Leveraging Programmable Dataplanes for a High Performance 5G User Plane Function

Abhik Bose[*]
Department of Computer Science & Engineering Indian Institute of Technology Bombay, India
abhik@cse.iitb.ac.in

Diptyaroop Maji[†]
Department of Computer Science & Engineering Indian Institute of Technology Bombay, India
diptyaroop@cse.iitb.ac.in

Prateek Agarwal
Department of Computer Science & Engineering Indian Institute of Technology Bombay, India
prateekag@cse.iitb.ac.in

Nilesh Unhale
Department of Computer Science & Engineering Indian Institute of Technology Bombay, India
nileshunhale@cse.iitb.ac.in

Rinku Shah
Department of Computer Science & Engineering Indian Institute of Technology Bombay, India
rinku@cse.iitb.ac.in

Mythili Vutukuru[‡]
Department of Computer Science & Engineering Indian Institute of Technology Bombay, India
mythili@cse.iitb.ac.in

ABSTRACT

Emerging 5G applications require a dataplane that has a high forwarding throughput and low processing latency, in addition to low cost and power consumption. To meet these requirements, the state-of-the-art 5G User Plane Functions (UPFs) are built over high performance packet I/O mechanisms like the Data Plane Development Kit (DPDK), and further offload some functionality to programmable dataplane hardware. In this paper, we design and implement several standards-compliant UPF prototypes, beginning with a software-only DPDK-based UPF, progressing to designs which offload different functions to programmable hardware. We evaluate and compare the performance of these designs, to highlight the costs and benefits of these offloads. Our results show that offload techniques employed in prior work help improve performance in certain scenarios, but also have their limitations. Overcoming these limitations and fully realizing the power of programmable hardware requires offloading more complex functionality than is done today. Our work presents a preliminary implementation towards a comprehensive programmable dataplane-accelerated 5G UPF.

[*]Abhik Bose and Diptyaroop Maji are student authors with equal contribution.

APNet 2021, June 24–25, 2021, Shenzhen, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8587-9/21/06...$15.00
https://doi.org/10.1145/3469393.3469400

CCS CONCEPTS

• **Networks** → **In-network processing**; **Programmable networks**; *Network performance analysis*; **Mobile networks**.

KEYWORDS

5G core, cellular networks, programmable networks, DPDK

ACM Reference Format:
Abhik Bose, Diptyaroop Maji, Prateek Agarwal, Nilesh Unhale, Rinku Shah, and Mythili Vutukuru. 2021. Leveraging Programmable Dataplanes for a High Performance 5G User Plane Function. In *5th Asia-Pacific Workshop on Networking (APNet 2021) (APNet 2021), June 24–25, 2021, Shenzhen, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3469393.3469400

1 INTRODUCTION

The growth in mobile services and subscribers has resulted in an exponential increase in mobile signaling and data traffic [3, 41, 47, 59]. The upcoming 5G standards aim to support applications with diverse traffic characteristics and requirements like enhanced mobile broadband, dense deployments of IoT devices, self-driving cars, and AR/VR [34, 54]. These applications require high throughput, very low processing latencies, and stringent Quality-of-Service (QoS) enforcement. The mobile packet core, which connects the radio access network to external networks, comprises of control plane components that process signaling messages and a dataplane that forwards user traffic. The User Plane Function (UPF) is the main entity in the dataplane of the future 5G mobile packet core, and has a significant impact on the performance that users are going to perceive with 5G.

Most state-of-the-art UPFs are built as multicore-scalable software packet processing appliances running over commodity servers, and process traffic using a high performance packet I/O mechanism like the Data Plane Development Kit (DPDK) [20]. However, given the stringent performance requirements of 5G networks [34, 54], and ever increasing network speeds running into hundreds of Gbps, offloading some UPF packet processing to programmable dataplane hardware can lead to improved performance, along with cost and power savings. We establish this benefit of offload in Table 1, which shows the data forwarding capacity per unit cost/power for various programmable hardware platforms as well as a general purpose server CPU core that all run a UPF. We obtain this table as follows: the UPF throughput values for the first two columns (single core server and Agilio CX 2x10GbE) were measured using our standards-compliant UPF implementations (§3), while we assumed that the other hardware platforms were capable of handling offloaded UPF processing at linerate (a reasonable assumption from our experience with one platform). The cost and power consumption were obtained from the hardware specifications. We see from the table that offloading UPF processing to programmable hardware can result in significant cost and power savings across a wide variety of programmable dataplane platforms. The idea of accelerating UPF using programmable hardware is not new—prior work has proposed offloading the logic of steering packets to multiple CPU cores of the UPF [10, 12, 15, 42], as well as the dataplane forwarding itself [9, 12, 15, 24, 26]. However, to the best of our knowledge, none of the existing works systematically enumerates all the possible ways in which UPF functionality can be offloaded to programmable hardware, nor do they precisely quantify the costs and benefits of such offloads.

In this work, we begin with an industry-grade software based UPF built over DPDK, and progressively offload functions to programmable hardware, to come up with several different UPF prototypes (§3). We then measure the throughput and latency characteristics of such UPFs to analyze the pros and cons of offloading UPF functionality to programmable hardware (§4). For example, we find that offloading the logic of steering packets to the multiple CPU cores reduces UPF processing latency by up to 37% and increases throughput by 45%. However, performing packet steering in hardware is less flexible than doing so in software, and performs badly during scenarios involving dynamic scaling and skewed traffic distribution across users. Another interesting observation is that, while forwarding data directly from the programmable dataplane hardware (rather than via userspace) leads up to 24% lower latency in the dataplane as expected, it significantly worsens the control plane performance. This is because the UPF continues to process signaling messages (from the control plane that configures forwarding rules) within userspace

	Server [22, 23]	Agilio CX SmartNICs [19, 55] [56]	[57]	[58]	Tofino switch [21]
Mpps per USD	0.03	0.33	0.28	0.24	0.4
Mpps per Watt	0.14	2.96	2.96	2.96	8.52

Table 1: Performance per unit cost and power.

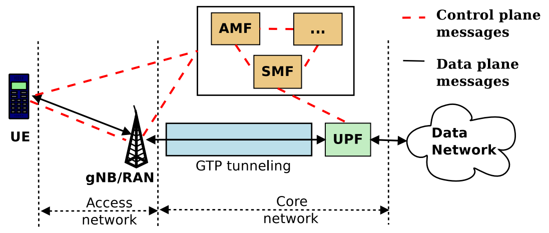

Figure 1: 5G Architecture.

itself, and in this split architecture, the communication between the UPF software and the programmable hardware becomes the bottleneck.

In this paper, we posit that truly leveraging the performance of programmable hardware to accelerate the 5G dataplane requires the UPF to process and respond to signaling messages from the hardware itself, in order not be limited by the hardware configuring capacity of userspace software. This is challenging to do for several reasons, including the variable-sized message formats of UPF signaling messages. This paper provides a preliminary implementation that solves some of these challenges, and provides initial results that show the promise of offloading these complex UPF functions to programmable hardware. Our work paves the way towards a high-performance 5G UPF that fully leverages the power of programmable dataplanes.

2 BACKGROUND & RELATED WORK

5G architecture. Figure 1 shows a high-level overview of the 5G architecture [5]. A 5G network consists of the wireless radio access network (RAN), which includes the User Equipment (UE) and the Base Station/g-NodeB (gNB), and the wired packet core network. In the control plane of the 5G core, the Access and Mobility Function (AMF) deals with registration and mobility management, and the Session Management Function (SMF) manages the UE's data sessions. The data plane comprises of one or more User Plane Functions (UPFs) that forward user data through the packet core. The control and data plane components communicate using Packet Forwarding Control Protocol (PFCP) messages that are exchanged between the SMF and UPF over UDP [4]. In the dataplane, a UE's IP datagram packets are encapsulated in GPRS Tunnelling Protocol (GTP) headers when transiting through the packet core; tunnelling aids easy mobility among other things. The GTP-encapsulated IP datagrams are transmitted over a UDP link between the gNB and the UPF.

The UPF is responsible for encapsulating downlink packets entering the core with GTP headers and correspondingly decapsulating uplink packets leaving the core. The UPF is also responsible for usage reporting and charging, and enforcing Quality-of-service (QoS), e.g., via rate limiting.

This paper deals with UPF implementation, so we describe the internal state and data structures at the UPF in more detail. A UE sets up one or more "sessions" to forward data through the core, and each data session will cause the SMF to install one or more Packet Detection Rules (PDRs) at the UPF via PFCP messages. A PDR helps associate an incoming data packet to a session based on its GTP/IP packet headers. A PDR has several types of actions associated with it, that instruct the UPF on how to handle the packet. For example, the Forward Action Rules (FARs) specify the forwarding behavior of the packet (e.g., GTP tunnel identifiers for encap), and the QoS Enforcement Rules (QERs) specify the QoS processing to be performed. An example of a field in the QER is the Aggregate Maximum Bit-Rate (AMBR) of the session. On receiving a PFCP message from the SMF, the UPF creates/updates the required rules, and sends an acknowledgement back to the SMF. On receiving a data packet, the UPF first identifies the matching PDR. For packets belonging to a valid session, the UPF executes forwarding and QoS behavior specified by the FARs and QERs linked to that PDR, in addition to updating usage and charging counters. Packets belonging to "oversubscribed" sessions that exceed the rate limit specified in the QoS rules are buffered for an appropriate duration, and scheduled for transmission suitably.

Programmable dataplanes. Programmable dataplanes allow networking hardware to be easily programmed to perform complex functions, via code written in a high-level language like P4 [25]. Packet processing pipeline specifications written in P4 can be compiled to a variety of programmable dataplanes, e.g., programmable hardware ASICs [1, 2, 46, 48], NPUs [8, 56], and FPGAs [7, 11]. The programmable hardware platforms have several limitations put in place, in order to ensure linerate processing. They have limited expressiveness in terms of the supported instruction set and programming constructs, and lack dynamic data-structures. The packets cannot stall during the switch pipeline processing—they have to be either forwarded or dropped. The amount of onboard memory on such hardware is limited (~few tens of MBs). Despite these limitations, researchers have observed substantial performance benefits by offloading applications to programmable hardware, via creative solutions that address the hardware limitations [27, 29–33, 36, 37, 39, 40, 43–45, 51, 53, 61].

State-of-the-art UPFs. Most production grade UPFs are built over kernel-bypass techniques like DPDK to achieve high dataplane throughput in software. Metaswitch [10] uses a specialized processing engine (CNAP) in the software itself to achieve high throughput. Some UPFs also use programmable hardware or specialized processing engines to offload some part of the UPF processing to hardware. Few proposals [12, 15, 24, 26] offload the GTP encap/decap based forwarding to hardware, while some [42] offload packet steering to cores via deep packet inspection (DPI) of the inner IP header. Kaloom [9] offloads a subset of QoS processing (bit rate policing) along with GTP processing to the programmable hardware. Our work explores more offload based designs than those considered in prior work, and systematically analyzes the costs and benefits of the offloads. TurboEPC [52] offloads the subset of 4G core signaling messages to the programmable hardware, while our work explores the signaling message offload problem in the context of 5G which has a different architecture.

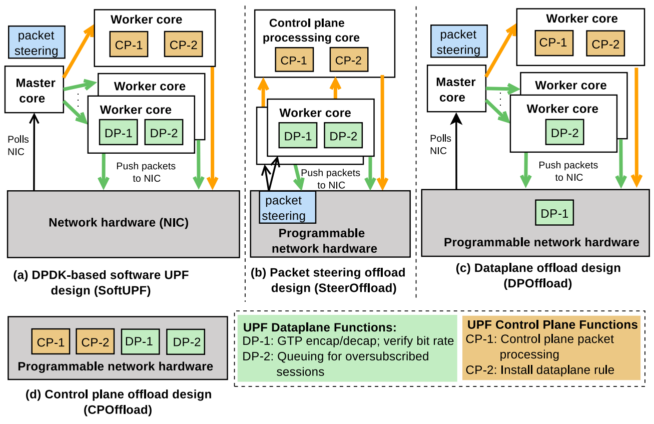
packet steering
Master core
Worker core
CP-1
CP-2
Worker core
Worker core
DP-1
DP-2
Polls NIC
Push packets to NIC
Network hardware (NIC)
(a) DPDK-based software UPF design (SoftUPF)
Control plane processsing core
CP-1
CP-2
Worker core
DP-1
DP-2
Push packets to NIC
packet steering
Programmable network hardware
(b) Packet steering offload design (SteerOffload)
packet steering
Master core
Worker core
CP-1
CP-2
Worker core
Worker core
DP-2
Polls NIC
Push packets to NIC
DP-1
Programmable network hardware
(c) Dataplane offload design (DPOffload)
CP-1
CP-2
DP-1
DP-2
Programmable network hardware
(d) Control plane offload design (CPOffload)
UPF Dataplane Functions:
DP-1: GTP encap/decap; verify bit rate
DP-2: Queuing for oversubscribed sessions
UPF Control Plane Functions
CP-1: Control plane packet processing
CP-2: Install dataplane rule

Figure 2: 5G User Plane Function designs.

3 DESIGN & IMPLEMENTATION

This section describes the various UPF prototypes compared in this paper, beginning with a software-based UPF, and progressively moving towards UPFs that offload more functionality to programmable hardware, as shown in Figure 2.

3.1 DPDK-based software UPF

We begin with describing our purely software DPDK-based UPF (Figure 2a) that is representative of the most common UPF design used in production networks today. Our implementation is based on a fully standards-compliant UPF obtained from [16, 17]. Our UPF supports GTP-based forwarding and AMBR-based QoS enforcement (using a variant of the algorithm in [50]), among other features. Our implementation spans 6.5K lines of code. Our UPF has a pipeline-based design, with multiple master and worker threads, each pinned to separate CPU cores. The master threads receive packets from the NIC via the polling-based DPDK APIs, and distribute them to the worker threads for further processing. PFCP packets and dataplane packets are processed by separate worker cores. The inter-core communication between the master and workers is done via the lockless shared rings provided by DPDK, for efficiency and high performance. The worker threads continuously poll the shared rings for received packets, process them, and transmit the output.

When steering packets to worker cores, we would like to ensure that the traffic of a UE is processed by the same worker, in order to avoid splitting the state of a particular UE (forwarding rules, buffered packets, and so on) across multiple workers. This steering is achieved by using the hash over the TCP/IP headers of the"inner" IP datagram (the datagram originated by the UE, which has been encapsulated within GTP for transit through the core) to partition traffic to worker cores. Note that we cannot simply use a hash over the "outer" UDP/IP header fields because dataplane traffic of all UEs between a gNB-UPF pair arrives on the same UDP link, so the outer IP header fields cannot be used to differentiate UEs. Most modern NICs have the capability to distribute packets to multiple CPU cores via RSS [60]; however, the set of header fields used for RSS is restricted to the outer IP header fields. Therefore, a purely software-based DPDK design cannot rely on the NIC to perform packet steering based on the inner IP header fields, and must perform this steering in software. (RSS based on outer header fields is still useful to distribute traffic to multiple master cores for performance scaling.) Packet steering in software also allows the UPF to efficiently rebalance load across worker cores in case some worker cores are more overloaded than others, and to dynamically scale to more worker cores quickly on demand. In our design, the master cores periodically monitor the queue lengths of the lockless rings shared with the worker cores, and reassigns UEs across workers if it finds that a worker core is overloaded (as inferred from a persistently high queue length), and another underloaded. If all worker cores are overloaded, the master can spawn a worker on new CPU core (when available). We have only implemented a simple load balancing algorithm, but more complex algorithms that assign special classes of UEs (e.g., high priority UEs) to specific worker cores are also possible.

3.2 Packet steering offload

In our next design (Figure 2b), the steering of UE traffic to cores happens not in software but within the NIC itself. This design relies on advanced NICs that allow RSS based on inner IP headers. For example, Intel NICs support the Dynamic Device Personalization [49] feature on 40Gbps+ NICs, which enables parsing of the GTP header and inner TCP/IP header fields for hash computation. Because the input to the hash function now contains the UE's IP address and GTP tunnel identifier, the packets of a UE are redirected to the same receive queue and CPU core via RSS. The multiple worker threads of the DPDK-based software UPF are assigned dedicated hardware receive queues and directly receive traffic from their corresponding queues. The worker threads then process the received packets in a run-to-completion model. The control plane traffic is redirected to (and processed on) dedicated cores as before. This design represents the simplest possible offload that can be done to programmable hardware, and is used by state-of-the-art UPFs [10, 12, 15, 42].

Offloading packet steering to hardware provides higher throughput and lower latency, due to minimal inter-core communication in software and faster hash computation in hardware. However, this design is also less flexible as we have lesser control on assigning UEs to cores. For example, the DPDK i40e poll mode driver [13] does not support dynamic load balancing by remapping queues to cores based on load. Further, dynamic scaling is also complicated by the fact that one needs to stop and reconfigure the port in order to change the number of receive queues.

3.3 Dataplane offload

Our next design (Figure 2c) offloads the complete dataplane processing of the UPF to a programmable hardware-based NIC or switch that can intercept packets destined to the software UPF. Without loss of generality, we assume that the dataplane is offloaded to a programmable NIC; our current implementation uses the Agilio CX 2x10GbE smartNIC [56]. In this design, the on-NIC programmable parser extracts 5G header fields from incoming packets. The control plane PFCP messages are directed to the software UPF and processed by the control plane worker thread. After processing the PFCP messages, the worker thread communicates with the NIC firmware to install/update session forwarding rules in the match-action tables. The worker-NIC communication happens via our custom C++ library that invokes the NIC's Thrift API [18] to install hardware rules. The incoming dataplane packets are matched against these rules for suitable forwarding. Note that the number of concurrent active user sessions that can be supported by such a system is inherently limited by the amount of memory available to store forwarding rules in the hardware; our current prototype can store the forwarding state of only 10K users but higher-end programmable hardware [46] can do more.

Prior work has also proposed similar designs that offload GTP-based forwarding to programmable hardware [9, 15, 24, 26], but does not provide much detail on how rate limiting for QoS is handled in the offloaded design. In our design, all traffic that is within the QoS-prescribed rate limit is forwarded directly from hardware, and traffic that cannot be transmitted immediately is sent to the userspace for buffering. Our implementation presently supports enforcement of session-wide AMBR (aggregate maximum bit rate), which requires us to rate limit the aggregate traffic of a user's data session (across all flows) to a maximum value and queue up the traffic that exceeds this limit. One key challenge we had to overcome in this implementation was to identify the packets that exceeded a session's rate limit. Our initial implementation started with using P4 register arrays (stateful memory available for packet processing in programmable hardware) to track incoming rates of various sessions. Because these hardware registers can be accessed concurrently by multiple packets across different stages of the packet processing pipeline, we must use mutual exclusion in accessing and updating these registers. Unfortunately, we found that updating P4 registers under mutual exclusion significantly impacted the performance of our UPF (dropping to 350 Mbps from the linerate of 10Gbps). To overcome this limitation, we moved to using the P4 meter primitive [28, 38] to identify packets exceeding the rate limit. P4 meter is implemented as a device specific extern, that sets the "color" of a packet in its metadata to different values based on whether the session's rate exceeds a pre-defined rate limit. While there was no slowdown due to P4 meters, we found that the hardware dataplane can no longer compute the time that a packet needs to be buffered for correctly, because the meter interface does not expose any rate calculations beyond the color. Therefore, once a session exceeds its rate limit, we forward all subsequent packets of that session to userspace, and let the software handle the session's dataplane processing.

3.4 Control plane offload

In the dataplane offload design (Figure 2c), signaling messages that configure the dataplane are still handled in userspace. As a result, a workload with a large number of signaling messages may slow down the hardware dataplane, due to a bottleneck at the software that configures hardware rules. To overcome this limitation, we implement a UPF design which offloads signaling message processing also to programmable hardware (Figure 2d). In this design, we move away from storing forwarding/QoS rules of a session in match-action tables (which need to be updated via the controller running in userspace), and store them instead in P4 register arrays. The register data structure supports update operations to register array state within the dataplane pipeline at linerate, but cannot perform key-based lookup. When a control plane PFCP message for a particular session arrives, we map the 64-bit session identifier to a 24-bit index into the register array. We can then access or manipulate the corresponding session state directly from within the dataplane, without requiring the intervention of the userspace controller. Since our index calculation is (currently) not collision-free, we store additional state in the register array to validate if we are accessing the correct session. In case of a collision or session mismatch, the control plane message is forwarded to the userspace for processing. We also make a few simplifying assumptions when parsing PFCP messages in our current implementation. For example, we assume a PFCP header with fixed structure that fits within the hardware memory, whereas in practice, the PFCP header comprises of a variable length component that consists of recursive PDR, FAR, and QER (§2) headers. We defer the comprehensive handling of complex PFCP messages to future work.

4 EVALUATION

We now evaluate our various UPF designs and quantify the performance gains of offloading UPF functionality.

Experiment Setup. All our experiments run on three servers with Intel Xeon processors (2.2Ghz, 24 cores) and 128GB RAM. The first server runs a RAN emulator/load generator that generates emulated traffic to the UPF, comprising of control plane PFCP messages and uplink/downlink dataplane traffic from a large number of UEs. The RAN emulator runs over DPDK to generate traffic at a high rate and we ensure that it can generate enough load to saturate the control/data plane of the UPF in all experiments. The second server in our setup runs one of the versions of our UPF. The third server runs a sink application that generates downlink traffic and consumes uplink traffic from the RAN emulator. Our RAN emulator and sink span 12K lines of code. We use two sets of NICs in our experiments. For experiments which evaluate the impact of offloading packet steering, we connect the servers using Intel XL710 i40e 40Gbps NICs that are capable of offloading packet steering. For the rest of the experiments, we use Agilio CX 2x10GbE programmable NICs [56].

Parameters and metrics. We generate different traffic mixes for dataplane experiments by varying the packet sizes, across $64B$, $1400B$ and a typical traffic mix ("IMIX") found in real user traffic [14]. We use an uplink to downlink traffic ratio of 1:2 [6, 35]. Control plane traffic consists of sets of session creation, modification, and deletion PFCP messages processed one after the other—we refer to this set as a control plane "procedure". All results reported are for an experiment conducted for 300 seconds, unless mentioned otherwise. We show error-bars that denote the maximum and minimum values where applicable. The performance metrics measured are the dataplane throughput (bps or pps), control plane throughput (procedures/sec), and average end-to-end response latency (RTT, where the response is mirrored by sink for data packets, and is generated by the control plane processing for signaling packets), as measured at UPF saturation. For experiments evaluating the effect of offloading packet steering, we send data from $65K$ concurrent users. For experiments involving dataplane and control plane offload to smartNICs, we emulate $1K$ concurrent users.

Packet Steering Offload. We begin by evaluating the impact of packet steering to programmable NICs, and compare a pure software-based UPF (§3.1) with a UPF that offloads packet steering to programmable hardware (§3.2). Figure 3 shows the throughput and latency of both the designs when

UPF design	Latency (in μs) with packet size		
	64B	IMIX	1400B
SoftUPF	138	176	294
DPOffload	130	140	222

Table 2: SoftUPF vs. DPOffload: Dataplane latency.

Performance metric	SoftUPF	DPOffload	CPOffload
Throughput (messages/sec)	5.1K	666	2.05M
Latency (μs)	113	1646	26

Table 3: Control plane performance.

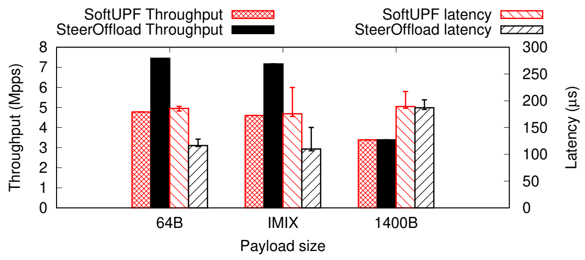

Figure 3: SoftUPF vs. SteerOffload: Performance.

we dedicate 2 cores for dataplane processing—one for uplink traffic and another for downlink traffic. As expected, we find that offloading packet steering leads to 45% higher throughput and up to 37% lower latency (except when both designs saturate linerate at maximum packet size), due to lesser overhead of intercore communication and packet steering in the offloaded design. However, the performance gains due to this offload also come with a loss in flexibility. We consider a scenario where a UPF has to dynamically scale the number of cores it is running on due to an increase in incoming traffic. Figure 4 shows the UPF throughput during scaleup, and we see the adverse impact of having to restart the port and reconfigure hardware queues in the offload design. Next, we configure our RAN emulator in such a way that 20% of the UEs processed by a single worker core generate a sudden burst of traffic, and Figure 5 shows how the offload design suffers high latency due to its inability to balance load across cores in the presence of such "heavy-hitter" UEs. *In summary, we find that offloading packet steering to NICs leads to significant performance gains, but comes at a loss of flexibility that can hurt operators that expect to perform dynamic scaling frequently or see skewed traffic across users.*

Dataplane Offload. Next, we evaluate the benefits of offloading dataplane processing to programmable hardware by comparing the performance of the pure software UPF (§3.1) with one that offloads dataplane processing to a programmable NIC (§3.3). We find that both designs saturate the 10Gbps linerate in our setup, and Table 2 shows end-to-end latency measurements for the various traffic scenarios. As expected, dataplane offload improves latency by up to 24%.

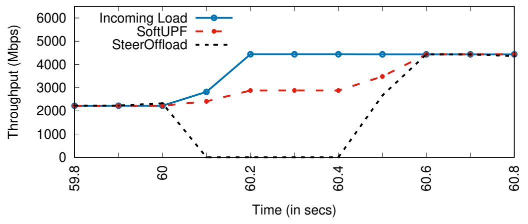

Figure 4: SoftUPF vs. SteerOffload: Dynamic scaling.

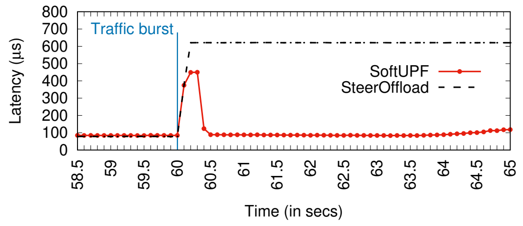

Figure 5: SoftUPF vs. SteerOffload: Heavy hitters.

Control Plane Offload. Next, we compare the control plane performance of three designs: the pure software UPF (§3.1), the design that offloads only the dataplane forwarding to programmable hardware but processes control plane packets in userspace (§3.3), and our preliminary implementation of a UPF that offloads even control plane processing to programmable hardware (§3.4). Table 3 shows the average rate of processing signaling messages, and the processing latency, for all three designs. We observe that only offloading dataplane processing to hardware has a significant adverse impact on control plane performance, because the performance is limited by the overheads of control plane message processing in software and the subsequent communication with NIC firmware. Therefore, the design that offloads only dataplane processing has 86% lower control plane throughput and 15× higher control plane latency, as compared to the pure software UPF. However, the UPF that offloads control plane processing has 402× higher throughput as compared to the pure software design, and $3K\times$ higher throughput as compared to the dataplane offload design. It also has 77% latency reduction compared to the pure software design, and 98% latency reduction compared to the dataplane offload design. This result points to the conclusion that offloading GTP-based dataplane processing alone to programmable hardware is likely to cause additional performance concerns for signaling message processing, especially for future 5G networks that expect to have frequent signaling with IoT devices. *A comprehensive programmable dataplane accelerated UPF must offload signaling message processing to hardware as well, in addition to offloading dataplane processing.*

5 CONCLUSION AND FUTURE WORK

Our work evaluates several designs of the 5G user plane function (UPF) that leverage programmable dataplane hardware in different ways to improve performance. Our evaluation highlights the costs and benefits of each offload strategy, and provides lessons for future research in this area. We find that UPF offload designs proposed in prior work improve performance, but hurt flexibility in dynamic workload scenarios, and cannot process signaling traffic efficiently. To fully realize the power of programmable hardware offload for UPFs, we need to overcome the technical challenge of being able to parse complex 5G signaling messages in hardware. Our work is a first step towards using programmable dataplanes to realize the vision of 5G.

ACKNOWLEDGMENTS

We thank the 5G testbed project, funded by the Department of Telecommunications, Govt. of India, for access to 5G UPF.

REFERENCES

[1] Cisco highlights next big switch. (2013). https://www.biztechafrica.com/article/cisco-announces-next-big-switch/5448/

[2] Cavium Xpliant ethernet switch product line. (2015). https://people.ucsc.edu/~warner/Bufs/Xpliant-cavium.pdf

[3] Ericsson Mobility Report. (2016). http://mb.cision.com/Main/15448/2245189/661253.pdf

[4] 3GPP Ref #: 29.244. System architecture for the 5G System (5GS). (2017). https://www.3gpp.org/ftp/Specs/archive/29$_s$eries/29.244/

[5] 3GPP Ref #:23.501. System architecture for the 5G System (5GS). (2017). https://www.3gpp.org/ftp/Specs/archive/23$_s$eries/23.501/

[6] Minimum requirements related to technical performance for IMT-2020 radio interface(s). (2017). https://www.itu.int/dms$_p$ub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf

[7] Altera. (2019). https://www.mouser.in/manufacturer/altera/

[8] EZchip. (2019). https://www.radisys.com/partners/ez-chip

[9] The Kaloom 5G User Plane Function (UPF). (2019). https://www.mbuzzeurope.com/wp-content/uploads/2020/02/Product-Brief-Kaloom-5G-UPF-v1.0.pdf

[10] Lighting Up the 5G Core with a High-Speed User Plane on Intel Architecture. (2019). https://builders.intel.com/docs/networkbuilders/lighting-up-the-5g-core-with-a-high-speed-user-plane-on-intel-architecture.pdf

[11] Xilinx. (2019). https://www.xilinx.com/

[12] 5G User Plane Function (UPF) - Performance with ASTRI. (2020). https://networkbuilders.intel.com/solutionslibrary/5g-user-plane-function-upf-performance-with-astri-solution-brief

[13] I40E Poll Mode Driver. (2020). https://doc.dpdk.org/guides/nics/i40e.html

[14] Internet Mix (IMIX) Traffic. (2020). https://en.wikipedia.org/wiki/Internet$_M$ix

[15] Optimizing UPF performance using SmartNIC offload. (2020). https://mavenir.com/wp-content/uploads/2020/11/Mavenir$_U$PF$_S$olution$_B$rief.pdf

[16] 5G testbed at IIT Bombay. (2021). https://www.cse.iitb.ac.in/~5gtestbed/

[17] 5G testbed, DoT, Govt. of India. (2021). https://5gtestbed.in/

[18] Apache Thrift - Home. (2021). https://thrift.apache.org/

[19] Cost of Agilio CX SmartNICs. (2021). https://colfaxdirect.com/store/pc/showsearchresults.asp?IDBrand=38&iPageSize=50

[20] DPDK Overview. (2021). https://doc.dpdk.org/guides/prog$_g$uide/overview.html

[21] Edgecore Networks AS9516-32D (Tofino-2). (2021). https://stordirect.com/shop/switches/400g-switches/edgecore-networks-as9516-32d/

[22] Intel Xeon E5-2670 V3 Dodeca-core (12 Core) 2.30 Ghz Processor. (2021). https://www.amazon.com/Intel-Xeon-E5-2670-Dodeca-core-Processor/dp/B00NFA7ILQ#HLCXComparisonWidget$_f$eature$_d$iv

[23] Intel XL710-BM2 Dual-Port 40G QSFP+ PCIe 3.0 x8, Ethernet Network Interface Card. (2021). https://www.fs.com/products/75604.html

[24] Ashkan Aghdai et al. 2018. Transparent Edge Gateway for Mobile Networks. In *IEEE 26th International Conference on Network Protocols (ICNP)*.

[25] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Computer Communication Review* 44 (2014).

[26] Carmelo Cascone and Uyen Chau. 2018. Offloading VNFs to programmable switches using P4. In *ONS North America*.

[27] Eyal Cidon, Sean Choi, Sachin Katti, and Nick McKeown. 2017. AppSwitch: Application-layer Load Balancing Within a Software Switch. In *Proceedings of the Asia-Pacific Workshop on Networking (APNet)*.

[28] Edgar Costa. P4 Meter Application. (2020). https://github.com/nsg-ethz/p4-learning/tree/master/examples/meter

[29] Huynh Tu Dang et al. 2018. Consensus for Non-Volatile Main Memory. In *IEEE 26th International Conference on Network Protocols (ICNP)*.

[30] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. 2015. NetPaxos: Consensus at Network Speed. In *Proceedings of the the Symposium on SDN Research (SOSR)*.

[31] Hans Giesen, Lei Shi, John Sonchack, Anirudh Chelluri, Nishanth Prabhu, Nik Sultana, Latha Kant, Anthony J McAuley, Alexander Poylisher, André DeHon, et al. 2018. In-network computing to the rescue of faulty links. In *Proceedings of the 2018 Morning Workshop on In-Network Computing (NetCompute)*.

[32] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. 2018. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research (SOSR)*.

[33] Rob Harrison, Shir Landau Feibish, Arpit Gupta, Ross Teixeira, S Muthukrishnan, and Jennifer Rexford. 2020. Carpe Elephants: Seize the Global Heavy Hitters. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure (SPIN)*.

[34] R. E. Hattachi. Next Generation Mobile Networks, NGMN. (2015). https://www.ngmn.org/wp-content/uploads/NGMN$_5$G$_W$hite$_P$aper$_V$1$_0$.pdf

[35] Harri Holma and Antti Toskala. LTE Advanced: 3GPP Solution for IMT-Advanced. (2012). https://www.oreilly.com/library/view/lte-advanced-3gpp/9781118399422/c01anchor-3.html

[36] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. 2019. Blink: Fast Connectivity Recovery Entirely in the Data Plane. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[37] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*.

[38] Jaco Joubert. P4 Loadbalancer and Metering. (2017). https://github.com/open-nfpsw/p4$_b$asic$_l$b$_m$etering$_n$ic

[39] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. HULA: Scalable Load Balancing Using Programmable Data Planes. In *Proceedings of the the Symposium on SDN Research (SOSR)*.

[40] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band network telemetry via programmable dataplanes. In *ACM Special Interest Group on Data Communication (SIGCOMM)*.

[41] Dr. Kim. 5G stats. (2017). https://techneconomyblog.com/tag/economics/

[42] DongJin Lee, JongHan Park, Chetan Hiremath, John Mangan, and Michael Lynch. Towards achieving high performance in 5G mobile packet core's user plane function. (2018). https://builders.intel.com/docs/networkbuilders/towards-achieving-high-performance-in-5g-mobile-packet-cores-user-plane-function.pdf

[43] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*.

[44] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[45] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. AccelTCP: Accelerating Network Applications with Stateful TCP Offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[46] Barefoot networks. NoviWare 400.5 for Barefoot Tofino chipset. (2018). https://noviflow.com/wp-content/uploads/NoviWare-Tofino-Datasheet.pdf

[47] David Nowoswiat. Managing LTE Core Network Signaling Traffic. (2013). https://www.nokia.com/en¡nt/blog/managing-lte-core-network-signaling-traffic

[48] Recep Ozdag. Intel Ethernet Switch FM6000 Series - Software Defined Networking. (2019). https://people.ucsc.edu/~warner/Bufs/ethernet-switch-fm6000-sdn-paper.pdf

[49] Brian Johnson Robin Giller, Andrey Chilikin. Intel® Ethernet Controller 700 Series GTPv1 - Dynamic Device Personalization. (2018). https://builders.intel.com/docs/networkbuilders/intel-ethernet-controller-700-series-gtpv1-dynamic-device-personalization.pdf/

[50] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[51] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan R. K. Ports, and Peter Richtárik. Scaling Distributed Machine Learning with In-Network Aggregation. (2020). arXiv:cs.DC/1903.06701

[52] Rinku Shah, Vikas Kumar, Mythili Vutukuru, and Purushottam Kulkarni. 2020. TurboEPC: Leveraging Dataplane Programmability to Accelerate the Mobile Packet Core. In *Proceedings of the Symposium on SDN Research (SOSR)*.

[53] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the the Symposium on SDN Research (SOSR)*.

[54] Gábor Soós, Ferenc Nándor Janky, and Pál Varga. 2019. Distinguishing 5G IoT Use-Cases through Analyzing Signaling Traffic Characteristics. In *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*.

[55] Netronome systems. Agilio CX SmartNICs. (2018). https://www.netronome.com/products/agilio-cx/

[56] Netronome systems. Agilio CX 2x10GbE SmartNIC. (2020). https://www.netronome.com/media/documents/PB$_A$gilio$_C$X$_2$x10GbE-7-20.pdf

[57] Netronome systems. Agilio CX 2x25GbE SmartNIC. (2020). https://colfaxdirect.com/store/pc/viewPrd.asp?idproduct=3144&idcategory=0

[58] Netronome systems. Agilio CX 2x40GbE SmartNIC. (2020). https://colfaxdirect.com/store/pc/viewPrd.asp?idproduct=2871

[59] Sami Tabbane. Core network and transmission dimensioning. (2016). https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/Events/2016/Aug-WBB-Iran/Wirelessbroadband/core%20network%20dimensioning.pdf

[60] Amy Viviano, David Coulter, Nick Schonning, Duncan MacMichael, and Bill Latimer. Receive Side Scaling (RSS). (2017). https://docs.microsoft.com/en-us/windows-hardware/drivers/network/introduction-to-receive-side-scaling

[61] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets)*.