Electronic Preprint for Journal of Information Processing Vol.30

Regular Paper

A NIC-driven Architecture for High-speed IP Packet Forwarding on General-purpose Servers

Yukito Ueno[1,3,a)] Ryo Nakamura[2] Yohei Kuga[2] Hiroshi Esaki[1]

Received: May 31, 2021, Accepted: December 3, 2021

Abstract: We propose a high-speed packet forwarding architecture on general-purpose servers, in which a Network Interface Card (NIC) drives packet forwarding by direct packet transfer to other NICs via a PCIe switch. The demand for high-speed packet forwarding technology on general-purpose servers is increasing with the spread of networking concepts such as Network Function Virtualization (NFV). However, the current architecture, which processes packets by CPU, cannot achieve the similar degree of performance that hardware routers can provide because the processing capacity of the CPU and the bandwidth of the main memory constrain the performance. Our proposed method, called P2PNIC, overcomes this constraint by eliminating the CPU and the main memory from the entire packet forwarding. In the P2PNIC architecture, a NIC determines to which NIC to forward the packets and directly transfers the packets to the NIC over the PCIe. We evaluate the P2PNIC architecture by comparing it with the DPDK applications as examples of the current architecture. The evaluation shows that the P2PNIC architecture achieves 3.44 times higher throughput and up to 79% lower latency than the DPDK applications. This study offers a new approach in software-based network infrastructure for achieving comparable performance with hardware routers in the future.

Keywords: Ethernet NIC, packet forwarding, PCIe, Peer-to-Peer DMA

1. Introduction

In commercial Internet Service Provider (ISP) networks, there is a strong demand for faster IP router implementations, given the continuous growth of Internet traffic. To meet this demand, ISPs usually use hardware routers, which can accommodate several tens of 100 Gbps ports. Hardware routers are composed of multiple boards called linecards, connected by a switching fabric. The linecard consists of integrated Ethernet ports and dedicated ASICs, which have packet forwarding capacity at the rate of tens of Tbps. The packet forwarding performance of hardware routers is sufficient for the demand of today's ISP backbone routers.

On the other hand, software routers have not yet achieved the same level of performance as hardware routers, although researchers and network service providers are actively pursuing ways to accelerate them [2], [8]. The major reasons for the technical difficulties arise from the limitation of the processing capacity of the CPU and the bandwidth of the main memory on general-purpose servers. In general-purpose servers used as software routers, the Network Interface Card (NIC) receives packets from the network and transfers the packets to the main memory so that the CPU processes them, as shown on the left-hand side of **Fig. 1**. For this architecture, which we refer to as CPU-driven architecture in this paper, the bottleneck arises from the unavoidable involvement of the CPU and main memory in the packet forwarding process. For example, the current fastest DDR SDRAM standard has a bandwidth of 51.2 GB/s, which would accommodate eight 100 Gbps ports with four memory channels. Although the bandwidth currently meets the level a single host would require, it is insufficient for the backplane capacity of the ISP backbone routers that accommodate several tens of 100 Gbps ports.

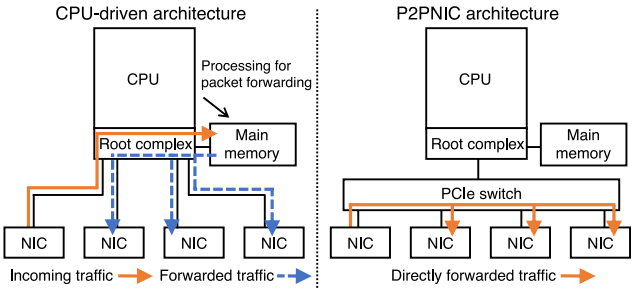


Fig. 1 The packet forwarding path of the current CPU-driven architecture and the proposed P2PNIC architecture when the packets are incoming from a NIC and forwarded to the other three NICs.

As a resolution of this bottleneck, we propose an IP router architecture for general-purpose servers, called P2PNIC, which achieves the packet forwarding without the involvement of CPU and main memory. We also refer to this architecture as NIC-driven architecture because the Ethernet NIC itself drives packet forwarding; a NIC that received a packet (ingress NIC) determines a NIC that transmits the packet to the network (egress NIC) by IP routing. Based on the determination, the ingress NIC transfers the packet to the egress NIC via a PCIe switch, as shown on the right-hand side of Fig. 1. By adopting the P2PNIC architecture, the packet forwarding performance can be improved regard-

[1] Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[2] Information Technology Center, The University of Tokyo, Kashiwa, Chiba 277–0882, Japan
[3] Innovation Center, NTT Communications, Minato, Tokyo 108–8118, Japan
[a)] eden@g.ecc.u-tokyo.ac.jp

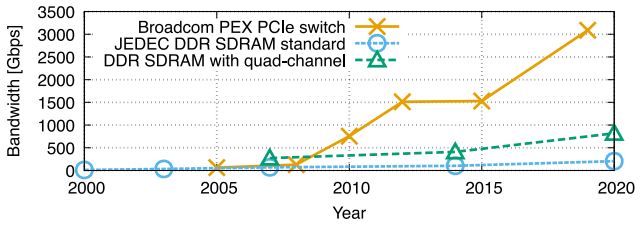


Fig. 2 Comparison of the aggregate bandwidth of the Broadcom PEX series PCIe switches and JEDEC DDR SDRAM standards. Because DDR memory operates in half-duplex, the available bandwidth for packet forwarding is calculated as half of its theoretical bandwidth.

less of the processing capacity of the CPU and the bandwidth of the main memory. Instead of the performance limit of the CPU and main memory, the aggregate bandwidth of the PCIe switches on the server determines the performance of packet forwarding in the P2PNIC architecture.

Figure 2 illustrates the advancements in the aggregate bandwidth of the DDR memory and PCIe switches used in general-purpose servers, revealing the potential for performance improvement by the P2PNIC architecture. The current fastest PCIe switch has more than 3 Tbps aggregate bandwidth, and it can accommodate up to 30 100 Gbps ports, which is higher than that of the current fastest DDR SDRAM standard. Figure 2 also shows that the aggregate bandwidth of PCIe switches has tended to improve more rapidly in recent years compared with the DDR memory. This is due to the emergence of applications such as machine learning and data mining that require higher PCIe bandwidth for direct communication between GPUs [24], [31]. Thus, the possibility of significant improvement for the packet forwarding performance on general-purpose servers exists if the main memory can be eliminated in the process of packet forwarding.

In this work, we have implemented some variants of P2PNIC architecture on Netronome's SmartNICs [34] and conducted an evaluation by comparing them with the DPDK applications as examples of CPU-driven architecture. We extend our previous work [43], which proposed the basic concept of the P2PNIC architecture; whereas the previous method supports only forwarding packets to a single fixed port, this extended version supports each NIC dynamically determining the destination port to which to forward packets from multiple ports by IP routing. Moreover, we have added measurements for the extension with several traffic patterns that load multiple ports simultaneously and confirmed that its throughput scales up to 160 Gbps at maximum. The evaluation shows that the P2PNIC architecture has 3.44 times higher throughput and up to 79% lower latency compared with a DPDK application in the highest load situation. The contributions of this paper are as follows.

- We have proposed an architecture called P2PNIC, in which a NIC directly communicates with other NICs to accelerate packet forwarding inside general-purpose servers.
- We have demonstrated performance advantages of the P2PNIC architecture, including throughput and latency compared with DPDK applications in the evaluation.
- We have extended P2PNIC to support multiple Ethernet ports and succeeded in scaling its performance up to 160 Gbps at maximum.

2. Related Work

Although hardware routers are used in ISP backbone networks, the importance of software routers is increasing because it is a fundamental technology for newer networking concepts such as Network Function Virtualization (NFV). Software routers process packets on general-purpose servers, and we refer to the current architecture as CPU-driven architecture because all packets are transferred to the main memory and processed by the CPU. To date, the evolution of CPU and memory bandwidth and optimizations by previous studies have improved the performance of software routers based on the CPU-driven architecture. Nevertheless, the performance of CPU-driven architecture has not yet caught up with the evolution of network link speeds. This section reviews the advancements in hardware and software routers and describes the limitation for performance improvement in the CPU-driven architecture.

2.1 Hardware Routers

Hardware routers provide high bandwidth capacity on the peer-to-peer communication form with multiple linecards and switching fabric [1], [45]. Nowadays, the bandwidth capacity of hardware routers based on the switching fabric reaches as high as 160 Tbps [9]. The multiple linecards installed in a hardware router process and transfer packets to other linecards through the switching fabric independently. The distributed packet processing and communication form in a single chassis contribute to the total bandwidth in the equipment by allowing a large number of linecards and the corresponding number of high-speed Ethernet ports to be accommodated.

One of the factors that improves the performance of hardware routers is the optimizations of the ASIC chip, an example of which is the miniaturization of the manufacturing process and the adoption of shared buffers. Broadcom's Tomahawk 4 [4] is one of the flagship switching ASIC chips processed by TSMC's 7 nm process, which has a 25.6 Tbps throughput capacity. The chip adopts a shared buffer architecture, in which all ports share a single large pool for packet buffers. The shared buffer architecture improves the tolerance for burst traffic while maintaining the achievable throughput for all ports.

2.2 Software Routers

With the need for rapid deployment of new features that hardware routers cannot provide—but that concepts such as NFV can provide—software routers are actively studied and developed despite the spread of hardware routers. The Click Modular Router [22] builds a software router by composing simple functions in the form of a graph. The Click Modular router has been extended in subsequent studies because the abstraction allows for both scalability and performance of software routers. An example is RouteBricks [10], which achieves 35 Gbps throughput on general-purpose servers by parallelizing router functionality based on the Click Modular Router.

Another approach to improve the performance of software routers is to use coprocessors to accelerate packet processing in software routers. PacketShader [19] and subsequent stud-

ies [16], [42], [44] use GPUs to accelerate packet processing as part of software routers. Adopting GPUs can improve the throughput of the packet processing by exploiting their massive parallelism while concealing the communication latency between CPU and GPU by means of their batching manner. As a result, PacketShader achieves 39 Gbps forwarding performance.

As a culmination of the technologies and studies mentioned above, packet processing frameworks that focus on accelerating the I/O part of networking applications have been proposed. DPDK [12] is one of the most popular packet processing frameworks that provides a variety of packet processing libraries for networking applications. Netmap [41] proposes a packet processing framework by modifying in-kernel NIC device drivers with supports for zero-copy, batching, and multiple queues. These frameworks aim to accelerate various networking applications, whereas previous studies focused on accelerating software routers [22].

A recent approach to reducing the pressure on the processing capacity of the CPU and the bandwidth of the main memory is to offload a part of the packet processing to SmartNICs. A SmartNIC is a NIC that has a high-speed Ethernet port, which is typically faster than 10 Gbps and provides a programmable mechanism to offload the part of the packet processing that the CPU had processed. UNO [23] proposes a framework to offload packet processing functions to SmartNICs transparently. Other proposals have been made to offload packet processing to SmartNICs to speed up networking applications [27], [28], [32], [33]. The common idea behind these technologies is to reduce data transfer between the CPU and NIC by processing packets as close to the network as possible.

2.3 The Problem of Packet Processing by CPU

Although the CPU and main memory are inevitable in CPU-driven architecture, it is becoming clear that the degree of their improvement cannot keep pace with the evolution of network link speeds. In recent years, DDR memory has doubled in bandwidth with each generation. However, as shown in Fig. 2, the bandwidth of the DDR memory is unlikely to reach the level that can accommodate several tens of 100 Gbps ports within a few years. For example, the current fastest DDR SDRAM standard (DDR5-6400) with four memory channels cannot provide sufficient bandwidth for more than eight 100 Gbps ports. Moreover, the processing capacity of the CPU would be the next bottleneck for packet forwarding in this architecture. According to the recent benchmark of DPDK [35], accommodating a single 100 Gbps port requires two CPU cores or more. Because the number of the CPU cores used in the benchmark is 24, the CPU cannot accommodate more than 12 100 Gbps ports. In addition, the increase in the processing capacity of a single CPU core and the number of CPU cores has stagnated [20].

In recent years, an approach to reducing the amount of traffic that needs to be forwarded between the CPU and NICs at devices closer to the network has been gaining attention. For that purpose, techniques to offload packet processing to NICs are being actively studied and developed; these studies offload specific applications (e.g., data analytics, transaction processing, and key-value store [27]) or protocol stacks (e.g., IPsec [23] and TCP [32]) to NICs. With these techniques, a NIC modifies the content of the packets to achieve the offloaded features and, if necessary, transfers the processed contents to the CPU. This form of processing reduces not only the load on the CPU but also the communication amount between the CPU and NICs via PCIe.

However, except for primitive techniques such as IP checksum offload, these techniques cannot be applied to IP routing across NICs. The major reason is that modifying the content of the packets except for the IP header violates the design principle of IP that IP routers provide datagram transport only. For example, Large Receive Offload (LRO) improves the throughput of the TCP stack at the end host by combining segmented TCP packets into a single large packet in the NIC. However, if the technique is applied to the intermediate IP routers, the routers can not restore the original packets from the combined packets when sending them because of the absence of original size information. Consequently, the modification of the packet contents by the IP router would result in unexpected problems in end-to-end communication.

3. P2PNIC Architecture

The rapid performance improvement of PCIe switches and the remaining bottlenecks of software routers led us to the approach of eliminating the CPU and the main memory from the entire packet forwarding. Based on this approach, we propose a novel packet forwarding architecture called P2PNIC. In this architecture, the CPU and memory are not involved in packet forwarding. Instead, a NIC directly transfers packets to other NICs via PCIe, as illustrated on the right-hand side of Fig. 1. Whereas the CPU performs packet forwarding in the CPU-driven architecture, the NICs perform packet forwarding in the P2PNIC architecture, which we refer to as a NIC-driven architecture in this paper.

By completing the communication between NICs under a PCIe switch, the P2PNIC architecture can overcome the limits of performance improvement caused by the processing capacity of the CPU and the bandwidth of the main memory. In addition, the P2PNIC architecture can reduce the latency to forward a packet by halving the number of packet transfers in the host. The CPU-driven architecture requires at least two packet transfers between the NIC and the main memory to forward a packet; one is from the NIC to the main memory for processing the packet by the CPU, and the other is from the main memory to the NIC to send the packet to the network. In contrast, the P2PNIC architecture requires a single packet transfer to forward a packet; the ingress NIC decides to which NIC to send the packet, and then the NIC transfers the packet to the egress NIC over PCIe. This effect would be greater than the bare data transfer latency of PCIe because it also eliminates the processing for managing the packet transfer, such as manipulating ring buffers.

To achieve direct packet transfer between NICs, the P2PNIC architecture exploits the conventional data transfer method between CPU and NICs over PCIe called Direct Memory Access (DMA). Although DMA is mainly used as the data transfer method between NICs and the main memory, the PCIe specification does not constrain the target memory region of the PCIe transaction into the main memory. That is, if a memory region

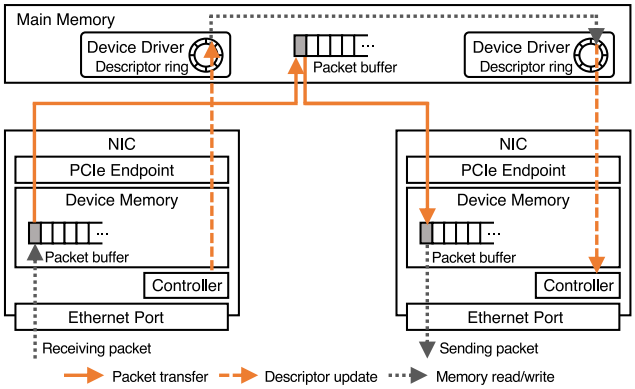


Fig. 3 Components and behavior of the CPU-driven architecture.

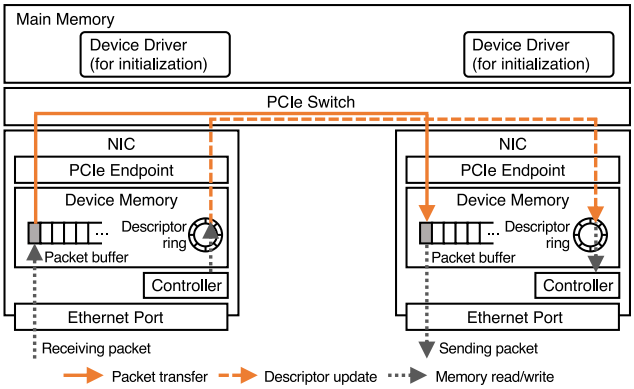


Fig. 4 Components and behavior of the P2PNIC architecture.

of a PCIe device is mapped into the physical address space, other PCIe devices can issue DMA to the memory region of the PCIe device. The DMA between PCIe devices is called Peer-to-Peer DMA (P2P DMA), and the technique can be used with any PCIe device, including commodity Ethernet NICs and SmartNICs in principle, although recent studies focus on its application to GPU and NVMe storage [3], [46]. The P2PNIC architecture is the first scheme that applies the P2P DMA to packet forwarding between Ethernet NICs to improve the performance of an IP router.

In the P2PNIC architecture, the configuration of PCIe on the motherboard defines the performance limits of packet forwarding. In other words, the bottleneck shifts from the bandwidth of the main memory to that of the PCIe bus. For example, the upper limit of the aggregate bandwidth would be 3 Tbps for the current highest grade PCIe switch [6]. Because it is expected that the performance will continue to improve with the generation of PCIe and its speed is faster than that of the DDR memory (Fig. 2), the shift of the packet transfer path from the DDR memory to the PCIe switch would be preferable for improving packet forwarding performance.

3.1 The Difference between CPU-driven and P2PNIC Architectures

A major difference between the CPU-driven and P2PNIC architectures is the placement of ring buffers for transferring packets. Commodity NICs send/receive packets through one or more ring buffers allocated on the main memory, as shown in **Fig. 3**. Each element of the ring buffer contains the metadata to describe a single packet; the element is called a descriptor, and the entire ring buffer is called a descriptor ring. The descriptor ring for CPU–NIC communication is placed on the main memory and updated by both the host CPU and NIC to synchronize the state of packet sending and receiving. Inside the NICs, the controller manages the manipulation of the descriptor rings and the packet transfer to/from the main memory.

In the CPU-driven architecture, the device driver manages NICs for sending and receiving packets. The workflow of the device driver can be classified into the initialization and packet I/O. For the initialization, the device driver configures the NICs by setting various parameters before it starts packet I/O. For example, the device driver sets the starting address of the descriptor ring on the main memory to each NIC, which is used by the NIC to update the descriptor ring. After the initialization, the device driver manages the packet I/O through the descriptor ring. For packet receiving, the device driver informs the NIC of the memory address of the packet buffer through the descriptor ring. When the NIC receives a packet, if unused packet buffers remain, the NIC transfers the content of the packet buffer to the main memory via PCIe. After finishing the transfer of the packet buffer, the NIC informs the device driver of the arrival of the new packet through the descriptor ring. On the other hand, when sending, the device driver informs the NIC of the memory address of the packet buffer waiting to be sent through the descriptor ring. When the NIC detects the presence of the packets waiting to be sent, the NIC retrieves the packets from the main memory to its packet buffers via PCIe and sends the packets from its Ethernet ports. The initialization and packet I/O are fundamental features in the device driver for commodity NICs.

In P2PNIC architecture, NICs pass the packets to the destination NIC directly through the descriptor rings allocated on the device memory inside the ingress and egress NICs, as shown in **Fig. 4**. The descriptor ring is still required in the P2PNIC architecture as well as the CPU-driven architecture to pass the metadata of packets between the ingress and egress NICs. However, in contrast to the CPU-driven architecture, the descriptor ring is placed on the device memory of each NIC. The NIC that received packets directly updates the descriptor ring placed on the egress NIC through a PCIe switch to inform the egress NIC of the presence of the newly arrived packets. In addition, the NIC also transfers the contents of the packet buffer to the egress NIC directly through the PCIe switch, as well as the update of the descriptor ring. In this way, because all the information necessary for packet forwarding is transferred under a PCIe switch, the communication between NIC and main memory does not occur in the packet forwarding.

The P2PNIC architecture also requires the device driver to initialize each NIC before starting packet forwarding, while the device driver is not involved in the packet forwarding process, unlike the CPU-driven architecture. To achieve direct communication between NICs, each NIC has to know the physical addresses of the packet buffers and descriptor rings of other NICs. Therefore, the initialization includes informing each NIC of the physical addresses of other NICs' packet buffers and corresponding descriptor rings, for instance. On the other hand, after the initial-

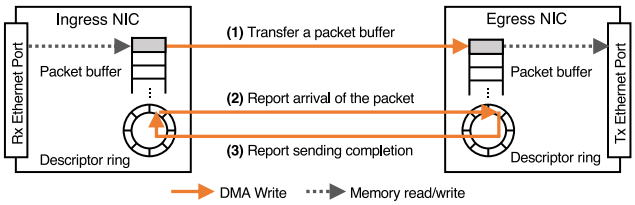


Fig. 5 Packet forwarding procedure of the P2PNIC architecture for DMA Write.

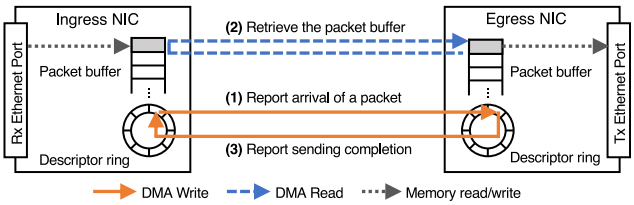


Fig. 6 Packet forwarding procedure of the P2PNIC architecture for DMA Read.

ization, the packet forwarding is driven by each NIC without the involvement of the device driver.

3.2 Methods for Direct Packet Transfer between NICs

In the P2PNIC architecture, the method to transfer the packets between NICs could be either DMA Write or Read. **Figure 5** shows the packet forwarding procedure for the DMA Write version of the P2PNIC architecture. To forward a packet, (1) the NIC that received the packet issues a DMA Write to transfer the packet to the device memory of the egress NIC. After finishing the packet transfer, (2) the NIC notifies the egress NIC of the presence of the new packet by updating the egress NIC's descriptor ring. When the egress NIC detects the presence of the packet, the NIC sends the packet via its Ethernet port. (3) The egress NIC notifies the ingress NIC of the sending completion by updating the ingress NIC's descriptor ring. Both ingress and egress NICs use DMA Write as the way to update the descriptor ring.

One concern about using DMA Write in conjunction with P2P DMA is a possible DMA failure due to the congestion at the intermediate PCIe switches. In DMA between NICs, the required bandwidth is not always ensured across the entire path between the PCIe devices, while it is ensured between CPU and PCIe devices. In the transaction layer of the PCIe, which adopts packet-based communication, if data congestion occurs at a component (e.g., PCIe switch) on the communication path, the component may drop the PCIe packets called Transaction Layer Packets (TLPs). To address this situation, either the detection of DMA failure by dedicated mechanisms such as checksum at the PCIe endpoint or the congestion control at intermediate PCIe components is required.

As a way to maximize the performance in terms of throughput and latency, this extended version of the P2PNIC architecture assumes that the PCIe switch is equipped with the congestion control feature, which prevents the DMA failure due to TLP drop. In fact, Broadcom's PEX series, which is one of the most common PCIe switches, provides this feature [5]. The PCIe switch forwards TLPs and holds them in internal buffers shared by all links until the PCIe switch confirms successful transfer of the TLPs by receiving acknowledgment packets from the destination PCIe device. When the buffers are exhausted in the PCIe switch due to insufficient bandwidth on the destination PCIe device, the PCIe switch will suspend the transmission of TLPs from the source PCIe device by credit-based control. Therefore, if the source PCIe device was a NIC in the P2PNIC architecture, the NIC stops transferring TLPs of received packets to other NICs. In this way, if all NICs are directly connected to this PCIe switch, the TLP drop due to congestion and resulting DMA failure will not occur even for the DMA Write version of the P2PNIC architecture. If the internal packet buffers in the NIC are also exhausted while waiting for the resolution of the buffer exhaustion at the PCIe switch, the ingress NIC drops packets at the Ethernet controller for the receiving port, as is the usual behavior of NICs.

Another method for packet transfer between NICs is based on DMA Read. **Figure 6** shows the packet forwarding procedure for the DMA Read version of the P2PNIC architecture. The major difference from the DMA Write version is that the egress NIC retrieves the content of a packet buffer, whereas the ingress NIC transfers the content in the DMA Write version. Upon receiving a packet, (1) the NIC that received the packet informs the egress NIC of the presence of the newly received packet through the descriptor ring. When the egress NIC detects the presence of the packet, (2) the NIC issues a DMA Read to retrieve the packet to its device memory. After finishing the packet transfer, the egress NIC sends the packet via its Ethernet port. (3) The egress NIC notifies the ingress NIC of the sending completion through the descriptor ring.

3.3 Supporting Multiple Ethernet Ports

In this work, we extend the P2PNIC architecture to support multiple Ethernet ports by using multiple descriptor rings and corresponding buffers in each NIC. To support multiple Ethernet ports, there is a limitation on the scalability of the number of NICs due to the capacity of the NIC's device memory. In the P2PNIC architecture, each NIC has to maintain a dedicated descriptor ring per egress NIC because the ring buffer is a data structure for a single reader and a single writer. In this design, because each NIC needs to hold descriptor rings and buffers for each of the other NICs, the limit of the number of accommodatable NICs could be constrained by the capacity of the NIC's device memory. However, the actual number will vary depending on the condition, such as the link speed of Ethernet ports and the implementation of the firmware for the NIC. In this work, we demonstrate the feasibility of accommodating four 40 Gbps Ethernet NICs while maintaining high throughput and low latency by the P2PNIC architecture.

The P2PNIC architecture supports IP routing as a method to organize multiple Ethernet ports. In the P2PNIC architecture, an ingress NIC has to determine the direction to which to forward a packet because it forwards a packet to the egress NIC directly, whereas the CPU plays that role in the CPU-driven architecture. Therefore, a NIC that received a packet performs Longest Prefix Matching (LPM) to determine to which NIC to forward the packet before starting the procedure to transfer the packet. After the LPM is finished, the NIC starts transferring the packet to the

NIC that the result of LPM indicates.

4. Implementation

To show the feasibility of P2PNIC architecture and its performance advantage compared with the applications of CPU-driven architecture, we implemented two types of P2PNIC architecture as separated SmartNIC firmware that varies according to whether DMA Write or Read is used for the communication between NICs as described in Section 3.2. We refer to the variation that uses DMA Write as P2PNIC-Wr and DMA Read as P2PNIC-Rd. In addition, we implemented an example firmware of CPU-driven architecture based on P2PNIC-Wr and P2PNIC-Rd named P2PNIC-Bn, which transfers packets once to the main memory. The purpose of P2PNIC-Bn is to observe the performance impact of the differences between CPU-driven and P2PNIC architectures while minimizing implementational differences.

To implement the P2PNIC architecture, fine-grained control of the NIC's processing and its DMA engine is a mandatory feature. As a SmartNIC that meets the requirements, we adopt Netronome Agilio CX 2x40GbE NICs, which equip two 40 Gbps Ethernet ports and provide programmability of its network processing unit (NPU) called NFP-4000 by customized C in a firmware. Based on the programmability of the NIC, the P2PNIC-Wr consists of the 1,148 LoC of the firmware and the 12,728 LoC of the DPDK-derived device driver for NFP-4000.

On the other hand, it is currently difficult to implement the P2PNIC architecture on NICs other than Netronome's SmartNICs. To implement the P2PNIC architecture, there are two conditions: the performance of the hardware and its programmability. As for hardware performance, other SmartNICs [7], [37] already meet the requirements with components such as 100 Gbps Ethernet ports, PCIe 4.0, and large device memory. Their large device memory would be able to provide a larger number of the descriptor rings than Netronome's SmartNICs, for instance. However, they do not expect and provide programmability for some usage required to implement the P2PNIC architecture. The usage includes addressing all received packets by a programmable mechanism in the NICs, exposing their device memory to other PCIe devices, and issuing DMA Read and Write autonomously to arbitrary addresses. Therefore, the enhancement of the programmability is the condition to implement the P2PNIC architecture on other NICs.

Because the P2PNIC architecture performs as an IP router, and the CPU is not involved in packet forwarding in P2PNIC architecture, the NIC has to perform LPM to determine the direction to forward the packet. For P2PNIC-Wr, P2PNIC-Rd, and P2PNIC-Bn, we adopt the state-jump table [25] as the LPM method because of its ease of implementation and performance. In addition to the LPM processing, the implementation supports the mandatory features required for actual IP routing, including TTL decrement, checksum calculation, and MAC address rewriting.

As described in Section 3.1, the P2PNIC architecture requires a dedicated device driver to initialize each NIC. In the initialization process for each NIC, the device driver enables Ethernet ports and the PCIe endpoint of each NIC. In addition, the device driver gathers physical addresses of the descriptor rings and packet buffers allocated on the device memory of each NIC to inform each NIC of the physical addresses. We implemented the device driver in the user-space of the Linux OS using the VFIO mechanism [26]. The VFIO mechanism enables device drivers in the user-space to have the same level of functionality in the kernel-space by exporting access permission of PCIe devices to user-space applications.

To reduce the overhead for DMA Write to update the descriptor ring of the egress NIC on a per-packet basis, we adopt the batching manner. With the batching manner, the ingress NIC checks the presence of the packet waiting to be processed after it processes an arrived packet. If the next packet has already arrived, the NIC holds off updating the descriptor ring until there are no more packets to process or the specified number of pending packets is accumulated. The batching manner improves the throughput by reducing the processing overhead to update the descriptor ring while increasing the latency to forward a single packet due to waiting for the processing of the next packet.

5. Evaluation

We evaluate the P2PNIC architecture from the following three aspects by comparing it with the DPDK applications as examples of the CPU-driven architecture.

- How do throughput and latency differ between CPU-driven and P2PNIC architectures when transferring a single flow with the minimum configuration? (Section 5.1 and Section 5.2).
- How much does the throughput of the CPU-driven and P2PNIC architectures scale with multiple Ethernet ports? (Section 5.3).
- What are the characteristics of the CPU-driven and P2PNIC architectures with respect to CPU usage and power consumption? (Section 5.4).

The DPDK applications we used for the evaluation were TestPMD [14] and L3FWD [13]. TestPMD forwards packets without any processing on the packet data for the performance test of device drivers in DPDK. On the other hand, L3FWD forwards packets based on IP routing.

For the evaluations, we used two hosts connected with four 40 Gbps Ethernet links with direct attach cables, as shown in **Fig. 7**. One of the hosts is for generating and receiving test traffic, and the other is for forwarding the traffic with the implementations of the P2PNIC architecture and DPDK applications. We refer to these two hosts as tester and forwarder hosts, respectively. When P2PNIC-Wr and P2PNIC-Rd run on the forwarder host,

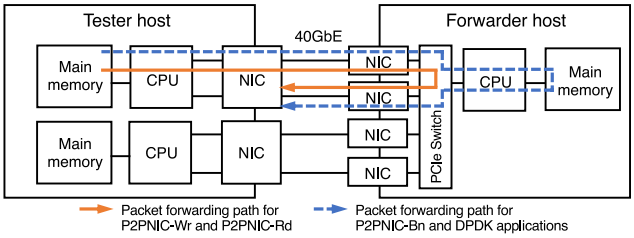


Fig. 7 Measurement setup for the evaluation: two Napatech NT200A02 NICs were installed in the tester host, and four Netronome Agilio CX 2x40G NICs were installed in the forwarder host. The two links below were used for the measurements in Section 5.3.

the CPU and main memory are not involved in the packet forwarding process because each NIC directly forwards packets to other NICs through a PCIe switch. On the other hand, P2PNIC-Bn and DPDK applications require the CPU and main memory to forward every packet. Note that for the measurement with a single flow, we used only two Ethernet ports for both tester and forwarder hosts.

For each measurement, we used two Napatech NT200A02 NICs as the traffic generator and receiver installed in the tester host. The NT200A02 NIC supports measuring the throughput at a rate of over 40 Gbps for 64-byte packets and the latency with the accuracy of several nanoseconds by utilizing its hardware. In the forwarder host, we used a single Ethernet port of each Netronome Agilio CX 2x40GbE NIC because this NIC does not provide sufficient PCIe bandwidth for accommodating two 40 Gbps Ethernet ports running at the wire-rate as its specification. The NICs in the forwarder host were installed under the same PCIe switch. The tester host was a machine based on the SuperMicro 4029GP-TRT with two Intel Xeon Gold 6246R 16 core CPUs and four 16 GB DDR4-2933 memories (two memory channels per CPU), and the forwarder host was based on the SuperMicro 4029GP-TRT2 with two Intel Xeon Gold 6246R 16 core CPUs, four 16 GB DDR4-2933 memories (two memory channels per CPU), and two Broadcom PEX 9797 PCIe switches. Note that the packet forwarding performance of P2PNIC-Bn and DPDK applications could be improved by the chipset's feature called Data Direct I/O Technology (DDIO) [21]. With DDIO, the chipset transfers received packets to the last-level CPU cache before the CPU access them. We enabled DDIO in the forwarder host for all measurements.

For the configuration basis in this evaluation, all methods were configured to maximize their throughput as long as there was no significant latency increase. The parameters set on this basis were the batch size of the NIC's descriptor updating (8) in P2PNIC-Wr, P2PNIC-Rd, and P2PNIC-Bn, and the number of queues (4), the corresponding CPU cores (4), and the batch size of the CPU's processing (8) in P2PNIC-Bn and DPDK applications. We made the implementation and parameters (e.g., the batch size) of the P2PNIC-Bn and other P2PNIC methods as common as possible to observe the bare impact of going through memory.

5.1 Throughput

To reveal the advantage of P2PNIC from the aspect of the packet forwarding architecture, we evaluate the throughput of the P2PNIC implementations and the DPDK applications. In this section, we measure the throughput of unidirectionally forwarded traffic between single fixed ports (1in-1out) to clarify the baseline performance of each method. Each packet in the traffic has the same destination IP address and 256 different source IP addresses to utilize all CPU cores for the DPDK applications, which rely on the Receive Side Scaling (RSS) feature of the NIC. We observed the throughput for packets of 64, 128, 256, 512, 1,024, 1,280, and 1,518 bytes according to the hardware counter of the NIC for an average of 10 seconds, excluding the 10 seconds before and after. We indicate the theoretical maximum bandwidth for the packets in **Fig. 8**, which is calculated by

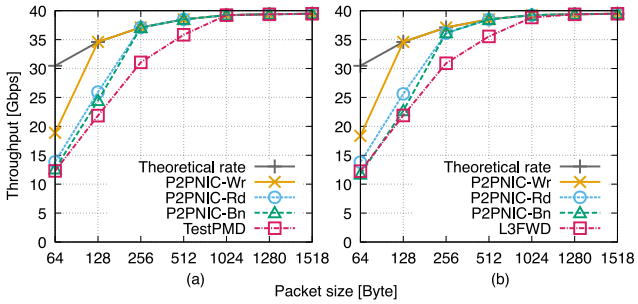


Fig. 8 Throughput of unidirectionally forwarded traffic between single fixed ports (1in-1out) along with the packet size for (a) without IP routing and (b) with IP routing.

$$\text{Theoretical rate (Gbps)} = \left(\frac{\text{Link speed (Gbps)}}{(\text{Preamble, SFD, and IFG (bytes)} + \text{Packet size (bytes)}) \times 8}\right) \times (\text{Packet size (bytes)} \times 8). \quad (1)$$

Note that the aggregate size of Ethernet Preamble, Start Frame Delimiter (SFD), and Inter-Frame Gap (IFG) is 20 bytes.

As a result, the throughput of P2PNIC-Wr with 64-byte and 128-byte packets outperforms other methods in both settings for forwarding packets to a fixed port (Fig. 8 (a)) and by IP routing (Fig. 8 (b)). When the IP routing is enabled, P2PNIC-Wr achieved 35.85 Mpps (60% of the wire-rate) for 64-byte packets and 33.78 Mpps (100% of the wire-rate) for 128-byte packets, which is 1.50 times and 1.58 times higher than L3FWD, respectively. Up to 1,024-byte packet size, P2PNIC-Wr, P2PNIC-Rd, and P2PNIC-Bn show higher throughput than L3FWD.

There is a major performance gap between P2PNIC-Wr and P2PNIC-Rd; the reason is that P2PNIC-Wr has a better balance of the memory pressure and processing load between the ingress and egress NICs. For P2PNIC-Wr, the ingress NIC executes DMA Write to transfer a packet buffer, and the egress NIC executes busy-waiting to detect a newly transferred packet. On the other hand, P2PNIC-Rd makes the egress NIC process both DMA Read to retrieve a packet buffer and busy-waiting. Therefore, the memory pressure and processing load are biased in the egress NIC. In addition, the performance gap between P2PNIC-Rd and P2PNIC-Bn shows the bare overhead to forward packets through the main memory. The difference is small for the condition below 40 Gbps but increases along with the traffic volume (Section 5.3). L3FWD is based on the CPU-driven architecture as with P2PNIC-Bn, but shows lower throughput; this is due to the implementational differences between DPDK (L3FWD) and ours (P2PNIC-Bn) in firmware and device drivers.

5.2 Latency

Latency is also a key metric for packet forwarding architectures, and thus we evaluated unidirectional latency on the P2PNIC implementations and the DPDK applications. In this section, the measurement for 1 Gbps background traffic of 64-byte packets with IP routing is used as the baseline performance. Comparing with the baseline performance, we observed the latency characteristics of each method with several traffic patterns that varied in the presence of IP routing, the amount of background traffic, and the packet size. Due to the limitation of

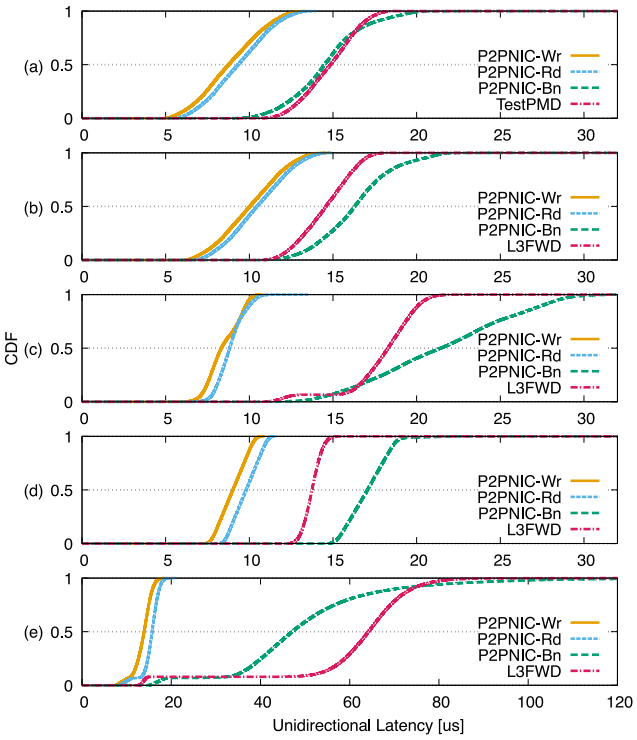


Fig. 9 CDF of the unidirectional latency in the following conditions: (a) 1 Gbps background traffic of 64-byte packets with fixed destination port, (b) 1 Gbps background traffic of 64-byte packets with IP routing (all the following are with IP routing), (c) 10 Gbps background traffic of 64-byte packets, (d) 1 Gbps background traffic of 1,518-byte packets, and (e) 40 Gbps background traffic of 1,518-byte packets.

the NIC in the tester host, only 1 Kpps traffic was counted by hardware timestamps, and other packets were forwarded without counting. We observed the latency in the aggregation of 30 seconds; thus, around 30,000 samples were counted for each measurement.

For all measurements shown in **Fig. 9**, P2PNIC-Wr and P2PNIC-Rd show lower latency than the methods of CPU-driven architecture (i.e., P2PNIC-Bn, TestPMD, and L3FWD). In the measurement for the baseline performance (Fig. 9 (b)), P2PNIC-Wr has 24% lower latency than L3FWD at the 90th percentile in 64-byte packets and 1 Gbps background traffic with IP routing. In addition, by comparing the latency when the destination port is fixed (Fig. 9 (a)) with that of the baseline performance (Fig. 9 (b)), we confirm that the processing of the IP routing by the ingress NIC increases the latency by 9% for P2PNIC-Wr at the 90th percentile. Nevertheless, the latency is still lower than that of the CPU-driven architecture.

We also see that increasing packet size and traffic volume change the distribution of latency but do not necessarily increase the latency itself. From the comparison between latency of the baseline performance (Fig. 9 (b)) and that with 10 Gbps background traffic (Fig. 9 (c)), the latency decreases by 22% at the 90th percentile. The reason for the decrease is that the P2PNIC implementations adopt the batching manner to update descriptors. With the batching manner, the egress NIC quickly detects incoming packets when the batch on the ingress NIC fills up quickly by 10 times higher packet rate than that of the baseline. In addition, from the comparison between latency of the baseline performance (Fig. 9 (b)) and that with 1,518-byte packets (Fig. 9 (d)), the latency decreases by 19% at the 90th percentile. In the condition for Fig. 9 (d), by increasing the packet size from 64 bytes to 1,518 bytes, the arrival rate of the packets becomes approximately 5% compared with the baseline condition. In such a low packet rate, the latency to forward a packet is decreased because the DMA queue for PCIe and the processing capacity in the NICs have more room to operate. Moreover, the descriptor will be updated without waiting for the batch to be filled because the decision to wait for the next packet is based on whether the next packet is incoming when the processing of one packet is finished.

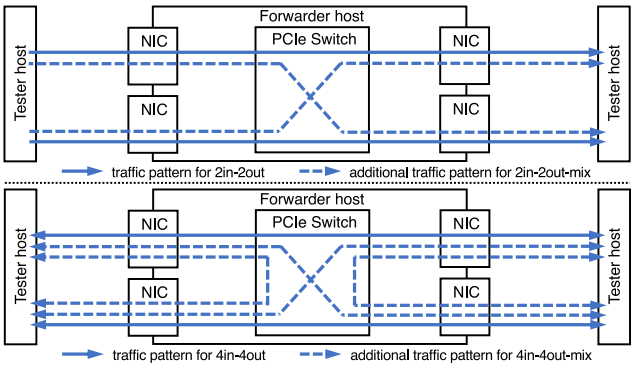


Fig. 10 The traffic patterns for 2in-2out, 2in-2out-mix, 4in-4out, and 4in-4out-mix. Each flow was directed by destination IP addresses. In the tests for 2in-2out-mix and 4in-4out-mix, flows destined for all other ports were generated in addition to the flow for 2in-2out and 4in-4out, respectively.

The result with 1,518-byte packets and 40 Gbps background traffic (Fig. 9 (e)) shows increased latency compared with that with the same packet size and 1 Gbps background traffic (Fig. 9 (d)). The increase of the latency in P2PNIC-Wr and P2PNIC-Rd is due to the memory pressure and processing load in the NICs. However, the increased degree of P2PNIC-Wr (1.57 times) is much lower than P2PNIC-Bn (3.80 times) and L3FWD (5.13 times), and thus the latency of P2PNIC-Wr is 79% lower than L3FWD at the 90th percentile.

5.3 Scalability with Multiple Ethernet Ports

In this section, we evaluate how the throughput of each method scales with the numbers of Ethernet ports because it would reveal the performance capacity of the CPU-driven and P2PNIC architectures. For the measurements, we prepared four traffic patterns: 2in-2out, 2in-2out-mix, 4in-4out, and 4in-4out-mix, as shown in **Fig. 10**. The 2in-2out is unidirectional and consists of two flows, where the tester host sends traffic to two ports in the forwarder host simultaneously, and the forwarder host sends back the traffic from another two ports to the tester host. For this pattern, each packet in each flow has the same destination IP address and 256 different source IP addresses, as well as the measurement in Section 5.1. The 2in-2out-mix is almost the same as the 2in-2out, but the packets have the same percentage of two different destination IP addresses that are directed to two outgoing ports evenly. The purpose of the 2in-2out-mix pattern is to confirm the performance when multiple flows merge into a single outgoing port. The maximum aggregate throughput for 2in-2out and 2in-2out-mix is 80 Gbps. The 4in-4out and 4in-4out-mix are the

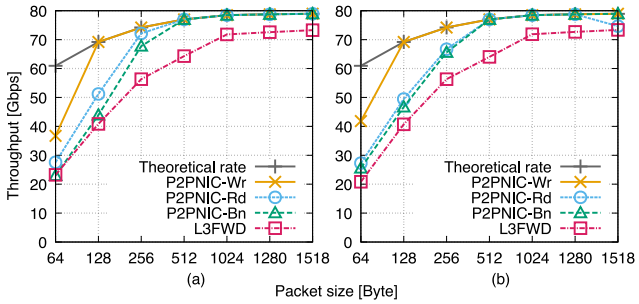


Fig. 11 Aggregate throughput for (a) 2in-2out and (b) 2in-2out-mix traffic patterns.

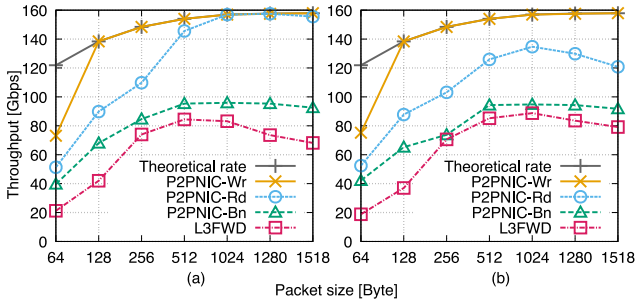


Fig. 12 Aggregate throughput for (a) 4in-4out and (b) 4in-4out-mix traffic patterns.

bidirectional versions of 2in-2out and 2in-2out-mix, respectively, where the traffic comes and goes out of the same four ports. For the 4in-4out-mix, the packets have the same percentage of three different destination IP addresses to direct them to the other three ports. The maximum aggregate throughput for 4in-4out and 4in-4out-mix is 160 Gbps. The theoretical maximum bandwidth for the packets indicated in **Fig. 11** and **Fig. 12** is calculated by Equation 1 mentioned in Section 5.1.

From the comparison between the result of 2in-2out (Fig. 11 (a)) and 4in-4out (Fig. 12 (a)), we can clearly see the difference in scalability between the CPU-driven and the P2PNIC architectures. In 2in-2out, the performance difference between the methods of P2PNIC architecture (i.e., P2PNIC-Wr and P2PNIC-Rd) and that of CPU-driven architecture (i.e., P2PNIC-Bn and L3FWD) is relatively close, as well as the measurement in Section 5.1. When the packet sizes are 64, 128, and 1,518 bytes, the throughput of P2PNIC-Wr are 1.58 times, 1.69 times, and 1.08 times higher than L3FWD, respectively. By contrast, in 4in-4out, the difference of throughput between the methods of CPU-driven and P2PNIC architectures increases significantly. For 4in-4out, when the packet sizes are 64, 128, and 1,518 bytes, the throughput of P2PNIC-Wr are 3.44 times, 3.30 times, and 2.32 times higher than L3FWD, respectively. This result shows that the scalability limit of the CPU-driven architecture is lower than that of the P2PNIC architecture.

Another noteworthy result is that P2PNIC-Wr achieves the wire-rate for all measurements when the packet size is above 64 bytes. The result indicates that P2PNIC-Wr still has room for more traffic volume, even when it forwards the maximum traffic rate that can be generated by four 40 Gbps Ethernet ports.

There is no noticeable change in trend between 2in-2out (Fig. 11 (a)) and 2in-2out-mix (Fig. 11 (b)) and between 4in-4out (Fig. 12 (a)) and 4in-4out-mix (Fig. 12 (b)), except for P2PNIC-Rd. In other words, methods other than P2PNIC-Rd can support the pattern of multiple flows merging at a single outgoing port without significant overhead. For P2PNIC-Rd, the throughput degradation occurs when the packet size increases because it lacks the balance of memory pressure and processing load between ingress and egress NICs compared with P2PNIC-Wr, as mentioned in Section 5.1. For example, in P2PNIC-Rd, the egress NIC processes both DMA Read to retrieve a packet buffer and busy-waiting. Moreover, the lack of balance strongly affects its throughput when the egress NIC processes multiple descriptor rings. This is because the number of available RISC processors per descriptor ring at a time decreases compared with processing a single descriptor ring.

Table 1 CPU usage per core of each method for forwarding 1,518-byte packets in 40 Gbps.

	P2PNIC-Wr	P2PNIC-Rd	P2PNIC-Bn	L3FWD
CPU usage	0%	0%	100%	100%

5.4 CPU Usage and Power Consumption

We clarify the CPU usage and the associated power consumption of the CPU-driven and P2PNIC architectures because these characteristics would be a constraint in increasing the number of Ethernet ports. We measured the CPU usage in 40 Gbps traffic load with 1,518-byte packets using the sysstat [17] tool to confirm that the P2PNIC architecture achieves high-speed packet forwarding on general-purpose servers without using the CPU. In addition, we measured the power consumption of P2PNIC-Wr and L3FWD with the same traffic pattern as in Section 5.1 and Section 5.3 to reveal how the load on the methods affects their power consumption. For the measurement of the power consumption, we sampled the instantaneous power consumption value at one-second intervals in watts through the Intelligent Platform Management Interface (IPMI). Although the accuracy of the power consumption value read from IPMI is not sufficient for high-resolution measurement, it can be used to grasp trends from averages of samples taken at intervals of one second or more [18]. We observed the CPU usage and power consumption for an average of 10 seconds, excluding the 10 seconds before and after.

As shown in **Table 1**, P2PNIC-Wr and P2PNIC-Rd do not show any increase in CPU usage, while P2PNIC-Bn and L3FWD show 100% CPU usage. This is because P2PNIC-Wr and P2PNIC-Rd forward packets without the involvement of the CPU and main memory, whereas P2PNIC-Bn and L3FWD use busy-waiting on the CPU to detect the arrival of packets. No use of the CPU in the P2PNIC architecture is an architectural advantage because it allows for scaling the number of Ethernet ports without considering the lack of CPU resources.

Moreover, as shown in **Fig. 13**, the power consumption of P2PNIC-Wr and L3FWD shows a significant difference. While L3FWD consumes about 70–80W, P2PNIC-Wr consumes less than 10W of power in addition to the idle power consumption. As noted, the accuracy of IPMI is not sufficient for high-resolution measurement and should be limited to understanding trends based on averages over a few seconds. However, there is clearly a difference in power consumption between the CPU-driven and

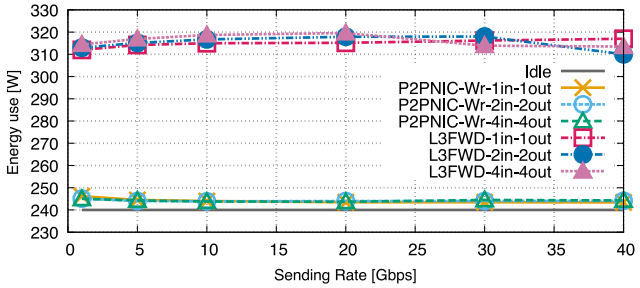


Fig. 13 Power consumption during packet forwarding for 1,518-byte packets when the traffic pattern is 1in-1out, 2in-2out, and 4in-4out.

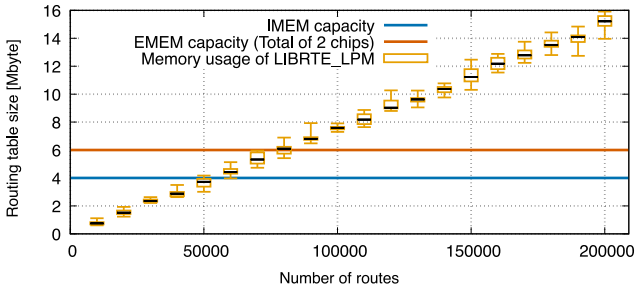


Fig. 14 Routing table size of LPM library in DPDK along with the number of routes. The routes are randomly selected from the BGP full route of an actual ISP.

P2PNIC architectures. The power consumption of the P2PNIC architecture would be lower than that of the CPU-driven architecture even if the number of NICs in the P2PNIC architecture were increased several times. The larger margin for power consumption of P2PNIC-Wr is also an architectural advantage for scaling the number of Ethernet ports.

6. Discussion

In this section, we discuss the limitation, extensibility, and application of the P2PNIC architecture compared with existing architectures.

6.1 Limitation on Scalability of Routing Table Size

The P2PNIC architecture has some limitations on scalability due to the capacity of the device memory that the NIC can provide compared to the CPU-driven architecture. The most severe constraint, when compared to the CPU-driven architecture, would be the number of routes that the NIC can accommodate. In this section, to reveal how many routes the NIC can accommodate, we compared the routing table size and capacity of NIC's device memory. In addition, the capacity of NIC's device memory could also constrain the number of accommodatable NICs in a single router, as mentioned in Section 3.3.

Figure 14 shows the routing table size of the LPM library in DPDK called librte_lpm, along with the number of routes. We executed the routing table construction ten times for each number of routes, with the routes randomly selected from 841,941 routes that make up the Internet (BGP full route). The routes were sampled from an actual ISP on September 6, 2021. In addition, we show the capacity of the device memory that the Netronome Agilio CX 2x40G NIC has. This NIC has several types of SRAM-based cache memories called Internal Memory (IMEM) and External Memory (EMEM). The SRAM-based cache memories are suitable to accommodate routing tables while maintaining high performance in packet processing because of their low access latency. However, these memories are not as large as that of DRAMs due to memory density constraints and resulting costs. Although some SmartNICs, including Netronome Agilio CX 2x40G NIC, are equipped with large DRAMs, the DRAMs are not suitable for accommodating routing tables because of their longer access latency than SRAM-based cache memories.

As a result, we can clearly see that accommodating the entire BGP full route requires a larger capacity of the device memory than the SmartNIC has. The result means that the routing table of the P2PNIC architecture is less scalable than that of the CPU-driven architecture. The routing table size of librte_lpm exceeds the capacity of the IMEM (4 MBytes) for 60,000 routes. The routing table size also exceeds the total capacity of two chips of the EMEM (6 MBytes) for 80,000 routes. Based on the above results, the P2PNIC architecture would be suitable for being used in the underlay part of overlay networks by technologies such as VxLAN [29] and Segment Routing [15]. These technologies have been rapidly deployed for not only data centers but also the ISP backbone networks in recent years [30]. The number of routes on the underlay part is typically small because the part is responsible only for reachability between the routers. On the other hand, the routers in the underlay part are required to accommodate a large amount of traffic between hosts in the overlay part of the network. Therefore, for the routers in the underlay part, the packet forwarding performance is more important than the scalability for the number of routes.

6.2 Extensibility

In the P2PNIC architecture, flexibility on packet forwarding (e.g., access control list and decapsulation/encapsulation for tunneling) can be achieved by implementing the features on NICs as with IP routing. In addition, most stateful data plane features (e.g., session-based firewall) can also be achieved by having only one side of the NICs, either ingress or egress, manage their state. However, some of the stateful features would require additional consideration for state consistency between NICs. For example, the traffic pattern where packets of a single session going from and returning to different ports requires state consistency between NICs. The consistency between NICs would not be achieved as a feature of a single NIC but instead as a management feature over multiple NICs in the equipment.

For recent network equipment, OSes running on the CPU are important to achieve qualitative aspects such as manageability, portability, interoperability, and security. For example, an OS controls dedicated hardware with switching chips for Ethernet while managing routing protocols. The OS also provides management features such as logging and statistics for the equipment. This approach has become mainstream with the rise of networking OSes for white-box switches [36], [38], [39], and the approach can also be directly applied to the P2PNIC architecture, except that the switching chips become NICs. Adding a control plane to the P2PNIC architecture would achieve the same level of operational features and costs with the CPU-driven architecture and white-box switches.

6.3 Impact on Other PCIe Communication

Because a NIC in the P2PNIC architecture forwards packets over PCIe switches on general-purpose servers, it may occupy the bandwidth of the PCIe switch and disrupt PCIe communication by other devices such as GPU and NVMe storage. A solution to this is devising the design of the PCIe topology. Because multiple PCIe switches can coexist in a single general-purpose server, accommodating NICs for the P2PNIC architecture and other PCIe devices by different PCIe switches would eliminate the interference with each other.

6.4 Application Area

The P2PNIC architecture can be a high-performance alternative to the CPU-driven architecture. For example, the P2PNIC architecture can achieve high-performance packet forwarding with more programmability than hardware-based network equipment. The characteristic of the P2PNIC architecture is suitable for the network operators who need high-performance and customized IP routers, for instance. On the other hand, the development cost for adding new features to the P2PNIC architecture would be higher than that of the CPU-driven architecture because of the need for NIC customization. With regard to this point, the development cost of adding new features to NICs is decreasing with the advent of SmartNICs such as those that can be programmed with P4 [40] or run applications on general-purpose OSes by the embedded CPUs [7], [37]. This means that the best choice depends on the requirements of developers and operators, just as the white-box switches have become an alternative to dedicated Ethernet switches. The white-box switches can achieve the same performance as dedicated Ethernet switches with a lower initial investment, but the operator may be burdened with a different operation method from the traditional dedicated Ethernet switches [11].

7. Conclusion

In this paper, we have proposed the P2PNIC architecture, which significantly improves packet forwarding performance on general-purpose servers by eliminating CPU and main memory from the packet forwarding path. In the P2PNIC architecture, the NIC itself processes the packet forwarding procedure and directly transfers the packets to other NICs over PCIe. By adopting this architecture, the throughput of packet forwarding on general-purpose servers can scale beyond the processing capacity of the CPU and bandwidth of the main memory, and the latency can be significantly reduced. A key part of the P2PNIC architecture is to apply P2P DMA to packet forwarding between Ethernet NICs, which has not been achieved so far. As a result, P2PNIC-Wr, which is one of the methods of the P2PNIC architecture, shows 3.44 times higher throughput and up to 79% lower latency than L3FWD in the measurement for the highest load. The results of this work will lay the foundation to make software-based network infrastructure achieve performance comparable with hardware routers.

Acknowledgments This work was supported by JSPS KAKENHI Grant Number JP20K19781.

References

[1] Aweya, J.: *Architectures With Bus-Based Switch Fabrics: Case Study-Cisco Catalyst 6000 Series Switches*, Wiley-IEEE Press (2018).
[2] Barbette, T., Soldani, C. and Mathy, L.: Fast Userspace Packet Processing, *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (*ANCS 2015*), pp.5–16, IEEE (2015).
[3] Bergman, S., Brokhman, T., Cohen, T. and Silberstein, M.: SPIN: Seamless Operating System Integration of Peer-to-Peer DMA Between SSDs and GPUs, *ACM Trans. Comput. Syst.*, Vol.36, No.2, pp.1–26 (2019).
[4] Broadcom Inc.: Broadcom Ships Tomahawk 4, Industry's Highest Bandwidth Ethernet Switch Chip at 25.6 Terabits per Second, Broadcom Inc. (online), available from ⟨https://jp.broadcom.com/company/news/product-releases/52756⟩ (accessed 2021-10-04).
[5] Broadcom Inc.: Dynamic Buffer Pool White Paper, Broadcom Inc. (online), available from ⟨https://docs.broadcom.com/docs/PEX8600_Dynamic_Buffer_Pool_WP_v1.0_17Oct07.pdf⟩ (accessed 2021-10-04).
[6] Broadcom Inc.: PEX88096: 98 lane, 98 port, PCI Express Gen 4.0 ExpressFabric Platform, Broadcom Inc. (online), available from ⟨https://jp.broadcom.com/products/pcie-switches-bridges/expressfabric/gen4/pex88096⟩ (accessed 2021-10-04).
[7] Broadcom Inc.: Stingray PS250, Broadcom Inc. (online), available from ⟨https://docs.broadcom.com/doc/PS250-PB⟩ (accessed 2021-10-04).
[8] Brouer, J.D. and Høiland-Jørgensen, T.: XDP - challenges and future work, *Proc. Linux Plumbers Conference* (*LPC 2018*) (2018).
[9] Cisco Systems Inc.: Cisco ASR 9000 Series Aggregation Services Routers Data Sheet, Cisco Systems Inc. (online), available from ⟨https://www.cisco.com/c/en/us/products/collateral/routers/asr-9000-series-aggregation-services-routers/data_sheet_c78-501767.html⟩ (accessed 2021-10-04).
[10] Dobrescu, M., Egi, N., Argyraki, K., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Manesh, M. and Ratnasamy, S.: RouteBricks: Exploiting Parallelism To Scale Software Routers, *Proc. ACM Symposium on Operating Systems Principles* (*SOSP'19*), pp.15–28, ACM (2009).
[11] Doyle, L.: White-box switches yield initial savings but pose challenges, Network World from IDG (online), available from ⟨https://www.networkworld.com/article/3453828/white-box-switches-yield-initial-savings-but-pose-challenges.html⟩ (accessed 2021-10-04).
[12] DPDK Project: DPDK: Home, DPDK Project (online), available from ⟨https://www.dpdk.org⟩ (accessed 2021-10-04).
[13] DPDK Project: L3 Forwarding Sample Application, DPDK Project (online), available from ⟨https://doc.dpdk.org/guides/sample_app_ug/l3_forward.html⟩ (accessed 2021-10-04).
[14] DPDK Project: Testpmd Application User Guide, DPDK Project (online), available from ⟨https://doc.dpdk.org/guides/testpmd_app_ug/⟩ (accessed 2021-10-04).
[15] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S. and Shakir, R.: Segment Routing Architecture, RFC 8402 (2018).
[16] Go, Y., Jamshed, M.A., Moon, Y., Hwang, C. and Park, K.: APUNet: Revitalizing GPU as Packet Processing Accelerator, *Proc. USENIX Symposium on Networked Systems Design and Implementation* (*NSDI'17*), pp.83–96, USENIX (2017).
[17] Godard, S.: Sysstat utilities home page, Sysstat utilities home page (online), available from ⟨http://sebastien.godard.pagesperso-orange.fr⟩ (accessed 2021-10-04).
[18] Hackenberg, D., Ilsche, T., Schöne, R., Molka, D., Schmidt, M. and Nagel, W.E.: Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison, *Proc. IEEE International Symposium on Performance Analysis of Systems and Software* (*ISPASS 2013*), pp.194–204, IEEE (2013).
[19] Han, S., Jang, K., Park, K. and Moon, S.: PacketShader: A GPU-Accelerated Software Router, *SIGCOMM Comput. Commun. Rev.*, Vol.41, No.4, pp.195–206 (2011).
[20] Hennessy, J.L. and Patterson, D.A.: A New Golden Age for Computer Architecture, *Comm. ACM* (online), available from ⟨https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext⟩ (accessed 2021-10-04).
[21] Intel Corporation: Intel Data Direct I/O Technology, Intel Corporation (online), available from ⟨https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html⟩ (accessed 2021-10-04).
[22] Kohler, E., Morris, R., Chen, B., Jannotti, J. and Kaashoek, M.F.: The Click Modular Router, *ACM Trans. Comput. Syst.*, Vol.18, No.3, pp.263–297 (2000).
[23] Le, Y., Chang, H., Mukherjee, S., Wang, L., Akella, A., Swift, M.M. and Lakshman, T.: UNO: Unifying Host and Smart NIC Offload for Flexible Packet Processing, *Proc. ACM Symposium on Cloud Computing* (*SoCC'17*), pp.506–519, ACM (2017).

[24] Li, A., Song, S.L., Chen, J., Li, J., Liu, X., Tallent, N.R. and Barker, K.J.: Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect, *IEEE Trans. Parallel Distrib. Syst.*, Vol.31, No.1, pp.94–110 (2019).

[25] Li, Y., Zhang, D., Liu, A.X. and Zheng, J.: GAMT: A Fast and Scalable IP Lookup Engine for GPU-based Software Routers, *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (*ANCS 2013*), pp.1–12, IEEE (2013).

[26] Linux Kernel Organization: VFIO - Virtual Function I/O, The Linux Kernel Archives (online), available from ⟨https://www.kernel.org/doc/Documentation/vfio.txt⟩ (accessed 2021-10-04).

[27] Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S. and Gupta, K.: Offloading Distributed Applications onto SmartNICs using iPipe, *Proc. Conference of the ACM Special Interest Group on Data Communication* (*SIGCOMM'19*), pp.318–333, ACM (2019).

[28] Liu, M., Peter, S., Krishnamurthy, A. and Phothilimthana, P.M.: E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers, *Proc. USENIX Annual Technical Conference* (*ATC 19*), pp.363–378, USENIX (2019).

[29] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M. and Wright, C.: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, RFC 7348 (2014).

[30] Matsushima, S., Filsfils, C., Ali, Z., Li, Z. and Rajaraman, K.: SRv6 Implementation and Deployment Status, Internet-Draft (2021).

[31] Mayer, R. and Jacobsen, H.-A.: Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools, *ACM Comput. Surv.*, Vol.53, No.1, pp.1–37 (2020).

[32] Moon, Y., Lee, S., Jamshed, M.A. and Park, K.: Acceltcp: Accelerating network applications with stateful TCP offloading, *Proc. USENIX Symposium on Networked Systems Design and Implementation* (*NSDI'20*), pp.77–92, USENIX (2020).

[33] Moro, D., Peuster, M., Karl, H. and Capone, A.: FOP4: Function Offloading Prototyping in Heterogeneous and Programmable Network Scenarios, *Proc. IEEE Conference on Network Functions Virtualization and Software Defined Networking* (*NFV-SDN 2019*), pp.1–6, IEEE (2019).

[34] Netronome: Agilio CX SmartNICs, Netronome (online), available from ⟨https://www.netronome.com/products/agilio-cx/⟩ (accessed 2021-10-04).

[35] NVIDIA Corporation: DPDK 21.02 Mellanox NIC Performance Report, DPDK Project (online), available from ⟨https://fast.dpdk.org/doc/perf/DPDK_21_02_Mellanox_NIC_performance_report.pdf⟩ (accessed 2021-10-04).

[36] NVIDIA Corporation: NVIDIA CUMULUS LINUX, NVIDIA Corporation (online), available from ⟨https://www.nvidia.com/en-us/networking/ethernet-switching/cumulus-linux/⟩ (accessed 2021-10-04).

[37] NVIDIA Corporation: NVIDIA Mellanox BlueField SmartNIC for Ethernet, NVIDIA Corporation (online), available from ⟨https://www.mellanox.com/files/doc-2020/pb-bluefield-smart-nic.pdf⟩ (accessed 2021-10-04).

[38] Open Compute Project: Open Network Linux, Open Compute Project (online), available from ⟨http://opennetlinux.org⟩ (accessed 2021-10-04).

[39] Open Compute Project: SONiC: Software for Open Networking in the Cloud, Open Compute Project (online), available from ⟨https://azure.github.io/SONiC/⟩ (accessed 2021-10-04).

[40] Pensando Systems: Pensando DSC-100 Distributed Services Card, Pensando Systems (online), available from ⟨https://pensando.io/documents/pensando-dsc-100-distributed-services-card/⟩ (accessed 2021-10-04).

[41] Rizzo, L.: netmap: a novel framework for fast packet I/O, *Proc. USENIX Annual Technical Conference* (*ATC 12*), pp.101–112, USENIX (2012).

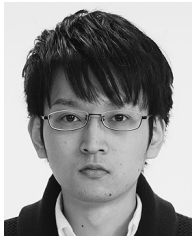
[42] Sun, W. and Ricci, R.: Fast and flexible: Parallel packet processing with GPUs and click, *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (*ANCS 2013*), pp.25–35, IEEE (2013).

[43] Ueno, Y., Nakamura, R., Kuga, Y. and Esaki, H.: P2PNIC: High-Speed Packet Forwarding by Direct Communication between NICs, *Proc. IEEE Conference on Computer Communications Workshops* (*IEEE INFOCOM 2021 WKSHPS*), pp.1–6, IEEE (2021).

[44] Vasiliadis, G., Koromilas, L., Polychronakis, M. and Ioannidis, S.: GASPP: A GPU-Accelerated Stateful Packet Processing Framework, *Proc. USENIX Annual Technical Conference* (*ATC 14*), pp.321–332, USENIX (2014).

[45] Wu, W.: *Packet forwarding technologies*, CRC Press (2007).

[46] Zhang, J., Donofrio, D., Shalf, J., Kandemir, M.T. and Jung, M.: NVMMU: A Non-Volatile Memory Management Unit for Heterogeneous GPU-SSD Architectures, *Proc. International Conference on Parallel Architecture and Compilation Techniques* (*PACT 2015*), pp.13–24, IEEE (2015).



Yukito Ueno is a Ph.D. course student at the Graduate School of Information Science and Technology, the University of Tokyo, Japan. He received a Master's degree in Media and Governance from Keio University, Tokyo, Japan, in 2016. He is currently with the department of Innovation Center in NTT Communications Corporation. His research interests include network architecture and software packet processing.

Ryo Nakamura received his Ph.D. degree in information science and technology from the University of Tokyo, Tokyo, Japan, in 2017. He is a research associate with the Information Technology Center at the University of Tokyo. His research interests include the networking aspect in operating systems, network virtualization, and network operation. He is a member of the IEICE and IPSJ.

Yohei Kuga received his Ph.D. degree in media and governance from Keio University, Tokyo, Japan, in 2015. He is an Associate Professor with the Information Technology Center at the University of Tokyo. His current research interests are systems aspects of networking hardware and high-performance computing and networking. He is a member of the IPSJ.

Hiroshi Esaki received his Ph.D. degree from the University of Tokyo, Japan, in 1998. In 1987, he joined Research and Development Center, Toshiba Corporation. From 1990 to 1991, he was with Applied Research Laboratory, Bell-Core, Inc., Murray Hill, NJ, USA, as a Residential Researcher. From 1994 to 1996, he was with the Center for Telecommunication Research, Columbia University, New York, NY, USA. Since 1998, he has been serving as a Professor with the University of Tokyo, and as a Board Member of WIDE Project. He is currently the Executive Director of IPv6 Promotion Council, the Vice President of JPNIC, IPv6 Forum Fellow, the Director of WIDE Project, and the Chief Architect of Japan Digital Agency. He is a member of the IEICE and IEEE.