Protocol Independent NIC Offloading for Overlay Networks

Ryo Nakamura*, Yohei Kuga†, Yuji Sekiya*, Hiroshi Esaki*
*University of Tokyo, †Keio University

ABSTRACT

We propose a novel technique for offloading multiple tunneling protocols on NIC hardware. IP overlays have been proposed and developed for various purposes in the data centers. In the use cases such as IaaS clouds, a typical end point of the overlays is host OS and its software network stack. However, the per-packet encapsulation causes CPU resource consumption and performance degradation at the end hosts. The proposed technique enables the OS network stack to offload look-up destination address for inner packets and outer IP encapsulation without regard to differences of tunneling protocols. In this paper, we describe the approach and a prototype NIC implementation using a commodity FPGA card.

Categories and Subject Descriptors

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

Keywords

NIC Offloading; Tunneling Protocol; Overlay Networks

1. INTRODUCTION

IP overlays are widely used for various purposes. The most significant use case is Infrastructure as a Service cloud environment. The multi-tenancy and multi-point tunneling in the clouds are key functionalities by tunneling protocols such as VXLAN [1] and NVGRE [2]. In such a cloud, hyper-visors connect the overlay networks directly and forward packets between virtual machines and the overlays. Therefore, the tunneling protocols are usually implemented in the OS network stack.

NIC offloading is an effective technique for improving the tunneling performance. Figure 1 shows the performance of transmitting 64-byte UDP packets with Linux kernel 3.16 and an Intel X520 10-Gigabit Ethernet card. The result indicates that tunneling protocols cause about 20% performance degradation due to the per-packet encapsulation overhead. Encapsulation requires the memory access to add outer headers and the routing table look-up for outer IP header. Therefore, some NICs begin to equip several functions for tunneling protocol offloads [3, 4].

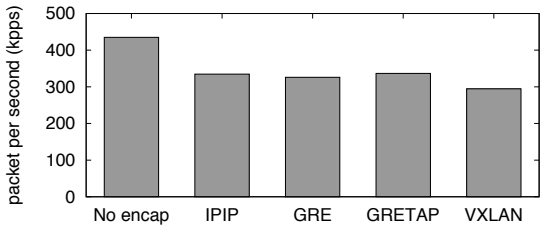

Figure 1: Transmitting performances of tunneling protocols with 64-byte packet.

On the other hand, new tunneling protocols for various purposes have been proposed continuously, for instance service chaining and Network Service Header (NSH) [5]. It is possible to follow the changes in the tunneling protocols by using the software implementation. However, the existing tunnel offloads lack extensibility because existing NIC hardware has no APIs for managing the tunneling protocols and users cannot change the offloaded protocols. Thus, NIC must be replaced new one with new ASIC to obtain high performance on new protocols. In order to follow the rapid growth of new tunneling protocols and achieve high performance at the same time, more flexible offloading NICs are needed.

In this paper, we propose a novel offloading technique for multiple tunneling protocols. All tunneling protocols have same two operations: look-up destination IP address corresponding some identifiers embedded in packets and outer IP encapsulation. The proposed technique offloads these operations on a NIC. In following sections, we describe the approach and show a prototype implementation using a commodity FPGA card.

2. APPROACH

Figure 2 shows the Linux network architecture of typical tunneling protocols and our proposed method. The outer IP encapsulation and the IP routing look-up in the encapsulation process are common among different tunneling protocols. These OS functions can be offloaded on NIC. However, the decision processes for a destination address of the outer IP header are based on each

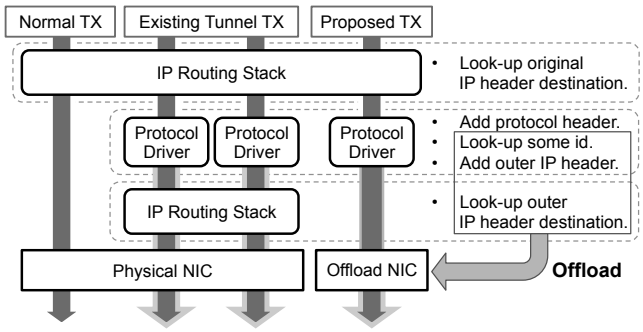

Figure 2: System design of tunneling protocols in the OS network stack.

protocol context and identifier. For instance, point-to-point tunneling (e.g., IPIP) has only a destination and multi-point tunneling (e.g., VXLAN) decides a destination IP address in accordance with a destination MAC address of the inner packet. In order to offload the look-up based on each context, we define **offset** and **length** for the identifiers like BPF [6] approach.

Although the identifier for look-up is different from each tunneling protocol, all destination look-up only check the particular byte string embedded in packets. The identifiers can be specified relatively by **offset** and **length**. The **offset** indicates the beginning of the identifier and the **length** indicates the bit length of the identifier in the packets. In case of VXLAN, **offset** is 16-byte for UDP and VXLAN headers and **length** is 48-bit for destination MAC address. In case of NSH over VXLAN-GPE [7], **offset** is 24-byte for UDP, VXLAN and NSH base headers and **length** is 32-bit for Service Path ID and Service Index. In this manner, the protocol specific look-up can be handled as a single common operation and offloaded on a NIC.

In order to offload these operations, the NIC has a table for the destination look-up and a configuration API. The table is composed of pairs of a configured length identifier and a destination address for outer IP header. The NIC does not have concern with what the identifier is. The identifier is just a byte string and only defined by the **length**. The configuration API has functions to set IP header parameters (protocol number, ToS and TTL), **offset** and **length**, and add or delete the table entries.

3. PROTOTYPING

We implemented a part of proposed functions in a prototype NIC using a NetFPGA-1G card [8] with Linux. Our NIC implementation is still in the first stage; however, we confirm that it is possible to develop our technique on commodity NIC hardware. The current implementation has the only function for the offloading outer IP encapsulation without the table look-up. The prototype consists of the NetFPGA reference NIC design and our encapsulation module. The module adds an outer IP header with specified source and destination IP addresses, TTL, ToS and protocol number.

Table 1: Measuring transmit throughput with outer IP encapsulation offloading (kpps).

packet size	no encap	IPIP (offload) Off	IPIP (offload) On	VXLAN (offload) Off	VXLAN (offload) On
64	112.00	106.88	114.51	97.52	106.53
1024	29.64	27.68	30.00	27.55	27.33

We modified IPIP and VXLAN drivers on Linux. In Linux kernel, the tunneling protocol drivers call the `ip_tunnel_xmit()` function that processes outer IP encapsulation, IP routing and transmitting. This function in the drivers is replaced with the `nf2c_tx()` function that enqueues a packet to the TX buffer of the NetFPGA NIC. Our IPIP driver places IP packets to the NIC TX buffer directly, and our VXLAN driver places an Ethernet frame encapsulated in UDP and VXLAN headers to the buffer through the `nf2c_tx()`.

The transmitting performance will suggest an effect of the offloading per-packet outer IP encapsulation. Thus, we measured and compared 64-byte packet transmitting performances as shown in Table 1. The experiment platform was Linux kernel 3.16 and Intel Core i7-3770K CPU. To receive and count transmitted packets, we used netmap [9]. When IPIP encapsulation is offloaded, the performance reaches the no encapsulation performance. Our IPIP driver carries transmitting packets to the NIC via the `nf2c_tx()` immediately after a few error checks. There is no overhead for the `ip_tunnel_xmit()`. In the case of VXLAN, the offloading also contributes to the performance improvement; however, the throughput is lower than the no encapsulation. The cause of the lack is that the destination look-up before the `nf2c_tx()` is not offloaded in the current implementation. As a result, the per-packet encapsulation is overhead, and the offloading it on NIC is a credible way for the performance.

4. SUMMARY AND FUTURE WORK

The per-packet encapsulation of tunneling protocols is an essential overhead at the end hosts. Our proposed technique enables multiple tunneling protocols in the OS network stack to offload the destination look-up and the outer IP encapsulation on a NIC. We have implemented a prototype NIC using NetFPGA-1G and measured its performance. The result indicates that the offloading per-packet outer IP encapsulation on NIC contributes to the performance improvement. We plan to implement the whole proposed technique in a FPGA card with 10-Gigabit Ethernet. Finally, we aim to evaluate and discuss the offloading technique in more detail.

5. REFERENCES

[1] 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, August 2014.

[2] Pankaj Garg and Yu-Shun Wang. Nvgre: Network virtualization using generic routing encapsulation. Internet-Draft draft-sridharan-virtualization-nvgre-08, April 2015.

[3] 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`.

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

[5] Paul Quinn and Uri Elzur. Network service header. Internet-Draft draft-ietf-sfc-nsh-00, March 2015.

[6] 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.

[7] Paul Quinn, Rajeev Manur, Lawrence Kreeger, Darrel Lewis, Fabio Maino, Michael Smith, Puneet Agarwal, Lucy Yong, Xiaohu Xu, Uri Elzur, Pankaj Garg, and David Melman. Generic protocol extension for vxlan. Internet-Draft draft-ietf-nvo3-vxlan-gpe-00, May 2015.

[8] 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.

[9] 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.