User Plane Hardware Acceleration in Access Networks: Experiences in Offloading Network Functions in Real 5G Deployments

Ralf Kundel, Tobias Meuser, Timo Koppe, Rhaban Hark, Ralf Steinmetz
Techincal University of Darmstadt
{firstname.lastname}@tu-darmstadt.de

Abstract

Fulfilling the ambitious Quality of Service demands of today's wireless networks, especially low latency, high bandwidths and availability, is a big challenge for researchers, network architects, and operators. Each networking component on the data path between the user equipment and the destination data network, e.g., the Internet, must provide the highest performance to meet these requirements. This work demonstrates how different network elements of the user plane, describing the whole path of user traffic, can be sped up with different hardware acceleration technologies. For that, we demonstrate how to build up a 5G standalone campus network for evaluation, working end-to-end with real user equipment and open-source software components. Further, we analyze the user plane network functions of 5G networks from the radio access network to the core. Based on our real 5G setup, the practical evaluation of the analysis results shows up how the 5G user plane hardware can be accelerated best.

1. Introduction & Background

One main objective of 5G access networks is an increased Quality of Service (QoS), especially high throughput and low latency. All involved network functions and the underlying hardware must perform at the highest possible level to accomplish this.

One enabler for innovation in mobile access networks and computer networks, in general, is Software Defined Networking (SDN). SDN describes a paradigm for disaggregating network functionality [1]. Network switches are divided into a user plane (also known as data plane) and a control plane. The user plane is responsible for forwarding network packets based on simple rules only, while the control plane is responsible for computing and managing these rules. By introducing well-defined interfaces between control and user plane, the replacement of only a single user plane function or control function is eased.

Achieving the postulated QoS requirements in 5G networks (*e.g.*, one millisecond round trip time from devices to the core) requires focusing on the user plane for acceleration. In most of today's mobile networks, including modern 4G and 5G deployments, all network functions are realized in software environments fulfilling the performance goals neither in terms of throughput nor latency. However, this performance can be increased by offloading some functionality on programmable hardware accelerators. Besides the increase in performance, hardware acceleration raises new challenges and not every approach is suitable for every network function to be accelerated.

5G networks can operate either in non-standalone mode with a 4G-based anchor cell for authentication or as a pure 5G-based network, named 5G standalone. We focus on 5G standalone networks in the following. Figure 1 shows the 5G standalone architecture according to the 3GPP specification [2]. The functionality can be divided according to the previously introduced SDN terminology into a user plane and a control plane. The User Equipment (UE) connects with the Data Network (DN) (typically the Internet) through the 5G network.

For that, the UE connects via the Radio Access Network (RAN) with the 5G-core at the specified reference points N1, N2 for the control plane and N3 for the user plane. The RAN functionality can be further divided into sub-functions, often named as "split". In the terminology of the Open RAN Alliance (O-RAN) and the "7.2-split", this is the Radio Unit (RU), Distributed Unit (DU) and Central Unit (CU). In other split scenarios, the DU and CU are combined to a CU/DU or BaseBand Unit (BBU) in the 4G terminology. However, the overall functionality remains unchanged.

As part of the 5G-core, the User Plane Function (UPF) terminates the Protocol Data Unit (PDU) session of many UEs and forwards the packets to and from the data network. On the N6 side of the UPF, standard IPv4/6 protocols are used; on the N3 side, for each PDU session a GPRS

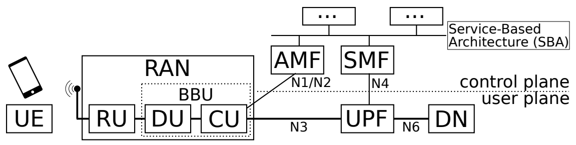

Figure 1. User plane centric topology of the 5G-core network and RAN.

Tunneling Protocol (GTP) tunnel is established. One UE can be connected to multiple DNs in parallel and two GTP-tunnels, one for upstream and one for downstream, are established for each DN.

The two network functions, RAN and UPF, represent all involved components in the user plane.

In general, the usage of 5G standalone networks can be divided into two phases: (1) Registration: The UE authenticates itself towards the 5G core with a challenge-and-response mechanism [3] and flow rules are installed in the RAN and UPF of the user plane. This happens typically only once for each UE and therefore does not affect the end-to-end performance strongly. (2) Connection: The UE is successfully connected and can send/receive IP packets to/from the data network. Only the user plane is involved in this phase.

The UE connects via the N1 reference point to the Authentication Management Function (AMF) for authentication. These N1 messages are received on the air interface by the RAN and forwarded to the AMF. In addition, messages between the RAN and the AMF are exchanged during the authentication and PDU session establishment procedure via the N2 reference point.

Once the UE is successfully authenticated, it can request a PDU session from the 5G-core, configured in the UE, RAN and UPF by the AMF and the Session Management Function (SMF). Within the 5G-core, all control plane network functions, including the AMF, SMF and many more, are orchestrated in a Service-Based Architecture (SBA). This means, all network functions communicate via a HTTP/2 REST interface and can be deployed as a cloud service. Through this deployment, network functions can be instantiated multiple times in parallel, *e.g.*, having multiple SMF and UPF functions in order to distribute the load. By that, high scalability of 5G deployments for high network loads can be achieved.

To improve the overall 5G network performance, several different hardware acceleration approaches can be applied within the RAN and UPF network function to achieve the desired performance. In order to ensure correct behavior as well, researchers and engineers need to validate new approaches in a real working end-to-end system. Therefore, we first describe the difficulties in setting up a modular end-to-end 5G standalone testbed, allowing the easy exchange of network functions, and discuss several hardware acceleration approaches.

The outline of the following paper is as follows: First, we present our experiences in deploying an end-to-end 5G network for research purposes. Second, we discuss several hardware acceleration approaches and technologies for 5G network functions in the user plane. Last, we present measurement results as well as our experiences in deploying these approaches within our test network.

2. A 5G standalone Testbed Setup

As validations performed in simulations commonly have limited expressiveness, it is crucial also to test novel approaches for accelerating 5G networks in testbeds. There are two main approaches to perform these test: (i) in an isolated environment with test benches and stimulus generators for the particular network function under test or (ii) integrated into a working end-to-end setup, i.e., a 5G capable UE connected via a RAN and a 5G-core with a data network to perform end-to-end testing. In general, testing in an end-to-end setup is far more expressive than tests based on test benches, as the 5G ecosystem is very complex and test benches rarely cover all cases. Additionally, a functional end-to-end test guarantees that no major functionality is missing or behaving faulty.

In this section, we will introduce a disaggregated 5G standalone testbed for research and development. We describe the deployment of this testbed and important design decisions to be considered. This testbed is easily extensible, as we rely on open-source components wherever possible.

2.1. Overall Architecture

The overall testbed architecture is shown in Figure 2. It relies mainly on the free5gc open-source project to realize 5G-core network functions [4]. The free5gc project is licensed under the Apache 2.0 license and especially well suited due to its modularity: Each network function is realized as its own process and communicates with other network functions over the well-defined reference points in the 5G service-based architecture. This enables the replacement or modification of individual network functions and eases new features to be developed and tested. For example, the UPF implementation can be replaced by a novel prototype with advanced features just like any other control plane function. Note that the authors of this work are neither the founders nor the main contributors in this project. Therefore we would like to acknowledge the work of the contributors in free5gc explicitly.

In our testbed architecture, depicted in Figure 2, the 5G-core network functions are realized within two off-the-shelf servers and one specialized component realizing the RAN functionality as described later in detail in Section 2.2. The two servers are running on Ubuntu 18.04 (kernel 5.0.0-23), but we do not expect any issues with other Linux distributions. The only restriction is the kernel version, which is required by the UPF implementation of free5gc.

The *server #1* realizes all control plane functions of the 5G-core as well as the first UPF instance for “ims”-traffic and an IMS-server as described later in Section 2.3. The *server #2* realizes the UPF instance for the data network “internet”. For each UPF instance, a dedicated SMF network function instance is required, both running on *server #1*.

Due to current limitations of free5gc, it is impossible to run these two UPF instances on the same server within the same Linux network namespace, and therefore the second server is reasonable. However, we successfully tested running both UPF instances on the same server in two different namespaces as well. As one of our goals is to replace the UPF network function, the encapsulated dual server approach is more suitable for us. Nevertheless, research projects having their focus on the control plane, *e.g.*, on 5G authentication methods, two namespaces would be a proper solution.

PDU sessions to the data network “internet”, terminated on the UPF of *server #2*, are used for all normal traffic of nowadays UEs and multiple networks are typically not in use. As the IP addresses assigned to the UEs in this data network are not public available (in the example they are in the range of $10.1.0.1 - 10.1.255.254$), a Network Address Translation (NAT) must be performed before forwarding the packets to the Internet. This NAT functionality is implemented using the basic Linux *masquerade* functionality.

All network functions within the 5G core must be able to connect to each other. For that, unique IP addresses must be assigned to each function. The service-based architecture interfaces within *server #1* can be assigned to the IP address range $127.0.0.x/24$ as they are all located within the same machine. Only the interfaces N1/N2 of the AMF, the N4 interface of the “internet” SMF and the N3 interfaces of both UPF functions must be assigned to IP addresses in the management network (*i.e.*, 192.168.1.2 and 192.168.1.3) as they need to be accessible by *server #2*. Further, the N3 address of the AMF must be configured within the RAN, which connects at startup to the AMF. Within the 5G-core, no further IP addresses must be configured, as the network repository function of the 5G core, allows dynamic locating of any network function.

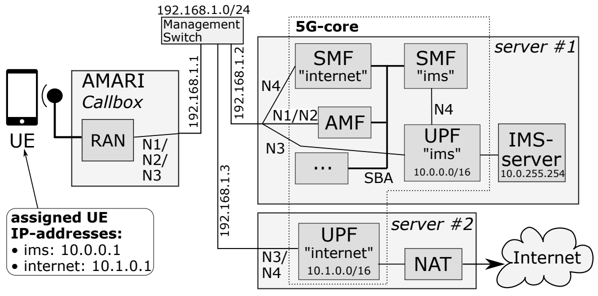

Figure 2. End-to-end 5G standalone testbed.

The last major step is to configure the 5G parameters within the RAN and the core. A Public Land Mobile Network (PLMN) identifier must be set within the RAN configuration file, the AMF configuration file and in the SIM card of the UE. The UE uses this PLMN to identify the operator network it should register to. If the PLMN of the RAN mismatch with the PLMN of the SIM card, the registration process will not be initiated. Further, a programmable SIM card must be programmed with all 5G-AKA credentials as well as a unique SIM card ID (IMSI), and these same credentials must be inserted into the 5G core database.

2.2. Radio Access Network

In our work, we focused on the commercial product *AMARI Callbox Classic* [5] which was the only available 5G standalone RAN for research purposes at the time of starting this work.

Nevertheless, open-source initiatives exist, which realize this functionality with free programmable Software Defined Radio (SDR) boards. In the recent past, the *openairinterface5g* initiative [6] presented a first working 5G standalone RAN based on SDR cards. However, since we did not have the necessary SDR hardware in our lab at the time of writing, we could not yet investigate this.

As previously mentioned, the O-RAN project [7] specified a disaggregated RAN consisting of a RU, DU and CU. First vendors already presented hardware of the RU containing only the high-frequency functionality. The functionality of DU and CU can be realized within software and we expect such RU hardware to be supported in conjunction with open-source software such as *openairinterface5g* in the future.

2.3. Voice over New Radio

In 5G standalone networks, the UE must use Voice over New Radio (VoNR) to provide telephony service. For that, the UE requests an additional PDU session from the 5G core to the data network “IMS” (IP

Multimedia Subsystem) and connects to an IMS-server within this network. The IP address of the IMS-server will be forwarded by the AMF to the UE during the PDU session setup and must be within the IP-range of the PDU subnet. In the example setup of Figure 2, the UE IPv4 address for the IMS data network is $10.0.0.1/16$ and the IMS-server has the IPv4 address $10.0.255.254$, which is in the same subnet. Suppose the UE requests VoNR functionality but the 5G core provides no IMS data network or no IMS-server exists. In that case, the registration process terminates unsuccessfully for some devices and prevents a successful test.

In order to circumvent this, the test network must provide an IMS data network and an IMS-server must respond to the UE initial request. The UE does not need to registers successfully to the IMS-server, even an error message is sufficient to complete the registration.

However, one of the tested UEs has a fallback mechanism to accept connections to 5G standalone networks without VoNR functionality after 1 to 3 minutes of unsuccessful IMS registration attempts.

In our work, we circumvent this issue by configuring two data networks (“internet” and “ims”) in the 5G-core and running an IMS-server. We achieved a working setup for VoNR with multiple server implementations, including the Amarisoft IMS, the open-source Kamailio SIP [8] and a self-developed error message responder.

2.4. Available 5G Standalone UEs

Not all UEs capable of 5G support 5G standalone, even if the datasheet claims this. Further, not every UE works with 5G standalone on every radio frequency and every Public Land Mobile Network (PLMN) identifier. We hat best experience on 5G channel `n78` with the PLMN *00101*, *999*** and selected PLMNs of operators which support already 5G standalone. The configured channel bandwidth and MIMO factor also influence the success of the registration process. We observed that the only working configuration for all tested 5G capable UEs was 2x2 MIMO with $50\ MHz$ bandwidth per MIMO-channel. At the time of this work, we successfully tested the following devices with at least one of the previously mentioned configurations: Huawei P40, OnePlus 8T and Oppo Find X2 Pro. The list of unsuccessfully tested devices will not be presented, as not all parameter configurations were tested and a software update might enable 5G standalone functionality in the future. In addition, we observed changes in the behavior after firmware updates of the UEs, sometimes even to the worse. As sending on some frequencies and PLMNs might not be allowed by local regulations, a Faraday cage should be considered.

2.5. Reproducing the Setup

While building this 5G demonstration setup, we experienced several minor bugs and missing features. However, we proposed bug fixes as pull requests for all of them and starting with free5gc v3.0.6 the setup described in this section can be deployed with only adapting the configuration files and the UE database. Special hardware is only required for the RAN functionality; all other components can be realized with commodity servers and open-source software.

3. Hardware Acceleration Technologies

This section provides an overview of existing approaches for hardware acceleration approaches for the 5G user plane. The 5G standalone architecture implements concepts and functionality of two domains, traditional networks as well as of data centers. Therefore, we will consider in the following acceleration technologies from both domains. First, in Section 3.1 the evaluation metrics are introduced. Second, we discuss multiple acceleration technologies for 5G user plane network functions in Section 3.2. The discussion results are summarized in Table 1 on a scale from good (++) to bad (- -).

3.1. Evaluation criteria

For the assessment of different approaches, we will introduce review classes in the following which can be grouped into three categories: (i) functionality, (ii) flexibility and (iii) performance.

Functionality: *Functionality* is captured by three properties: header processing, QoS-functions, and cryptography. Header processing refers to the ability of the user plane network functions to understand and process special packet header protocols which are not supported by non-programmable off-the-shelf networking hardware [9], *e.g.*, the GTP encapsulation protocol of 5G. QoS-functions refer to prioritizing and queuing packets to ensure that the UE is not exceeding its bandwidth or in case of handing over a UE from one RAN cell to another[10]. Cryptography refers to the en-/de-cryption and en-/de-coding of packets within the RAN, not only the packet headers. In the O-RAN split shown in Figure 1, this encryption functionality would be realized in the Central Unit (CU), while the channel encoding would be realized in the Distributed Unit (DU).

Flexibility: In this work, *Flexibility* describes the adaptability of the acceleration technology to changing requirements. Concretely, the scaling of an approach, *i.e.*, how easily new instances of a network

	Software				**Hardware**			
	kernel space	user space	SR-IOV + kernel space	SR-IOV + user space	NPU	GPU	FPGA	P4 switch
functionality:								
header processing	-	++	-	++	++	+	+	++
QoS-functions	+	++	+	++	-	+	++	-
cryptography	-	+	-	+	+	++	+	- -
flexibility:								
scaling	+	+	++	++	-	+	-	-
reconfiguration	++	++	++	++	-	+	-	+
#GTP_sessions	++	++	++	++	+	++	-/+	-
performance:								
throughput	-	+	-	+	+	+	+	++
latency	-	+	-	+	+	-	++	++
jitter	-	+	-	+	+	-	++	++
packet loss	-	+	-	+	+	-	++	++

Table 1. Overview of software concepts and hardware technologies for accelerating 5G network functions.

function can be created or destroyed, analogous to the cloud computing principles. Additionally, it is very beneficial if new functionality introduced by 3GPP can be easily added to existing network functions by updating the software or reconfiguring the hardware. For example, in the 5G standalone specification, QoS flow identifiers were used as a GTP header extension. This kind of flexibility is considered in the review class *reconfiguration*. Last, the total amount of state which can be stored within the device should be considered, *i.e.*, how many GTP sessions can be installed in the UPF at the same time or how many UEs can be registered to a RAN station.

Performance: As the last category, Quality of Service (QoS) metrics to measure the performance of approaches need to be considered. Throughput describes the number of bytes processed and forwarded by the network function in a certain time period. The latency describes the time a packet is processed in a particular function. Compared to that, the total end-to-end latency of a system is the sum of the latency of all network functions. As the total end-to-end latency is the latency perceived by the UE, it is a crucial metric in 5G. The variance in latency is described as latency jitter. Especially for functionality like the RAN network, which has strict QoS requirements, this is very important. It influences the longest latency of an individual packet and is a pivotal metric to assess the expectable latency of a network function. Packet loss is the last metric to be considered. A lost packet must be retransmitted and therefore has a similar negative effect than a high latency jitter. In computer networks with real-time guarantees, *e.g.*, time-sensitive networks for real-time applications, typically no packet loss and only low jitter is required, and a certain throughput can be guaranteed.

3.2. Acceleration Approaches

Kernel space: Standard software applications get access through the operating system kernel. The kernel typically includes a full networking stack containing all protocols. The advantage of this approach is that the application does not have to take care on the network and transport layer. However, the use of special protocols in 5G requires to load additional kernel modules and is therefore not ideally suited for header processing. QoS-functionality can be realized within the kernel; additional queueing disciplines can be defined but is quite challenging compared to a pure user space implementation. As all functionality is realized in software, the flexibility in general is very high. The only disadvantage is the scaling aspect, as one physical port of the network interface card (NIC) is always mapped to one kernel stack implementation and no parallelism is given. As the kernel functionality is made to be universal and a lot of functionality is implemented, the performance is, compared to the other approaches, low but acceptable.

One advantage of a kernel-based network function is that no specialized hardware is needed. To summarize, network functions can be deployed as software components upon the kernel networking stack, however, the main issue is a bad performance.

User space: The most prominent framework for user space network functions is the Data Plane Development Kit (DPDK).In contrast to kernel space functions, the network interface card will be decoupled from the operating system driver and is directly attached to the application. By that, receiving data will be stored directly within the application's memory space and, as a consequence, will never be copied around. User space drivers usually have a significantly higher performance in terms of throughput and latency as the kernel is not involved [11]. Further, the latency jitter is lower as at least one dedicated CPU core is exclusively reserved and no interrupts or thread changes can occur. The expressiveness of an application in the user space is comparable to the kernel space. However, as the raw packet is hand over to the application, header processing can be performed much better as all header fields are available. Further, advanced QoS schedulers can be implemented as no restrictions of the kernel complicate the process. Same, cryptography functionality can be performed with the assistance of existing user space libraries.

For user space drivers, hardware support by the network interface card is required, which is, however, given by most nowadays network adapters. Therefore, this approach can be seen as hardware-assisted acceleration even though the network function is still running in software and it is no network function offloading to hardware. In summary, user space drivers are more suitable in all categories than kernel space drivers and should be used if a software-based network function implementation is intended.

SR-IOV: Single Root I/O Virtualization (SR-IOV)is a technology that allows to split up one physical resource,*i.e.*, the physical network interface card port or, more concrete, the corresponding PCIe device, into multiple virtual devices. SR-IOV requires hardware support by the network interface card and by the CPU. Towards the software, multiple interface ports occur, which can be configured with different IP addresses, and on each a network function can be bound to. By that, the flexibility in terms of scaling increases, as multiple instances, *e.g.*, as docker container or virtual machines, can be dynamically added and removed.

SR-IoV technology can be applied for kernel and user space network functions in a very similar way. As this approach causes no overhead in software and the hardware implementation should cause no or only little overhead within the network interface card, this approach should not strongly affect the performance. Measurements in related work have shown only a marginally negative impact on the performance by SR-IOV [12]. Kourtis et al. presented similar results and came to the conclusion that SR-IOV in combination with a user space driver results in a very good performance and flexibility for virtualized network functions [13].

NPU: Network Processing Units (NPU) are a special kind of processor architecture optimized for processing packets at high rates. The available functionality of the NPUs varies depending on the vendor, but header processing is generally supported. Cryptographic units, *i.e.*, for encrypting the packets in the 5G RAN, are also often available. However, they are typically made for data center applications and therefore do not offer a sufficient number of queues for shaping the traffic of all connected UEs. In contrast to conventional software network functions, the programmability is more restricted. Nevertheless, flow rules are stored in large DDR3/4 memories and only the most frequent rules are cached within the NPU. This storage strategy allows more session installations compared to a P4 switch. The performance is similar to user space software implementations but uses considerable less energy.

GPU: Graphics Processing Units (GPU) are massive parallel computing accelerators initially build for rendering computer graphics. In addition to their initial task, they are also very well suited for many other computing tasks, including network functions, due to their highly parallel and universal processing architecture. An advantage of the GPUs is their high programmability, which is comparable to standard software applications. Civerchia et al. [14] have shown that this architecture is generally well suited for 5G RAN functionality. However, they observed issues regarding the latency of data transfers. This issue was addressed by Kundel et. al. [15] by proposing a direct data I/O from the network to the GPU. In general, this kind of accelerator benefits from huge memory capacities and fast memory controllers.

FPGA: Field Programmable Gate Arrays (FPGA) can be configured to represent any boolean logic. While the development process is very challenging and configuration updates typically require a restart, they are very powerful for offloading network functions. Building 5G RAN functionality on top of FPGA technology was discussed in literature many times before. For example, Ricart-Sanchez et al. [16] proposed a framework for 5G network slicing upon FPGAs. They showed that FPGAs are well suited for the required QoS functionality in the separated network slices. Depending on the FPGA, different memory technologies for storing flow rules exist. Nevertheless, all currently available FPGAs provide very good performance and deterministic behavior. However, the throughput is reduced compared to P4 switches but still equal or better than software-based approaches. FPGAs can be used to realize either RAN or UPF functionality in 5G networks.

P4 switch: The P4 programming language allows the easy reconfiguration of partially programmable network switches [17]. By that, any networking protocol can be parsed and processed by the switch. Modifications of the payload, *e.g.*, encryption, is not possible. Therefore they are only well suited for accelerating the 5G UPF. The memory for storing flow rules is realized within the switching hardware. This limits the number of parallel GTP sessions but can provide a very high throughput of up to 12.8 $Tbps$. For QoS functionality the limited number of queues is strongly restricting the possible feature set. However, basic QoS management can also be performed within programmable switches [18].

Hybrid Solutions: Combinations of FPGAs and P4 switches have been discussed in literature before. The work "OpenBNG" [19] combined an FPGA and P4 switch to realize a Broadband Network Gateway with FPGA-assisted QoS support. Katta et al. [20] proposed the idea of combining a software network function and a hardware switch to combine the benefits of both. This approach might be promising for 5G network functions.

To summarize, hardware acceleration approaches, in general, are an up-and-coming alternative to pure software-based virtual network functions in 5G, as they can satisfy hard latency and throughput demands in the user plane. The UPF and NAT functionality could be realized on any of the previously mentioned technologies, while RAN functionality requires reconfigurable hardware.

4. Experimental Results and Experiences of HWA in 5G

In this section, we extend our theoretical analysis by a prototypical implementation in our 5G testbed and provide the measurement results as well as our experiences in deploying these approaches. In particular, we investigate different implementations of the User Plane Function (UPF). For that, we focus on four different approaches as Device under Test (DUT): (1) a kernel based software implementation on servers with dedicated NIC ports (KS), (2) a user space software implementation on a server with dedicated NIC ports (US), (3) a kernel space software implementation with virtual SR-IOV NIC ports (KSv) and (4) Intel Tofino based P4 implementation.

For the kernel space implementation (1 and 3), we used the existing free5gc UPF implementation, which operates upon the Linux kernel networking stack. In case of the user space (2), we investigated the `dpdk_gtp_gateway` [21], which is open-source available on Github. As this implementation was not 5G standalone (3GPP release 15 or newer) compliant, we added a GTP protocol extension for QoS flow identifiers. As no 5G standalone compliant P4 implementation of an UPF exists, we build our own prototype implementation on an Intel Tofino switch (4).

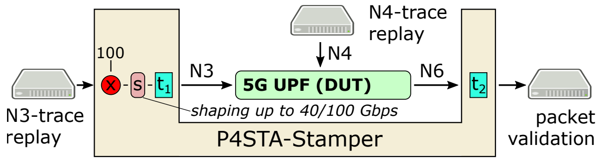

Figure 3. Testbed for performance benchmarking multiple UPF implementations.

4.1. Measurement methodology

The previously introduced end-to-end testbed setup in Section 2 allows capturing all user and control plane messages on all available interfaces. For benchmarking the different UPF implementations in a reproducible manner, these messages can be captured and replayed. In this particular setup, we capture the control messages of the N4 interface and the user plane traffic at the N3 and N6 interfaces. Based on the captured messages, we generate synthetic replay bots behaving similarly to the real end-to-end testbed. In addition to replaying a captured trace, these bots can modify the packets slightly, *e.g.*, replacing timestamps by the current date and time. Replaying of messages has three main advantages: (i) The GTP session id can be assumed to be static which simplifies the setup and server coordination, (ii) the UE cannot enter the idle mode causing unexpected N4 messages, and (iii) the generated load is exactly the same in all experiments ensuring reproducible results.

Our measurement setup is shown in Figure 3. As we focus on the performance criteria of the DUT, concrete throughput, latency, jitter and packet loss, a measurement tool for these metrics is needed. For that, we build upon the open-source framework P4STA, allowing us to measure these performance criteria with a very high accuracy [22].

In our measurement setup in Figure 3, the server on the top replays the N4 traffic, which uses the PFCP protocol to install the forwarding rules in the UPF. After installing the forwarding rules, the server on the left side sends a GTP encapsulated data stream with the previously installed GTP tunnel ID. Before entering the DUT, the packets are replicated, shaped and timestamped by P4STA. By replicating each packet 100 times, a rate of up to 100 $Gbps$ can be generated by software-based load generators such as the Linux tool `tcpreplay`. Timestamping each packet before and

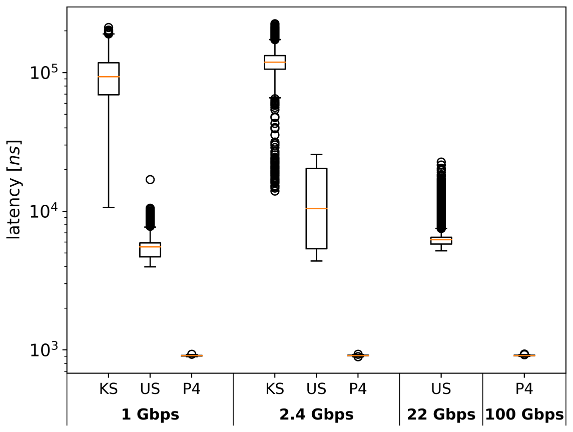

Figure 4. Kernel space (KS), User Space (US) and P4 switch (P4) latency distribution at different input rates.

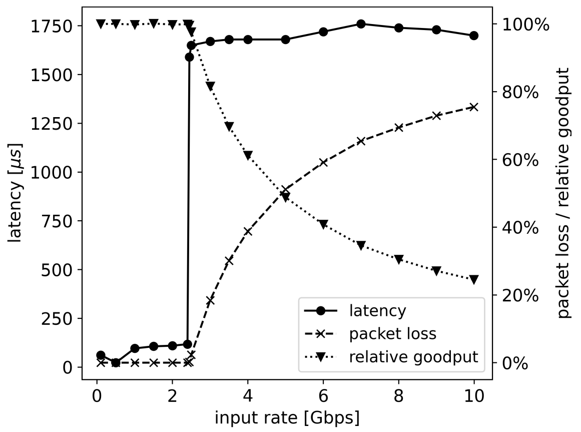

Figure 5. Performance characteristics of the kernel space UPF implementation in dependence of the input rate.

after the DUT allows latency and jitter measurements, as well as loss detection. By shaping the packet flow to a specific rate, the load on the DUT can be varied. The DUT will receive the GTP encapsulated data on the N3 port and processes them. In particular, the UPF parses all headers and removes the GTP encapsulation, causing a packet size reduction of 44 byte. The replayed N3-trace consists of 1000-Byte packets containing IPv4/TCP payload, resulting in packets of 956-Byte on the N6 side. Last, the packet validation server captures the packets and they can be validated for correctness after the experiment.

The server used for benchmarking the software UPF implementation was equipped with a dual-core Intel Xeon E5-2670v3, 256 GB DDR4 RAM and two Intel XL710 network interface cards. The operating system was Ubuntu 18.04 (kernel 5.0.0-23). The CPU clock frequency was configured statically to the maximum frequency, and therefore no clock throttling is allowed. In the case of the user space UPF, the application was pinned to a CPU socket directly attached to the network interface card. In each run, between $1.2\ million$ and $122.5\ million$ test packets were sent into the DUT, and consequently all presented results are significant.

4.2. Results

The measurement results in Figure 4 depict the differences between the investigated UPF implementations. Note that all results only depict data rates that are lower than the maximum throughput of the investigated DUT and therefore, (i) no packet loss occurred and (ii) no large packet queue should be build up within the DUT. The vertical spread of the boxplots is an indicator of the latency jitter.

In the left part of the figure, measurement results for the latency at $1\ Gbps$ input rate are shown. The kernel space implementation has the highest average latency and also a very heavy jitter. The DPDK-based user space implementation latency is around ten times lower and the jitter is reduced. The P4 implementation shows up an average latency of only $\sim 739\ ns$ and no measurable jitter. In total, we can conclude that the P4 UPF shows up a latency around $100x$ times lower than the kernel based implementation. Thus, kernel-based implementations seem to be less suitable for UPF implementations, as strict QoS requirements can generally not be satisfied due to the high latency and heavy jitter.

Increasing the input rate from $1\ Gbps$ to $2.4\ Gbps$ shows similar results. The kernel and user space implementation latency has increased in both scenarios, even though no packet loss occurs. Slightly above $2.4\ Gbps$, the first packet loss occurs in the KS implementation and the latency increases drastically.

While increasing the rate further to $22\ Gbps$, which is close to the maximum throughput of the user space UPF, its average latency remains rather constant. However, it is noteworthy that the distribution is much tighter than the measurement at $2.5\ Gbps$. We could reproduce this behavior but did not find out the reason behind this. We assume that this might be caused by either the PCIe bus of the server or the used NIC, which sends packets to the user space application in batches dependent on the input rate.

Increasing the rate further to $100\ Gbps$, the performance of the P4 UPF still remains unchanged. This confirms the statement from Section 3, hardware can provide very deterministic and good performance.

In order to understand the performance

characteristics of the software implementations of the UPF, we investigate the point of failing in detail. Figure 5 depicts the latency, packet loss and relative goodput for varying input rates for the kernel space implementation with no SR-IOV. Relative goodput describes the number of processed packets, *i.e.*, the number of packets at the N6 interface, compared to the number of packets sent into the DUT, *i.e.*, at the N3 interface. For an input rate between $100\ Mbps$ and $2.4\ Gbps$, the latency remains low and the packet loss is zero. Increasing the input rate from $100\ Mbps$ to $500\ Mbps$, it can be observed that the latency decreases. This is caused neither by the UPF software implementation nor a measurement error. Instead, it is caused by the Linux kernel I/O performance, which is worse if the number of requests is too low. Starting with $500\ Mbps$, the latency is constantly increasing until the first point of packet loss at $2.45 Gbps$. At this point, the latency increases drastically. Increasing the rate even further, the latency and absolute goodput remain constant at $2.29 Gbps$. The difference between the input and output rate is caused by the packet decapsulation functionality within the UPF, reducing the packet size by $44 byte$. The latency is caused by the overflowing receive buffer of the network interface card.

In the case of the DPDK UPF, the observed behavior is similar. The point of failure is at $22.8\ Gbps$ and the latency increases to only $350\ \mu s$ due to the higher rates.

Last, we investigate the impact of SR-IOV. As discussed in Section 3, this technology should have almost no impact on the performance. Surprisingly, our measurement results have shown a slight performance increase with SR-IOV and the kernel space UPF. In case of no SR-IOV enabled (KS) and configured, we observed the first packet loss at $2.45\ Gbps$, while the kernel space driver operating on an SR-IOV interface (KSv) started dropping packets at $2.5\ Gbps$. One reason behind this could be that the kernel module used for the virtual interfaces (Intel *iavf-4.1.1*) was newer than the kernel module of the native port (Intel *i40e-2.7.6*. The research results of Jiuxing Liu [12], performing a similar investigation 11 years ago, have shown slightly different results: Depending on the packet size, they observed an almost equal or even worse throughput in the case of SR-IOV. However, their results are generated with older drivers and hardware.

Note that the presented results in this section were created based on different UPF implementations of different authors with different design goals. Therefore an exact quantitative comparison must be regarded with care. Nevertheless, the performance characteristics and differences can be generalized and are valid for any other implementation of the underlying technology.

4.3. Transferring Results

The presented measurement results show clearly that the P4 switch provides by far the best performance. Therefore we integrated the implementation of this UPF into our end-to-end testbed. In order to provide a minimal working setup, we determined the following functionality to be implemented: (i) The UPF must be able to associate with the SMF via a PFCP association handshake on the N4 interface. (ii) GTP session creation for upstream and downstream GTP sessions. (iii) As real UEs periodically can switch between an idle and active state, which results in changing GTP tunnel IDs, GTP session modifications must be added.

Only rate policing and no traffic shaping was supported due to QoS limitations of the P4 switch. However, extending this implementation by an FPGA with QoS functionality would build a very powerful UPF with all required features. Our end-to-end tests with real UEs show that a smooth operation with open-source components is possible.

5. Conclusion

5G networks are expected to provide very high performance regarding bandwidth and latency to the user equipment. While these performance requirements are demanding by themselves, the correctness validation of developed approaches is a challenge in itself.

In this work, we describe how to setup a working *end-to-end 5G standalone* test network as a basis for research activities focusing on fulfilling these requirements. By that, the functional correctness of novel approaches can be shown. In addition, we analyzed several existing technologies for accelerating 5G network functions based on related work. It turns out that not every approach fits well for every user plane network function, however, hardware acceleration brings great potentials in general.

Last, we evaluated various UPF implementations (userspace, kernel, hardware) to analyze their performance regarding bandwidth, latency, jitter and packet loss. In general, our results show that the selection of the underlying technology strongly influences the network function performance. The outcomes clearly show that hardware acceleration in the user plane is a key asset to fulfill the high QoS requirements of 5G and beyond networks.

5.1. Future work

In future work, we will focus on bringing the previously shown approaches into production. For that, we will set up a large-scale 5G standalone campus testbed with $\pm 40.000\ m^2$, multiple O-RAN base stations and many stationary and moving UEs. This setup will rely on open-source components and a P4 switch UPF implementation with FPGA support for QoS functionality. For that, we will improve the feature support of our P4-UPF implementation and seek an open-source publication.

Acknowledgment

This work has been funded by the Federal Ministry of Education and Research (BMBF, Germany) within the Software Campus Project ”5G-PCI” and by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 MAKI. Further, this work has been supported by Deutsche Telekom through the Dynamic Networks 9 project. We would like to thank all contributors of open-source projects this work builds upon.

References

[1] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, “Software-defined networking (sdn): Layers and architecture terminology,” tech. rep., 2015. RFC 7426.

[2] 3GPP, “System architecture for the 5G System (5GS),” Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 12 2020. Version 15.12.0.

[3] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, “A formal analysis of 5g authentication,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’18, (New York, NY, USA), p. 1383–1396, Association for Computing Machinery, 2018.

[4] Chi Chang, Fu-Cheng Chen, Jyh-Cheng Chen, et al., “free5gc.” https://github.com/free5gc/free5gc, 2019. Accessed on: 09. June 2021.

[5] Amarisoft, “Amari callbox.” https://www.amarisoft.com/. Accessed on: 02. June 2021.

[6] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, “Openairinterface: A flexible platform for 5g research,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, p. 33–38, Oct. 2014.

[7] L. Gavrilovska, V. Rakovic, and D. Denkovski, “From cloud ran to open ran,” *Wireless Personal Communications*, pp. 1–17, 2020.

[8] Kamailio, “Kamailio - the open source sip server.” https://github.com/herlesupreeth/kamailio, 2001. Accessed on: 01. June 2021.

[9] L. Nobach, J. Blendin, H.-J. Kolbe, G. Schyguda, and D. Hausheer, “Bare-metal switches and their customization and usability in a carrier-grade environment,” in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pp. 649–657, 2017.

[10] J. Prados-Garzon, O. Adamuz-Hinojosa, P. Ameigeiras, J. J. Ramos-Munoz, P. Andres-Maldonado, and J. M. Lopez-Soler, “Handover implementation in a 5g sdn-based mobile network architecture,” in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–6, 2016.

[11] G. P. Katsikas, T. Barbette, D. Kostić, R. Steinert, and G. Q. M. Jr., “Metron: NFV service chains at the true speed of the underlying hardware,” in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, (Renton, WA), pp. 171–186, USENIX Association, Apr. 2018.

[12] J. Liu, “Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support,” in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pp. 1–12, 2010.

[13] M.-A. Kourtis, G. Xilouris, V. Riccobene, M. J. McGrath, G. Petralia, H. Koumaras, G. Gardikis, and F. Liberal, “Enhancing vnf performance by exploiting sr-iov and dpdk packet processing acceleration,” in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 74–78, 2015.

[14] F. Civerchia, M. Pelcat, L. Maggiani, K. Kondepu, P. Castoldi, and L. Valcarenghi, “Is opencl driven reconfigurable hardware suitable for virtualising 5g infrastructure?,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 849–863, 2020.

[15] R. Kundel, T. Burkert, C. Griwodz, and B. Koldehofe, “Chaining of hardware accelerated virtual network functions in pcie environments,” in *Proceedings of the 20th International Middleware Conference Demos and Posters*, Middleware ’19, (New York, NY, USA), p. 13–14, Association for Computing Machinery, 2019.

[16] R. Ricart-Sanchez, P. Malagon, A. Matencio-Escolar, J. M. Alcaraz Calero, and Q. Wang, “Toward hardware-accelerated qos-aware 5g network slicing based on data plane programmability,” *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 1, p. e3726, 2020. e3726 ett.3726.

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, “P4: Programming protocol-independent packet processors,” vol. 44, pp. 87–95, Association for Computing Machinery, 2014.

[18] R. Kundel, A. Rizk, J. Blendin, B. Koldehofe, R. Hark, and R. Steinmetz, “P4-codel: Experiences on programmable data plane hardware,” in *IEEE International Conference on Communications (ICC)*, pp. 1–6, 2021.

[19] R. Kundel, L. Nobach, J. Blendin, W. Maas, A. Zimber, H.-J. Kolbe, G. Schyguda, V. Gurevich, R. Hark, B. Koldehofe, and R. Steinmetz, “OpenBNG: Central office network functions on programmable data plane hardware,” *International Journal of Network Management*, 2021.

[20] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, “Cacheflow: Dependency-aware rule-caching for software-defined networks,” in *Proceedings of the Symposium on SDN Research*, pp. 1–12, 2016.

[21] C. Chang, “Kamailio - the open source sip server.” https://github.com/edingroot/dpdk_gtp_gateway, 2016. Accessed on: 01. June 2021.

[22] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, and B. Koldehofe, “P4STA: High performance packet timestamping with programmable packet processors,” in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium(NOMS)*, IEEE, 2020.