.4.4 [**Communications management**]: Network communication

Keywords

Network Stack; Network Overlay; Tunneling Protocol

1. INTRODUCTION

A significant benefit from using network overlays is the separation of complex and ossified physical networks and virtual networks. Logical topologies isolated from physical networks can be changed agilely to respond to various demands. This characteristic is essential for today's virtualized environments such as clouds and network function virtualization (NFV). However, overlaying networks requires additional protocol processing to transmit packets. This can lead to overheads, especially in host operating systems.

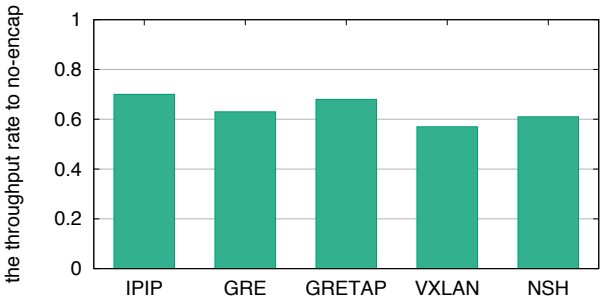

Figure 1: The rate of transmitting throughput via tunneling protocols compared with the normal transmission on Linux kernel 4.2.0.

A typical end point of network overlays is software network stack of host OS. In clouds, hypervisors connect to overlays directly and virtual machines on the hypervisors are accommodated in the overlaid inner networks. In NFV, software-based middleboxes as service functions are connected by overlays [1]. Moreover, load balancers sometimes utilize overlays for topology isolation [2, 3]. Tunneling protocols that build network overlays are implemented as pseudo network interface drivers at the host OS. A pseudo interface becomes an entry point to an overlay network. Packets sent to a pseudo tunnel interface **via a network stack** are encapsulated by the interface and transmitted to underlay physical networks **via a network stack again**.

This conventional OS network stack design of tunneling protocols involves overhead and degrades throughput. To demonstrate the degradation at the OS, we measured transmitting throughput with 5 tunneling protocols: IPIP, GRE, GRETAP (Ethernet over GRE), VXLAN and NSH[1] on Linux kernel version 4.2.0 with Intel Core i7-3770K 3.50-GHz CPU and Intel X520 10-Gbps NIC. The test traffic was a single UDP flow generated in kernel space. Figure 1 shows the result of

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

[1]https://github.com/upa/nshkmod was used for NSH.

this experiment. The x-axis refers to the rate of transmitting throughput via each tunneling protocol to the normal transmission (no encapsulation). As Figure 1 shows, using tunneling protocols in the host OS causes over 30% to 43% throughput degradation.

In this study, 1) we show a fine-grained bottleneck analysis of tunneling protocols on a Linux network stack. The resulting analysis shows the required time for each part of the network stack through tunneling protocols. Based on the analysis, 2) we propose a new forwarding information base (FIB) architecture for tunneling protocols. This architecture provides a shortcut for the second network stack processing. Furthermore, the FIB is protocol independent, so that the proposed architecture can be adapted to existing and future tunneling protocols. We implemented the proposed FIB using a software dummy interface and an Intel 10-Gbps NIC driver. The evaluation results show that the transmitting throughput is improved in 5 tunneling protocols and the kernel throughput is approximately doubled in particular protocols.

2. CASE STUDY: LINUX TUNNEL IMPLEMENTATION

Although new tunneling protocols are being proposed continuously for various purposes, all tunneling protocols can be classified into three models by logical topology type: point-to-point, multipoint-to-multipoint and path-setup. Point-to-point is a classic model that interconnects two hosts via a virtual wire. Typical protocols are IPIP and GRE. The multipoint-to-multipoint model constructs full-mesh topology between multiple hosts. A typical multipoint-to-multipoint protocol is VXLAN. Finally, path-setup, which is used for NFV, constructs virtual circuits on overlays. Network Service Header (NSH) is a typical tunneling protocol of the path-setup type. We consider that the bottleneck of transmission (TX) path via tunneling protocols may change, based on the topology models. Hence, we selected 5 tunneling protocols from three topology models as measurement targets: IPIP, GRE, GRETAP, VXLAN and NSH.

We measured the time required for each part of the network stack processing to occur through 5 tunneling protocols on a current Linux kernel. Figure 2 shows the TX path of a Linux kernel and the experiment overview. The TX path through a tunnel is classified into three parts. **Inner-TX** processes transmitted packets after IP routing for layer-3 (e.g., fragmentation), layer-2 (e.g., ARP) and delivers them to a tunnel interface via `dev_queue_xmit()` in the same manner as physical interfaces. **Pseudo Tunnel Interface** processes packets according to a tunneling protocol of this interface such as destination lookup on an overlay and headroom allocation for outer headers (**Tunnel Processing**) and encapsulates the packets in the tunnel header and outer IP header (**Encapsulation**). After that, **IP Routing** runs for the outer destination IP address. Finally, the encapsulated packets are delivered to the network stack again via `ip_local_out_sk()`. **Outer-TX** processes the encapsulated packets for layer-3 and layer-2 and delivers them to an outgoing interface in the same manner as **Inner-TX**.

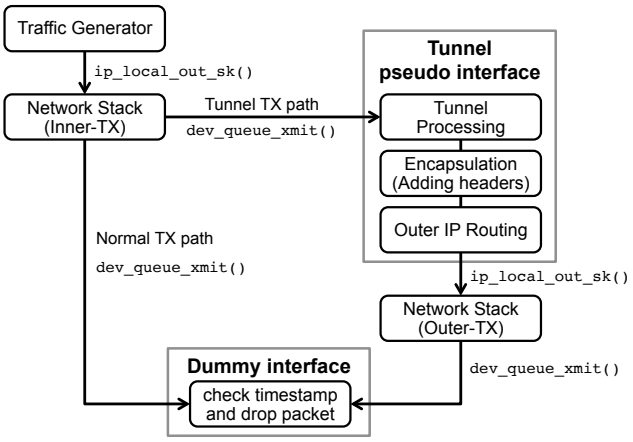

Figure 2: Overview of measurement of the TX path of Linux kernel 4.2.0.

To measure the time required for these parts, we used a modified Linux kernel. We added timestamp fields to the packet buffer (`struct sk_buff`), and inserted RDTSC instructions to the start and end of each part. In this experiment, a packet generated by the traffic generator went through each part and transit times were stored in the timestamp fields of the packet buffer. The dummy interface shown in Figure 2 saved the timestamp fields and dropped the packet immediately when it received a packet. By using this method, we measured the time required for each part on the TX path with 5 tunneling protocols. For this experiment, we used a modified Linux kernel 4.2.0 with an Intel Core i7-3770K 3.50-GHz CPU and test packets were 64-byte UDP packets generated in kernel space.

Figure 3 shows the result of this experiment. The y-axis refers to the time required as the number of CPU clock cycles. The value of each part is the median of the results of 300 runs for each protocol. The x-axis refers to tunneling protocols; NoEncap means the normal TX (no encapsulation), so that there is only Inner-TX on NoEncap. From the result, it can be seen that additional processing time due to tunneling cannot be ignored. In vxlan and NSH, the required CPU time to transmit a packet is approximately doubled compared with the normal TX.

In VXLAN and NSH, the required CPU time for IP routing for the outer IP header (the red stack in Figure 3) is larger than for the other cases. This is ascribable to differences of the topology models. IPIP, GRE and GRETAP are point-to-point models, so that the destination of the outer IP header is always the same. Therefore, the routing cache is used instead of outer IP routing. By contrast, in multipoint-to-multipoint

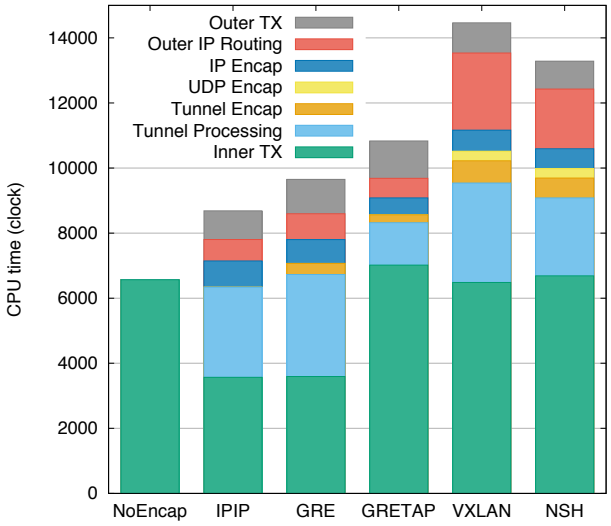

Figure 3: Required CPU time for each part on the Linux TX path via 5 tunneling protocols.

(VXLAN) or path-setup (NSH) models, the destination IP address of the outer IP header is determined for each transmitting packet by their lookup mechanisms. Thus, IP routing cannot be omitted, so that the outer IP routing occupies over 16% of the TX path.

The reason why the Inner-TX of IPIP and GRE is shorter than other protocols is that packet copy due to headroom allocation for outer headers occurs in the Tunnel Processing instead of the Inner-TX. Layer-2 overlay protocols allocate necessary headroom for the inner Ethernet header, tunnel header, outer IP and Ethernet headers together in the Inner-TX. On the other hand, layer-3 overlay protocols allocate headroom in the Tunnel Processing because the Inner-TX does not need to allocate headroom for the inner Ethernet header.

In addition to the CPU time measurement, we measured TX throughput of the Linux kernel. In this experiment, the traffic generator kept generating test traffic and the dummy interface kept dropping the packets. Then, we counted the number of transmitted packets in 60 seconds. Figure 4 shows the result of the experiment. Using tunneling protocols causes over 40% throughput degradation on the Linux kernel. Moreover, the result indicates that the current kernel network stack cannot make full use of the link speed. With 1500-byte packets, TX throughput of IPIP was 26 Gbps and VXLAN was 18.9 Gbps. These results are not sufficient for today's and future interface link speeds of 25 Gbps, 40 Gbps, 50 Gbps and 100 Gbps. As described, using tunneling protocols on a host OS involves additional overhead and highly degrades TX throughput.

3. APPROACH

In this study, we aim to shrink the gap of throughput with/without the tunneling protocols described in Section 2. Before describing the approach, *protocol independence* has to be considered. New tunneling protocols are being proposed on a daily basis. Thus, a method specialized in a particular protocol will soon be out of date. Moreover, such dedicated methods are difficult to deploy in the real world.

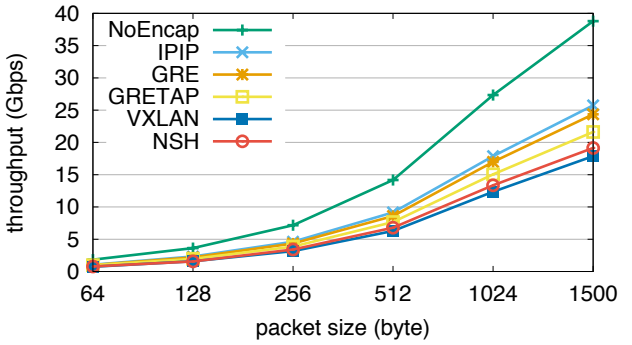

Figure 4: Measuring TX throughput of Linux kernel 4.2.0 with a dummy interface.

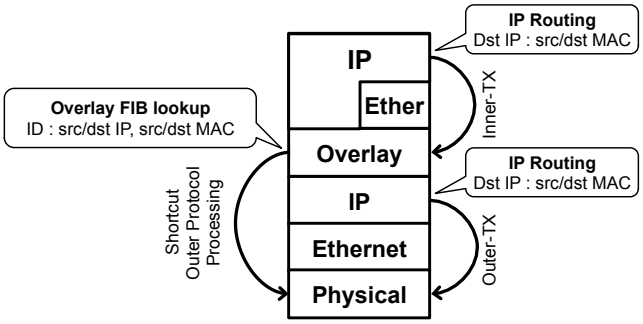

Figure 5: Overlay FIB lookup omits second protocol processing for outer IP headers.

Accordingly, we propose a protocol-independent forwarding information base (FIB) architecture for network overlays. This architecture omits the second protocol processing after encapsulation as shown in Figure 5. The original layer-3 FIB is an IP forwarding table composed of entries of a destination prefix as a key and gateway information such as a destination MAC address. In contrast to the layer-3 FIB, the proposed FIB, called overlay FIB, is composed of entries of some sort of identifier as a key, a destination and parameters for encapsulation. In the normal TX path, layer-3 and layer-2 protocol processing run twice for inner and outer headers of a packet. By using the overlay FIB, protocol processing for outer headers is omitted.

3.1 Overlay Forwarding Information Base

This section describes the overlay FIB architecture and its protocol independence. Even if protocols or their topology models are different, the task of tunneling to transmit packets is the same: determining the destination and encapsulating the packets. The overlay

Table 1: Identifiers embedded in packets can be specified by offset and length.

Protocol	Offset	Length	Identifier
IPIP, GRE	none	none	none (point-to-point)
VXLAN	16-byte	48-bit	Inner destination MAC address
NVGRE	8-byte	48-bit	Inner destination MAC address
MPLS over GRE	4-byte	20-bit	MPLS label
NSH over VXLAN-GPE	20-byte	32-bit	Service Path ID and Service Index

FIB corresponds to these two tasks.

Identifiers for determining the destination address of the outer IP header are different for each tunneling protocol. For instance, point-to-point protocols do not have identifiers because the tunnel has only one destination. Typical multipoint-to-multipoint protocols use the inner destination MAC address as an identifier to determine a destination. NSH, a path-setup model, determines a destination in accordance with the Service Path ID and Service Index embedded in the NSH header [4]. To handle these different lookup mechanisms as a single logic process for protocol independence, we define **offset** and **length** for the identifiers used in, for example, the BPF [5] approach.

Although identifiers for lookups are different for each tunneling protocol, they can be considered as the same type of processing: checking a particular byte-string embedded in a packet and determining a destination. Identifiers are always embedded in packets; therefore, identifiers can be specified by **offset** and **length**. The parameter **offset** indicates the beginning of the identifier and the **length** indicates the bit length of the identifier in the packets. Table 1 shows **offset** and **length** values for major tunneling protocols. In the case of VXLAN, **offset** is 16 bytes for UDP and VXLAN headers and **length** is 48 bits for destination MAC address. In the case of NSH over VXLAN-GPE, **offset** is 20 bytes for UDP, VXLAN and NSH base headers and **length** is 32 bits for Service Path ID and Service Index. In this manner, protocol-specific lookup mechanisms can be handled as a single operation on the overlay FIB.

An overlay FIB entry consists of a byte-string as an identifier and a destination IP address. In addition to the destination, necessary parameters for encapsulation are also stored: source IP address, IP header parameters such as ToS and TTL, destination MAC address (gateway router's MAC address), source MAC address, and an outgoing physical interface. When transmitting a packet via a tunnel interface, the overlay FIB finds an entry corresponding to the identifier embedded in the packet. Thus, the packet is encapsulated in outer IP and Ethernet headers with parameters stored in the found entry from the FIB, and finally transmitted to a physical interface. In this TX path, there is neither IP routing nor protocol processing.

The overlay FIB is updated by entry operations and lower layers. The layer-3 FIB is updated by layer-3 routing table operations (route add or delete) and lower layers such as ARP table change (layer-2) and link down or up (layer-1). The overlay FIB update is similar to that in the layer-3 FIB except that the overlay FIB is placed on layer-3. The overlay FIB is updated by entry add or delete and in lower layers such as IP routing table change (layer-3), ARP table change (layer-2) and link down or up (layer-1). Altogether, update notifications are yielded from lower layers to the overlay FIB layer. Moreover, when an overlay FIB entry is added, update notifications are yielded from the overlay FIB layer to lower layers; then, the gateway's IP and MAC addresses are resolved and stored into the FIB entry.

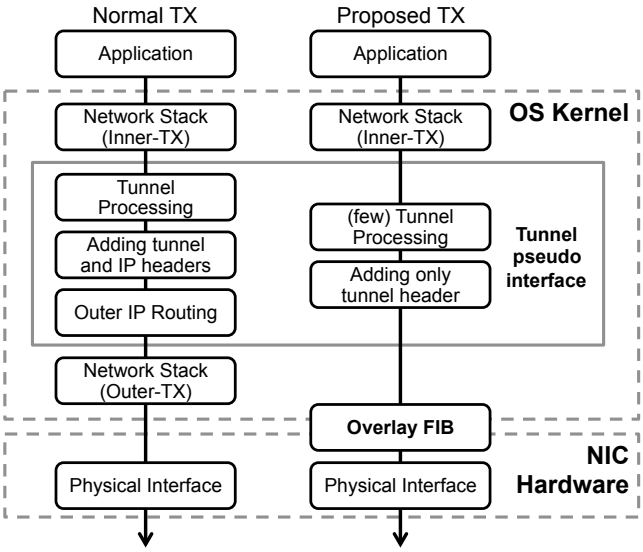

Figure 6: The normal and the proposed TX pathes through network overlays in the host OS.

3.2 Offloading Second Protocol Processing to the Overlay FIB

The overlay FIB is placed between tunnel pseudo interfaces and outgoing physical interfaces in the TX path. Figure 6 shows the normal TX path and the proposed path with the overlay FIB on a Linux kernel. In the normal TX path, the tunnel pseudo interface determines an outer destination IP address of a packet, encapsulates the packet in tunnel and IP headers, runs IP routing for the outer destination IP address and finally delivers the encapsulated packet to the network stack again.

In the proposed TX path, determination of the destination and encapsulation for the outer IP and Ethernet headers are offloaded from the tunnel pseudo interface to the overlay FIB. Consequently, IP routing for the outer IP header is omitted. Tunnel pseudo interfaces only encapsulate packets in the tunnel header and deliver the packets to the overlay FIB. When the overlay FIB receives packets, the FIB finds entries for the packets based on the lookup mechanism described in Section 3.1. Then the packets are encapsulated in outer IP and Ethernet headers and delivered to the physical outgoing interface in accordance with the FIB entries. As a result, the overlay FIB architecture reduces the CPU time needed to transmit a packet by omitting IP routing and the second protocol processing for the outer IP header. In addition to outer IP and Ethernet header encapsulation, UDP encapsulation can also be offloaded to the overlay FIB.

The overlay FIB can be implemented in both software and hardware. As hardware, it is implemented in NICs. NICs have the FIB table and a configuration API that has functions to set **offset** and **length**, add and delete an entry composed of an identifier and corresponding parameters. This API is similar to switchdev API [6] introduced to Linux kernel version 3.19. The switchdev provides kernel with hardware independent configuration API to notify layer-2 and layer-3 FIB entries to NIC. The overlay FIB entries can be notified from kernel to NIC in a similar way. Moreover, proposed architecture can be implemented in hardware by using various technique like FPGA [7], hardware programming [8, 9] and dedicated ASIC certainly.

4. EVALUATION

The proposed overlay FIB architecture provides a shortcut for second protocol processing on the TX path via tunneling protocols. By using the architecture, TX throughput with tunneling protocols is improved. In this section, we investigate throughput improvement due to the proposed architecture from two aspects: 1) measuring the required CPU time to transmit a packet and 2) actual throughput. For this evaluation, we modified IPIP, GRE, GRETAP, VXLAN and NSH drivers to support the overlay FIB. All modifications for drivers are less than a few dozen lines of code.

Source codes for all experiments described in this paper are available at https://github.com/xxx/xxx.

4.1 Measuring CPU Time on the TX Path

First, we measured the CPU time required for each part on the TX path with the overlay FIB in the same manner as for the experiment discussed in Section 2. In this experiment, we assumed that the overlay FIB was completely implemented in the NIC hardware; thus, lookup identifier, second protocol processing, outer UDP, IP and Ethernet header encapsulation are omitted from the TX path of the kernel. Tunnel pseudo interfaces delivered packets to the dummy interface immediately after encapsulation. We experimented with modified Linux kernel version 4.2.0 and an Intel Core i7-3770K 3.50-GHz CPU machine.

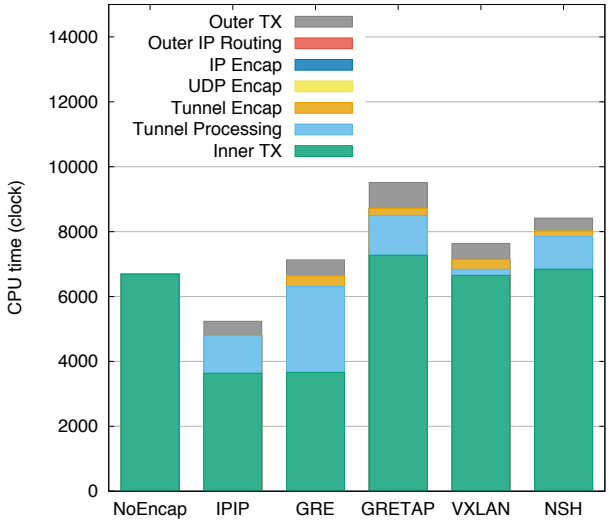

Figure 7: Outer UDP, IP encapsulation and IP routing are omitted from the TX path compared with the normal TX path shown in Figure 3.

Figure 7 shows the CPU time required to send a packet when the FIB is implemented in NICs. In Figure 7, Outer-TX means the time required from the end of tunnel interface to the start of dummy interface processing. IP routing, IP and UDP encapsulation are completely removed from TX paths compared with Figure 3. As a result, the CPU time required to send a packet in the kernel is reduced in all tunneling protocols.

In addition, IPIP with the FIB is shorter than NoEncap. That is because of the packet copy for headroom allocation. In the normal TX path of IPIP, headroom for outer IP and Ethernet headers is allocated in the Tunnel Processing as described in Section 2. By contrast, allocating headroom is not needed if outer encapsulation is offloaded to NIC hardware by the proposed method. As a result, CPU time for IPIP becomes shorter than for no encapsulation, which requires headroom allocation for the Ethernet header in the Inner-TX. On the other hand, time for the Tunnel Processing of GRE, which uses layer-3 tunneling, does not decrease, because headroom allocation is still needed for the GRE header.

In VXLAN, the Tunnel Processing is almost removed. When using the FIB, protocol processing of VXLAN only involves adding a VXLAN header to a packet. Lookup VXLAN FDB, determining destination IP address, encapsulating VXLAN and UDP headers are removed from the Tunnel Processing. As a result, the CPU time required to transmit a packet via the VXLAN tunnel with the FIB has decreased by 47% compared with the normal VXLAN.

Table 2: Measurement result of 5 protocols with/without the overlay FIB.

		IPIP		GRE		GRETAP		VXLAN		NSH	
	packet size (Byte)	64	1500	64	1500	64	1500	64	1500	64	1500
dummy	Normal TX (Gbps)	1.05	25.74	1.00	24.36	0.98	21.62	0.74	17.87	0.79	19.14
	TX with overlay FIB (Gbps)	2.20	51.94	1.58	31.86	1.28	27.82	1.43	31.10	1.36	29.48
	Throughput rate (%)	210	202	158	131	131	129	193	174	172	154
	packet size (Byte)	64	1024	64	1024	64	1024	64	1024	64	1024
ixgbe	Normal TX (Gbps)	0.72	9.58	0.64	9.55	0.69	9.43	0.58	8.23	0.62	8.71
	TX with overlay FIB (Gbps)	0.84	9.58	0.86	9.55	0.81	9.73	0.85	9.32	0.80	9.13
	Throughput rate (%)	117	100	132	100	117	103	147	113	129	105

4.2 Measuring Throughput

We next evaluated throughput using the dummy interface and the modified ixgbe driver, which is the Intel 10-Gbps NIC driver. With the dummy interface, transmitted packets were dropped immediately the packets were delivered to the dummy interface; this is the same as for the experiment shown in Figure 4. Altogether, the throughput of the dummy interface means the throughput of the Linux kernel itself. In addition, we implemented the overlay FIB as software into an ixgbe driver. With the modified ixgbe driver, packets delivered to an ixgbe interface were encapsulated in outer IP and Ethernet headers in accordance with the overlay FIB in the device driver. Then, the packets were transmitted to a wire via X520 NIC. We measured throughput of the modified ixgbe at an receiver machine connected to the NIC. For this experiment, we used the same machine as that used in the experiment described in Section 4.1.

Table 2 shows transmitting throughput using the dummy interface and the modified ixgbe driver. In all cases, the proposed method improves TX throughput. Throughputs of the dummy interface with IPIP and VXLAN with 64-byte packets are also approximately doubled as, expected from the result of the experiment about CPU time in Section 4.1 On the other hand, the overlay FIB software implementation also improves actual throughput to the wire when using the ixgbe driver in all tunneling protocols. When it was implemented in the NIC hardware, the throughput of the ixgbe was improved at the same rate as for the dummy interface.

5. DISCUSSION

Other identifiers: The proposed method assumes that the identifier is embedded in the packet and is just a byte-string. This assumption is correct for MAC addresses or labels; however, it does not work for identifiers that have some sort of semantics. Locator Identifier Separation Protocol (LISP) [10] utilizes the inner destination IP address as the identifier to determine a destination on a LISP overlay network. Thus, the overlay FIB has to be capable of the longest prefix match in order to support LISP.

Other bottlenecks on the TX path: Many bottlenecks on the TX path are well known. For instance, socket API, system call overhead and per-packet processing have been mentioned for high-speed packet I/O approaches [11, 12, 13, 14]. Meanwhile, network protocol processing never disappears from host OS. This means, no matter how fast the network stack is, protocol processing becomes the bottleneck on the TX path. In the Arrakis operating system [15], which has a very optimized network stack, network protocol processing occupies 44.7% of its TX path. When protocol processing becomes a bottleneck as it does in Arrakis, the current design of overlays where packets go through the network stack twice involves serious overheads. Therefore, the proposed method, which avoids the second protocol processing, can achieve throughput improvement even if other bottlenecks are also relieved.

Hardware offloading techniques: The proposed method can coexist with existing NIC offloading techniques. For example, the generic segmentation offload (GSO) delivers large (over 60-kbyte) packets to NICs. Then, NICs divide large packets into small packets corresponding to MTU size. GSO reduces the number of packets that are handled by the network stack, so that it achieves TX throughput improvement. By applying the overlay FIB before the GSO to NICs, large packets are encapsulated in outer IP and Ethernet headers and then divided into small packets. In addition, UDP tunnel offload [16, 17], checksum offload can also coexist.

6. CONCLUSION

In this paper, we have investigated the bottleneck of tunneling protocols in the Linux kernel network stack. Based on the investigation, we have proposed a new FIB architecture for network overlays. The overlay FIB provides a shortcut for second protocol processing of tunneling protocols. Moreover, this architecture is protocol independent, so that it can be adapted to existing and future tunneling protocols. We demonstrated that our proposed method could improve transmitting throughput with 5 protocols, and the kernel throughput was approximately doubled in IPIP and VXLAN. We plan to implement the architecture in a FPGA card with 10-Gbps NIC. Finally, we aim to evaluate and discuss the architecture in more detail in the future.

7. REFERENCES

[1] J. Halpern and C. Pignataro. Service function chaining (sfc) architecture. RFC 7665, RFC Editor, October 2015.

[2] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 523–535, Santa Clara, CA, March 2016. USENIX Association.

[3] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. *Commun. ACM*, 54(3):95–104, March 2011.

[4] Paul Quinn and Uri Elzur. Network service header. Internet-Draft draft-ietf-sfc-nsh-04, IETF Secretariat, March 2016. http://www.ietf.org/internet-drafts/draft-ietf-sfc-nsh-04.txt.

[5] Steven McCanne and Van Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference*, USENIX'93, pages 2–2, Berkeley, CA, USA, 1993. USENIX Association.

[6] Jiri Priko. Hadware switches - the open-source approach. Technical report, Netdev 0.1, The Technical Conference on Linux Networking, 2015.

[7] Jad Naous, Glen Gibb, Sara Bolouki, and Nick McKeown. Netfpga: Reusable router architecture for experimental research. In *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '08, pages 1–7, New York, NY, USA, 2008. ACM.

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[9] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with flexnic. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 67–81, New York, NY, USA, 2016. ACM.

[10] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The locator/id separation protocol (lisp). RFC 6830, RFC Editor, January 2013. http://www.rfc-editor.org/rfc/rfc6830.txt.

[11] Luigi Rizzo and Matteo Landi. Netmap: Memory mapped access to network devices. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 422–423, New York, NY, USA, 2011. ACM.

[12] Intel. Intel data plane development kit. http://dpdk.org/.

[13] Kenichi Yasukata, Michio Honda, Douglas Santry, and Lars Eggert. Stackmap: Low-latency networking with the os stack and dedicated nics. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, Denver, CO, June 2016. USENIX Association.

[14] The fast data project. https://fd.io/.

[15] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The operating system is the control plane. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 1–16, Broomfield, CO, October 2014. USENIX Association.

[16] Intel. Intel ethernet controller 10 gigabit and 40 gigabit xl710 family. http://www.intel.com/content/www/us/en/embedded/products/networking/ethernet-controller-xl710-family.html.

[17] Mellanox Technologies. Mellanox connectx-3 pro product brief. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-3_Pro_Card_EN.pdf.