# Performance Characteristics and Guidelines of Offloading Middleboxes onto BlueField-2 DPU

Fuliang Li, *Member, IEEE*, Qin Chen, Jiaxing Shen
Xingwei Wang *Member, IEEE,* and Jiannong Cao, *Fellow, IEEE*

***Abstract*—With the rapid growth in data center network bandwidth far outpacing improvements in CPU performance, traditional software middleboxes running on servers have become inefficient. The emerging data processing units aim to address this by offloading network functions from the CPU. However, as DPUs are still a new technology, there lacks comprehensive evaluation of their capabilities for accelerating middleboxes. This paper benchmarks and analyzes the performance of offloading middleboxes onto the NVIDIA BlueField-2 DPU. Three key DPU capabilities are explored: flow tables offloading, ARM subsystem packet processing, and connection tracking hardware offload. By applying these to implement representative middleboxes for firewall, packet scheduling, and load balancing, their performance is characterized and compared to conventional CPU-based versions. Results reveal the high throughput of flow tables offloading for stateless firewalls, but limitations as pipeline depth increases. Packet scheduling using ARM cores is shown to currently reduce performance versus CPU-based scheduling. Finally, while connection tracking hardware offload boosts load balancer bandwidth, it also weakens connection creation abilities. Key lessons on efficient middleboxes offloading strategies with DPUs are provided to guide further research and development. Overall, this paper offers useful benchmarking and analysis of emerging DPUs for accelerating middleboxes in modern data centers.**

***Index Terms*—Data processing unit, performance characteristics, offloading middleboxes.**

## I. INTRODUCTION

WITH the advent of the post-Moore era [1], the rapid growth in NIC performance has caused data center network (DCN) bandwidth to increase at a rate that far exceeds improvements in CPU computing power. Forwarding and processing network traffic on servers in modern data centers (DCs) leads to higher general CPU resource consumption and becomes more inefficient. This results in the performance of middleboxes on these servers gradually being unable to meet the demands of high-speed networks. The emergence of the DPU (Data Processing Unit) [2] aims to solve these problems. It releases the general computing resources of servers in modern DCs by offloading traffic processing from the CPU. The DPU also utilizes special hardware to accelerate packet processing, achieving the goals of decreasing costs and increasing benefits. Consequently, offloading middleboxes onto DPUs is a promising and efficient solution.

Currently, there have been some works on offloading network traffic processing onto DPUs. IO-TCP [3] focuses on offloading the TCP protocol stack onto the DPU. Its main idea is to offload data plane tasks onto the DPU while reserving control plane tasks for the CPU on the host. This saves significant CPU resources and greatly improves transmission performance using hardware accelerators. NanoBPF [4] target offloading QUIC encryption tasks with high CPU usage onto the DPU, which significantly increases QUIC throughput [5]. BluesMPI [6] achieves maximum overlap of communication and computation for applications by offloading communication to the DPU, thus substantially reducing program execution times. Kaushik et al identify bottlenecks in point-to-point and collective communication patterns and design a generic framework for offloading communication to the DPU [7]. Anton's work [8] concentrates on offloading the networking stack of overlay networks onto the DPU, intending to demonstrate decreased host overhead and improved performance of running processes.

Although related works on offloading network services onto DPUs have gradually increased and received growing attention from industry and researchers, DPUs remain a new type of network infrastructure lacking comprehensive performance evaluation data. This data could provide crucial references for industry and researchers when offloading network functions onto DPUs. Therefore, we explored the BlueField-2 DPU, a representative and popular product from NVIDIA, one of the largest NIC suppliers. We focused on testing and analyzing the performance of offloading middleboxes onto this DPU. Our main contributions are as follows:

- We analyzed the architecture and features of the BlueField-2 DPU, and benchmarked three key functions: flow tables offloading, ARM subsystem packet processing, and connection tracking [9] hardware offload. This provided insights into the maximum performance and guidelines when utilizing these functions.
- We leveraged these key functions to offload typical middleboxes that address hot issues in modern DCs, including stateless firewalls, per-packet rerouting packet schedulers, and L4 load balancers.
- We evaluated the performance of these offloaded middleboxes and compared them to CPU-based middleboxes. This analysis revealed the disadvantages and benefits of offloading from CPU to DPU.
- We validated problems identified from BlueField-2 DPU

F. Li, Q. Chen, and X. Wang are with Northeastern University, Shenyang 110819, China. E-mail: lifuliang@cse.neu.edu.cn, chenqin@stumail.neu.edu.cn, wangxw@mail.neu.edu.cn.
J. Shen is with Lingnan University, Hong Kong, E-mail: jiaxingshen@LN.edu.hk.
J. Cao is with the Hong Kong Polytechnic University, Hong Kong, E-mail: csjcao@comp.polyu.edu.hk.
*(Corresponding authors: Xingwei Wang and Jiannong Cao.)*

on BlueField-3 DPU which is the newest generation of NVIDIA DPU with better performance, to check whether these issues still existed and were being improved.

Based on our benchmark results and application performance comparisons, we had the following key findings:

- The performance of flow tables offloading decreased with more used entries and deeper pipes, but stateless firewalls using it still significantly outperformed iptables while saving substantial CPU resources. Our lesson is that developers should avoid routing traffic through excessive pipes.
- Packet scheduling utilizing the ARM subsystem's processing incurred major transmission performance reductions. The maximum throughput of DPU-based packet schedulers was far worse than CPU-based ones, despite DPU saving CPU consumption for scheduling. We learned that currently, offloading the entire data plane onto the ARM subsystem is unwise and inefficient for flexible packet processing.
- Connection tracking hardware offload did not drastically lower bandwidth or increase latency but significantly weakened connection establishment capabilities. Compared to LVS [10], L4 load balancers using it had higher bandwidth, lower latency, and zero CPU consumption but worse connection creation abilities. Our lesson is that applications leveraging connection tracking hardware offload presently cannot handle high concurrency short connections well.
- Most of the problems found on BlueField-2 DPU still exist on BlueField-3 DPU, although these issues have all been improved to a certain extent.

## II. Background and Motivation

### A. Background

**Middleboxes running on CPU have shown lower efficiency and high consumption of general-purpose computing resources.** Middleboxes are network devices used to transform, inspect, filter, and manipulate traffic, not just for packet forwarding. Typical middleboxes include firewall, NAT(network address translator), and L4 load balancer[11]. Nowadays, with the rapid development of cloud computing, these middleboxes are widely deployed on modern DC servers as infrastructures to improve network security and performance, relying on the general computing resources of the CPU to forward and process traffic as shown in the left part of Fig.1. However, with the rapid growth of flowing data in DC, the NIC has gradually evolved from 10G to 25G or even 100G, resulting in a far faster growth rate of network bandwidth than the increase in CPU performance. The traditional method of forwarding and manipulating traffic based on CPU has shown significant performance bottlenecks, and the CPU-based middleboxes provide lower efficiency, too. For example, when we use iptables as the firewall to maintain 20K filtering rules and filter TCP traffic with a bandwidth of 5Gbps, it will cause an additional significant consumption of 2 CPU cores. For modern data centers, consuming too many general-purpose computing resources on packet processing is not cost-effective.

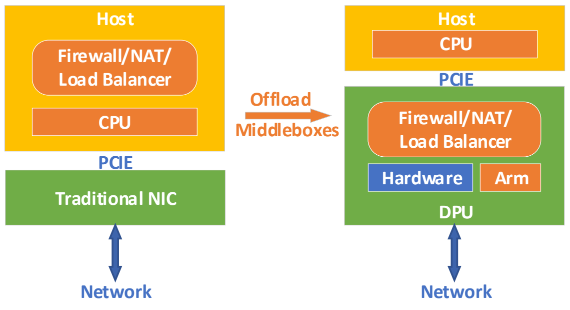


Fig. 1: Offload Middleboxes onto DPU

**Offloading Middleboxes onto DPU has higher efficiency and saves general-purpose computing resources.** DPU has emerged to address the problems of I/O performance bottlenecks and virtualization technology development constraints in the age of post-Moore's Law. DPU is a dedicated processor built around data, supporting the offloading of infrastructure services such as network, storage, and security. DPU utilizes dedicated hardware to complete network I/O, saving general computing resources for CPU, and significantly improving I/O performance. More importantly, it also provides network programmable capabilities, enabling customization of traffic processing logic. For example, when offloading the firewall as shown in the right part of Fig.1, DPU helped us save significant CPU consumption and get higher throughput when maintaining the same amount of filtering rules as iptables. Therefore, offloading middleboxes onto DPU has become a promising and efficient solution.

**Although accelerating and offloading network services is the most basic function of DPU, there is still a lack of comprehensive performance evaluation data about it.** DPU is a very cutting-edge technology for the industry, and most developers and researchers lack understanding of it. Having comprehensive evaluation data is of great reference significance for the industry and developers to understand the characteristics of DPU and develop offloaded applications. At the same time, offloading network services is the most fundamental function of DPU, so the performance evaluation on it is essential. Therefore, we have decided to analyze the network services offloading performance of the BlueField-2 DPU, which is the most representative product of NVIDIA.

### B. Related works

Currently, the most relevant work for characterizing the performance of BlueField-2 DPU is [12], where authors measure how much delay between packet transmissions before throughput drops by using Linux's pktgen [13] tool to analyze the maximum available processing headroom left by DPU to the applications when transmitting a batch of packets and measure how well DPU performs several targeted computational tasks by using stress-ng [14] tool to analyze which types of operations are profitable to be offloaded onto DPU. However, their work did not characterize the offloading performance of

DPU through widely deployed applications in DC and did not analyze the benefits of the hardware accelerators on DPU for offloading according to the specific architecture of DPU.

Another relevant work [15] for characterizing the performance of DPU comes from Tong Xing et al. They built up a testing framework for Linux host systems equipped with DPU based on general-purpose processors and used it to test Broadcom Stingray PS225 DPU and Mellanox BlueField-2 DPU. They are convinced that more users wish to use mature and feature-rich Linux regular socket APIs, so the framework focuses on measuring the performance of regular Linux kernel stacks running either on host or DPU subsystem equipped with general purpose processors, mainly including end-to-end latency, throughput, and multi-core scalability. Although their work is portable and measures the performance of offloading widely deployed applications, they only focused on the impact of DPU architecture on latency and throughput, as well as the performance of general-purpose processors on DPU, and still did not analyze the benefits of accelerators on DPU.

Another work [16] that is very similar to ours is from Georgios et al. They tested the performance of NVIDIA ConnectX series SmartNICs and BlueField-1 DPU and analyzed the problem of using the hardware-based packet classifier to help high-speed middleboxes track connections. However, their work did not analyze the performance of BlueField-2 DPU and BlueField-3 DPU, which support DPU's efficient software framework DOCA while BlueField-1 DPU did not. Their work did not include an analysis of the DPU Arm subsystem or an analysis of NVIDIA's new exclusive technologies, such as connection tracking HW-offload and DOCA-Flow. As a result, our testing is more comprehensive and up-to-date for understanding the performance of the current DPU.

Zhen Ni et al. implemented hardware-based load balancing using Netronome Agilio SmartNIC to distribute traffic evenly to different NFs on the server in [17]. Tianyi Cui et al. used Marvell LiquidIO3 SmartNIC to offload L4 load balancer and L7 load balancer in [18]. However, their SmartNICs are typical On-Path NICs, while the BlueField-2 DPU we use is a typical Off-Path NIC, and there are significant differences in their architectures. In off-path NICs, a NIC-level fabric switch provides connectivity between the network, the CPU, and the NIC cores, like eSwitch(embedded Switch) on BlueField-2 DPU.

## C. Motivation

Our testing goals are motivated by a specific project we have worked on. The purpose of the project is to offload an existing per-packet load balancing algorithm onto DPU and achieve multipath transmission of RDMA to overcome throughput reduction caused by hash collision when using ECMP [19] in the network with many redundant paths [20], to verify the algorithm can improve RDMA throughput and links utilization. Although we ultimately found that the RDMA protocol stack on BlueField-2 DPU did not support customization, we still found an alternative method to implement the prototype and validate the scheduling algorithm, which is based on the second tested function of DPU. Finally, We successfully extracted RDMA traffic in the Arm subsystem and implemented RDMA multipath transmission. The three functions we will test in this article also correspond to the three exploration stages below of implementing this prototype and they are the most three important functions for offloading network services onto DPU.

**Stage 1: Exploring offloading of flow tables to the DPU.** We found this function does not support customization and cannot achieve very flexible field matching and modification capabilities. For example, in the above project, we need to achieve per-packet modification of UDP checksum, but it does not meet our needs. However, we found that although its flexibility is limited, DPU basically can offload most of the flow tables in OpenFlow [21] through eSwitch, which can accelerate the matching and modification of typical fields in packets header and is very suitable for offloading stateless firewall from the host.

**Stage 2: Explore the flexible packet processing capability of the Arm subsystem.** We found that the Arm subsystem is the most flexible part of the DPU, capable of extracting all kinds of traffic from the host in ECPF(Embedded CPU physical function) mode, including RDMA. By leveraging the general-purpose processors of the Arm subsystem and the high-performance packet processing capabilities of the DPDK [22], we can achieve flexible traffic scheduling. In the end, we implemented our per-packet scheduling algorithm through this function but also discovered many potential offloading issues, such as offloading the whole data plane onto the Arm subsystem would consume a lot of computing resources and it's very inefficient.

**Stage 3: Explore the connection tracking HW-offload.** We found connection tracking HW-offload is an important function for achieving collaboration between the Arm subsystem and eSwitch. Although we have offloaded the per-packet load balancing algorithm onto the Arm subsystem, we're still unable to utilize hardware-based eSwitch to accelerate packet processing and forwarding. Finally, based on the principle of offloading OvS (Open vSwitch) [23] onto DPU, we understand that a more suitable traffic processing method for DPU should be to offload the data plane onto eSwitch and offload the control plane onto the Arm subsystem. The connection tracking HW-offload is the most typical manifestation of this idea and we can offload L4 load balancer from the host based on it.

After these three exploration stages, we have also understood that the most basic method for offloading network services onto DPU is to transform, inspect, filter, manipulate, and forward traffic through the flow tables offloading function, the flexible packet processing capabilities of the Arm subsystem and the connection tracking HW-offload function, which is to develop middleboxes on DPU. Therefore, we benchmarked these three important functions to get their maximum performance, implement three typical middleboxes closely integrated with production based on these three basic functions to compare with CPU-based middleboxes, to analyze the advantages and problems of offloading middleboxes from CPU onto DPU, and validate problems identified from BlueField-2 DPU on BlueField-3 DPU to check whether these issues still existed and were being improved.

## III. Measurement Methodology

### A. Testbed Setup

The main features of the BlueField-2 DPU under test are shown in Tab.I. We installed our tested DPU on the DUT(Device Under Test) and used the testing machine to send traffic to the DUT, receive traffic from the DUT, and obtain our test results. Both DUT and testing machine are equipped with 11th Gen Intel (R) Core (TM) i7-11700 CPU clocked at 2.50GHz, 32G DDR4 RAM clocked at 2666MHz, BlueField-2 DPU with the speed of 100GbE, and run the Ubuntu 20.04 (kernel v5.14) operating system. In most experiments, we connected one port of the DUT and one port of the testing machine back-to-back through a 100GbE link. During the functionality verification of the DPU-based packet scheduler, we connected two Barefoot Wedge100BF-32X switches to the DUT and the testing machine with 40GbE links respectively, and built a multipath network between the two switches. When testing the performance of the L4 load balancer, we connected two backend servers to port p0 of DUT through a Huawei CE6860 switch and three 100GbE links while port p1 of DUT is connected with the testing machine back-to-back. Both backend servers were equipped with ConnectX-6 DX 100GbE SmartNIC. As for problem validation on BlueField-3 DPU, DUT and testing machine are equipped with AMD EPYC 7302 16-Core CPU clocked at 2.50GHz, 256G DDR4 RAM clocked at 3000MHz, BlueField-3 DPU with the speed of 100GbE, run the Ubuntu 22.04 (kernel v5.15) operating system and connected back-to-back.

TABLE I: Main Features of BlueField-2 DPU

| | |
|---|---|
| **Processor** | 8 Cortex-A72 cores (64-bit) @ 2.5 GHz |
| **Cache** | 1MB L2 cache per 2 cores, 6MB L3 cache |
| **DRAM** | 16 GB on-board DDR4-1600 |
| **Storage** | eMMC flash memory |
| **Speed** | 10/25/40/50/100 GbE |
| **PCIe** | Gen 4.0 x 16 |
| **OS** | DOCA_1.5.0_BSP_3.9.3_Ubuntu_20.04-11 |
| **Kernel** | 5.4.0-1049-bluefield |
| **DPDK Version** | 20.11.6 |
| **DOCA Version** | 1.5.0 |

### B. Measurements.

In all experiments, our tested DPU was running in ECPF mode to manage the whole traffic of DUT. In benchmark, we mainly test the maximum performance of flow tables offloading, packet scheduling on the Arm subsystem, and connection tracking HW-offload separately, and compare them with the situation without introducing these three main functions to understand their impact on forwarding performance. In comparison tests, the stateless firewall was mainly implemented based on flow tables offloading and compared with iptables on the host, DPU-based packet scheduler was implemented based on the flexible packet processing capability of the Arm subsystem to achieve per-packet rerouting and compared with the CPU-based packet scheduler on the host, and L4 load balancer was implemented based on connection tracking HW-offload and compared with LVS on the host. The main indicators we tested were bandwidth, packet rate, latency, and CPU utilization, using tools such as DPDK-pktgen [24], iperf [25], dperf [26], sockperf [27], perftest [28], sar [29], etc. When testing the performance of the DPU-based packet scheduler, we also tested the maximum FCT using iperf and perftest to analyze the improvement of the DPU-based packet scheduler on tail latency and further validate its functionality. In addition, we also tested cps and tps using dperf and netperf [30] during the connection tracking benchmark and compared the performance of the L4 load balancer by testing cps and rps using dperf and Apache ab [31] to analyze the impact of connection tracking HW-offload on the ability to create new connections. The method of measurements on BlueField-2 DPU and BlueField-3 DPU are the same, we only provide results of BlueField-3 DPU in problem validation and results from other parts belong to BlueField-2 DPU in default.

## IV. Analysis of Offloading Middleboxes

### A. Analysis of Offloading Stateless Firewall

In this section, we first benchmarked the flow tables of DPU and analyzed the main factors that affect the performance. Then, we compared it with the iptables running on the host and analyzed the performance improvement and CPU resource savings that offloading can bring. Finally, we validated the problems found in the benchmark part on BlueField-3 DPU. We conclude that although the performance of the flow tables decreases with the increase of the number of used entries and the depth of the pipe on BlueField-2 DPU, the performance of the stateless firewall implemented by it is still significantly higher than that of iptables, and it can save high CPU consumption caused by packet filtering. Only under the 20k filtering rules, it saves 2 CPU cores for filtering 10 TCP flows with a bandwidth of 5Gbps. As for BlueField-3 DPU, the performance of the flow tables will not decrease with the increase of the number of used entries but still decrease with the depth of the pipe on BlueField-3 DPU.

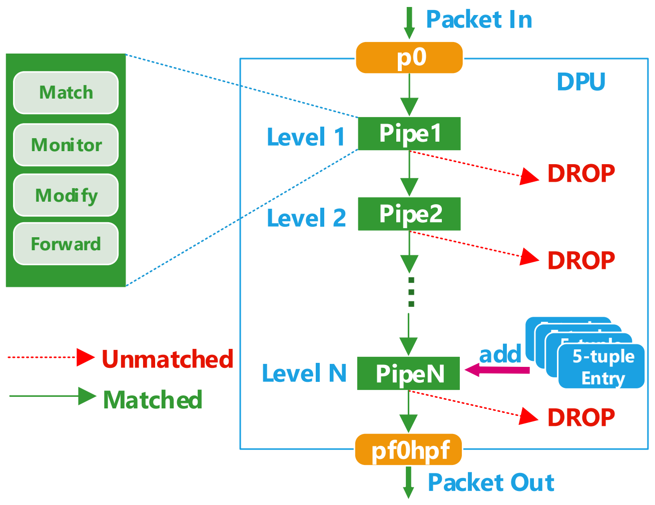


Fig. 2: Architecture of the tested DOCA-Flow program

*1) Benchmark of Flow Tables:* NVIDIA provides DOCA-Flow API to offload flow tables, so we use DOCA-Flow to

**Algorithm 1** Sketch of Tested DOCA-Flow Program

```
Input: $entryAmount$
Output: None
1: $iter \leftarrow 0$
2: while $iter < entryAmount$ do
3:     $src\_ip \leftarrow 192.168.200.1$
4:     $dst\_ip \leftarrow 192.168.200.2$
5:     $src\_port \leftarrow iter \mod 65536$
6:     $dst\_port \leftarrow 5001 + \lfloor iter/65536 \rfloor$
7:     $match \leftarrow (src\_ip, dst\_ip, UDP, src\_port, dst\_port)$
8:     ADD_ENTRY($pipeN$,$match$)
9:     $iter \leftarrow iter + 1$
10: end while
```

represent the flow tables of DPU during the subsequent tests. Through this API, we can build many pipes, each composed of match criteria, monitoring, modification, and forwarding. Traffic can be discarded and forwarded to the next port or pipe based on the match criteria. The connections between pipes are like a tree, and the pipe tree only has one root pipe. In addition, users need to add pipe entries to match the specific traffic and perform subsequent operations while a pipe just creates a traffic pattern, and usually unmatched traffic will be discarded.

**Goals.** In this part, we mainly aim to explore the impact of the number of pipe entries and pipe depth on forwarding performance through the benchmark. Therefore, we tested the bandwidth, packet rate, and latency under different numbers of flow entries and pipe depths to characterize these effects. According to the relevant official documents provided by NVIDIA, we have learned that the flow entries will be added to the flow tables on eSwitch, which can store up to 4 million flow entries. Therefore, we have set the maximum number of flow entries to 4M(4000K).

**Scenario.** We used DOCA-Flow API to write a simple tested program, whose architecture is shown in Fig.2 and algorithm sketch is shown in Algorithm.1. The entire tested program running in the Arm subsystem is mainly used to add flow entries to eSwitch and achieve the 5-tuple matching and traffic forwarding. It takes over the uplink representative port p0 and the host physical function representative port pf0hpf to filter the traffic from the network to the host. The level-N pipe discards unmatched traffic and forwards the matched traffic according to 5-tuple entries offloaded to hardware while other pipes only add a single entry to perform unconditional forwarding, which means that allowed traffic will pass through all pipes. The testing machine is connected back-to-back with the DUT through a 100GbE link. When testing bandwidth and packet rate, we ran DPDK-Pktgen on the testing machine, using 14 tx and 14 CPU cores to send different numbers of UDP flows. We also ran another DPDK program using 14 rx and 14 CPU cores on the tested machine to receive packets and calculate bandwidth and packet rate. When testing bandwidth, we sent 1518B-sized packets with 100Gbps bandwidth, when testing the packet rate, we sent 64B-sized packets with 117Mpps packet rate. Finally, we ran sockperf on both of them to measure the ping-pong latency of 1 UDP flow. As for the baseline of throughput and latency, we set the pipe depth of the tested program to 1, and the root pipe doesn't match any headers of packets. We only add a single entry into the root pipe to forward any type of traffic from the network(p0) to the host(pf0hpf). The maximum throughput and minimum latency tested in this situation are the baseline we used here, which are marked as green dashed lines in Fig.3.

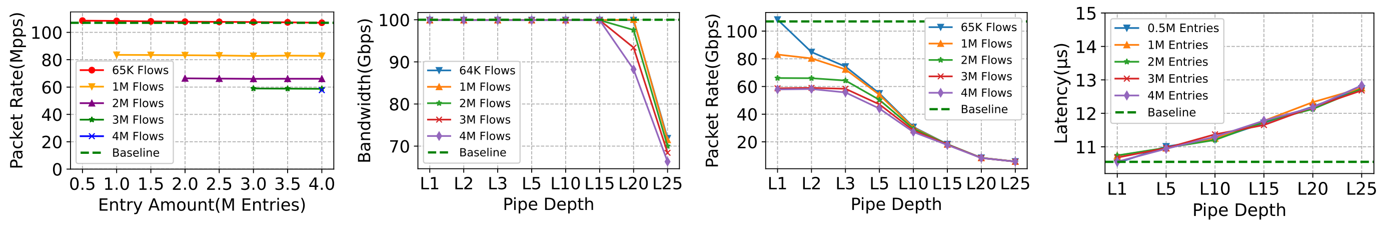
**The packet rate will decrease as the number of used entries increases.** We set the pipe depth of the program to 1, so the root pipe matches the 5-tuple. Each UDP flow corresponds to a pipe entry and flows without corresponding pipe entry will be discarded. Therefore, we only recorded the results when the number of entries $N_{Entries}$ is greater than or equal to flow amount $N_{Flows}$. We called the pipe entries corresponding to UDP flows used entries, with a quantity of $N_{UsedEntries}$, while the extra entries are called unused entries, with a quantity of $N_{UnusedEntries}$.The relationship between them is as equation 1. We tested the bandwidth and packet rate of different numbers of UDP flows using the two DPDK programs mentioned above under different numbers of entries and tested the UDP latency of a single flow using sockperf. We found that within the 4000k entries and 4000k UDP flows, the maximum bandwidth still can reach 100Gbps, and there is no significant change in the UDP latency($\sim$ 10.5 µs). As is shown in Fig.3(a), under the same number of UDP flows, increasing the number of entries will not significantly reduce the packet rate. However, increasing the number of UDP flows will significantly reduce the packet rate under the same number of entries. When all 4000k entries are used, the packet rate decreases by $\sim 40\%$.

$$\begin{aligned} &65K <= N_{Flows} <= N_{Entries} <= 4000K \\ &N_{Entries} = N_{UsedEntries} + N_{UnusedEntries} \\ &N_{Flows} = N_{UsedEntries} \end{aligned} \tag{1}$$

**The bandwidth and packet rate will significantly decrease as the depth of the pipe increases.** When exploring the impact of pipe depth on bandwidth and packet rate, we set the number of entries in the level-N pipe to the maximum value of 4000k and tested the bandwidth and packet rate of UDP flows at different pipe depths. The bandwidth results are shown in Fig.3(b), and the packet rate results are shown in Fig.3(c). We found that the maximum bandwidth begins to be less than 100Gbps when the depth of the pipe is greater than 20 and the maximum bandwidth decreases by $\sim 30\%$ when the depth is 25. More importantly, the packet rate significantly decreases with the increase of pipe depth and the packet rate even decreases by $\sim 90\%$ when the pipe depth reaches 25.

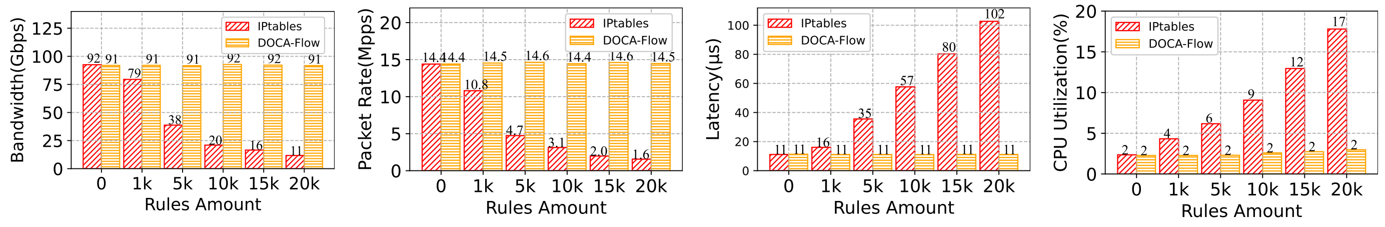
**The latency will increase as the depth of the pipe increases.** When exploring the impact of pipe depth on latency, we tested the ping pong latency of 1 UDP flow at different pipe depths, and the results are shown in Fig.3(d). We found that UDP latency increases with the increase of pipe depth, and the latency increases by $\sim$ 2µs when the pipe depth is 25.

*2) Performance Comparison:* In this part, we mainly want to compare the performance of the stateless firewall implemented by DOCA-Flow with iptables running on the host CPU. Therefore, we tested and compared the bandwidth,



(a) Packet Rate(Mpps) (b) Bandwidth(Gbps) (c) Packet Rate(Mpps) (d) Latency(μs)

Fig. 3: Bandwidth, packet rate and latency of DOCA-Flow under the different numbers of pre-installed entries and different pipe depths when using BlueField-2 DPU, (a) installed different number entries in the pipe at the first level while (b) and (c) installed 4M entries under different pipe depths.



(a) Bandwidth(Gbps) (b) Packet Rate(Mpps) (c) Latency(μs) (d) CPU Utilization(%)

Fig. 4: Performance comparison between iptables and DOCA-Flow under the different numbers of filtering rules based on 5-tuple when using BlueField-2 DPU.

packet rate, latency, and CPU utilization of DOCA-Flow with iptables under the different numbers of 5-tuple filtering rules. After our preliminary testing, we found the performance of iptables has significantly decreased while retaining over 20k rules, so we set the maximum number of rules at 20k.

**Scenario.** We still use the DOCA-Flow program mentioned above to compare with iptables. We set the pipe depth of DOCA-Flow to 1 and only allow matched traffic to pass through. For iptables, we only add 5-tuple rules in the INPUT chain. The testing machine, DUT, and topology are still the same as the benchmark, but we replaced the testing tool of bandwidth and packet rate with iperf because we had to use protocol stack and netfiler in the Linux kernel. We sent 10 TCP flows with 1518B-sized packets on the testing machine when testing the bandwidth, we sent 10 TCP flows with 158B-sized packets when testing the packet rate, and we calculated the bandwidth and packet rate on DUT using the statistics provided by the ethtool. Then, we tested the latency of a single TCP flow using the sockperf tool and the CPU consumption generated by DUT receiving 10 TCP flows with a total bandwidth of 5Gbps using the sar command.

**The performance of DOCA-Flow is significantly better than that of iptables and it can save high CPU consumption generated by packet filtering.**Fig.4(a), Fig.4(b) and Fig.4(c) show the bandwidth, packet rate and latency under the different numbers of filtering rules. We found that as the number of rules increases, the impact of iptables is very significant. When the number of rules only increases to 20K, the bandwidth decreases by $\sim 85\%$ the packet rate decreases by $\sim 90\%$, and the latency has also increased by $\sim$ 90μs, while the related performance of DOCA-Flow has remained nearly unchanged. Fig.4(d) shows the CPU consumption generated by receiving the same bandwidth TCP traffic under different filtering rules. We found that when the rules reached 20k, iptables generated a very significant CPU consumption, even 7 times the consumption generated by iperf receiving these packets, occupying approximately 2 more CPU cores while DOCA-Flow helped the host save this large amount of CPU consumption.

*3) Problems Validation:* In this part, we further explored whether the issues identified in the benchmark part also exist on BlueField-3 DPU, which is the next-generation DPU of NVIDIA. We conducted the same experiment as the benchmark part on BlueField-3 DPU by using DOCA 2.5.0. Tab.II shows the packet rate of two generations of DPU at different numbers of used entries, and Tab.III shows the performance of BlueField-3 DPU at different pipe depths. Through these comparative experiments, we have the following conclusions:

- The reason why the packet rate of BlueField-2 DPU in the benchmark part sharply decreases with the number of used entries is that we enabled the counter function for each entry, which is an important function that helps developers monitor the real-time rate of traffic.
- The packet rate will not decrease with the number of used entries on BlueField-3 DPU, regardless of whether the counter function is enabled for each entry.
- BlueField-3 DPU will also experience performance degradation with an increase in pipe depth, resulting in

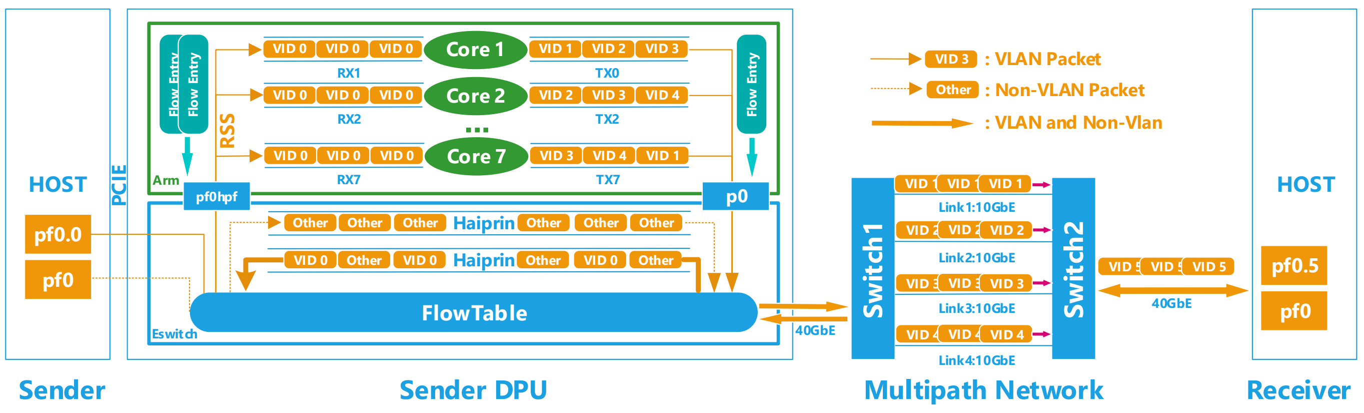


Fig. 5: Architecture of the Packet Scheduler on DPU

a decrease in packet rate and an increase in latency. Although compared to BlueField-2 DPU, BlueField-3 DPU has improved throughput on the same pipe depth, especially bandwidth.

TABLE II: Packet Rate(Mpps) of BlueField-2 and BlueField-3 under Different Amount of Used Entries

| DUT | Counter | Amount of Used Entries | | | | |
|---|---|---|---|---|---|---|
| | | 64k | 1000k | 2000k | 3000k | 4000k |
| BlueField-2 | Enable | 107.33 | 86.21 | 64.58 | 56.89 | 55.89 |
| BlueField-2 | Disable | 108.86 | 108.53 | 105.33 | 100.05 | 93.24 |
| BlueField-3 | Enable | 119.85 | 119.99 | 119.91 | 119.89 | 119.69 |
| BlueField-3 | Disable | 119.98 | 119.97 | 119.92 | 119.89 | 119.74 |

TABLE III: Performance of BlueField-3 under Different Pipe Depths

| Pipe Depth | Performance Indicators | | |
|---|---|---|---|
| | Bandwidth(Gbps) | Packet Rate(Mpps) | Latency(μs) |
| 1 | 99.99 | 119.69 | 13.66 |
| 3 | 99.99 | 110.67 | 14.13 |
| 5 | 99.99 | 66.82 | 14.45 |
| 10 | 99.99 | 57.58 | 14.93 |
| 25 | 99.99 | 26.47 | 16.62 |

## *B. Analysis of Offloading Packet Scheduler*

In this section, we mainly analyze the performance of BlueField-2 DPU for fine-grained packet scheduling. Because the eSwitch of the DPU does not have flexible field modification and computing capabilities, the main work of the entire scheduling is concentrated in the Arm subsystem of the DPU. We hope that this part can characterize the impact of offloading packet scheduling onto the BlueField-2 DPU Arm subsystem on throughput, delay, FCT, and CPU utilization. The DPU-based packet scheduler can not only achieve load balancing by offloading scheduling algorithms to save host CPU resources but also achieve traffic multipath transmission without modifying existing commercial switches and host protocol stacks. What's more, We found that BlueField-2 DPU not only supports scheduling traffic from the kernel protocol stack but also supports scheduling RDMA traffic from the special hardware on DPU. However, BlueField-2 DPU belongs to a typical Off-Path SmartNIC and Arm cores have relatively weak processing performance. Our results show that packet scheduling based on DPU significantly reduces transmission performance. When compared with CPU-based packet scheduling, although the DPU-based packet scheduler can save CPU consumption generated by scheduling, the maximum transmission performance is worse than the CPU-based packet scheduler. The problem exists on both BlueField-2 DPU and BlueField-3 DPU although BlueField-3 DPU is more capable of processing packets in the ARM subsystem than BlueField-2 DPU.

*1) Benchmark of Packet Scheduling:* In the Arm subsystem of BlueField-2 DPU, we mainly use DPDK for flexible programming on the data plane, including packet forwarding and scheduling. In addition, we can also extract critical flows by adding flow entries to eSwitch and redirect them to the Arm subsystem for fine-grained operations, and use the offloaded hairpin queues to forward irrelevant traffic to the network through eSwitch directly.

**Goals.** In this part, we mainly want to test the impact of the packet scheduler on forwarding performance and verify the effectiveness of packet scheduling through the benchmark. Therefore, we first tested the bandwidth and packet rate of the packet scheduler under different numbers of Arm cores, as well as the Ping-Pong latency of a single flow when hosts connect back-to-back. Then, we tested the maximum flow completion time(maximum FCT) of different numbers of TCP and RDMA flows in our multipath network and performed functionality verification of the DPU-based packet scheduler by comparing it with ECMP. Noted that performing throughput and latency testing back-to-back is to eliminate the impact of multipath networks and facilitate obtaining the optimal forwarding performance of the packet scheduler.

**Scenario.** We selected VLAN ID as the key hash entropy of ECMP to implement per-packet rerouting and implemented the framework of the packet scheduler using DPDK and the Round Robin algorithm. The specific architecture of the packet scheduler and the whole procedure is shown in Fig.5. The packet scheduler will add three flow entries during the initialization phase. One entry will separate the VLAN flows from the upstream traffic and distribute them to different queues in the Arm subsystem through RSS(Receive Side Scaling) [32].

The other one will forward the remaining traffic directly to the network through a hairpin queue, and the last one will forward all downstream traffic directly to the host through a hairpin queue. Each Arm core of the packet scheduler modifies the VLAN ID of the data packets in the queue to different values according to the scheduling algorithm(here we used the Round Robin algorithm) and forwards them to the network. This way, the packets of each flow can be distributed onto different redundant paths, allowing traffic to pass through multiple paths more evenly to improve links utilization. We first connected the sender (DUT) and receiver(testing machine) back-to-back through a 100GbE link. The sender sent 1K UDP flows through pktgen to test the maximum bandwidth and maximum packet rate. The receiver used the same receiving program as PartA to receive the traffic and calculate the result. When testing the bandwidth, the packet size was 1520B, and when testing the packet rate, the packet size was 64B. Then, we used sockperf to test the Ping-Pong latency of 1 UDP flow and 1 TCP flow and tested the Ping-Pong latency of RDMA send operation through perftest. Finally, we connected the sender and receiver to our multipath network using two 40GbE links, connected the two switches with four 10GbE redundant links, and configured ECMP in them. We tested maximum FCT for different numbers of TCP flows using iperf, and maximum FCT for different numbers of RDMA flows using perftest. In the back-to-back experiment, we found that the bandwidth of the packet scheduler can only reach up to 85Gbps. Therefore, we conservatively adjusted the port rate for the validation experiment from 100GbE to 40GbE.

**The packet scheduling based on the DPU Arm subsystem resulted in a significant decrease in bandwidth and packet rate.** We ran our packet scheduler and DPDK-Testpmd with different numbers of Arm cores in the Arm subsystem of DPU and used the performance of forwarding traffic without any other operation through eSwitch and DPDK-Testpmd as our reference. The bandwidth results are shown in Fig.6(a). We found that compared with eSwitch, packet scheduling resulted in a significant decrease in bandwidth, and with the increase of Arm cores and queues, there was a further decrease, even reaching $\sim 40\%$. Compared with DPDK-Testpmd, there was only a slight decrease. The results of packet rate are shown in Fig.6(b). We found that compared with eSwitch, the packet scheduler resulted in a significant decrease in packet rate, with the maximum packet rate even decreasing by $\sim 60\%$. Compared with DPDK-Testpmd, the packet rate only showed a significant decrease when using a few cores. When using six or more cores, the packet scheduler and DPDK-Testpmd were able to achieve the same packet rate. Therefore, we know that even if we use a relatively simple scheduling algorithm, packet scheduling based on the Arm subsystem still faces significant throughput reduction issues.

**The packet scheduling based on the DPU Arm subsystem resulted in a significant increase in latency.** We ran our packet scheduler and DPDK-Testpmd using 7 Arm cores and used the latency of forwarding traffic through eSwitch, Hairpin queues, and DPDK-Testpmd as our reference. The specific test results are shown in Tab.IV. Compared with eSwitch and Hairpin queues, after passing through the packet scheduler, the

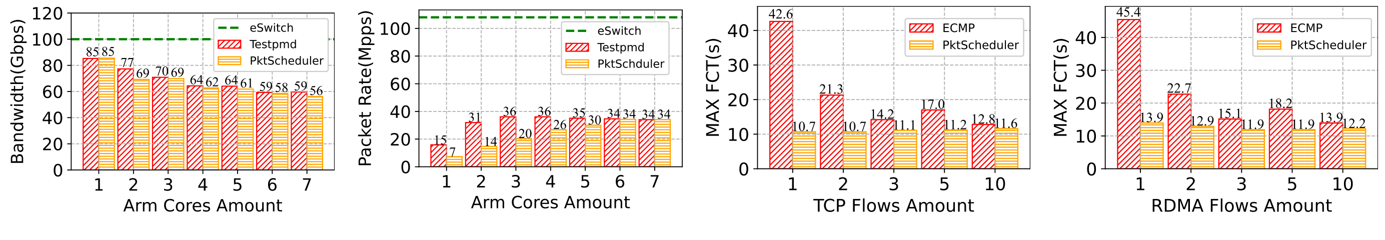
TABLE IV: Ping-Pong Latency(μs) of Different Kinds of Traffic when Using BlueField-2 DPU

| Forward Engine | Traffic Type | | |
|---|---|---|---|
| | RDMA | TCP | UDP |
| eSwitch | 2.18 | 11.03 | 11.33 |
| Hairpin | 3.79 | 13.61 | 12.74 |
| DPDK-Testpmd | 6.92 | 16.08 | 14.93 |
| PktScheduler | 5.23 | 15.20 | 14.26 |

Ping-Pong latency of all types of traffic showed a significant increase($\sim$ 3 μs). However, compared with DPDK-Testpmd, the latency of the packet scheduler has decreased. The specific reason is that DPDK-Testpmd forwards both upstream and downstream traffic using the Arm cores, while our packet scheduler only forwards upstream traffic using the Arm cores, while downstream traffic is forwarded using the hairpin queue with lower latency. The testing tools use RTT to calculate the latency, so the latency of both upstream and downstream needs to be considered. eSwitch is the most efficient forwarding engine on DPU, and we use its results($T_{eSwitch}$) as the latency baseline for the entire system. Using Hairpin queues will increase $T_{H\_Queue}$ in the whole system latency($T_{Hairpin}$), and using the Arm cores forwarding will increase $T_{Arm}$ in the whole system latency($T_{Testpmd}$). Therefore, we can infer that our packet scheduler implementing a simple scheduling algorithm will make the whole system latency about half of the sum of $T_{Hairpin}$ and $T_{Testpmd}$. The reasoning is shown in Eq.2, and our test results are also quite consistent with this speculation.

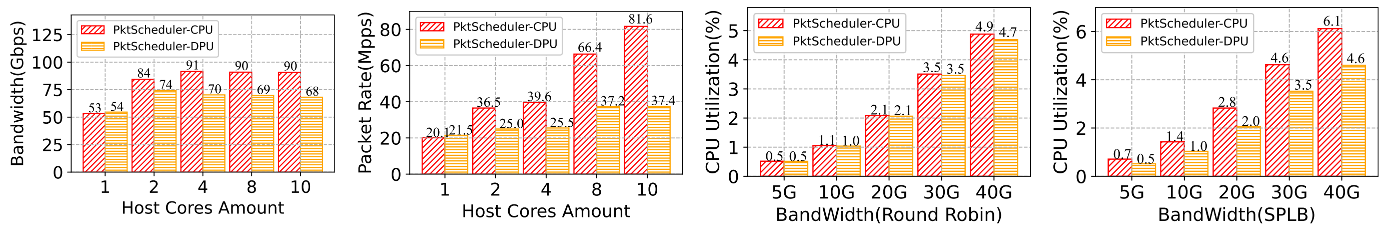
$$\begin{aligned}
T_{Hairpin} &= (T_{eSwitch} \times 2 + T_{H\_Queue} \times 2)/2 \\
T_{Testpmd} &= (T_{eSwitch} \times 2 + T_{Arm} \times 2)/2 \\
T_{PktScheduler} &= (T_{eSwitch} \times 2 + T_{Arm} + T_{H\_Queue})/2 \\
T_{PktScheduler} &= (T_{Testpmd} + T_{Hairpin})/2
\end{aligned} \tag{2}$$

**Compared with ECMP, packet scheduling based on the DPU Arm subsystem can effectively improve flow completion time.** We first connected the sender and receiver to the multipath network mentioned above, and then ran our packet scheduler with 7 Arm cores and compared the maximum FCT of TCP flow with ECMP. The specific results are shown in Fig.6(c). When the number of flows is less than 4, even if ECMP distributes all flows to different paths, the whole bandwidth cannot approach 40Gbps, and the maximum bandwidth of each flow will not exceed 10Gbps. However, the packet scheduler can make TCP occupy almost all bandwidth and distribute it evenly to each flow; when the number of flows is 5 and 10, and ECMP cannot evenly hash all traffic onto four redundant paths. Therefore, it is inevitable that the bandwidth allocated to the flow on paths with more flows will be very low. Therefore, in these five cases, the maximum FCT of TCP when using ECMP will be higher than using the packet scheduler. Noted that in these 5 cases, the amount of data we sent is 50GB. When the number of flows is 1, it sends data of 50GB, when the number of flows is 2, each sends data of 25GB, and so on. Therefore, the final TCP maximum FCT of the packet scheduler is around 11s. Finally, we also verified the effectiveness of the packet scheduler in scheduling



(a) Bandwidth(Gbps) (b) Packet Rate(Mpps) (c) Maximum FCT of TCP(s) (d) Maximum FCT of RDMA(s)

Fig. 6: Maximum throughput and functionality verification of DPU-based packet scheduler when using BlueField-2 DPU, (a) and (b) also show the bandwidth and packet rate of DPDK-Testpmd and eSwitch as a reference while (c) and (d) show the maximum FCT of ECMP to verify the functionality of DPU-based packet scheduler.



(a) Bandwidth(Gbps) (b) Packet Rate(Mpps) (c) CPU Utilization(%) (d) CPU Utilization(%)

Fig. 7: Performance comparison between DPU-based packet scheduler and CPU-based packet scheduler when using BlueField-2 DPU, (a) and (b) show the maximum throughput of them under the different number of CPU cores while (c) and (d) show the maximum CPU utilization of them under different bandwidth and scheduling algorithms.

RDMA. We ran our packet scheduler with just one Arm core and compared it with ECMP. The specific results are shown in Fig.6(d). Similarly, in these five cases, the maximum FCT of RDMA when using ECMP is significantly higher than when using the packet scheduler, too.

*2) Performance Comparison:* In this part, we mainly compared the performance of the CPU-based packet scheduler and the DPU-based packet scheduler. Therefore, we tested and compared the bandwidth, packet rate, latency, and CPU utilization of these two packet schedulers that achieve the same scheduling algorithms. In addition, we not only compared the performance of two schemes in implementing the Round Robin algorithm but also compared the CPU utilization of two schedulers in implementing the SPLB [33] algorithm. The Round Robin algorithm is relatively simple, so it does not bring much additional consumption to the CPU. SPLB is an algorithm that requires sending and receiving detection packets to obtain global congestion states and distribute traffic, which will bring obvious additional CPU consumption. Through the SPLB algorithm, we can more clearly see the CPU resources saved by offloading the scheduling algorithm onto DPU.

**Scenario.** We implemented a DPDK-based testing program on the sender host for sending UDP packets of different sizes and testing latency under different loads, then still set the port rate to 100GbE and connect the sender and receiver back-to-back. And we implemented the Round Robin algorithm in the host's testing program, which is the same as the packet scheduler in DPU. Then, we tested the maximum bandwidth and packet rate of CPU-based scheduling and DPU-based scheduling when using the different number of CPU cores in divide by sending flows with 1520-sized packets and 64B-sized packets. After that, we tested their latency at different packet rates. Finally, we implemented the SPLB algorithm on both the testing program of the host and the DPU-based packet scheduler, then compared the CPU utilization of DPU-based and CPU-based schedulers when using different scheduling algorithms to schedule traffic with different bandwidths.

**Although the packet scheduler based on the DPU Arm subsystem can save CPU consumption caused by scheduling, its performance is significantly weaker than that of the CPU-based packet scheduler.** Fig.7(a) and Fig.7(b) show the maximum throughput of a DPU-based packet scheduler and a CPU-based packet scheduler with the same number of CPU cores when using BlueField-2 DPU. We can observe that offloading the Round Robin scheduling algorithm onto the DPU only slightly improves throughput when we just use one CPU core. However, as the number of CPU cores increases, the traffic generated by the host has gradually exceeded the bottleneck of the DPU Arm subsystem. Therefore, we can maintain higher throughput even if we do not offload the Round Robin algorithm. When using 10 CPU cores, The packet rate of CPU-based schemes is even twice that of DPU-based schemes. Tab.V shows the overall latency impact of the two schemes at the same packet rate when using BlueField-2 DPU. Noted that 0.05Mpps is equivalent to the load when testing Ping-Pong latency by the testing tools like sockperf and perftest. We found that DPU-based scheduling also has a significant impact on latency. At a packet rate of 4Mpps, the latency is already four times that of the CPU-based scheme, it also dramatically reflects the negative effects of BlueField-2 DPU off-path architecture and weak processors on fine-grained packet processing. Fig.7(c) and Fig.7(d) show the CPU consumption generated by using Round Robin algorithm and SPLB algorithm to schedule the same bandwidth traffic in two schemes. As the entire testing program on the host is based on DPDK, the sar command may not accurately measure CPU consumption due to the polling mode. Therefore, inspired by the dperf, we use the real CPU cycles consumed by generating traffic and scheduling divided by the total cycles as the real CPU consumption. The results show that offloading a simple algorithm like Round Robin cannot significantly save CPU resources. However, offloading the SPLB algorithm which requires detecting path congestion status can significantly save CPU resources. When scheduling 40Gbps traffic, it can save 1.5% of CPU consumption, approximately 0.25 CPU core.

TABLE V: Latency(μs) under Different Packet Rate while Using BlueField-2(BF2) DPU and BlueField-3(BF3) DPU

| Forward Engine | DPU Model | Packet Rate of UDP Flow | | | | |
|---|---|---|---|---|---|---|
| | | 0.05Mpps | 1Mpps | 2Mpps | 3Mpps | 4Mpps |
| PktScheduler-CPU | BF2 | 9.16 | 9.62 | 10.74 | 11.37 | 12.18 |
| PktScheduler-DPU | BF2 | 15.85 | 50.85 | 56.57 | 52.17 | 52.33 |
| PktScheduler-CPU | BF3 | 7.83 | 7.86 | 10.66 | 11.07 | 11.18 |
| PktScheduler-DPU | BF3 | 11.89 | 14.64 | 16.94 | 17.79 | 18.15 |

*3) Problems Validation:* In this part, we also conducted the same experiment to further explore whether the problems that occurred in the performance comparison existed on the BlueField-3 DPU. As shown in Tab.VI and the last two rows of Tab.V, we have attached the differences in throughput and latency between the CPU-based packet scheduler and the DPU-based packet scheduler when running the Round Robin algorithm by using BlueField-3 DPU and DOCA 2.5.0. It should be noted that our BlueField-3 DPU and BlueField-2 DPU run on two different models of servers, so another set of data from the CPU-based packet scheduler will appear here. We can conclude that:

- When using BlueField-3 DPU, the DPU-based packet scheduler still has lower packet rates and higher latency compared to the CPU-based packet scheduler.
- Compared to using BlueField-2 DPU, DPU-based packet schedulers have significant improvements in latency and throughput, especially in bandwidth when using BlueField-3 DPU.

TABLE VI: Throughput of Packet Scheduler while using BlueField-3 DPU

| Forward Engine | Throughput Indicators | Host Cores Amount | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 4 | 8 | 10 |
| PktScheduler-CPU | Packet Rate(Mpps) | 24.01 | 47.32 | 86.08 | 79.85 | 79.66 |
| PktScheduler-DPU | Packet Rate(Mpps) | 25.01 | 49.68 | 63.42 | 60.38 | 59.13 |
| PktScheduler-CPU | Bandwidth(Gbps) | 65.95 | 99.91 | 99.91 | 99.91 | 99.91 |
| PktScheduler-DPU | Bandwidth(Gbps) | 69.03 | 92.65 | 99.91 | 99.91 | 99.91 |

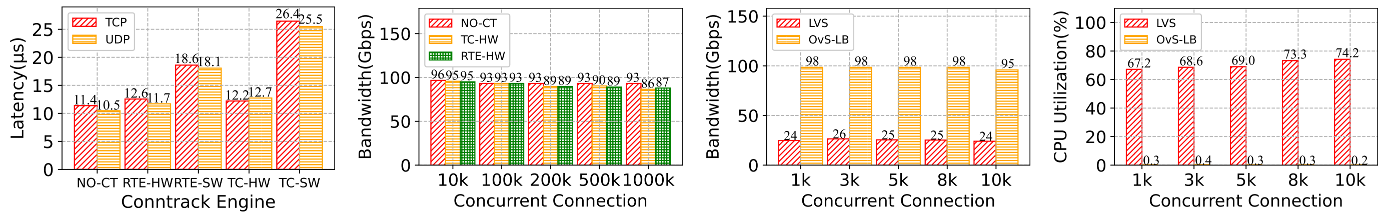
## *C. Analysis of Offloading L4 Load Balancer*

In this section, we first benchmarked connection tracking HW-offload and analyzed the impact on forwarding performance. Then, we compared the performance of LVS and the L4 load balancer based on connection tracking HW-offload. Finally, we validated the problem found in the benchmark part on BlueField-3 DPU. Our conclusion is that when the number of concurrent connections is within 1M, connection tracking HW-offload will not significantly reduce bandwidth, but will slightly increase latency and significantly weaken connection establishment ability. The problem of weakening connection establishment ability still exists on BlueField-3 DPU but has been improved compared to BlueField-2 DPU. The L4 load balancer based on connection tracking HW-offload can achieve higher bandwidth, and lower latency, and does not consume any CPU resources compared with LVS, but its ability to establish new connections is significantly inferior to LVS.

*1) Benchmark of Connection Tracking:* To offload connection tracking, users only need to enable the OvS hardware offload and add flow entries to eSwitch through the OvS command on the Arm subsystem. Whenever a new connection enters the DPU, eSwitch will redirect it to the Arm subsystem and hand it over to OvS for inspection. After the processing is completed and the connection is established, all packets from this connection will be directly forwarded through eSwitch to the next hop and will no longer enter the Arm subsystem. It's known that OvS has two common datapaths, namely system, and netdev. When DPU uses system as the datapath, OvS processes data packets in the kernel and adds flow entries to eSwitch through tc [34] to offload connection tracking. When netdev is used as the datapath, OvS processes data packets at the user level through DPDK and adds flow entries to eSwitch through rte-flow to offload connection tracking.

**Goals.** In this part, we mainly want to test the impact of offloading connection tracking on forwarding performance and connection establishment ability through benchmark. So we first tested the ping-pong latency when using different datapaths and maximum bandwidth under different concurrent connections to characterize the impact on forwarding performance. Then, we tested cps and tps when using different data paths to characterize the impact on connection establishment ability.

**Scenario.** We use OvS to connect the host physical function representative port pf0hpf and the uplink representative port p0 and use the ovs-ofctl command to add flow entries to enable connection tracking. The successfully established connection will be offloaded onto eSwitch for forwarding without other operations. The DUT and testing machine are still connected back-to-back using a 100GbE link. During testing ping-pong latency, we ran the sockperf server on the DUT and sockperf client on the testing machine to send TCP and UDP traffic. When testing bandwidth and cps, we ran the dperf server with 8 rx, 8 tx, and 8 CPU cores on DUT, and the dperf client with 8 rx, 8 tx, and 8 CPU cores on the testing machine to send TCP traffic. During the bandwidth tests, the packet size for upstream traffic was 1518B, and the packet size for downstream traffic was 164B. When testing tps, we ran the netperf server on the



(a) Latency(µs) (b) Bandwitdh(Gpps) (c) Bandwitdh(Gpps) (d) CPU Utilization(%)

Fig. 8: Latency and the maximum bandwidth of connection tracking HW-offload under different numbers of concurrent connections and performance comparison between OvS-LB and LVS when using BlueField-2 DPU, (c) and (d) show the maximum bandwidth and the CPU utilization of OvS-LB and LVS under different numbers of concurrent connections.

DUT and the netperf client on the test machine using a single thread.

**Enabling connection tracking HW-offload resulted in a slight increase in latency.** When testing the impact of connection tracking on ping-pong latency, we used sockperf to test the TCP and UDP latency without connection tracking as a reference and also tested the latency when disabling HW-offload as a reference. Fig.8(a) shows the specific test results of ping-pong latency. NO-CT means to disable connection tracking, RTE-HW, and RTE-SW mean to use DPDK to process packets at the user level and enable connection tracking, while TC-HW and TC-SW mean to process packets in the kernel and enable connection tracking. HW means to enable HW-offload, only part of the packets will enter the Arm subsystem, SW means to disable HW-offload, and all packets will enter the Arm subsystem. Firstly, from the comparison between RTE-SW, TC-SW, RTE-HW, and TC-HW, it can be seen that packets entering into the Arm subsystem still lead to a significant increase in latency, and the latency increase caused by the Arm subsystem kernel is more significant than using DPDK to process packets. Secondly, the comparison between RTE-HW, TC-HW, and NO-CT shows that enabling connection tracking HW-offload still increases latency by $\sim 1.2$µs.

**When the number of concurrent connections is within 1M, enabling connection tracking HW-offload does not cause a significant reduction in bandwidth.** We established different numbers of TCP concurrent connections using dperf, tested the maximum bandwidth of OvS-DPDK and OvS-kernel when enabling connection tracking HW-offload, and used the maximum bandwidth when disabling connection tracking as a reference. Fig.8(b) shows the specific test results. We found that within 1M concurrent connections, enabling connection tracking and offloading still ensures a relatively high bandwidth, which is only $\sim 6\%$ lower than NO-CT.

**Enabling connection tracking HW-offload will result in a significant decrease in cps and tps.** We ran netperf with a single thread and did TCP_CRR test to measure tps while enabling connection tracking HW-offload. Then, dperf was run on 8 CPU cores to test its cps. Similarly, we also tested tps and cps while disabling connection tracking and HW-offload on BlueField-2 DPU as references. The specific results of BlueField-2 DPU are shown in Tab.VII. Firstly,

TABLE VII: CPS(Kcps) and TPS(Ktps) under Different Conntrack Engine while Using BlueField-2(BF2) DPU and BlueField-3(BF3) DPU

| Performance Indicators | DPU Model | Conntrack Engine | | | | |
|---|---|---|---|---|---|---|
| | | NO-CT | RTE-HW | RTE-SW | TC-HW | TC-SW |
| TPS | BF2 | 10.86 | 6.21 | 6.76 | 4.58 | 5.52 |
| CPS | BF2 | 8000.00 | 84.96 | 130.00 | 30.00 | 64.95 |
| TPS | BF3 | 6.69 | 6.12 | 6.41 | 4.67 | 4.78 |
| CPS | BF3 | 10000.38 | 298.91 | 409.71 | 45.31 | 110.01 |

by comparing NO-CT with RTE-HW and TC-HW, we can find that the HW-offload of connection tracking significantly weakens the ability to create new connections, reducing tps by at least $\sim 37\%$ and cps by at least $\sim 98\%$. Secondly, through RTE-HW and TC-HW, it can be found that the HW-offload of connection tracking through DPDK is faster than the hardware offloading of connection tracking through the kernel. Finally, the comparison among RTE-HW, TC-HW, and RTE-SW, TC-SW shows that enabling HW-offload has a weakened ability to create new connections than disabling HW-offload, which also implies that the decline in connection establishment ability is not only due to the wimpy Arm cores, tracking connections through eSwitch and offloading established connections from the Arm subsystem to eSwitch is also a significant factor in the decline of cps and tps.

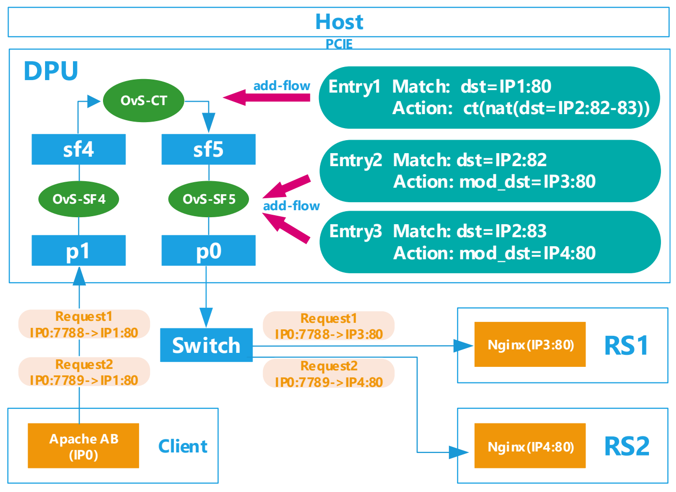


Fig. 9: Architecture of OvS-LB on DPU

*2) Performance Comparison:* In this part, we mainly want to compare the performance of the L4 load balancer based on connection tracking HW-offload with LVS running on the host CPU. Therefore, we tested and compared their ping-pong latency, rps, cps, and maximum bandwidth and CPU utilization under different numbers of concurrent connections. We developed an L4 load balancer called OvS-LB on DPU based on connection tracking HW-offload as Fig.9. For example, when we implement the DNAT of LVS and balance the requests to the nginx services with TCP port 80 on the backend real servers, we can treat the DPU as a virtual server. Firstly, we filter out the corresponding request traffic of the service through the OvS flow tables. Then we hash the traffic of different requests to different TCP ports of the same temporary IP through the DNAT function of connection tracking. Finally, we bind these TCP ports to different backend real servers' nginx services through the OvS flow tables. After preliminary experiments, we found that we can achieve DNAT load balancing in two-arm mode in the Arm subsystem of the DPU, without requiring the host CPU to participate in any work.

**Scenario.** We built the same testing environment shown in Fig.9, using the DPU of the DUT as a virtual server and running OvS-LB. We connected the testing machine as a client to the p1 port of the DPU through a 100GbE link and connected the other two backend real servers to the same switch with the p0 port of the DPU through three 100GbE links. Noted that the two physical ports of the DPU correspond to two PCIe devices with different PCIe IDs on the Arm subsystem, p0 and SF4 belong to one of the PCIe devices, and p1 and SF5 belong to the other PCIe device. When comparing with LVS, we only need to disable the OvS-LB of the DPU, let the traffic enter the host, and forward them to LVS for processing through the CPU. We only moved the sockperf server and dperf server to two real backend servers, then we tested the ping-pong latency of OvS-LB and LVS using sockperf and tested their cps and maximum bandwidth under different numbers of concurrent connections using dperf. In addition, we also deployed nginx services on backend servers and tested rps for obtaining 15B and 50MB files using Apache ab, then tested the CPU utilization under different numbers of concurrent connections.

**The L4 load balancer based on DPU connection tracking HW-offload can more easily achieve high bandwidth, and low latency, and save significant CPU consumption. However, the current performance of connection establishment is still significantly inferior to the CPU-based solution.** Fig.8(c) shows the maximum bandwidth of OvS-LB and LVS under different numbers of concurrent connections. We can find that OvS-LB can easily reach 100Gbps, while LVS can only reach around 26Gbps. Fig.8(d) shows the CPU consumption generated by OvS-LB and LVS sending 20Gbps traffic under different numbers of concurrent connections. LVS's CPU consumption increases with the number of connections, while OvS-LB does not cause any CPU consumption, which means that when sending 20Gbps traffic with 10k TCP connections, OvS-LB can save 12 CPU cores compared with LVS. Tab.VIII shows the ping-pong latency, rps, and cps of OvS-LB and LVS. We can see that OvS-LB can provide lower latency compared with LVS, but its connection establishment ability is greatly weakened, with a $\sim 53\%$ reduction in cps compared with LVS. In addition, when requesting 15B files, OvS-LB's rps decreased by $\sim 43\%$ compared with LVS, and when requesting 50MB files, OvS-LB's rps and LVS were nearly the same. These phenomenons also clearly indicate that programs developed based on DPU connection tracking HW-offload are currently not suitable for handling short connections with high concurrency, but more suitable for handling long connections with low concurrency.

TABLE VIII: Latency(μs), RPS(rps) and CPS(Kcps) of LVS and OvS-LB when Using BlueField-2 DPU

| **L4 Load** | **Performance Indicators** | | | | |
|---|---|---|---|---|---|
| **Balancer** | **Latecny(TCP)** | **Latecny(UDP)** | **RPS(15B)** | **RPS(50MB)** | **CPS** |
| LVS | 27.30 | 26.07 | 63311.17 | 22.49 | 130.00 |
| OvS-LB | 21.29 | 20.88 | 36091.09 | 21.69 | 60.00 |

*3) Problems Validation:* In this part, we still use the same experiments to verify whether the issues found in the benchmark part still exist on the BlueField-3 DPU. The last two rows of Tab.VII show the tps and cps of BlueField-3 DPU when enabling connection tracking HW-offload. We still tested tps and cps while disabling connection tracking and HW-offload as references. From the experimental data, we can conclude that:

- Enabling connection tracking HW-offload on BlueField-3 DPU still weakens connection creation ability. Although the weakening of tps by connection tracking HW-offload is less obvious because of the low baseline, the weakening of cps is still significant like BlueField-2 DPU.
- Compared to BlueField-2 DPU, BlueField-3 DPU has a significant improvement in connection creation ability when enabling connection tracking HW-offload. Especially when we use OvS-DPDK, cps has more improvement than OvS-Kernel.

## V. Discussion

**DPU should be equipped with a more flexible programmable datapath accelerator.** When having requirements for the higher flexibility of the data plane, developers must place the overall data plane in the Arm subsystems of Off-Path DPUs like NVIDIA BlueField-2 and Broadcom Stingray, which achieves the requirements but sacrifices performance. Meanwhile, On-Path DPUs like Netronome Agilio, rely on programmable multi-core processors to maintain a more flexible data plane and high performance, although its development difficulty is usually higher[35]. Therefore, DPU requires a more flexible, high-performance programmable datapath accelerator that balances the high performance of flow tables and the flexibility of the Arm subsystem.

**DPU should focus on improving the concurrency performance of connection tracking.** We've validated the issue about connection tracking HW-offload weakening connection creation abilities on BlueField-2 and BlueField-3 DPU and have seen improvements. AMD has also made significant improvements on this aspect in their Pensando DPU with a similar "SoC+ASIC" architecture, which has no requirement to offload to Arm processor for stateful service delivery and providing higher connection creation abilities($\sim$ 3Mcps)[36].

## VI. Conclusion

Motivated by the trend of offloading network functions onto DPU in the age of post-Moore's Law, this paper investigates the performance of offloading typical middleboxes onto NVIDIA BlueField-2 DPU based on three key functions including flow tables offloading, packet processing of the Arm subsystem and connection tracking HW-offload. We found that the flow tables offloading with inflexible programmability has high performance in processing traffic, which can accelerate stateless firewall and save high CPU consumption, and the Arm subsystem has flexible packet processing capability and it can save CPU consumption for packet scheduling, but currently it still has worse performance in processing traffic than CPU, and connection tracking HW-offload improves the bandwidth and latency of L4 load balancer while releasing CPU resources, but it's still not suitable for handling short connections with high concurrency. Most of the problems found on BlueField-2 DPU still exist on BlueField-3 DPU, although they've been improved.

## Acknowledgments

We would like to thank the Editors and Reviewers of IEEE Transactions on Computers for their review efforts and helpful feedback. This work is supported by the National Natural Science Foundation of China under Grant Nos. 62072091, U22B2005, 62032013, and 92267206.

## References

[1] Charles E Leiserson et al. "There's plenty of room at the Top: What will drive computer performance after Moore's law?" In: *Science* 368.6495 (2020), eaam9744.

[2] Idan Burstein. "Nvidia Data Center Processing Unit (DPU) Architecture". In: *IEEE Hot Chips 33 Symposium, HCS 2021, Palo Alto, CA, USA, August 22-24, 2021*. IEEE, 2021, pp. 1–20.

[3] Taehyun Kim et al. "Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery". In: *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022*. USENIX. 2023.

[4] Jichang Wang et al. "QUIC Cryption Offloading Based on NanoBPF". In: *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE. 2022, pp. 1–4.

[5] Adam Langley et al. "The quic transport protocol: Design and internet-scale deployment". In: *Proceedings of the conference of the ACM special interest group on data communication*. 2017, pp. 183–196.

[6] Mohammadreza Bayatpour et al. "Bluesmpi: Efficient mpi non-blocking alltoall offloading designs on modern bluefield smart nics". In: *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proceedings*. Springer. 2021, pp. 18–37.

[7] Kaushik Kandadi Suresh et al. "A Novel Framework for Efficient Offloading of Communication Operations to Bluefield SmartNICs". In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2023, pp. 123–133.

[8] Anton Njavro et al. "A DPU Solution for Container Overlay Networks". In: *arXiv preprint arXiv:2211.10495* (2022).

[9] P Ayuso. "Netfilter's connection tracking system". In: *LOGIN: The USENIX magazine* 31 (2006), pp. 34–39.

[10] Patrick O'Rourke and Mike Keefe. "Performance Evaluation of Linux Virtual Server." In: *LISA*. 2001, pp. 79–92.

[11] Brian E. Carpenter and Scott W. Brim. "Middleboxes: Taxonomy and Issues". In: *RFC* 3234 (2002), pp. 1–27.

[12] Jianshen Liu et al. "Performance Characteristics of the BlueField-2 SmartNIC". In: *CoRR* abs/2105.06619 (2021).

[13] Robert Olsson. "Pktgen the linux packet generator". In: *Proceedings of the Linux Symposium, Ottawa, Canada*. Vol. 2. 2005, pp. 11–24.

[14] Colin Ian King. "Stress-ng". In: *URL: http://kernel. ubuntu. com/git/cking/stressng. git/(visited on 28/03/2018)* (2017), p. 39.

[15] Tong Xing et al. "Towards portable end-to-end network performance characterization of SmartNICs". In: *APSys '22: 13th ACM SIGOPS Asia-Pacific Workshop on Systems, Virtual Event, Singapore, August 23 - 24, 2022*. ACM, 2022, pp. 46–52.

[16] Georgios P. Katsikas et al. "What You Need to Know About (Smart) Network Interface Cards". In: *Passive and Active Measurement - 22nd International Conference, PAM 2021, Virtual Event, March 29 - April 1, 2021, Proceedings*. Vol. 12671. Lecture Notes in Computer Science. Springer, 2021, pp. 319–336.

[17] Zhen Ni et al. "A SmartNIC-based Load Balancing and Auto Scaling Framework for Middlebox Edge Server". In: *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2021, Heraklion, Greece, November 9-11, 2021*. IEEE, 2021, pp. 21–27.

[18] Tianyi Cui et al. "Offloading load balancers onto SmartNICs". In: *APSys '21: 12th ACM SIGOPS Asia-Pacific Workshop on Systems, Hong Kong, China, August 24-25, 2021*. ACM, 2021, pp. 56–62.

[19] David Thaler and Christian E. Hopps. "Multipath Issues in Unicast and Multicast Next-Hop Selection". In: *RFC* 2991 (2000), pp. 1–9.

[20] Mohammad Alizadeh and Tom Edsall. "On the Data Path Performance of Leaf-Spine Datacenter Fabrics". In: *IEEE 21st Annual Symposium on High-Performance Interconnects, HOTI 2013, Santa Clara, CA, USA, August 21-23, 2013*. IEEE Computer Society, 2013, pp. 71–74.

[21] Nick McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *Comput. Commun. Rev.* 38.2 (2008), pp. 69–74.

[22] Ivano Cerrato, Mauro Annarumma, and Fulvio Risso. “Supporting fine-grained network functions through Intel DPDK”. In: *2014 Third European Workshop on Software Defined Networks*. IEEE. 2014, pp. 1–6.

[23] Ben Pfaff et al. “The design and implementation of open vSwitch”. In: *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 2015, pp. 117–130.

[24] Guo Li et al. “Toward energy-efficiency optimization of pktgen-DPDK for green network testbeds”. In: *China Communications* 15.11 (2018), pp. 199–207.

[25] Ajay Tirumala. “Iperf: The TCP/UDP bandwidth measurement tool”. In: *http://dast. nlanr. net/Projects/Iperf/* (1999).

[26] Jianzhang Peng. *dperf*. https://github.com/baidu/dperf. 2023.

[27] Mellanox. *sockperf*. https://github.com/Mellanox/sockperf. 2023.

[28] linux-rdma. *perftest*. https://github.com/linux-rdma/perftest. 2023.

[29] David Gavin. “Performance monitoring tools for Linux”. In: *Linux Journal* 1998.56es (1998), 1–es.

[30] R Jones. “Netperf: A network performance monitoring tool”. In: *http://www. netperf. org/netperf/NetperfPage. html* (2001).

[31] Yuelan Liu and Yuefan Liu. “Research of Advertisement performance measure system based on Apache Flink and AB testing”. In: *2021 6th International Conference on Multimedia and Image Processing*. 2021, pp. 78–82.

[32] Intel Server Adapters. *Receive Side Scaling on Intel Network Adapters*.

[33] Cong Xu et al. “Dual Channel Per-packet Load Balancing for Datacenters”. In: *39th IEEE Conference on Computer Communications, INFOCOM Workshops 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 2020, pp. 157–164.

[34] Werner Almesberger et al. *Linux network traffic control—implementation overview*. 1999.

[35] Xingda Wei et al. “Characterizing Off-path SmartNIC for Accelerating Distributed Systems”. In: *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 2023, pp. 987–1004.

[36] Deepak Bansal et al. “Disaggregating stateful network functions”. In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023, pp. 1469–1487.

**Fuliang Li** (Member, IEEE) received the BSc degree in computer science from Northeastern University, China in 2009, and the PhD degree in computer science from Tsinghua University, China in 2015. He is currently an associate professor at the School of Computer Science and Engineering, Northeastern University, China. He has published more than 50 Journal/conference papers. His research interests include network management and measurement, cloud computing, and network security.

**Qin Chen** received the BSc degree from Chang’an University, Xi’an, China, in 2021. He is currently pursuing the MS degree at Northeastern University, Shenyang, China. His research interests include Programmable Networks, SmartNIC, and High-performance transport protocols.

**Jiaxing Shen** is an Assistant Professor with the Department of Computing and Decision Sciences at Lingnan University. He received the B.E. degree in Software Engineering from Jilin University in 2014, and the Ph.D. degree in Computer Science from the Hong Kong Polytechnic University in 2019. He was a visiting scholar at the Media Lab, Massachusetts Institute of Technology in 2017. His research interests include mobile computing, data mining, and IoT systems. His research has been published in top-tier journals such as IEEE TMC, ACM TOIS, ACM IMWUT, and IEEE TKDE. He was awarded conference best paper twice including one from IEEE INFOCOM 2020.

**Xingwei Wang** (Member, IEEE) received the BS, MS, and PhD degrees in computer science from Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively. He is currently a professor with the College of Computer Science and Engineering, Northeastern University, Shenyang, China. He has authored or co-authored more than 100 journal articles, books, and book chapters, and refereed conference papers. His research interests include cloud computing and future Internet. He was the recipient of several best paper awards.

**Jiannong Cao** (Fellow, IEEE) received the BSc degree in computer science from Nanjing University, China, in 1982, and the MSc and PhD degrees in computer science from Washington State University, USA, in 1986 and 1990 respectively. He is currently a chair professor of distributed and mobile computing with the Department of Computing, The Hong Kong Polytechnic University. He is also the director of Internet and Mobile Computing Lab in the department and the Director of University Research Facility in Big Data Analytics. His research interests include parallel and distributed computing, wireless networks and mobile computing, big data and cloud computing, pervasive computing, and fault tolerant computing. He has co-authored five books in Mobile Computing and Wireless Sensor Networks, co-edited nine books, and published more than 500 papers in major international journals and conference proceedings.