Industry Paper: On the Performance of Commodity Hardware for Low Latency and Low Jitter Packet Processing

Industry Paper: On the Performance of Commodity Hardware for Low Latency and Low Jitter Packet Processing

Charalampos Stylianopoulos, Magnus Almgren
Olaf Landsiedel, Marina Papatriantafilou
Chalmers University of Technology
Gothenburg, Sweden
{chasty,magnus.almgren,olafl,ptrianta}@chalmers.se

Trevor Neish, Linus Gillander
Bengt Johansson, Staffan Bonnier
Ericsson
Gothenburg, Sweden
{trevor.neish,linus.gillander}.ericsson.com
{staffan.bonnier,bengt.e.johansson}.ericsson.com

ABSTRACT

With the introduction of Virtual Network Functions (VNF), network processing is no longer done solely on special purpose hardware. Instead, deploying network functions on commodity servers increases flexibility and has been proven effective for many network applications. However, new industrial applications and the Internet of Things (IoT) call for event-based systems and midleware that can deliver ultra-low and predictable latency, which present a challenge for the packet processing infrastructure they are deployed on.

In this industry experience paper, we take a hands-on look on the performance of network functions on commodity servers to determine the feasibility of using them in existing and future latency-critical event-based applications. We identify sources of significant latency (delays in packet processing and forwarding) and jitter (variation in latency) and we propose application- and system-level improvements for removing or keeping them within required limits. Our results show that network functions that are highly optimized for throughput perform sub-optimally under the very different requirements set by latency-critical applications, compared to latency-optimized versions that have up to 9.8X lower latency. We also show that hardware-aware, system-level configurations, such as disabling frequency scaling technologies, greatly reduce jitter by up 2.4X and lead to more predictable latency.

CCS CONCEPTS

• **Networks** → **Network measurement**.

KEYWORDS

packet processing, latency, jitter, Industry 4.0

ACM Reference Format:
Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, Marina Papatriantafilou, Trevor Neish, Linus Gillander, and Bengt Johansson, Staffan Bonnier. 2020. Industry Paper: On the Performance of Commodity Hardware for Low Latency and Low Jitter Packet Processing. In *The 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20), July 13–17, 2020, Virtual Event, QC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3401025.3403591

Olaf Landsiedel is also with Kiel University, Germany.

DEBS '20, July 13–17, 2020, Virtual Event, QC, Canada

ACM ISBN 978-1-4503-8028-7/20/07…$15.00
https://doi.org/10.1145/3401025.3403591

1 INTRODUCTION

High-speed networks are a key part of today's connected world. As the number of connected devices increases [7] and new event-based applications with strict performance requirements (e.g., those related to Industry 4.0 [10] that combines factory automation with communication technologies) become common, next-generation networks face new challenges with respect to scaling and meeting those requirements. In particular, in network architectures, the user-plane[1] [13] i.e. the component that is responsible for carrying and processing network traffic, needs to support reliable connectivity with minimum added delays (latency).

Over the last years, network technologies have shifted from specialized hardware platforms to a user-plane that can be deployed on commodity, off-the-shelf hardware, either natively or using virtualization and container technologies [19]. This shift allows flexibility in the design and deployment of network functions to support a variety of event-driven applications, reduces deployment cost and enables horizontal scaling. It also allows the deployment of Virtual Network Functions (VNF) where packet processing such as switching, firewalls and intrusion detection systems, is decoupled from the hardware that it is deployed on.

While the aforementioned shift to commodity hardware and VNF has been proven successful, e.g., for mobile broadband applications, such network technologies have not yet been put to use in more demanding applications such as Industry 4.0. In particular, one of the requirements of such applications is low latency. In upcoming event-based industrial applications, such as factory automation, machine-to-machine communication and autonomous driving, the network infrastructure is required to deliver low end-to-end latency (less than 10ms in some cases). Another, often overlooked requirement is that variation in latency (jitter) must be kept to a minimum. High jitter might lead to interruptions in service, e.g., in an automated production line where co-operating machinery might experience differences in latency and desynchronize.

The challenge to meet these requirements is particularly pressing for mobile networks that will play a central role in future Industry 4.0 deployments. In particular, the evolution of the core packet processing network, i.e. Evolved Packet Core (EPC) [17],

[1]The term data-plane is also used in the literature, we use the term user-plane throughout the paper.

needs to enable future mobile networks to meet the required performance.The packet processing involved in the user-plane of EPC involves many event-based network functions such as Deep Packet Inspection [22] and firewalls, but the bare minimum functionality is packet switching, which we focus on in this paper.

When considering software-based packet processing in general purpose servers, there are several factors that can contribute to high latency. They range from operating system interrupts or scheduling and timesharing with other tasks. Moreover, the packet processing application itself needs to be optimized to make best use of the underlying hardware [23] and focus on delivering low latency. Hence, the performance and feasibility of using commodity hardware for latency-critical event-based applications is still an open issue.

In this industrial experience report, we establish a baseline for the latency and jitter performance of software-based packet processing on commodity hardware. Our goal is to determine the feasibility of using software-based packet processing on commodity hardware for challenging and latency-critical event-based applications in Industry 4.0. As a starting point, we focus on the bare minimum functionality that the user-plane performs, namely packet forwarding. We design and perform experiments to identify sources of unnecessary latency and jitter and we propose ways to remove them, through optimizations in the application itself and at system level. Specifically, our evaluation shows that:

- In order to come close to the reliable and low latency network requirements of event-based applications, packet processing must be optimized to avoid buffering packets as much as possible, even if that comes at the cost of a reduced sustained throughput.
- System-level, hardware-aware configurations, such as disabling frequency scaling, can have a noticeable effect on latency and jitter.

The rest of the paper is organized as follows. Section 2 gives the required background on low-latency packet processing. In Section 3 we identify sources of latency and jitter and show how to mitigate them. We present our experimental methodology in Section 4 and the results from the experiments in Section 5. We discuss those results in connection with application requirements in Section 6. We present related work in Section 7 and conclude in Section 8.

2 PRELIMINARIES

In this section, we summarize relevant information on the requirements of Industry 4.0 applications, the packet processing involved in the packet core, as well as a brief background on user-space networking.

2.1 Ultra Reliable Low Latency Communication Requirements

Ultra Reliable Low Latency Communication (URLLC) is a collection of services supported by the upcoming fifth generation networks (5G), designed to address the needs of latency-critical event-based applications, mainly related to machine-to-machine communication and industrial applications [20].

The basic requirements and characteristics of URLLC services that relate to the network infrastructure they are deployed on are the following: (a) *Low latency*. While there is a plethora of studies mentioning different latency requirements for the same application, typical latency requirements range from 1 msec to 50 msec end-to-end latency. Note that this requirement is about the maximum guaranteed latency and not the average. (b) *High reliability*. Most use cases require a highly reliable network infrastructure that must deliver packets with 99.99% to 99.999% reliability. (c) *Low jitter*. On many industrial applications, jitter (variations in latency) can cause disruptions in service, even if the latency itself remains within the acceptable bounds. (d) *Low traffic rates*. Fortunately, in most cases, the traffic generated by industrial event-based applications is not both latency- and throughput-critical. Typical use cases generate roughly 50 Mbps or less, which is a very low traffic rate for modern networks.

2.2 The Evolved Packet Core

The Evolved Packet Core (EPC) is the core network of Long Term Evolution (LTE) that supports mobile broadband in current mobile telecommunication standards (4G) [17]. It consists of different network functions, such as mobility management, quality of service and lookup of subscription information [4]. The user-plane of EPC is responsible for processing packets between the nodes of the EPC and for connecting them with external networks. The user-plane includes many functions such as firewalls and deep packet inspection, but it is primarily responsible for packet forwarding (switching) [13]. That is the network function that we focus on in this paper.

2.3 User-space packet processing

Traditionally, software-based packet processing uses the kernel's network stack to send and receive packets. However, relying on the kernel for packet I/O involves issuing an interrupt every time there is a new packet to be received from the network card, which introduces overheads. Even though newer kernel versions reduce the number of interrupts generated [21], kernel-based networking cannot match the processing speed needed by high performance networks.

In the last few years, user-space solutions for packet processing have become popular, with the *Data Plane Development Kit* (DPDK) being the most common packet I/O framework [2]. In DPDK, packets that are received from the network card are sent directly to memory that is mapped to user-space. DPDK's driver keeps polling the memory for new packets, instead of waiting for an interrupt to be issued. Additionally, DPDK's library includes support for fast packet processing across many cores and heavily relies on batching multiple packets to amortize the per-packet processing cost, making it the de-facto choice for high speed networks.

3 LATENCY AND JITTER

In this section, we identify sources of latency and jitter in the context of packet processing on commodity hardware and propose methods to mitigate their effects.

Commodity hardware platforms are generally not designed for low latency and real-time processing. As a result, there are many sources of latency, stemming from, e.g., the scheduler, the operating system or packet processing application itself. While there are many suggestions on how to improve the real-time characteristics of commodity systems [16], we focus on specific parts and

measure their effects. Specifically, we target latency and jitter due to: (a) the packet I/O framework and forwarding application and (b) the operating system and the hardware platform itself.

3.1 Packet I/O and forwarding application

Choice of Packet I/O: As previously mentioned in Section 2, kernel-based packet processing introduces overheads. For this reason we use a user-space packet I/O framework (DPDK) and its own simple packet switching application (Layer 2 packet forwarding). We compare its performance with the traditional kernel-based packet I/O (NAPI [21]) using *Linux bridge* [24] for packet switching.

Note that DPDK is designed with high throughput as the primary goal. While the vast majority of that increased throughput comes from reducing the per-packet processing latency, DPDK's design is not necessarily oriented towards maintaining a low per-packet maximum latency. In fact, as we discuss next and experimentally show in Section 5, the per-packet latency can be high under low load, due to buffering.

Low Traffic Rate + Buffering = High Latency: For the DPDK version, as a staring point, we use the Layer 2 forwarding example application provided by DPDK's library, that simply forwards packets from one port to another. By default, the Layer 2 forwarding application will receive and buffer up to 32 packets before sending them to the outgoing port as a single batch. This reduces the number of times the application communicates with the network card and helps sustain a high processing rate when the incoming packet rate is high. However, buffering can severely impact latency at low traffic rates. Recall from Section 2 that URLLC applications usually have low traffic rates, so maintaining low latency under such conditions is critical. When the input rate is low, there is a significant delay until the buffer is full of packets and can be sent to the network card which causes: (a) high latency on the first packet that is buffered, since it has to wait for 31 more packets to arrive and be processed and (b) high variation in latency, due to the difference in waiting time between the first packet that gets buffered and the last one before the buffer is flushed. This effect on DPDK's Layer 2 forwarding application has been reported by Kawashima et al. [11].

Since our target applications require very low latency at low traffic rates, we mitigate the effects of buffering and sacrifice the performance at high traffic rates for low and consistent latency at low rates (in Section 6 we quantify experimentally how much throughput gets sacrificed). We set the size of the buffer to one packet to send every received packet to the outgoing port as soon as it is received. An alternative approach would be to set a timer that will flush the packet buffer at fixed intervals. We choose to disable buffering in order to come as close as possible to the lowest possible latency that we can achieve. We study the effect that disabling buffer has in latency and jitter in Section 5.

3.2 Operating System and Hardware Platform

We next identify and tackle sources of latency and jitter that come from the operating system and its interaction with the hardware platform. The goal is to ensure (to the extent that it is possible) that the user-space application handling the packet processing does not get any interruptions in its service.

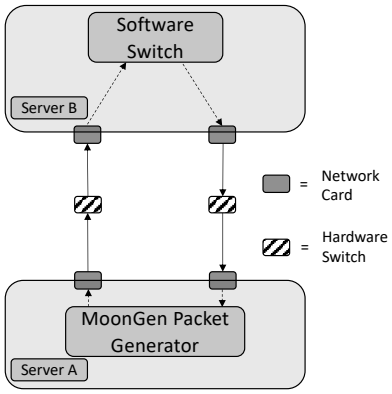


Figure 1: The experimental setup.

Nadathur et al. [16] suggest multiple kernel options that can contribute to lower and more stable latency, including real-time kernel patches specifically for this purpose. We follow several of these considerations and introduce additional ones. Specifically:

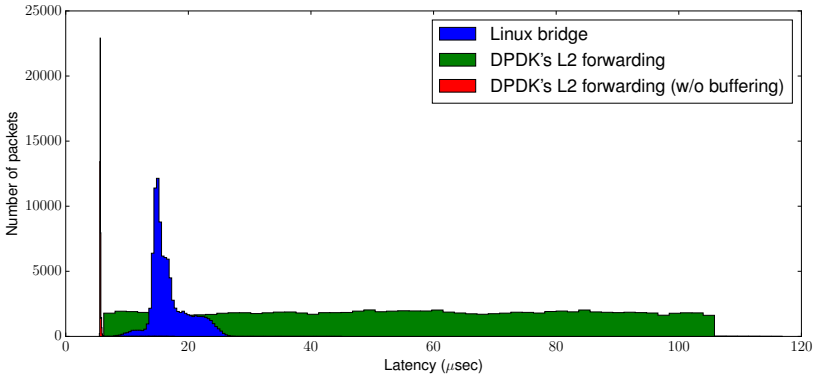
- **Thread isolation:** We isolate the cores that are used by DPDK from the kernel scheduler, to ensure that no other task will time-share or use resources from those cores.
- **Disable interrupt balancing:** We disable the daemon that dynamically distributes interrupts to cores, to avoid handling unrelated interrupts by the cores running DPDK.
- **Disable turbo-boost:** Turbo-boost is a technology used in Intel CPUs that allows scaling the frequency of a core dynamically during peak loads, even beyond the nominal values, if the power and temperature budget allows [9]. We find (and experimentally show in Section 5) that on some platforms, enabling turbo boost causes high variation in packet processing latency, possibly due to interruptions in processing involved when the CPU frequency changes.

4 EXPERIMENTAL METHODOLOGY

In this section, we present our experimental methodology. We describe the hardware platform used in our experiments, the network topology and the way test traffic is generated. We also present the metrics that are relevant in our evaluation.

Hardware Platform: We use servers and network infrastructure from CloudLab [3], a platform that allows bare metal access to a wide range of hardware devices. A summary of our experimental setup is shown in Figure 1. Each of the two servers is a 20-core NUMA platform with Xeon E5 at 2.26 Ghz that supports 2-way hyperthreading. Each server has two Intel 82599ES 10 Gb Ethernet network cards, connected to each other as shown in Figure 1, through an internal network of hardware switches. Each server also has 1Gb Ethernet card connected to the external network for control over the servers (not shown in Figure 1).

While it is currently not possible to connect the servers directly, we performed a loop-back test to establish a baseline on the latency introduced by the hardware switches. Our measurements showed an average latency of 1.1 μsec and 0.1 μsec difference between maximum and minimum latency per switch (these measurements



(a) Histogram of latency measurements using different packet processing versions. The spike at 5.7 μsec corresponds to the version that does not use buffering and has too low variation in latency to be clearly visible.

	Linux bridge	DPDK	DPDK (w/o buffering)
Average latency (μsec)	16.8	56.1	5.7
Minimum latency (μsec)	7.8	6.2	5.5
Maximum latency (μsec)	45.0	117.0	17.8
Deviation (μsec)	3.1	28.5	0.6

(b) Latency statistics for different packet processing versions.

Figure 2: Histogram and latency statistics for packet processing of different versions. Different versions have significant differences in both the average and the general distribution of latency.

include any overhead added from the packet generation software, which we describe next).

Traffic Characteristics and Generation: We use the MoonGen packet generator [5] to send traffic and measure latency and jitter. MoonGen uses DPDK to send and receive traffic and supports measurements with 100 nsec precision. Unless otherwise noted, we generate a low, constant load of 64 byte UDP packets at 100 Kpackets/sec and we measure latency every 1ms. We generate traffic in platform A, send it to platform B where the software switch application forwards it to a different port that is connected back to platform A. Latency is measured using hardware timestamps generated at the network cards.

Metrics: We use latency, reported as the round-trip-time of packets from the moment they leave the network card at server A until they return. We report jitter as the absolute difference in latency between two successive measurement samples [18]. Finally, we also report throughput as the rate at which we send traffic to the network at server A.

5 EMPIRICAL STUDY

In this section, we present the results from our experiments and show the effect of our mitigation techniques on latency and jitter.

5.1 Packet I/O framework

We start by studying the effect that the choice of the packet I/O framework has on latency. In Figure 2a we show the latency when using kernel-based packet processing (linux-bridge) versus user-space packet processing (DPDK). We include the latency statistics in the table of Figure 2 for completeness. The latency samples for the linux bridge are spread around an average of 16.8 μsec, with a minimum and maximum latency of 7.8 and 45 μsec respectively. On the other hand, the latency measurements of DPDK's original layer 2 forwarding application are evenly spread between 6.2 and 117 μsec, with an average of 65 μsec. This even spread is clearly an effect of buffering, where the latency of a packet depends greatly on how early or late it was placed on the buffer. As a result, we see that the original DPDK version of packet forwarding is not designed to perform well with respect to latency at low traffic rates.

In summary: Using a high-performance, user-space I/O framework does not, on its own, guarantee low latency at low traffic rates. Next, we show the effect of buffer-removal on latency.

5.2 Application Layer Optimizations

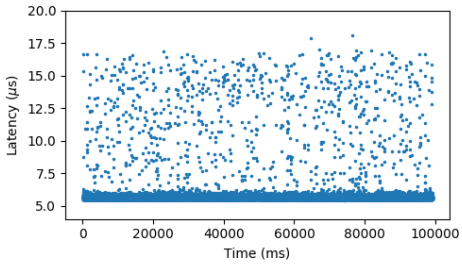
We now present the latency measurements of a modified version of DPDK's L2 forwarding application where packets are sent directly to the outgoing interface after they are received. We include those results in Figure 2a to compare against the other versions, especially against the version that uses buffering. By disabling packet buffering, the latency of most packets is concentrated at around 5.7 μsec, which is 9.8X lower than than the original version that buffers packets. However, the maximum reported latency was 17.8 μsec, which means that there are still sources of spurious latency spikes, that originate from the underlying hardware and operating system. We discuss their effect and mitigation next.

In summary: Disabling packet buffering in packet forwarding applications is necessary to achieve low latency when the traffic rate is low.

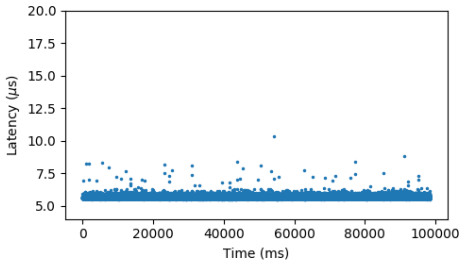
5.3 System Layer Configurations

After optimizing the application for low latency, we now focus on the system level configurations and their effect on latency. Figure 3a shows a different representation of the latency of DPDK's L2 forwarding version that does not use any buffering, where we plot all the latency samples gathered during our experiments. We see that the vast majority of samples have latency around the average of 5.7 μsec, but there are spurious increases in latency that range up to 18 μsec, indicating that latency increases unpredictably during the execution of our experiments. We also report jitter in the table of Figure 3. The average jitter is 0.17 μsec with a maximum reported jitter of 12.52 μsec.

In Figure 3b, we show the net effect of applying the system configurations presented in Section 3. The effect of those configurations



(a) Latency samples before system level configurations.



(b) Latency samples after system level configurations.

	Before changes	After changes
Average jitter (μsec)	0.17	0.07
Maximum jitter (μsec)	12.52	4.80
Deviation (μsec)	0.88	0.10

(c) Jitter statistics before and after system level configurations.

Figure 3: The effects of system level optimizations on latency. The system level configurations lead to more predictable latency that is concentrated around the average values and has less deviation than before applying the changes.

is immediately noticeable in this representation, where the vast majority of spurious increases in latency have been mitigated. The average reported jitter is 0.07 μsec, which is 2.4X lower than before applying the system level configurations. Performing longer runs, we find that the maximum latency is not affected (our maximum reported latency was 20 μsec), but latency is much more predictable and concentrated around the average.

In summary: System layer configurations such as thread isolation and disabling turbo boost significantly reduce jitter.

5.4 Latency vs Throughput

The mitigation techniques we introduced in Section 3 and evaluated so far focus on minimizing latency at low traffic rates. However, they come at a cost: the maximum sustained packet processing rate is reduced.

In Figure 4 we report the average and maximum latency of both the original (throughput-optimized) and the latency-optimized packet forwarding DPDK application as we increase the rate at which we generate traffic. When the traffic rate is low, the original version that focuses on throughput has high average and maximum latency, since it takes long for the packet buffer to fill. As the traffic rate increases, the effect of buffering on latency decreases and this version maintains a low latency at high rates. In fact, the original DPDK L2 forwarding application can easily support the maximum available bandwidth at the link (10 Gbps). The version that does not do any buffering has low latency at low traffic rates, which gradually increases with the traffic rates, until roughly 4.3 Gbps. After that point, this version cannot process packets at the same rate as they arrive. As a result the packet queues start to fill, latency increases and we start to see packets being dropped. However, as we discuss in Section 2.1, it is the low-throughput end (often less than 50Mbps) that is relevant for URLLC applications. At those low rates, our latency mitigation techniques manage to keep the latency an order of magnitude lower than what it originally was.

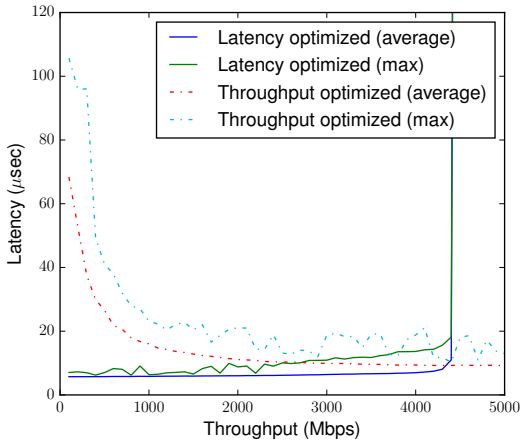


Figure 4: Average and maximum latency at different packet transmission rates. The latency-optimized version cannot sustain the incoming packets rates after roughly 4.3 Gbps.

6 DISCUSSION

The requirements of the URLLC on latency and jitter presented in Section 2, relate to the end-to-end application requirements and include the overhead of many components of the network, including wireless communication with the base station. As such, for an application with e.g. a 10ms latency requirement budget, only a small part of that budget can be allocated to the user-plane of the packet core and specifically to packet switching. However, it is hard to judge exactly how much the target latency for packet switching alone would be. In our experiments, the maximum reported latency was 20 μsec, which is likely to be sufficiently low for most industrial applications. However, since we examine only a small part of the network stack involved, it is hard to form a clear picture of the feasibility of deploying a network stack on commodity hardware for latency critical applications. Moreover, as we demonstrated in Section 5.4, our latency reduction techniques come at the cost of lower sustainable throughput at high rates, but are a good fit for URLLC applications that have low traffic rates.

7 RELATED WORK

In this section, we present related work on the topic of evaluating the packet processing performance of commodity hardware.

Emmerich et al. [6] perform extensive benchmarks using different packet I/O frameworks and identify performance bottlenecks

in hardware and software. Gallenmüller et al. [8] focus on the performance of I/O frameworks and present an analytical model to predict their performance. They also experiment with packet buffering and show its effect on latency, but only at high traffic rates. Kawashima et al. [11] evaluate the performance of packet forwarding applications across many packet I/O frameworks and execution environments, focusing on both throughput and latency. They also identify that the original DPDK L2 forwarding application uses buffering which negatively affects latency at low traffic rates.

A large body of research evaluates commodity hardware in the context of NFV. Anderson et al. [1] evaluate packet I/O frameworks in different environments, including virtual machines and containers. They provide results on latency and jitter, but do not use fast, user-space I/O frameworks such as DPDK. Kourtis et al. [12] evaluate the processing throughput of deep packet inspection applications with DPDK on both bare-metal and virtualized environments.

Focusing on mobile broadband networks, Lange et al. [14] evaluate the performance a Serving Gateway that involves both the user-plane and control plane events, and show that user-space networking based on DPDK can greatly improve the per-packet processing time. Mao et al. [15] study the performance of a software-based Radio Access Network under strict latency requirements and show that light-weight virtualization with containers, together with frameworks like DPDK can lead to worst-case latency that is within the required bounds of latency-critical applications. Contrary to those approaches, in this paper we show the trade-off between latency and throughput, study the sources of latency and jitter and show system and applications configurations that can reduce them.

8 CONCLUSIONS

In this paper, we consider the performance of packet processing deployed on commodity hardware with respect to latency and jitter and propose a baseline on the feasibility of such platforms for the packet processing needs of Industry 4.0 applications. We identify sources of latency and jitter in the packet processing application as well as the underlying system and show ways to mitigate them. Specifically, we show that optimizing applications for latency rather than throughput (e.g. by disabling buffering of packets), greatly reduces average latency by up to 9.8X, at low traffic rates, which is important for event-based URLLC that usually have low volumes of traffic but are sensitive to latency. Moreover, we show that system level configurations, such as disabling dynamic frequency scaling makes latency more predictable and reduces jitter.

ACKNOWLEDGEMENTS

The research leading to these results has been partially supported by the Swedish Civil Contingencies Agency (MSB) through the projects RICS and RIOT, by the Swedish Foundation for Strategic Research (SSF) through the framework project FiC, by the Swedish Research Council (VR) through the project ChaosNet and the project AgreeOnIT, the Vinnova-funded project "KIDSAM", and from the European Community's Horizon 2020 Framework Programme under grant agreement 773717.

REFERENCES

[1] Jason Anderson, Hongxin Hu, Udit Agarwal, Craig Lowery, Hongda Li, and Amy Apon. 2016. Performance considerations of network functions virtualization using containers. In *2016 International Conference on Computing, Networking and Communications (ICNC)*. 1–7. https://doi.org/10.1109/ICCNC.2016.7440668

[2] DPDK. 2019. Data Plane Development Kit. https://www.dpdk.org.

[3] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[4] Romaric Duvignau, Marina Papatriantafilou, Konstantinos Peratinos, Eric Nordström, and Patrik Nyman. 2019. Continuous Distributed Monitoring in the Evolved Packet Core. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*. 187–192.

[5] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference 2015 (IMC'15)*. Tokyo, Japan.

[6] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Assessing soft-and hardware bottlenecks in PC-based packet forwarding systems. *ICN 2015* (2015), 90.

[7] Ericsson. 2016. Internet of Things forecast. https://www.ericsson.com/en/mobility-report/internet-of-things-forecast. Accessed: 2019-01-15.

[8] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. 2015. Comparison of frameworks for high-performance packet IO. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 29–38.

[9] Intel. 2019. Higher Performance When You Need It Most. https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html.

[10] Nasser Jazdi. 2014. Cyber physical systems in the context of Industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*. 1–4. https://doi.org/10.1109/AQTR.2014.6857843

[11] Ryota Kawashima, Hiroki Nakayama, Tsunemasa Hayashi, and Hiroshi Matsuo. 2017. Evaluation of Forwarding Efficiency in NFV-Nodes Toward Predictable Service Chain Performance. *IEEE Transactions on Network and Service Management* 14, 4 (Dec 2017), 920–933. https://doi.org/10.1109/TNSM.2017.2734560

[12] Michail Kourtis, George Xilouris, Vincenzo Riccobene, Michael Mcgrath, Giuseppe Petralia, Harilaos Koumaras, Georgios Gardikis, and Fidel Liberal. 2015. Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration. https://doi.org/10.1109/NFV-SDN.2015.7387409

[13] James Kurose and Keith Ross. 2016. *Computer networks and the internet. Computer networking: A Top-down approach*. London: Pearson.

[14] Stanislav Lange, Anh Nguyen-Ngoc, Steffen Gebert, Thomas Zinner, Michael Jarschel, Andreas Köpsel, Marc Sune, Daniel Raumer, Sebastian Gallenmüller, Georg Carle, and Phuoc Tran-Gia. 2015. Performance benchmarking of a software-based LTE SGW. In *2015 11th International Conference on Network and Service Management (CNSM)*. 378–383. https://doi.org/10.1109/CNSM.2015.7367386

[15] Chen-Nien Mao, Mu-Han Huang, Satyajit Padhy, Shu-Ting Wang, Wu-Chun Chung, Yeh-Ching Chung, and Cheng-Hsin Hsu. 2015. Minimizing Latency of Real-Time Container Cloud for Software Radio Access Networks. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. 611–616. https://doi.org/10.1109/CloudCom.2015.67

[16] Sundar Nadathur and Jiming Sun. 2018. NFV-I host configuration for low latency. https://01.org/packet-processing/blogs/nsundar/2018/nfv-i-host-configuration-low-latency.

[17] Magnus Olsson, Catherine Mulligan, Shabnam Sultana, Stefan Rommer, and Lars Frid. 2013. *EPC and 4G packet networks: driving the mobile broadband revolution*. Academic Press.

[18] S Poretsky, J Perser, S Erramilli, and S Khurana. 2006. RFC 4689–Terminology for Benchmarking Network-layer Traffic Control Mechanisms. *IETF, October* (2006).

[19] Ericsson Technology Review. 2019. Cloud-native application design in the telecom domain. https://www.ericsson.com/en/ericsson-technology-review/archive/2019/cloud-native-application-design-in-the-telecom-domain.

[20] Joachim Sachs, Gustav Wikstrom, Torsten Dudda, Robert Baldemair, and Kittipong Kittichokechai. 2018. 5G radio network design for ultra-reliable low-latency communication. *IEEE network* 32, 2 (2018), 24–31.

[21] Jamal Hadi Salim. 2005. When NAPI comes to town. In *Linux 2005 Conf*.

[22] Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, and Marina Papatriantafilou. 2017. Multiple Pattern Matching for Network Security Applications: Acceleration through Vectorization. In *2017 46th International Conference on Parallel ProceWsing (ICPP)*. 472–482. https://doi.org/10.1109/ICPP.2017.56

[23] Charalampos Stylianopoulos, Simon Kindström, Magnus Almgren, Olaf Landsiedel, and Marina Papatriantafilou. 2019. Co-Evaluation of Pattern Matching Algorithms on IoT Devices with Embedded GPUs. In *Proceedings of the 35th Annual Computer Security Applications Conference* (San Juan, Puerto Rico) *(ACSAC '19)*. Association for Computing Machinery, New York, NY, USA, 17–27. https://doi.org/10.1145/3359789.3359811

[24] Nuutti Varis. 2012. Anatomy of a Linux bridge. In *Proceedings of Seminar on Network Protocols in Operating Systems*. 58.