IP Packet Forwarding Performance Comparison of the FD.io VPP and the Linux Kernel

Melinda Kosák, and Gábor Lencse

***Abstract*—There are numerous free software solutions for IPv4 or IPv6 packet forwarding. The Fast Data Project / Vector Packet Processing (FD.io VPP) is a novel and prominent solution. This paper investigates its performance and scalability compared to that of the Linux kernel. The investigation was conducted in accordance with the requirements outlined in the relevant Request for Comments (RFC) documents (RFC 2544, RFC 4814, and RFC 5180) using the siitperf measurement software. Two different test environments were used to eliminate the potential hardware-specific side effects and to gain insight into the performance and scalability of the IPv4 and IPv6 packet forwarding capability of the two investigated solutions. It was found that FD.io VPP outperformed the Linux kernel by approximately an order of magnitude. The configuration of FD.io VPP, along with the details of the measurements, are provided, and the results are presented and analyzed in the paper.**

***Index Terms*—FD.io VPP, IPv4, IPv6, Linux kernel, packet forwarding**

I. INTRODUCTION

THE Internet is an essential part of our life. All data of our daily communication is carried in packets by the Internet Protocol (IP). At the time of writing, there are two versions of IP: the older IPv4 version and the newer, increasingly more adopted IPv6. For IPv4 and IPv6 packet forwarding, there are numerous free software solutions, such as the Linux kernel, and one with exceptionally high performance called FD.io VPP. This article focuses on FD.io VPP, as its developers claim: "The Fast Data Project (FD.io) is an open-source project aimed at providing the world's fastest and most secure networking data plane through Vector Packet Processing (VPP)." [1]. We examine this proposition from a performance perspective. In this paper, we compare the IPv4 and IPv6 packet forwarding performance and scalability of the Linux kernel and FD.io VPP. By scalability, we refer to how their performance increases with the number of CPU cores utilized. First, we complete the performance and scalability test of the Linux kernel to establish a basis for comparison. Then we continue with the examination of the performance and scalability of FD.io VPP. For our tests, we use two different hardware environments.

Submittted November 18, 2024.

M. Kosák is with the Cybersecurity and Network Technologies Research Group of the Faculty of Mechanical Engineering, Informatics, and Electrical Engineering, Széchenyi István University, Győr, Hungary (e-mail: kosakmeli@gmail.com).

G. Lencse is with the Department of Telecommunications, Széchenyi István University, Győr, Hungary (e-mail: lencse@sze.hu).

The remainder of this paper is organized as follows. In section II, a brief introduction is given to the theoretical background of the performance measurements of network interconnect devices on the basis of the relevant Internet Engineering Task Force (IETF) Request for Comments (RFC) documents. Section III presents a short survey of related works. Section IV is an overview of the software used for our measurements. Section V discloses the relevant details of our measurement environments. In section VI, our measurement results are presented and analyzed. Section VII covers the discussion and our plans for future research. Section VIII concludes our paper.

II. THEORETICAL BACKGROUND

A. RFC 2544

The main purpose of the RFC 2544 [2] is to define how to measure the performance of network interconnect devices in an objective and repeatable way. The most important aspects include measurement setup, Device Under Test (DUT) setup, frame format and frames sizes, and testing duration.

The RFC lists three different measurement setups of which we used the first one.

The RFC recommends the usage of the following Ethernet frame sizes: 64, 128, 256, 512, 1024, 1280, 1518. It defines TCP/IP over Ethernet frame formats. They are: learning frame, routing update frame, management query frame and test frame. The test frame is used for different benchmarking tests like throughput, latency, frame loss rate, back-to-back frames, system recovery, and reset. From among them, throughput is the most essential one.

To measure the performance of routers, an IP address range was reserved, 198.18.0.0/15. The lower half of the range (198.18.0.0/16) was used on the left side of the configuration shown in Figure 1, and the upper half (198.19.0.0/16) on the right side. The interfaces of the DUT were assigned the IP address ending in 1 of the domains (198.18.0.1 and 198.19.0.1).

It should be noted that RFC 2544 requires testing with bidirectional traffic and it was used in all our measurements.

B. RFC 4814

RFC 4814 [3] covers several different topics. During our measurements, we used the pseudorandom port numbers, which are recommended in section 4.5 of the aforementioned RFC. The network cards in use today are capable of distributing interrupts caused by the packet arrivals to different CPU cores for processing. This is done using a hash function that takes the source and destination IP addresses, as well as the TCP or UDP

DOI: 10.36244/ICJ.2025.2.5

source and destination port numbers in incoming packets, as input parameters. This process is called Receive-Side Scaling (RSS) [4], which aims to achieve scalability. To support RSS, RFC 4814 recommends the usage of pseudorandom port numbers. The recommended ranges are:

- for source port numbers: 1024 – 65535
- for destination port numbers: 1 – 49151

C. RFC 5180

RFC 5180 [5] is highly similar to RFC 2544 in several ways, but while RFC 2544 focused on IPv4, RFC 5180 deals with IPv6. Similar to IPv4, an IPv6 address space was reserved for benchmarking. The reserved range is 2001:2::/48. As in RFC 2544, the address range should be halved.

III. RELATED WORK

As for peer-reviewed research papers about measuring IPv4 and IPv6 packet forwarding performance of FD.io VPP according to the current industry standards laid down by the RFCs mentioned above, we found only our own conference paper [6], which is extended in our current paper with further measurements.

However, several other recent research papers recommended FD.io VPP for IPv4 or IPv6 packet forwarding.

For example, Slavic and Krajnovic [7] proposed that open-source software and commodity hardware may replace the router vendors' products. They recommended the usage of FD.io VPP and some further software. However, they did not present any benchmarking measurements that were RFC 2544 or RFC 5180 compliant.

FD.io VPP was used as the packet forwarding solution of a network emulator called “CNNet” [8].

FD.io VPP is integrated with a custom control plane to provide a high-performance, low-cost software cloud gateway for accelerating virtual cloud networks [9]. Although certain performance data are included in the paper, the testing conditions are not mentioned. Their measurements were not RFC 2544 compliant because of the traffic generator used.

Another paper gives an important insight into the issue of why user-space solutions may outperform the interrupt-based ones [10].

IV. SOFTWARE USED FOR MEASUREMENTS

In this section, a brief summary is provided on DPDK, the Linux kernel, Non-Uniform Memory Access (NUMA), FD.io VPP, and siitperf.

A. DPDK

The Data Plane Development Kit (DPDK) [11] is an open-source software, with Linux based user platform, that was designed to improve packet processing speeds. DPDK enables the rapid development of high-speed data packet networking applications. DPDK achieves fast packet processing by consisting of libraries and drivers that bypass the operating system’s network stack. DPDK-based programs can send and receive approximately an order of magnitude more packets per CPU core than those using the Linux kernel [11].

B. Linux kernel

During packet forwarding, the Linux kernel processes incoming network packets and forwards them to the appropriate destination. In the Linux kernel, both scalar packet processing and RSS play an important role in packet delivery. During the scalar packet processing, the kernel individually processes the incoming packets. Just one packet is taken by an interrupt function (by default) from the network interface, then it works through a series of functions [1]. This method is simple, but its efficiency can be limited as it requires the same call chain to be executed for each packet. This can be time-consuming and place a strain on the processor and caches. With RSS, the kernel can handle heavy loads more efficiently and distribute packet processing across multiple CPUs in the system, increasing performance and reducing latency.

C. Non-Uniform Memory Access

NUMA is a multiprocessor system design where memory access time depends on the position of memory relative to the processor: each processor accesses its own local memory faster than the local memory of another processor [12]. Whereas NUMA is necessary to support scalability, our results show the consequences of its usage when a CPU core belongs to a different NUMA node than the Network Interface Card (NIC) it communicates with.

D. FD.io VPP

The Fast Data Project (FD.io) [1] introduced Vector Packet Processing (VPP) that can handle high performance traffic. It can be used on multiple platforms. Vector packet processing can receive multiple packets at once and pass this group, known as a packet vector, to the processing function, which then processes it, thereby saving time. The Packet Processing Graph (PPG) is at the heart of the FD.io VPP design. FD.io VPP collects a vector of packets from the RX rings, up to 256 packets in a single vector. The received packets are then traversed through the nodes of the PPG in the vector, with each graph node representing network processing that is applied to each packet. FD.io VPP can be used with or without DPDK. We used it with DPDK.

E. Siitperf

Siitperf [13] was running on our Tester server. The name of siitperf comes from the fact that it was originally designed to measure the performance of Stateless IP/ICMP Translation (SIIT) gateways. Due to its flexibility, it is also suitable for measuring the performance of IPv4 and IPv6 packet forwarders (routers). Siitperf uses DPDK to achieve a sufficiently high performance [13]. It should be noted that siitperf reports the results as packets per second *per direction*. When bidirectional traffic is used, the number of all frames forwarded is double the value reported.

As siitperf supports the throughput, latency, frame loss rate and packet delay variation measurement procedures of RFC 8219 [14], and the throughput measurement procedure of RFC 2544 was incorporated in RFC 5180 and then in RFC 8219 without any changes, siitperf could be used for measuring IPv4 and IPv6 throughput according to the requirements of RFC 2544 and RFC 5180, respectively.

Siitperf is a collection of binaries and bash shell scripts. The binaries execute an elementary step of the given measurement procedure. For example, the siitperf-tp binary performs a 60-second long throughput test, and the binary-rate-alg.sh script performs the binary search by calling the siitperf-tp binary and providing the appropriate parameter values as command line arguments. In contrast, other parameters that do not change during the consecutive steps of the binary search are read from the siitperf.conf configuration file.

As required by the throughput measurement procedures of RFC 2544 or RFC 5180, siitperf-tp sends bidirectional IPv4 or IPv6 traffic and counts the received frames. Based on the reported values, the shell script determines throughput, which is the highest frame rate at which the DUT can forward all test frames without loss.

Further details of the design, implementation, operation and performance of siitperf can be found in several research papers. The original design, which relied on fixed port numbers, was disclosed in [13]. The extension to support RFC 4814 pseudorandom port numbers was documented in [15]. The accuracy of siitperf was checked by comparing the IPv4 throughput of the same DUT determined by siitperf and the RFC 2544-compliant commercial Anritsu MP1590B Network Performance Tester [16]. The theory and practice of extending siitperf for stateful tests was published in [17]. Finally, support for pseudorandom IP addresses was added, as described in [18].

V. TEST ENVIRONMENT

A. The Structure of the Test Network

Two test systems were used. The first one consisted of Dell PowerEdge R620 servers. Each had two 6-core Intel Xeon E5-2620 processors and 32 GB 1600 MHz DDR3 SDRAM configured as 1333 MT/s. The second one contained Dell PowerEdge R730 servers. Each had two 8-core Intel Xeon E5-2667 v4 CPUs and 128 GB 2666 MHz DDR4 SDRAM configured as 2400 MT/s. We installed an Intel X540 10G/1G network interface card (NIC) in each of them, using the two 10 GbE ports for the measurements. The servers were directly connected with Cat6 UTP patch cables. The test setup is shown in Fig. 1.

To achieve stable measurement results, we switched off hyper-threading and set the CPU clock frequency of the servers to a fixed rate at their nominal clock frequency using the tlp Linux package, namely to 2 GHz and to 3.2 GHz for the R620 and the R730 servers, respectively.

As for drivers for the 10GbE ports of the X540 NIC, ixgbe and uio_pci_generic were used with the Linux kernel and with FD.io.VPP.

It should be noted that all servers used had two NUMA nodes, where the 10GbE network interfaces and the CPU cores with even serial numbers (core 0, core 2, core 4, etc.) belonged to NUMA node 0 and the CPU cores with odd serial numbers (core 1, core 3, core 5, etc.) belonged to NUMA node 1.

B. Performance of the Tester

As we did not use a commercial network performance tester to perform the measurements but instead employed our own software tester called siitperf, which ran on the same type of servers as the DUT, it was important to avoid the situation that the Tester could become a bottleneck. To achieve this, we conducted a loopback test: the two interfaces of the Tester were interconnected by a direct cable, leaving out the DUT, and a throughput test was performed. (This was called the "self-test of the Tester" in our previous papers about siitperf.)

This test was only performed with the R620 Tester using IPv4. The result was highly stable: 6.03 Mfps. The test was not repeated with IPv6 traffic because, according to our experience, it would not make a significant difference. (Please refer to Table 4 and Table 5 of [18].) We did not need to perform the test with the R730 server because we knew from our previous experiments that the X540 NIC formed the bottleneck, as it can do about 7.1-7.2 Mfps. (As already mentioned, these rates were measured with bidirectional traffic and are to be understood as *per direction* rates.)

C. Configuration of FD.io VPP

We followed the installation guide from the official FD.io webpage [1]. We installed the following packages: libvppinfra, vpp, vpp-plugin-core, vpp-plugin-dpdk. During FD.io VPP measurements, we assigned our interfaces to DPDK with the uio_pci_generic driver. The configuration of FD.io VPP was done with the following commands:

```
set interface ip address \
TenGigabitEthernet1/0/0 198.18.0.1/24
set interface ip address \
TenGigabitEthernet1/0/1 198.19.0.1/24
set interface state TenGigabitEthernet1/0/0 up
set interface state TenGigabitEthernet1/0/1 up
set ip neighbor TenGigabitEthernet1/0/0 \
198.18.0.2 24:6e:96:3b:fb:00
set ip neighbor TenGigabitEthernet1/0/1 \
198.19.0.2 24:6e:96:3b:fb:02
```

The latter two commands were necessary because siitperf cannot respond to ARP requests. As a result, we had to set the ARP table entries manually.

The IPv6 configuration was similar, using IPv6 addresses instead of the IPv4 addresses. In that case we set the NDP table entries similar to the ARP table entries.

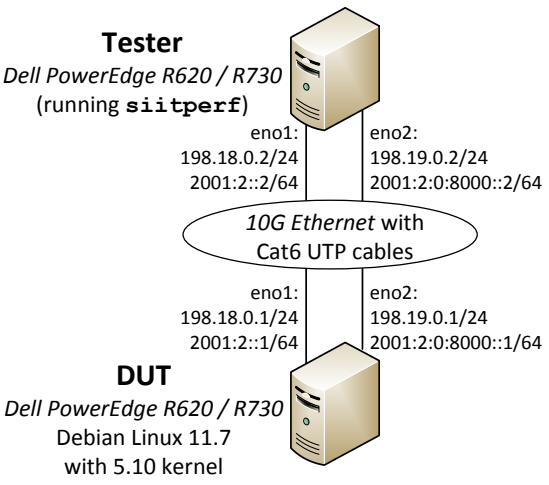

Fig. 1. Test setup for benchmarking FD.io VPP and the Linux kernel.

TABLE I
THROUGHPUT OF IPV4 PACKET FORWARDING OF THE LINUX KERNEL AS A FUNCTION OF THE NUMBER OF ACTIVE CPU CORES, E5-2620 @ 2 GHZ

Number of CPU cores	1	2	4	6	8	12
median (fps)	442,782	765,223	1,524,287	2,265,108	2,788,933	3,923,150
minimum (fps)	442,352	764,369	1,521,483	2,218,749	2,781,247	3,874,999
maximum (fps)	443,848	768,128	1,525,409	2,268,067	2,790,527	3,929,688
average (fps)	442,889	765,329	1,524,244	2,262,774	2,787,973	3,918,181
standard deviation	363.64	814.01	981.57	10,549.55	2,234.20	1,5163.08
dispersion (%)	0.34	0.49	0.26	2.18	0.33	1.39
relative to half as many cores	--	1.7282	1.9920	--	1.8297	1.7320
relative scale-up	1	0.8641	0.8606	0.8526	0.7873	0.7384

In VPP, for setting the number of CPU cores to be used, we had to modify the /etc/vpp/startup.conf configuration file. In the CPU segment, we changed the following settings:

```
main-core 2 # the main program runs on core 2
corelist-workers 4, 6 # 2 workers (on 4 and 6)
```

VI. MEASUREMENTS AND RESULTS

As for frame sizes, 64-byte and 84-byte test frames were used for IPv4 and IPv6 respectively. These are the smallest frame sizes allowed by siitperf. The rationale behind this choice was to make the CPU's processing capacity the bottleneck (limiting the maximum frame rate), rather than the packet transmission capacity of the network interface card.

It should be noted that since we used a general-purpose operating system, random events could occur during our measurements, potentially influencing the results. Therefore, each test was executed 20 times to achieve statistically reliable results.

As for summarizing function, both median and average was used. In the analysis, we primarily relied on median, because it is less sensitive to outliers than average.

To express the consistent or scattered nature of the results, we primarily relied on dispersion. It is defined by (1).

$$dispersion = \frac{maximum - minimim}{median} \cdot 100\% \qquad (1)$$

In addition, we also included the standard deviation.

As mentioned, in Section IV.E, siitperf reports the number of forwarded packets *per direction*. Therefore, our results should be interpreted accordingly (i.e., they should be multiplied by two to obtain the total number of forwarded packets per second).

A. Linux Kernel Packet Forwarding Performance of R620

The IPv4 and IPv6 packet forwarding performance of the Linux kernel was measured as a function of the number of active CPU cores. The number of active CPU cores was limited using the maxcpus=*n* kernel parameter, where *n* took the values of 1, 2, 4, 6, 8 and 12.

The IPv4 packet forwarding performance of the Linux kernel as a function of the number of active CPU cores is shown in Table I. At first glance, the results show that the performance of the DUT scaled up well with the increase of the number of active CPU cores. To facilitate a more detailed analysis of the results, the second-to-last line of the table shows the performance relative to having half as many active CPU cores, while the last line displays the relative scale-up, as defined by (2).

$$S(n) = P(n)/P(1)/n \qquad (2)$$

Where n is the number of active CPU cores, and P(n) is the performance measured (in frames per second) with n active CPU core. Its theoretical maximum value is 1.

A closer inspection of the results shows the following:

When using 2 CPU cores instead of 1 CPU core, the system performance did not double, but only increased by a factor of 1.7282. There are two main reasons for this phenomenon:

1. there is a performance cost of running a system with multiple cores compared to running a system with a single core,
2. CPU core 0 and the 10GbE Ethernet interfaces used for measurement belong to NUMA node 0, but CPU core 1 belongs to NUMA node 1, which means that CPU core 1 can only communicate with the network interface via core 0, and this communication overhead reduces the performance.

However, when we used 4 active CPU cores instead of 2, the performance doubled by a very good approximation (1.992 times). The explanation for this is very simple. The above factors were already present in the dual core system, so they did not cause any additional performance degradation in the quad core system.

TABLE II
THROUGHPUT OF IPV6 PACKET FORWARDING OF THE LINUX KERNEL AS A FUNCTION OF THE NUMBER OF ACTIVE CPU CORES, E5-2620 @ 2 GHZ

Number of CPU cores	1	2	4	6	8	12
median (fps)	425,782	732,299	1,466,585	2,176,369	2,658,915	3,766,926
minimum (fps)	424,951	726,561	1,374,999	1,999,999	2,617,186	3,761,709
maximum (fps)	426,026	733,467	1,466,974	2,178,284	2,660,414	3,772,095
average (fps)	425,628	731,888	1,461,745	2,156,176	2,656,771	3,766,972
standard deviation	291.61	1,481.93	20,459.50	54,645.21	9,443.81	2,679.58
dispersion (%)	0.25	0.94	6.27	8.19	1.63	0.28
relative to half as many cores	--	1.7199	2.0027	--	1.8130	1.7308
relative scale-up	1	0.8599	0.8611	0.8519	0.7806	0.7373

Further on, the performance of the 8-core system compared to a 4-core system is only 1.8297 times higher, and the comparison of performance of the 12-core system and the 6-core system shows only a 1.732 times increase. There may already be other reasons for this degraded increase, such as the fact that all CPU cores share the same network interfaces and, as their number increases, they are already interfering with each other to some extent by competing for the access to the network interfaces.

In conclusion, we observed that the performance of the system scaled up well: the throughput of the 12-core system instead the 1-core system increased from 442,782 fps to 3,923,150 fps, which means an 8.86 times increase.

The results of the throughput measurements for the IPv6 packet forwarding performance of the Linux kernel are shown in Table II. These results are essentially highly similar to those in Table I. There are two striking differences:

- The dispersion values are remarkably larger for 4 and 6 CPU cores. This is because one of the tests at 1,375,000 fps frame rate with 4 CPU cores and one of the tests at 2,000,000 fps frame rate with 6 CPU cores failed. These may be due to some relatively rare events in the system. The average is obviously affected by these outliers, which justifies our usage of median as summarizing function.
- The IPv6 throughput is slightly lower than the IPv4 throughput. There are two possible root causes for this: firstly, the frame size was larger for the IPv6 measurements, and secondly, the IPv6 addresses are four times as long as IPv4 addresses.

B. FD.io VPP Packet Forwarding Performance of R620

In order to measure the throughput of the FD.io VPP IPv4 and IPv6 packet forwarding, we used the CPU core 2 as the main core and 1 or 2 workers running on CPU cores with even or odd serial numbers to examine the performance of the test system and to gain insight into its scalability. When the CPU cores with even serial numbers were used with a single worker, we tested both core 4 and core 6 to check if there was a difference. Then they both were used with two workers. In the case of CPU cores with the odd serial numbers, the same was done with core 3, core 5, and finally, with cores 3 and 5.

At the time of our preliminary measurements using FD.io VPP with 2 workers, it was found that the Tester became the bottleneck, and thus we could not measure the true performance of the DUT. However, we considered it highly important to be able to measure the scalability of FD.io VPP at least up to two CPU cores. Therefore, the CPU clock frequency was set to 1.2 GHz (instead of 2 GHz, the nominal clock frequency of the CPU) to be able to perform the measurements using two workers.

The results of our throughput measurements of the IPv4 packet forwarding performance of FD.io VPP are shown in Table III. Core 4 and core 6 show nearly identical performance, with frame rates of 2,091,964 fps and 2,096,376 fps, respectively. When two workers were used across both cores the median frame rate roughly doubled to 4,187,904 fps. This indicates that the system scales efficiently, maximizing processing capacity without significant bottlenecks. The same can be said for the CPU cores with odd serial numbers. However, for those cores, the median value was more than 18% lower, which is clearly due to using the different NUMA node.

As for the quality of the results, with a single worker thread, the dispersion is always below 0.3%, so the results are highly stable. With two worker threads, the dispersion increased significantly, but still remained below 2%.

The results of our throughput measurements characterizing the performance of the FD.io VPP IPv6 packet forwarding are shown in Table IV. These results are basically very similar to the results in Table III. There is one visible difference: the IPv6 throughput is slightly lower than the IPv4 throughput. The possible reasons for this were explained in section A.

TABLE III
THROUGHPUT OF IPV4 PACKET FORWARDING OF THE FD.IO VPP AS A FUNCTION OF THE NUMBER AND INSTANCE OF THE ACTIVE CPU CORES, E5-2620 @ 1.2GHZ

Used CPU cores	**4th**	**6th**	**4th & 6th**	**3rd**	**5th**	**3rd & 5th**
Number of workers	**1**	**1**	**2**	**1**	**1**	**2**
median (fps)	2,091,964	2,096,376	4,187,904	1,700,940	1,697,002	3,432,319
minimum (fps)	2,089,688	2,093,309	4,183,576	1,699,461	1,695,153	3,390,624
maximum (fps)	2,093,880	2,099,060	4,238,282	1,702,287	1,698,181	3,448,609
average (fps)	2,091,944	2,096,289	4,191,721	1,700,981	1,696,889	3,431,086
standard deviation	1,501.18	1,721.36	12,034.38	849.40	726.75	13,274.60
dispersion (%)	0.20	0.27	1.31	0.17	0.18	1.69

TABLE IV
THROUGHPUT OF IPV6 PACKET FORWARDING OF THE FD.IO VPP AS A FUNCTION OF THE NUMBER AND INSTANCE OF THE ACTIVE CPU CORES, E5-2620 @ 1.2GHZ

Used CPU cores	**4th**	**6th**	**4th & 6th**	**3rd**	**5th**	**3rd & 5th**
Number of workers	**1**	**1**	**2**	**1**	**1**	**2**
median (fps)	1,919,142	1,926,082	3,876,009	1,526,365	1,587,369	3,188,186
minimum (fps)	1,916,951	1,923,811	3,866,209	1,517,536	1,585,936	3,124,999
maximum (fps)	1,921,052	1,928,529	3,908,204	1,528,194	1,588,904	3,195,313
average (fps)	1,918,898	1,926,321	3,878,199	1,525,300	1,587,528	3,185,315
standard deviation	1,120.34	1,119.19	10,495.01	3,163.36	746.77	14,657.38
dispersion (%)	0.21	0.24	1.08	0.70	0.19	2.21

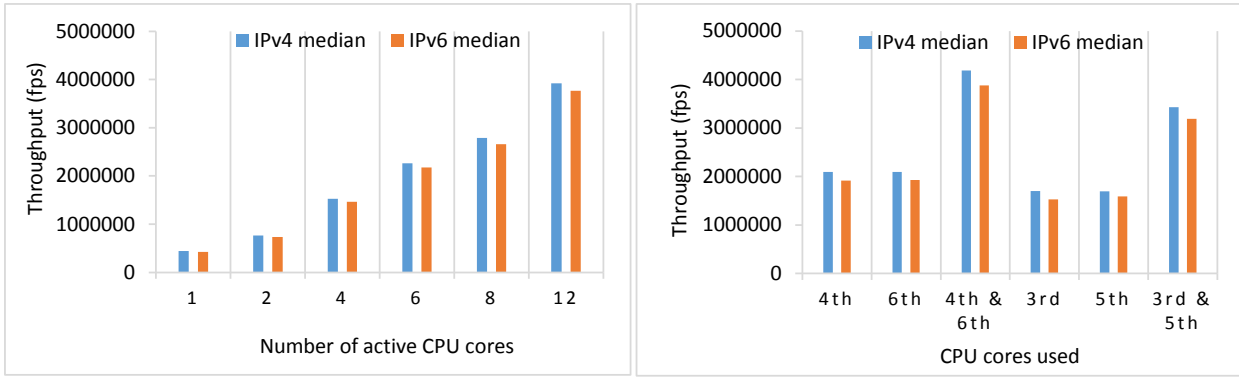

Fig. 2. Performance and scalability comparison of the Linux kernel using 1-12 active CPU cores (2GHz, left side) and FD.io VPP using one or two workers executed by the specified CPU cores (1.2GHz, right side) of a Dell PowerEdge R620 server with two 6-core Intel Xeon E5-2620 processors.

C. Comparison of the Performance of the Linux kernel and FD.io VPP using an R620 Server as DUT

The performance and scalability of the Linux kernel and FD.io VPP using a Dell PowerEdge R620 as the DUT is compared in Fig. 2. When considering their results, it is important to note that while the performance of the Linux kernel was measured at 2 GHz, the performance of FD.io VPP was measured at 1.2 GHz. Despite this disadvantage, FD.io VPP seriously outperformed the Linux kernel. While the Linux kernel on 1 CPU core delivered 442,782 fps IPv4 packet forwarding and 425,782 fps IPv6 packet forwarding performance, the FD.io VPP on 1 worker thread delivered more than 2 million IPv4 packets and more than 1.9 million IPv6 packets. Even when using CPU cores belonging to a different NUMA node than the NIC, the performance was still around 1.7 Mfps and 1.5 Mfps, for IPv4 and IPv6, respectively.

When the Linux system used all 12 CPU cores at the nominal 2 GHz clock frequency of the CPU, the performance was still only 3.92 Mfps and 3.77 Mfps for IPv4 and IPv6 packet forwarding, respectively. In contrast, FD.io VPP, using only 2 CPU cores (4 and 6), achieved more than 4.18 Mfps and 3.87 Mfps performance for IPv4 and IPv6 packet forwarding, respectively, at a clock frequency of 1.2 GHz. Our results prove that FD.io VPP is indeed a high-performance solution for IP packet forwarding.

D. Linux Kernel Packet Forwarding Performance on R730

The appropriate settings have been made using the methods described above. The same measurement parameters were used.

Each R730 server had 16 CPU cores, therefore we were able to set the active core numbers by the power of two (1, 2, 4, 8, 16). Our tests were performed using both 3.2 GHz and 1.2 GHz as the CPU clock frequency.

The IPv4 packet forwarding performance of the Linux kernel as a function of the number of active CPU cores at 3.2 GHz is shown in Table V. Using 2 CPU cores instead of 1 did not double system performance; it increased by only 1.7387 times. This phenomenon was discussed earlier. When we used 4 active CPU cores instead of 2, the performance doubled by a very good approximation (1.9293 times). The performance improvement of an 8-core system compared to 4 cores is now 1.9414 times. However, the performance increase at 16 cores was only 1.3374 times compared to that of an 8-core system. We contend that the main reason for this degradation was the insufficient performance of the NIC. To prove this, we repeated the tests by setting the CPU clock frequency to 1.2 GHz (the lowest possible value).

TABLE V
THROUGHPUT OF IPV4 PACKET FORWARDING OF THE LINUX KERNEL AS A FUNCTION OF THE NUMBER OF ACTIVE CPU CORES, E5-2667 V4 @ 3.2 GHZ

Number of CPU cores	**1**	**2**	**4**	**8**	**16**
median (fps)	746,706	1,298,271	2,504,729	4,862,591	6,503,402
minimum (fps)	744,141	1,249,991	2,499,022	4,808,590	6,492,093
maximum (fps)	750,001	1,301,758	2,509,173	4,880,386	6,516,618
average (fps)	746,678	1,295,252	2,503,804	4,859,680	6,504,297
standard deviation	1,239	10,875	3,477	21,168	7,033
dispersion (%)	0.78	3.99	0.41	1.48	0.38
relative to half as many cores	--	1.7387	1.9293	1.9414	1.3374
relative scale-up	1	0.8693	0.8386	0.8140	0.5443

TABLE VI
THROUGHPUT OF IPV4 PACKET FORWARDING OF THE LINUX KERNEL AS A FUNCTION OF THE NUMBER OF ACTIVE CPU CORES, E5-2667 V4 @ 1.2 GHZ

Number of CPU cores	**1**	**2**	**4**	**8**	**16**
median (fps)	299,758	552,915	1,093,546	2,169,302	4,046,615
minimum (fps)	298,741	548,765	1,085,934	2,160,155	4,030,760
maximum (fps)	300,659	554,762	1,095,856	2,172,245	4,056,641
average (fps)	299,694	552,454	1,093,201	2,168,249	4,045,039
standard deviation	622	1,954	1,988	3,546	6,120
dispersion (%)	0.64	1.08	0.91	0.56	0.64
relative to half as many cores	--	1.8445	1.9778	1.9837	1.8654
relative scale-up	1	0.9223	0.9120	0.9046	0.8437

TABLE VII
THROUGHPUT OF IPV6 PACKET FORWARDING OF THE LINUX KERNEL AS A FUNCTION OF THE NUMBER OF ACTIVE CPU CORES, E5-2667 V4 @ 3.2 GHZ

Number of CPU cores	1	2	4	8	16
median (fps)	728,805	1,144,471	2,233,474	4,420,087	6,505,859
minimum (fps)	718,749	1,140,602	2,124,968	4,374,999	6,374,999
maximum (fps)	730,621	1,148,438	2,239,532	4,437,501	6,562,577
average (fps)	728,192	1,143,963	2,226,907	4,417,140	6,505,697
standard deviation	2,588	2,104	2,4466	1,4143	4,0059
dispersion (%)	1,63	0,68	5,13	1,41	2,88
relative to half as many cores	--	1.5703	1.9515	1.9790	1.4719
relative scale-up	1	0.7852	0.7661	0.7581	0.5579

TABLE VIII
THROUGHPUT OF IPV6 PACKET FORWARDING OF THE LINUX KERNEL AS A FUNCTION OF THE NUMBER OF ACTIVE CPU CORES, E5-2667 V4 @ 1.2 GHZ

Number of CPU cores	1	2	4	8	16
median (fps)	288,704	511,763	1,017,832	1,990,212	3,861,507
minimum (fps)	281,249	499,999	1,007,688	1,937,483	3,749,999
maximum (fps)	289,093	512,894	1,019,907	1,996,223	3,875,001
average (fps)	288,348	511,299	1,016,904	1,988,055	3,856,157
standard deviation	1,685	2,693	3,329	12,125	25,856
dispersion (%)	2.72	2.52	1.20	2.95	3.24
relative to half as many cores	--	1.7726	1.9889	1.9553	1.9402
relative scale-up	1	0.8863	0.8814	0.8617	0.8360

The IPv4 packet forwarding performance of the Linux kernel as a function of the number of active CPU cores using 1.2 GHz CPU clock frequency is shown in Table VI. The results show that the performance of the DUT scaled up well with the increase in the number of active CPU cores in the entire range. With a 16-core system, we can observe a 1.8654 times increase compared to 8 cores. On one hand, this is certainly an improvement (especially compared to 1.3374); however, it is still lower than the increases seen when scaling from 2 to 4 or from 4 to 8 cores in the same test series, despite the NIC capacity being higher than the measured throughput. We attribute this small degradation to the fact that the 16 cores were competing to access the NIC. Overall, the system performance scaled up well: when using 16 CPU cores instead of 1 CPU core, the throughput increased from 299,758 fps to 4,046,615 fps by a factor of 13.5.

The IPv6 packet forwarding performance of the Linux kernel as a function of the number of active CPU cores using 3.2 GHz CPU clock frequency is shown in Table VII. These results are basically very similar to those in Table V with the difference that IPv6 throughput is slightly lower than IPv4 throughput.

Although IPv6 throughput is generally slightly lower than IPv4, the median values for 16 CPU cores are very similar (6,503,402 fps for IPv6 and 6,505,859 fps for IPv4). This suggests a hardware limitation, likely the network interface, in this case.

The IPv6 packet forwarding performance of the Linux kernel as a function of the number of active CPU cores using 1.2 GHz CPU clock frequency is shown in Table VIII. Overall, the system performance scaled well: when using 16 CPU cores instead of 1 CPU core, the throughput increased from 288,704 fps to 3,861,507 fps by a factor of 13.4.

E. FD.io VPP Packet Forwarding Performance on R730

The appropriate settings have been made using the methods described above. The measurement parameters used so far were also used for these measurements.

In order to measure the throughput of the FD.io VPP IPv4 and IPv6 packet forwarding, we used the CPU core 2 as the main core and 1 or 2 workers running on CPU cores with even serial numbers to examine the performance of the test system and to gain insight into its scalability. Previously, we also used CPU cores with odd serial numbers to examine the test system's performance. However, as discussed, there were no relevant differences in the results, aside from being lower due to their association with a different NUMA node than the NIC. This time, we focused on providing a clear comparison between different clock speeds (3.2 GHz and 1.2 GHz) and omitted the use of CPU cores with odd serial numbers.

The results of our throughput measurements of the IPv4 packet forwarding performance of FD.io VPP are shown in Table IX. Examining core 4 and core 6 at 3.2 GHz, we see that there is no significant difference in performance (6,947,546 fps and 6,960,875 fps) but compared to the results measured at 1.2 GHz (2,886,634 fps and 2,887,674 fps), there is a nearly 2.5-fold increase. For two workers, we can say that our results approximately doubled when comparing the single worker and the two worker results at 1.2 GHz.

As for the quality of the results, they are highly stable and consistent because all of the dispersions are below 1%.

The results of the throughput measurements of the IPv6 packet forwarding performance of FD.io VPP are shown in Table X. These results are basically very similar to the results in Table IX with the difference that IPv6 throughput is slightly lower than IPv4 throughput. Overall, we have highly stable results because all of the dispersions are below 1%.

It is salient that the results with FD.io VPP are more stable than those with the Linux kernel. The reason behind this is the following: during FD.io VPP measurements, the CPUs used for executing the workers were isolated (using the **`isolcpus`** kernel command line parameter). This means that no other task could be scheduled to the isolated CPU cores by the kernel. Conversely, the Linux kernel used all CPU cores for packet forwarding and the scheduler occasionally assigned them other tasks, as well.

TABLE IX
THROUGHPUT OF IPV4 PACKET FORWARDING OF THE FD.IO VPP AS A FUNCTION OF THE NUMBER AND INSTANCE OF THE ACTIVE CPU CORES, E5-2667 V4

Used CPU cores	**4th**	**6th**	**4th**	**6th**	**4th & 6th**
Number of workers	**1**	**1**	**1**	**1**	**2**
CPU clock frequency	**3.2 GHz**	**3.2GHz**	**1.2 GHz**	**1.2 GHz**	**1.2 GHz**
median (fps)	6,947,546	6,960,875	2,886,634	2,887,674	5,866,613
minimum (fps)	6,942,869	6,937,499	2,874,999	2,885,736	5,859,374
maximum (fps)	6,953,156	6,968,751	2,887,939	2,888,488	5,869,027
average (fps)	6,948,498	6,959,671	2,886,116	2,887,500	5,866,099
standard deviation	3,077	6,674	2,906	747	2,680
dispersion (%)	0.15	0.45	0.45	0.10	0.16

TABLE X
THROUGHPUT OF IPV6 PACKET FORWARDING OF THE FD.IO VPP AS A FUNCTION OF THE NUMBER AND INSTANCE OF THE ACTIVE CPU CORES, E5-2667 V4

Used CPU cores	**4th**	**6th**	**4th**	**6th**	**4th & 6th**
Number of workers	**1**	**1**	**1**	**1**	**2**
CPU clock frequency	**3.2 GHz**	**3.2GHz**	**1.2 GHz**	**1.2 GHz**	**1.2 GHz**
median (fps)	6,250,366	6,253,601	2,559,248	2,555,627	5,215,632
minimum (fps)	6,218,749	6,240,721	2,546,874	2,554,662	5,187,499
maximum (fps)	6,258,057	6,264,577	2,559,815	2,556,732	5,220,337
average (fps)	6,251,035	6,252,592	2,558,535	2,555,526	5,213,860
standard deviation	3,195	4,308	3,023	666	3,670
dispersion (%)	0.63	0.38	0.51	0.08	0.63

F. Comparison of the performance of the Linux kernel and FD.io VPP using an R730 Server as DUT

Unlike in the case of the R620 test system, the results of the Linux kernel and the FD.io VPP using the same CPU clock frequency on the R730 test system are directly comparable.

First, we compare the results produced using 1.2 GHz CPU clock frequency. The results are shown in Fig. 3. Whereas the Linux kernel on 1 CPU core delivered 299,758 fps IPv4 packet forwarding and 288,704 fps IPv6 packet forwarding performance, the FD.io VPP with 1 worker thread delivered more than 2.8 million IPv4 packets and more than 2.5 million IPv6 packets. When the Linux system used all 16 CPU cores of the CPU, the performance was only 4.05 Mfps and 3.86 Mfps for IPv4 and IPv6 packet forwarding, respectively. In contrast, FD.io VPP, using only 2 CPU cores (4 and 6), achieved 5.87 Mfps and 5.22 Mfps performance for IPv4 and IPv6 packet forwarding, respectively.

The 3.2 GHz results are compared in Fig. 4. The single core system of Linux kernel delivered 746,706 fps IPv4 packet forwarding and 728,805 fps IPv6 packet forwarding performance, whereas the FD.io VPP on 1 worker thread performance, whereas the FD.io VPP on 1 worker thread delivered more than 6.95 million IPv4 packets and more than 6.25 million IPv6 packets. Our results prove that FD.io VPP is indeed a high-performance solution for IP packet forwarding.

G. Comparison of the results of the R620 and R730 Servers

When comparing the two types of servers, it is important to note that the nominal CPU clock frequencies of the R620 and R730 servers are 2 GHz and 3.2 GHz, respectively.

When comparing the median values of their Linux kernel IPv4 packet forwarding results measured at their nominal CPU frequencies, a single CPU core of the R730 server outperformed the single CPU core of the R620 server with a factor of 1.6864 (746,706 fps vs. 442,782 fps) and similar statements can be made regarding their performances from 2 to 8 CPU cores. Alternatively, it can be noted that the performance of a single core of the E5-2667 v4 CPU at 3.2 GHz (746,706 fps) is nearly equivalent to the performance of two cores of the E5-2620 CPU at 2 GHz (765,223 fps).

As for the FD.io VPP results of the two types of CPUs, those measured at 1.2 GHz are directly comparable. The 2.89 Mfps

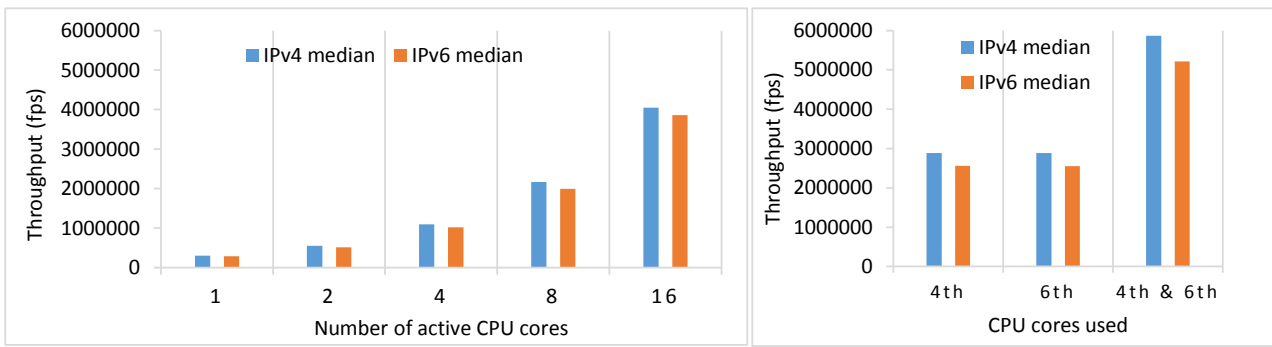

Fig. 3. Performance and scalability comparison of the Linux kernel using 1-16 active CPU cores (left side) and FD.io VPP using one or two workers executed by the specified CPU cores (right side) of a Dell PowerEdge R730 server with two 8-core Intel Xeon E5-2667 v4 processors @ 1.2GHz.

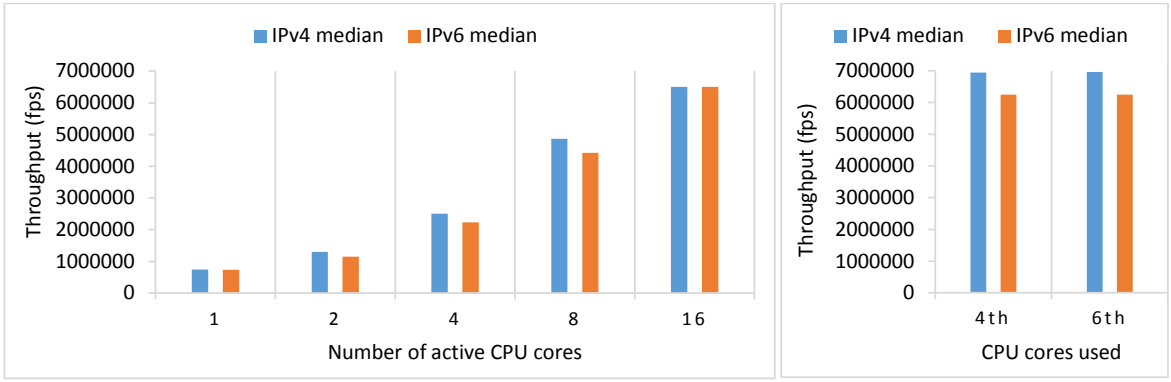

Fig. 4. Performance and scalability comparison of the Linux kernel using 1-16 active CPU cores (left side) and FD.io VPP using one worker executed by the specified CPU cores (right side) of a Dell PowerEdge R730 server with two 8-core Intel Xeon E5-2667 v4 processors @ 3.2GHz.

single core IPv4 throughput of the E5-2667 v4 CPU @ 1.2 GHz shows a 1.38-fold increase compared to the 2.09 Mfps single core IPv4 throughput of the E5-2620 CPU @ 1.2 GHz. The higher nominal clock frequency of the newer CPU further amplifies this difference.

VII. Discussion and Future Research

In our research group's previous benchmarking effort, the number of CPU cores in the test system was equal to powers of 2 (see [19]). This approach allowed us to consistently double the number of active CPU cores in each iteration and fully utilize all available CPU cores in the final test. During our first attempt to compare the performance of the Linux kernel and FD.io VPP, only servers with 12 CPU cores were available for the task. For this reason, we also conducted tests using 6 cores to provide a basis for comparison with the tests with 12 cores. Building on the success of our initial effort [6], we decided to repeat our experiments with 16-core servers, too.

Our current measurements only included throughput tests. In the future, we also plan to examine the latency. Since latency is measured at the frame rate previously determined by the throughput tests, our hypothesis is that FD.io VPP will exhibit higher latency. This is due to its vector packet processing mechanism, where a substantial number of frames are first accumulated into a vector before being processed together as they traverse the nodes of the packet processing graph. The latency results will complement the throughput results to give a more comprehensive view of the performance of the tested IP packet forwarding solutions.

Since both FD.io VPP and siitperf use DPDK, and siitperf only uses one core for sending and one core for receiving in each direction, we could evaluate the performance of FD.io VPP only up to two working threads. This remains the case even when using the trick of lowering the CPU clock frequency of the DUT. To be able to work with a significantly higher number of workers (4, 8, etc.) a more powerful Tester would be needed. The long-term plan of our research group includes the building of an FPGA-based tester that implements the functionalities of siitperf.

Another interesting research direction could be to evaluate the performance of the Linux kernel when using NAPI polling mode [20].

It would also be worth comparing the performance of FD.io VPP to that of Open vSwitch with DPDK (OvS DPDK) and eXpress Data Path (XDP).

Our results will encourage network operators to use FD.io VPP in production networks for IPv4 and IPv6 packet forwarding. Exhibiting high performance when running on commodity servers and being free software, FD.io VPP can be a good alternative to commercial routers. However, high performance and low cost are only two aspects. Network operators must consider at least two other ones: security and support. FD.io VPP runs on Linux, a general-purpose and widely used operating system. It is crucial to ensure the secure operation of the host machine. To that end, it must be carefully configured, and security updates must be regularly installed. Major version upgrades to the operating system could involve issues when DPDK and FD.io versions are upgraded. Fortunately, Debian and Ubuntu have long-term support (LTS) versions; thus, they can be operated for several years without changing the major version. Router vendors also provide support. As free software is basically provided “as is” without support, network operators either need to employ experts or buy support from a company that employs experts in Linux, DPDK and FD.io VPP.

VIII. Conclusion

We measured the performance of IPv4 and IPv6 packet forwarding of the Linux kernel and FD.io VPP. Regardless of the IP version, the Linux kernel’s packet forwarding performance showed a good scale-up with the increasing number of CPU cores, although minor performance variations were observed when utilizing different NUMA nodes. In contrast, FD.io VPP tests demonstrated exceptionally high performance with perfect scalability from 1 to 2 workers.

When a Dell PowerEdge R620 server with two 6-core E5-2620 CPUs was used as the DUT, FD.io VPP, using 2 workers executed by two CPU cores running at 1.2 GHz, outperformed

the Linux kernel, using 12 CPU cores running at 2 GHz.

When a Dell PowerEdge R730 server with two 8-core E5-2667 v4 CPUs was used as the DUT, the same two CPU clock speeds were used for testing both solutions. At 1.2 GHz CPU clock frequency, the Linux kernel on 1 CPU core delivered 299,758 fps IPv4 packet forwarding and 288,704 fps IPv6 packet forwarding performance, and the FD.io VPP with 1 worker thread delivered more than 2.8 million IPv4 packets and more than 2.5 million IPv6 packets. When the Linux system used all 16 CPU cores of the CPU, the performance was only 4.05 Mfps and 3.86 Mfps for IPv4 and IPv6 packet forwarding, respectively. In contrast, FD.io VPP, using only 2 CPU cores (4 and 6), achieved 5.87 Mfps and 5.22 Mfps performance for IPv4 and IPv6 packet forwarding, respectively. At a CPU clock frequency of 3.2 GHz, the Linux kernel's single-core system achieved an IPv4 packet forwarding performance of 746,706 fps and an IPv6 performance of 728,805 fps. In comparison, FD.io VPP, using a single worker thread, delivered over 6.95 million IPv4 packets and more than 6.25 million IPv6 packets. Our results confirm that FD.io VPP is indeed a high-performance solution for IP packet forwarding.

ACKNOWLEDGEMENT

The authors thank Bertalan Kovács for reviewing and commenting on the manuscript.

The authors thank Natasha Bailey-Borbély, Széchenyi István University, for the English language proofreading of the manuscript.

REFERENCES

[1] FD.io "What is the Vector Packet Processor (VPP)", VPP official website, https://s3-docs.fd.io/vpp/24.06/

[2] S. Bradner, and J. McQuaid, "Benchmarking methodology for network interconnect devices", IETF RFC 2544, 1999. DOI: 10.17487/RFC2544.

[3] D. Newman, T. Player, "Hash and stuffing: Overlooked factors in network device benchmarking", IETF RFC 4814, 2008. DOI: 10.17487/RFC4814.

[4] T. Herbert, W. de Bruijn, "Scaling in the Linux Networking Stack", https://www.kernel.org/doc/Documentation/networking/scaling.txt

[5] C. Popoviciu, A. Hamza, G. V. de Velde, and D. Dugatkin, "IPv6 benchmarking methodology for network interconnect devices", IETF RFC 5180, 2008, DOI: 10.17487/RFC5180.

[6] M. Kosák, and G. Lencse, "Performance comparison of IP packet forwarding solutions", *2024 47th International Conference on Telecommunications and Signal Processing (TSP)*, Virtual Conference, July 10-12, 2024, pp. 243–248. DOI: 10.1109/TSP63128.2024.10605773

[7] G. Slavic, N. Krajnovic, "Practical implementation of the vector packet processing software router", *2024 32nd Telecommunications Forum (TELFOR)*, Crowne Plaza, Belgrade, Serbia, November 26-27, 2024, DOI: 10.1109/TELFOR63250.2024.10819057

[8] Y. Peng, Y. Xiao, J. Duan, X. Zhang and W. Li, "Emulating high-performance networks with CNNet", *2023 IEEE 43rd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Hong Kong, Hong Kong, 2023, pp. 91–96, DOI: 10.1109/ICDCSW60045.2023.00024.

[9] S. Han, S. Wang, Y. Huang, T. Huang, Y. Liu, "High-performance and low-cost VPP gateway for virtual cloud networks" *2022 IEEE Global Communications Conference (GLOBECOM)*, pp. 4419–4424, DOI: 10.1109/GLOBECOM48099.2022.10001162

[10] P. Cai, M. Karsten, "Kernel vs. user-level networking: Don't throw out the stack with the interrupts", *ACM SIGMETRICS Performance Evaluation Review*, vol. 52, no. 1, June 13, 2024, pp. 43–44, DOI: 10.1145/3673660.3655061

[11] DPDK official website, https://www.dpdk.org/about/

[12] B. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann Publishers, 2008. DOI: 10.1016/B978-0-12-379751-3.X5001-2.

[13] G. Lencse, "Design and implementation of a software tester for benchmarking stateless NAT64 gateways", *IEICE Transactions on Communications*, vol. E104-B, no. 2, pp. 128–140, February 2021, DOI: 10.1587/transcom.2019EBN0010.

[14] M. Georgescu, L. Pislaru, and G. Lencse, "Benchmarking methodology for IPv6 transition technologies", IETF RFC 8219, August 2017, DOI: 10.17487/RFC8219.

[15] G. Lencse, "Adding RFC 4814 random port feature to siitperf: Design, implementation and performance estimation", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 9, no. 3, 2020, DOI: 10.11601/ijates.v9i3.291.

[16] G. Lencse, "Checking the accuracy of siitperf", *Infocommunications Journal*, vol. 13, no. 2, pp. 2–9, June 2021, DOI: 10.36244/ICJ.2021.2.1

[17] G. Lencse, "Design and implementation of a software tester for benchmarking stateful NATxy gateways: Theory and practice of extending siitperf for stateful tests", *Computer Communications*, vol. 172, no. 1, pp. 75–88, Aug. 1, 2022, DOI: 10.1016/j.comcom.2022.05.028.

[18] G. Lencse, "Making stateless and stateful network performance measurements unbiased", *Computer Communications*, vol. 225, September 2024, pp. 141–155, DOI: 10.1016/j.comcom.2024.05.018

[19] G. Lencse, and Á. Bazsó, "Benchmarking methodology for IPv4aaS technologies: Comparison of the scalability of the Jool implementation of 464XLAT and MAP-T", *Computer Communications*, vol. 219, April 2024, pp. 243–258, DOI: 10.1016/j.comcom.2024.03.007

[20] J. Sahoo, "Deep dive into NAPI: Optimizing network performance in the Linux kernel", March 3, 2024, https://www.linkedin.com/pulse/deep-dive-napi-optimizing-network-performance-linux-kernel-sahoo-opa1c/

Melinda Kosák received her BSc in electrical engineering from Széchenyi István University, Győr, Hungary in 2024.

She has been a member of the Cybersecurity and Network Technologies Research Group of the Faculty of Mechanical Engineering, Informatics and Electrical Engineering of Széchenyi István University, Győr, Hungary since 2022. Her research interests include performance analysis of FD.io VPP. She is now continuing her studies at Széchenyi István University as an MSc student in electrical engineering.

Gábor Lencse received his MSc and PhD degrees in computer science from the Budapest University of Technology and Economics, Budapest, Hungary in 1994 and 2001, respectively.

He has been working for the Department of Telecommunications, Széchenyi István University, Győr, Hungary since 1997 and has attained the rank of Professor. He has also been a part-time Senior Research Fellow at the Department of Networked Systems and Services, Budapest University of Technology and Economics since 2005. His research interests include the performance and security analysis of IPv6 transition technologies. He is a co-author of RFC 8219 and RFC 9313.