This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

Hybrid P4 Programmable Pipelines for 5G gNodeB and User Plane Functions

Suneet Kumar Singh*, Christian Esteve Rothenberg*, Jonatan Langlet†, Andreas Kassler‡
Péter Vörös§, Sándor Laki§ and Gergely Pongrácz¶
*University of Campinas, Brazil**, *Queen Mary University of London, UK†*, *Karlstad University, Sweden‡*
ELTE Eötvös Loránd University, Hungary§, *Ericsson Research, Hungary¶*
*Email: *ssingh@dca.fee.unicamp.br, *chesteve@dca.fee.unicamp.br, †j.langlet@qmul.ac.uk*
‡andreas.kassler@kau.se, §vopraai@inf.elte.hu, §lakis@inf.elte.hu, ¶gergely.pongracz@ericsson.com

***Abstract*—This paper focuses on hybrid pipeline designs for User Plane Function and next-generation NodeB leveraging target-specific features and an insightful discussion of P4 and target challenges and limitations. The entire or disaggregated UPF runs on P4 targets and allocates packet processing data paths in P4 hardware or DPDK/x86 software based on flow characteristics (e.g., heavy hitters) and QoS requirements (e.g., low-latency slices). For the hybrid gNodeB, most packet processing is executed in commodity Tofino hardware, while unsupported functions such as Automatic Repeat Request and cryptography are performed in DPDK/x86. We show that our hybrid UPF improves the scalability by 18× and reduces latency up to 50%. The results also suggest that careful traffic allocation to pipeline targets is required to optimize each target's strength and avoid processing delays. Finally, we demonstrate a QoS-oriented application of the hybrid UPF and present gNodeB buffer service benchmarks.**

***Index Terms*—5G, mobile network, P4, programmable networks, hybrid network**

1. Introduction

Next-generation mobile networks, including future versions of the currently deployed 5G networks, aim to support a diverse set of traffic mixes stemming from demanding applications such as Extended reality (XR), Industrial Internet of things (I-IoT), and self-driving vehicles [12]. To support the ever-increasing data rates while at the same achieving low and predictable latency for critical services (e.g., URLLC), the user plane of the mobile core network is a critical component because it deals with the packet forwarding between the user terminal (UE) and the packet gateway, which connects the operator network to the Internet. Consequently, the user plane faces complex requirements to ensure that diverse services can be delivered under tight latency bounds at scale.

Delivering new functionalities in a timely and customized manner demands a flexible user plane, which has led to the current softwarized packet core network design, where most packet processing is done on commodity servers in software [20]. However, such flexibility comes with several drawbacks. Firstly, softwarized packet processing has difficulties maintaining low and predictable latencies due to several bottlenecks that are intrinsic to the hardware design of modern servers, such as limited and varying PCIe bus transfer speeds, cache misses, and other phenomena [25], which makes it difficult to maintain stable packet processing latencies at high load. Secondly, softwarized packet processing on commodity servers typically relies on kernel bypass to process packets at user space using frameworks such as the DPDK [11], which fully utilizes CPU cores for constantly polling the NIC for packets, as shown in Figure 2(a). While reducing the latency, such polling wastes CPU cycles and leads to high energy consumption, significantly increasing operational costs. The main abbreviations used in this paper are listed in Table 1.

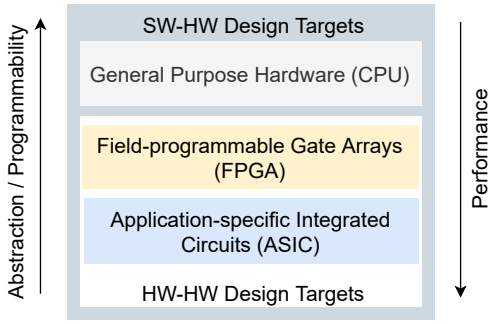
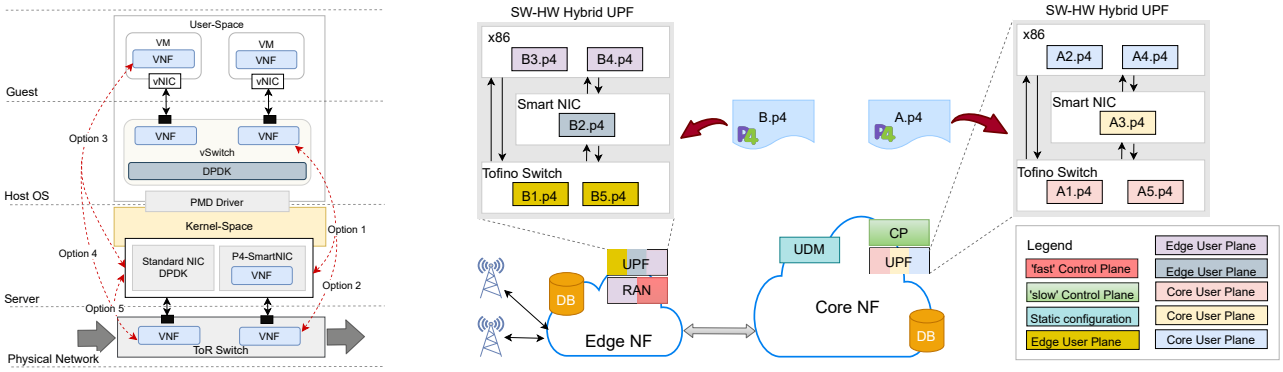


Figure 1: Performance and programmability trade-offs.

Recent efforts in the P4 [8] community provide support for a diverse range of various targets [37], i.e., ASICs, FPGAs, and x86, while providing flexibility to offload network functions to leverage the performance capabilities of different targets. Figure 1 shows the trade-off between programmability and performance. For example, x86-based packet processing provides the highest degree of programmability and flexibility at the expense of low throughput, high and unpredictable latency, and increased operational costs in CPU cores and energy consumption. On the other end, ASICs provide predictable latency at a line rate with a limited degree of programmability. On the other hand, FPGA offers more programmability than ASIC with good performance characteristics at increased footprint

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512



(a) VNF offloading options to accelerate the path or based on complexity or unsupported functions by P4

(b) 5G mobile network to show the disaggregation of a single P4 and run on different targets

Figure 2: (a) VNF offloading options and (b) disaggregation of a P4 code for UPF and gNodeB 5G data plane functions.

and power consumption.

To leverage emerging P4 programmable devices discussed above, most state-of-the-art [39] [36] compiles the user plane to P4 specific targets such as programmable ASIC, SmartNIC, or x86, but this approach limits the performance and functionality of the P4 program [10] [17]. For example, the P4 programmability and available resources such as stateful memory or match action table entries of a switching ASIC such as Tofino are limited. On the other hand, compiling a P4 program to a SmartNIC may leverage its micro-C-based programmability to implement more complex external functions. While the SmartNIC may have abundant resources, the performance may also be impacted, e.g., large table lookups. Finally, cryptographic operations, packet buffering, and retransmissions, which are required to implement the lower layer packet processing within a 5G data plane, may require the P4 program to be compiled to the x86 due to the needed flexibility for external function processing. To leverage the target specific features, some recent efforts [4] [22] split the UPF and offload unsupported functions, such as buffer services or hierarchical Quality of Service (QoS) from Tofino switch to x86 or SmartNIC. However, their solutions are either unable to answer how traffic allocation between targets impacts the overall performance or only focus on the specific target's performance.

In this paper, we propose *hybrid design approaches*, that P4 packet processing pipelines be split up into their basic processing units [41], individually compiled, and run on different, possibly heterogeneous, targets to enhance overall performance. The proposed data plane disaggregation approaches ~~to~~ split the functionality by carefully considering the mobile packet core packet processing requirements to combine each target's strength. Figure 2 shows the possible offloading options and illustrates the general view of a 5G mobile network architecture. The packet processing pipeline (Figure 2(b)) is disaggregated into small sub-programs and flexibly deployed from the edge to the core of the network across different targets. For example, assume that the program `A.p4` can be split into five sub-programs.

TABLE 1: This table describes the various abbreviations used throughout this paper.

Abbreviations	Definition
URLLC	Ultra-Reliable Low-Latency Communication
DPDK	Data Plane Development Kit
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
SmartNIC	Smart Network Interface Card
NetFPGA	Network Smart Network Interface Card
UPF	User Plane Function
P4	Programming Protocol-independent Packet Processors
NPL	Network Programming Language
RAN	Radio Access Network
GTP	General Packet Radio Service Tunneling Protocol
eBPF/XDP	Extended Berkeley Packet Filter/Express Data Path
SDNet	Software Defined Specification Environment for Networking
VHDL	VHSIC Hardware Description Language
BST	Base Station Transceiver
RLC-AM	Radio link control - Acknowledged Mode
RTT	Round Trip Time
QUIC	Quick UDP Internet Protocol
PDCP	Packet Data Convergence Protocol
ACL	Access Control List
BMv2	Behavioral Model version 2

`A2.p4` and `A4.p4` could be run on the x86 to utilize the high degree of programmability, functionality, and available resources while observing the target latency and throughput. The sub-program `A3.p4` could be run on SmartNIC to exploit the more predictable latency while releasing CPU cycles. Finally, 5G user plane functionality implemented in `A1.p4` and `A5.p4` that require limited programmability and resources could be run on Tofino switch ASICs. The same approach can be applied for edge network user plane functions. For example, the `B.p4` is split based on similar logic to `A4.p4`. Also shown in Figure 2(b) but not relevant to the scope of this work are the Unified Data Management (UDM) static configuration functions interfaced by 'slow' 5G control plane functions to access user data network profiles and handle access authorization and registration. The 'fast' control plane functions refer to low-level radio-related radio resource management per-user basis or control spectrum on a system level.

Our hybrid design models support diverse targets, ASIC, SmatNIC, and x86, focusing on 5G UPF and gNodeB, but

applicable to other datapath pipelines. We perform an extensive experimental evaluation to assess each design model. To the best of our knowledge, this is the first work to design, implement and experimentally evaluate different hybrid design models for 5G user plane packet processing functions embracing diverse P4 targets. We discuss the background and related work (Section 2) and 5G system implementation challenges and limitations using P4 (Section 3). We present different hybrid design models using three offloading axes and related use-cases (Section 4). Finally, we carry out an extensive evaluation for each design model (Section 5). We show that the proposed hybrid approaches are more scalable and programmable to make the user plane more flexible without compromising performance.

In summary, the main contribution of this paper is to realize the 5G components such as UPF and gNodeB as hybrid HW/SW systems by discussing the limitations and implementation challenges of the P4 language and target-specific nuances. We develop 5G unsupported user plane functions for buffering, encryption/decryption, and time-based re-transmission for P4-based x86/DPDK targets. In addition, we showcase a QoS-oriented application of the hybrid-UPF and discuss gNodeB buffer service benchmarks.

2. Background

2.1. 5G Network Architecture

The 5G network [12] is highly flexible than the previous generations because of the recent growth and implementation of softwarization and network virtualization of network services. The service-based architecture makes the 5G network more flexible and programmable. There are two significant benefits of service-based architecture, 1) each service can be updated without affecting the other services, and 2) easy to extend the new functionality. Figure 2 shows a high-level overview of the 5G mobile network architecture, consisting of RAN and Core network function. gNodeB in RAN contains a Centralized Unit (CU) and Distributed Units (DUs). The CU is further divided into control and user plane components. The user plane consists of Service Data Adaptation (SDAP) layers are responsible for user plane data transfer between UE and gNodeB.

UPF is connected with gNodeB, and data packets are exchanged using GTP. When a UE connects with the internet, an end-to-end tunnel between UE and UPF forms a PDU (Protocol Data Unit) session. UPF provides the connectivity between the mobile infrastructure and the Data Network to encapsulate and decapsulate packets using GPRS Tunneling Protocol. Also, UPF is responsible for handling QoS per-flow, including transport level packet marking for uplink and downlink, rate limiting, guaranteed bit rate, maximum bit rate, service data flow (SDF) mapping, and packet routing and forwarding. The detail about the functions of gNodeB and UPF is given in Sections 5.1 & 5.2.

Typically, UPF and gNodeB user plane functionalities are implemented as a cloud/edge application running on virtual machines or Docker containers [16] [26]. We can improve the overall performance by defining user-plane functions in the P4 language and instructing the networking hardware to process the packets. However, the complete user plane functionalities are challenging to run on the switch ASIC because of limited resources and functionalities (detailed in Section 4). For a more flexible solution, a hybrid design approach can be adopted, which distributes the overall packet processing on different hardware targets. Several P4-based user plane functions and hybrid design implementations have been proposed, and discussed in sub-sections 3.1 and 3.2.

2.2. Programmable Data Planes

Data plane programmability has opened the door for new applications [14] that require fast packet processing. Different solutions have emerged to support the programmability of hardware devices, including eBPF/XDP [43], DPDK's flow API [11], or domain-specific languages like P4 [8] or NPL [3]. Though the goals of these approaches are similar, they provide different abstraction levels. eBPF /XDP and DPDK are low-level C/C++ libraries supported mainly by smart and standard NICs, requiring deep domain-specific knowledge and programming skills. In addition, the data plane programming languages like P4 and NPL created for describing packet processing pipelines have a high abstraction levels and do not require deep programming skills, allowing network developers to solely focus on the problem to be solved. While NPL is designed to program switches, P4 intends to be more generic, supporting a variety of different targets through its architecture models in a common framework (e.g., same language, control plane, etc.). This makes P4 a good choice for designing and implementing hybrid solutions that integrate heterogeneous devices into a common platform.

The P4 language [8] is used to describe the pipeline behavior of the data plane of a forwarding element such as hardware or software switch and network interface card. The main purpose of designing this language are, 1) Target Independence: P4 programs can be compiled on different targets such as CPUs, FPGAs, system(s)-on-chip, network processors, and ASICs, 2) Protocol Independence: P4 allows to describe the header formats, and field names of the require protocols. 3) Reconfigurability: the P4 targets can change the forwarding pipeline anytime after they are deployed. Including these features, P4 also generates the control plane API for a given switch target. There are two versions of P4, $P4_{14}$, and $P4_{16}$. This article focuses on $P4_{16}$, which is much simpler, featured, structured, and flexible to run on different targets than $P4_{14}$.

3. Related Work

3.1. Programmable Hardware Acceleration

Offloading network functions running on general-purpose Commercial-Off-The-Shelf (COTS) servers (e.g.,

TABLE 2: The existing state-of-the-art on P4-based mobile network user functions and the proposed work. H-Split and V-Split indicate Horizontal and Vertical split the packet processing pipeline, detailed in Section 5.3.

Design	Core idea	Deployment	4/5G	User Plane		Implementation		
				H-Split	V-Split	CPU	SmartNIC	ASIC
vEPG [39]	Offload MPC UPF to P4 ASIC	Core NF	4G	❍	❍	✓	❍	✓
TurboEPC [36]	Analyze signaling in the data-plane	Core NF	4G	❍	❍	✓	✓	❍
5G-UPF [4]	Perform UPF in HW/SW and buffering service in SW	Core NF	5G	❍	⊙	✓	✓	❍
Model & Performant UPF [22]	Simplify interface with CP, achieve high performance	Core NF	5G	❍	⊙	✓	❍	✓
5G-Edge-to-Core [33]	5G Multi-tenant P4 based data-path	Backhaul	5G	N/A	N/A	❍	✓	❍
BNG [19]	Analyze BNG in P4 HW	BNG	5G	❍	✓	❍	✓	✓
Hybrid-UPF (Proposed)	Perform UPF combining different P4 target-specific features	Core NF	5G	✓	✓	✓	✓	✓
Hybrid-gNodeB (Proposed)	Define gNodeB in P4 and unsupported functions in DPDK	Edge NF	5G	❍	✓	✓	✓	✓

❍ Not covered ⊙ Covered but not evaluated ✓ Covered and evaluated

x86) [31] to the hardware switch can provide two significant benefits. First, network functions can perform with ultra-low latency and high throughput. Second, energy consumption can be reduced due to big data analytic applications or other data center computations [24]. As mentioned in the previous section, there are various network P4 programmable hardware accelerator platforms; this paper mainly focuses on x86, SmartNICs, and ASICs.

The P4→NetFPGA [13] provides an environment and workflow to use FPGAs as a switch to run P4 programs using the Xilinx P4-SDNet toolchain. The Alveo SN1000 [5] SmartNIC is a recent example of hybrid designs for Smart NIC-DPU based on the integration of FPGAs and CPU that allow full protocol-level offload acceleration customization and application-specific data paths and provide flexibility to program using P4 high-level language programming.

In [33], 5G backhaul is implemented on the NetFPGA P4 pipeline in two stages: Parsing and Match/Action. The proposed design supports the double encapsulation required to communicate between 5G edge and core network within the multi-tenant 5G networks. The evaluation results have shown a lower packet loss from 49% to 4% in the worst-case to provide QoS in network congestion. Similarly, in [32] [46], a 5G-based QoS-aware network slicing solution is performed on P4-NetFPGA to fulfill the SLA requirement in terms of end-to-end latency and bandwidth. Not only offloading the user plane functions but [36] redesigns mobile packet core and offloads a significant fraction of signaling procedures from the control plane to P4-programmable hardware (Netronome Agilio SmartNIC) or software switch (BMv2).

The performance of NetFPGA/SmartNIC lies in between a commodity x86 server and switch ASIC. However, some network applications require very high performance, e.g., very high throughput and ultra-low latency, which are difficult to achieve using NetFPGA or any SmartNIC. As a solution, many works have been proposed where network functions are offloaded to switch ASIC. For example, in [39], the virtual Evolved Packet Gateway (vEPG) in the 4G access network is designed and evaluated on Tofino switch ASIC. The evaluation results show a low end-to-end latency of less than two μsec. Also, in [18], 5G UPF is performed under four different approaches as a device under tests, such as a kernel and userspace implementation on servers with dedicated NIC or virtual R-IOV ports and P4-based UPF implementation on Tofino switch. The results confirm that the P4 switch performs far better than other approaches. Apart from the UPF implementation, complex algorithms such as enhanced content permutation algorithm [21] can be executed on the Tofino switch to encode/decode up to 6.4 Tb/sec to protect the 5G packets.

3.2. P4 Hybrid Datapaths

In the above discussion, we focus mainly on the existing works which offload network functions either on the switch ASIC or P4NetFPGA. However, network applications can be disaggregated and run on different platforms based on the application requirements to utilize the benefits from different targets [15]. Recently, a disaggregated UPF [35] [29] has been proposed commercially. Specifically, the P4 program is optimized and split into multiple components. Then, each component runs on different targets based on the complexity of function and application requirements. For example, running the same P4 based 5G UPF on an FPGA can scale the number of sessions from hundreds of thousands to millions compared to switch ASIC.

Similarly, in the ongoing Aether project [27] [22], which provides mobile connectivity and edge cloud services for distributed enterprise networks at Open Networking Foundation (ONF), two P4-based UPF designs are presented: (1) model-UPF and (2) performant-UPF. The Performant-UPF has been implemented on the Tofino switch ASIC. Also, supporting other functionalities such as packet buffering or hierarchical QoS to provide isolation for each UE, which are limited by the current versions of Tofino switches, are executed on P4-based SmartNIC or x86. The model-UPF is developed as an open-source model data-plane, written for the Behavioral Model version 2 (BMv2) software switch to simplify the interface with the control plane. In [22], the design and use-cases of Aether are discussed in detail; however, the performance is not evaluated. Also, it is unclear what scenario is to be adopted to distribute the traffic if UPF runs on different programmable targets. On the other hand, Aether proposes using different queues to prioritize the traffic flows but does not provide guarantees per traffic class.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

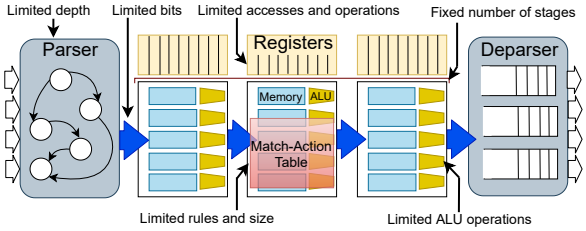


Figure 3: Resource limitations in P4 Switch ASIC.

As mentioned above, the HW switches can support QoS up to some extent with limited queues. However, handling the rate limit for QoS might be difficult because of the unsupported packet buffering features in HW switches. In [4], the UPF is performed on Agilio CX 2x10GbE SmartNICs. The packets which are required to be handled by buffer service are sent to the software. However, this work fails to show the overall impact of processing packets through different paths compared to SW or HW-based implementation. Finally, instead of UPF offloading, in [19], an open-source implementation of a broadband network gateway (BNG) is executed on NetFPGA and also performed on the Tofino switch ASIC. The packet queuing and scheduling are performed on NetFPGA.

Our work on hybrid designs for 5G UPF and gNodeB implementation differs from the existing works as follows (see Table 2 for a compact representation). Compared to related work on P4-based UPF, our Hybrid-UPF effectively utilizes the target-specific features based on vertical/horizontal split of packet processing pipeline and runs the entire or disaggregated UPF on different P4 targets to make UPF more flexible and programmable at high performance and scale. Furthermore, our gNodeB is the first work, where most packet processing is defined in P4 and other functionalities such as encryption/decryption, buffering, and retransmission services are implemented in DPDK. While our preliminary assessments of gNodeB were presented in [44], this article discusses in detail P4 language-specific limitations for gNodeB implementations and presents the P4 based hybrid gNodeB workflow, and the 5G buffer as a service with detailed algorithm and benchmark evaluation.

4. P4 for 5G User Planes: Rationale, Challenges and Limitations

As shown in Figure 2, 5G user plane functions can be defined in P4 and offloaded to the HW switches to satisfy the required 5G SLA. Using P4, we can leverage many *benefits* such as In-band Network Telemetry (INT) for traffic measurement, support of network slicing, cybersecurity using deep packet inspection capabilities of P4, and online DDoS (Distributed Denial-of-Service) attack mitigation, etc. Apart from these benefits, one of the significant advantages of using P4 is that it is easier for mobile operators to modify their pipeline (e.g., dynamically modify match conditions and forwarding actions as per the stateful parameters like meters or registers) on runtime.

The recent development of the P4-based programmable data plane opens an exciting opportunity to offload network functions and algorithms to various programmable targets. Each programmable target has its features and limitations to perform the different tasks and operations. Due to these limitations, some complex tasks need to be offloaded with performance expenses, as shown in Figure 1. In this section, we highlight the general P4 target-specific challenges focusing on the 5G network. Also, we discuss the P4 limitations to support 5G user planes based on the programmable targets and their available resources, including some functional limitations.

4.1. P4 Implementation Challenges

Programmable Switch ASIC. Figure 3 illustrates the resource limitations of a programmable switch ASIC. A limited number of ingress parsers per pipeline push their PHV (Packet Header Vector) into the ingress match-action pipeline. Suppose packet headers for parsing are longer than the ingress parser limit. In that case, the packet must be recirculated, and then the parser has to parse the remaining packet headers during the second phase. This directly impacts the performance of the switch. The precious PHV resources, a limited collection of bits, are allocated to match-action ingress and egress pipes based on the P4 program.

Each stage can have a fixed number of SRAM and TCAM tables, and each table can use a fixed number of bits for a single match. Although a table can span multiple stages, the number of stages also poses a limitation. So due to the limited SRAM and TCAM per stage and the limited number of stages, the size of big tables - such as the UE lookup table - is limited. We hit the maximum table size to perform UE matches (Table 5) during the 5G UPF execution on Tofino HW. To improve scalability, we propose a hybrid approach (Section 5.4).

Furthermore, arithmetic operations such as multiplications and divisions or comparison two variables operations are generally not supported or limited by traditional switch ASIC. The emerging P4 programmable HW (e.g., P4 Tofino) supports simple bit operations. For complex computations, sometimes, we have to rely on the average calculations. This paper uses the Heavy-Hitter detection algorithm (Section 5.4) based on Inter-Packet Gap (IPG) [38], where the exponential weighted moving average (EWMA) is performed. Since EMWA is difficult to compute accurately in Tofino switch HW, we depend on the average EWMA calculation. To execute such HH algorithms in programmable HW, we use registers to keep flow states, e.g., 5-tuple flow Id, counters. In such cases, we mostly hit the limitation to access the register memory. For example, when a packet enters the switch, it can access only a few addresses in the register memory because of per-stage timing demand. Also, the incoming packet accesses a register instance only once in its lifetime. Such limitations can force to re-circulate

the packet for re-accessing memory based on algorithm requirement.

In addition, to guarantee low and bounded per-packet latency, P4 programmable HW contains a fixed number of stages. To fit the given P4 program within the available stages, we require to avoid unnecessary table dependencies. We face this issue in executing the 5G UPF functionalities in parallel with the HH algorithm. Since the decision of packet routing is based on the output of the HH algorithm (Figure 5), it is hard to fit the hybrid design pipeline within the fixed number of stages (i.e., 12 Stages in Tofino). We optimize the code to efficiently use the table dependencies to fit the proposed design in the available stages. However, the upcoming P4 programmable switch ASIC [29] might have extra features, including the number of stages to overcome these issues. A detailed discussion about the hybrid design is given in Section 5.4.

Programmable SmartNIC. VNF packet processing can be accelerated using programmable components, and bringing it closer to endpoints inside the data center and at the edge is essential for reduced latency and improved scalability. However, this requires that those components support the required functionality (e.g., P4 programmability). For example, the function can be deployed on the NICs installed in the edge compute nodes leveraging the P4 programmability features of modern smartNICs. There are two classes of smartNICs that operate using flow processor cores that act more as small CPUs and are optimized for packet processing (e.g., Netronome Agilo CX, Pensando DSC, and Mellanox BlueField) or FPGA-based solutions (e.g., Xilinx Alveo or Intel PAC 3000).

Flow processor cores-based approaches are typically more flexible, where their generic architecture allows for highly complex and customizable processing logic not allowed in ASICs. For example, the Netronome platform comes with additional Micro-C language support that will enable implementations using custom user-defined external functions, expanding the P4 language with support, e.g., crypto acceleration, and various algorithms residing outside the *directed acyclical graph (DAG)* domain of P4. However, leveraging these techniques above might result in less deterministic packet processing performance, which in the worst cases lead to latency spikes, and potential packet loss. In addition, different operations (such as ternary matches) may need to be emulated in software as such NICs typically do not come with TCAM support. Consequently, complex and large table lookups may severely impact the performance of such flow processor core-based architectures, especially as those cores run typically at a lower frequency (e.g., 800 MHz) compared to host-based packet processing. Therefore, complex packet pipelines may lead to higher latency when deployed on smartNIC.

On the other hand, FPGA-based approaches provide more predictable performance but have less support for implementing complex packet processing logic. In addition, such hardware targets have a typically limited amount of resources (e.g., TCAM entries) and generally are more restricted in terms of external function support. Finally, smartNICs with programmable flow processor cores are easier to program as the P4 extern constructs are less restrictive (e.g., calculating modulus operation in micro-C in smartNIC is possible). At the same time, FPGA-based platforms require dedicated frameworks such as SDNet or Netcope to offer extended P4 programmability with the need to program such extensions using VHDL.

4.2. P4 Language Limitations for 5G support

Some 5G UPF and gNodeB functionalities are not possible to implement with the currently available features of the P4 language. We discuss some of them as follows:

Buffering and Timers. The re-transmission loop called RLC is between the gNodeB and the UEs. The goal of RLC in acknowledged mode (AM) is to confirm that packets transferred by the BST arrive at the corresponding UEs. This requires packet buffering and timers.

Packet buffering requires a relatively large amount of memory to sustain buffers long enough to hold an RTT-long stream of downlink packets. For example, a UE with good coverage can reach 1 Gbps and 20 msec RTT over the air. If the UE requires RLC-AM, the BST will allocate approximately 3 MB of buffer space for the in-flight packets. Furthermore, we have to multiply this amount by the number of (active) UEs.

Because fast memory is scarce in hardware switches, the P4 language lacks buffering and timer-based operations support. Because of that, we had to rely on a hybrid solution, where part of the pipeline (buffer-as-as-service, BaaS) runs external (e.g., x86/DPDK) to the P4 domain. In BaaS, a packet is received, and if the signal is del (i.e., delete), both packet and timer are deleted. On the other hand, the packet is re-sent and resets the timer if the timer reaches zero. The buffering and re-transmission timers are not specific to the RLC use-case; TCP and QUIC also use similar methods.

Clock-based events. There are clock-based events such as scheduler actions, where user packets are collected into a buffer, and the scheduler waits for the right slots where the user's device listens to the channel. Then, the scheduler sends the packets with high accuracy timing. These steps require buffering, as discussed above, and time sync (e.g., IEEE 1588-PTP), and internal clock-based events. Other possible use cases for these functions are traffic shaping and keep-alive messages.

Modularization. More complex dataplane pipelines ~~would~~ require code pieces from different developer teams or units. This would require a certain level of modularity where all teams could work on their respective code pieces while integration is still straightforward. In P4, this is only partially supported via control blocks: each developer or unit can develop its control module with a parser and a deparser that deal with headers necessary for the given functionality. This is good enough for simple testing, but integrating such functionalities into a larger pipeline lacks any support from P4. The common apply block and the parser have to be manually generated together with the used metadata and

TABLE 3: 5G User Plane Functions: Uplink and Downlink

UPF-UL	UPF-DL
Match Outer L2 and L3	
VxLAN Decapsulation	
Apply Metering	
Match 5 Tuple, set Queue Id	
Validate IPv4/GTP	
Match UPF IPv4	Match User Device IPv4
Apply global firewall rules	
GTP Decapsulation	GTP Encapsulation
IP Routing towards Internet	IP Routing towards eNodeB
VxLAN Encapsulation	
L3 Routing towards DCGW	
Outer L2 Forwarding	

header structures. Note, however, that this is not a functional limitation.

Stage-based pipeline setup. This limitation is only valid for the Tofino hardware target, but since this is currently the main P4 hardware, its limitations also limit the usability of the language. Tofino uses a stage-based approach for implementing the match-action blocks. The main limitation is that if a function uses a value as input, it cannot exist in the same stage as the function generating the given value. Also, a stage has access to multiple resources (SDRAM, TCAM, registers, etc.). If one type of resource becomes fully utilized, the pipeline will need to use multiple stages for the given functionality, in which case other resource types will be underutilized. This can be optimized a bit using packet recirculation which can be seen as a tradeoff between utilization and latency.

Others. Other missing lower-priority features include the multiple access to the same table, counter, or register (e.g., routing and encapsulation), utilizing conditional header field write in deparser, and better use of header stacks to support multi-level encapsulation. These were not mandatory to implement our use cases but would make the life of the network function developer easier.

5. P4-based hybrid 5G UPF Pipelines

This section first highlights the UPF and gNodeB functions. Then, we present the general architecture of hybrid design options for the data plane. Next, we focus on designing and implementing the proposed P4-based hybrid 5G UPF and gNodeB pipeline using different targets.

5.1. UPF P4 Pipeline

The UPF is one of the 5G core networks functions, as illustrated in Figure 2. As discussed, the UPF functionalities in Section 2.1, we define them in P4 for Tofino Native Architecture (TNA), as shown in Table 3. We discuss each of them as follows:

L2/Ethernet and Virtualization. For each incoming packet, lookups are performed for both source and destination MAC addresses. VxLAN is used to support the isolation between VNFs in multi-tenant environments. When the switch receives the packet, the VxLAN (de-)encapsulation of the L2 frame within the UDP header is performed. The routing applies for both UPF uplink (UPF-UL) and UPF downlink (UPF-DL) towards data center gateways (DCGWs).

QoS Support. Quality of Service (QoS) is used to provide better service to the selected network traffic. Without QoS, all the incoming packets are treated as best efforts. Our 5G UPF P4 code supports the QoS. The switch matches on five tuples to apply the meter and set the queue Id. Meter is used to drop the packets when the packet rate reaches the threshold. SDN controller pushes the entries to decide the queue Id based on the traffic type.

Firewall. The validity check of the IPv4 or GTP header decides the direction of the incoming packet, uplink, or downlink. Then, the global firewall rules are applied and block unauthorized traffic access to the server.

GTP Decap/Encap and IPv4 Routing. For the uplink direction, the GTP header is removed from the incoming packet and forwarded towards the data center getaway (DCGW) for accessing the internet. In the downlink direction, the tunnel endpoint identifier (TEID) allotted and the packet is routed towards the gNodeB.

5.2. gNodeB User Plane

Next-generation NodeB (gNodeB) is located in the 5G Radio Access Network (RAN) close to the base station. In the downlink direction, 5G UPF forwards subscribers' traffic to the appropriate gNodeB node encapsulated into GPRS Tunnel Protocol for user data (GTP-U). The gNodeB first decapsulates the downlink packets, ciphers the payload, adds PDCP (Packet Data Convergence Protocol) and RLC (Radio Link Control) headers, and then forwards the packet towards the user equipment. The gNodeB uses ARQ for reliable transport, meaning that all the downstream packets need to be stored in a packet buffer until an acknowledgment arrives. If the acknowledgment does not arrive within a timeout period, the packet is re-transmitted, and the appropriate acknowledgment timer is restarted. In the upstream pipeline, we have to handle acknowledgments (RLC control packets) that are not forwarded and uplink user data encapsulated in RLC/PDCP packets. Upstream user data must be deciphered and then forwarded via a GTP-U tunnel to a UPF instance in the 5G-Core. The complete pipeline is given in Figure 6.

5.3. General Architecture of Hybrid 5G Data Plane

Figure 4 shows the general architecture of hybrid UPF. Switch ASIC is connected with two different P4 targets: 1) x86 server behaving as a P4 SW switch, and 2) SmartNIC. The incoming packets at switch ASIC can be routed in three different paths based on the application's requirement. Three different routes have been shown in Figure 4.

In the fast path, incoming packets pass through NFs running on the P4 programmable switch ASIC and are sent to the output port. For slow path, packets can be routed

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

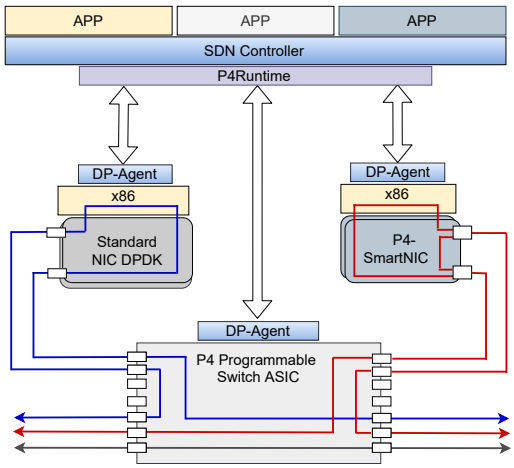


Figure 4: General architecture view of hybrid data plane

through an SW switch running on x86. For the slow path, there may be different options also in terms of hardware and programmability support on the x86. For example, if the host is equipped with a P4 programmable smart NIC, (parts of) the processing pipeline or parts of the overall traffic can be completely or partially processed by the SmartNIC (right side of the Figure 4). The remaining parts of the processing pipeline or user traffic can be processed in the CPU. If a standard NIC is available (left side of Figure 4), the NIC forwards user traffic to the x86 (e.g., through a software switch). Then, the remaining traffic or parts of the pipeline can be processed on the CPU using frameworks such as T4P4S [45], which integrates more general packet processing with software switch functionality or using, e.g., a P4 compiler that compiles from P4 to DPDK code directly.

There are two main approaches to partitioning the packet processing pipeline. In the first method (horizontal split), the pipelines have identical logic, but parts of the user packets are processed on each back-end, e.g., SmartNIC or host, observing the latency and capacity constraints of the different users (e.g., low latency traffic will be processed inside the programmable SmartNIC, while parts of the high capacity user traffic will be processed at the x86). This has the benefit that low latency traffic that is completely processed in SmartNIC does not need to transit the PCIe bus and is completely kept inside SmartNIC.

In the second method (vertical split), the logic of the pipeline is split into basic functional blocks; thus, processing the pipeline requires multiple processing backends to process each packet. Each backend will process parts of the pipeline (e.g., the SmartNIC performs a GTP de-tunneling operation while the host does table lookups and header rewriting). Using this approach, we need to consider the architectural constraints for hardware acceleration (e.g., a crypto operation should be run on a device with effective crypto support or a crypto chip). Also, this approach follows the Service Function Chain concept but is more challenging to realize due to the consideration of different hardware capabilities that need to be considered for allocating the individual atomic processing tasks.

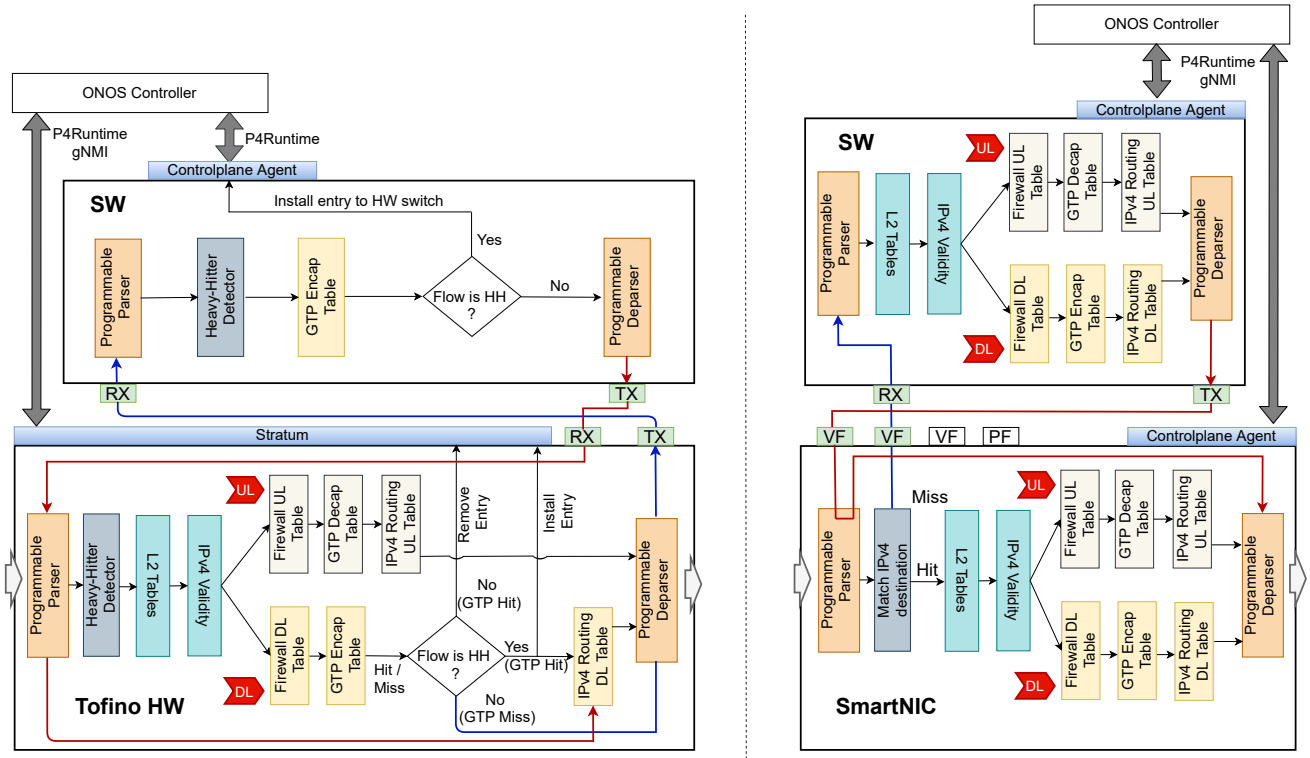
Such hybrid approaches can be cost-effective. SmartNICs do not significantly add up the cost because standard NICs can be replaced by SmartNICs, adding additional functionalities. On the other hand, the Tofino can be used instead of a ToR switch without much cost difference. So, the hybrid approach using SmartNICs and Tofino switches provides more programmability and performance with only a few additional economic expenses. Also, this approach would be more cost-effective than proprietary custom-made ASICs for beyond 5G use cases.

Next, we discuss our proposed design models using a vertical and horizontal split of the packet processing pipeline.

5.4. Hybrid-UPF using switch ASIC and x86

Tofino ASIC is a high-performance P4 programmable switch that can run up to 6.5 Tbit/sec speed. VNF can be offloaded to the Tofino switch ASIC for achieving high performance. VNF offloading depends on the complexity of the functions because of the limited resources and flexibility of programmable hardware to guarantee low and bounded per-packet latency (discussed in Section 4.1). Therefore, scalability for user equipment (UE) can be a significant issue for the 5G UPF running on programmable switch HW without compromising the performance. As given in Table 3, we perform a UE IPv4 address match with GTP encapsulation as an action for downlink UPF. In Tofino, there is limited SRAM memory per stage to keep the hash entries to perform the match. Instead, telecom operators require the limited and precious resources in the programmable switch ASIC for crucial control functions such as ACL rules, customized forwarding, and other network applications. Thus, we are allowed UE matches in thousands only, while the UEs can be in a million, especially when the 500 billion mobile devices are expected to be connected to the Internet by 2030 [1].

We can adopt a number of different strategies to overcome this problem, such as scaling out the full UPF pipeline across multiple programmable switch ASIC or disaggregating the UPF functions into different programmable targets. In real network traffic, the majority (i.e., 90-95%) are inactive UEs or non-HHs (i.e., low throughput), while the minority (i.e., 5-10%) are active UEs (i.e., HHs, high throughput) at a time [7] [23]. Also, most of the traffic in the network is contributed by HHs. For example, if we target to support 5 million UEs IoT (Internet of Things), 5-10% could be smartphones, 10-20% wideband IoT devices, and the rest can be narrowband IoT devices. The wideband and smartphones or HHs can be maintained on the Tofino switch to consuming fewer hardware resources. The rest flows (i.e., non-HHs or narrowband IoT devices) can go through the x86 server without degrading the Key Performance Indicator (KPI) for the 5G network. We propose a hybrid design based on Heavy-Hitter (HH) detection. The proposed hybrid design leverages the Inter Packet Gap (IPG) based HH detection method [38] entirely implementable in a P4 data planes [42] to reduce the control channel overhead with high



(a) P4 Based Hybrid Tofino-HW / x86

(b) P4 Based Hybrid SmartNIC-HW / x86

Figure 5: P4-based hybrid Tofino-HW/x86 and SmartNIC/x86 architectures for UPF.

detection accuracy. In this case, the switch relies on a push-based approach to report the flow as HH to the controller for fast reaction. The proposed hybrid solution can be based on any existing HH algorithm (e.g., [47] [48]) compatible with Tofino HW with a minor impact on overall performance based on how fast the algorithm reports the HH flows to the controller.

While the above approach can improve the overall scalability of the UPF, it does not guarantee that all non-HH flows are delay-critical. Offloading delay-critical flows to x86 can impact the required service level agreement (SLA). Other offloading criteria such as 3GPP slice definitions can be adopted for HW processing instead or in addition to HH-based offloading. For instance, traffic flows can be classified based on the definition of network slicing such as match on 6-tuple (i.e., 5G user source and destination IPs, 5G user source and destination ports, Differentiated Services Code Point (DSCP) and GTP Tunnel ID) to define the network slice and route the traffic accordingly [32]. The delay critical traffic can be forwarded to the programmable switch ASIC, and based on the priority; it will be assigned to the specific queues [34]. The network slicing implementation and queue management are out of the scope of this work, and will be considered in future work.

As shown in Figure 5(a), we offload the entire UPF pipeline to the Tofino switch while the GTP Encap table is maintained in both x86 and Tofino. For the uplink, the packet goes directly through the Tofino switch. In the case of downlink, the packet is routed based on the HH detector and UE address match (i.e., GTP hit/miss). Suppose the packet is detected as HH with GTP hit. In that case, the first packet is sent to the ONOS (Open Network Operating System) controller to install the corresponding table entry to the switch, and the rest of the packets of that flow follow the HW pipeline to exit. At GTP hit for non-HH flow, entry is removed from the table, while packet goes through the x86 server for GTP hit to perform the match on UE address.

5.5. Hybrid-UPF using SmartNIC and x86

As shown above, the SmartNIC can be used standalone to process packets. However, its operation becomes more flexible when combined with host-based packet processing. Indeed, in an NFV setup, there are typically multiple containers on the host that process packets. The SmartNIC can do more complex tasks (e.g., en/decapsulating headers) than forwarding packets between the host and the network. Consequently, several functions of the whole packet processing pipeline can be offloaded to the SmartNIC, including table lookups, firewalling, header rewriting, performing crypto operations, or load balancing among the different VNF processing cores in the host. Consequently, a processing

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

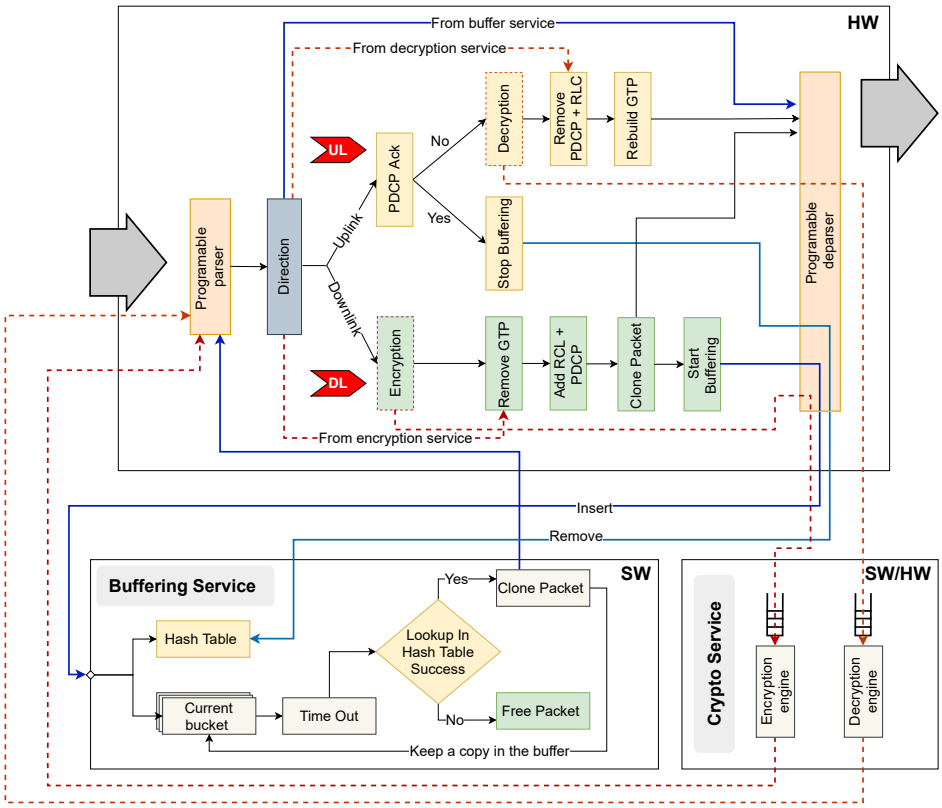


Figure 6: P4-based hybrid pipeline of the gNodeB .

pipeline can be partitioned, and some packets can be processed on the SmartNIC part of the pipeline and/or inside the host. Note, if part of the pipeline is inside the host, costly transitioning of the PCIe bus is required twice.

We have implemented an entire UPF pipeline following the horizontal split concept on the Netronome SmartNIC using a hybrid setup, where parts of the users are processed fully in SmartNIC. In contrast, other users are processed in the host. We created multiple SR-IOV (Single Root Input/Output Virtualization) Virtual Function (VF) ports in the host. We created a DPDK docker image that also implements the UPF pipelines using T4P4S, which compiles the UPF pipeline into DPDK code and wraps the data plane implementation into a docker container. Each container was bound to its own isolated CPU core and pinned to its own SmartNIC VF port. The P4 pipeline running on the SmartNIC has full control of packet forwarding to-and-from these VF ports. Such an approach requires load-balancing in the SmartNIC among the different processing containers in the x86 to select which VF port to send the packet to.

In our approach, packets entering the SmartNIC are parsed first in its P4 pipeline. We perform an additional exact match on the IPv4 UE address using an additional table (see Figure 5(b)) that determines which flows will be offloaded to the entire UPF pipeline in the SmartNIC (IPv4 destination). If there is no match, the SmartNIC will forward the packets towards the host on one of the VF ports for further processing using a hash-based load-balancing on the IP-destination address. If there is a hit, packets will be offloaded and processed fully on the SmartNICs UPF pipeline before emitting back on the physical ports, thus completely bypassing host processing. Consequently, determining which tunnels will be processed on the SmartNIC or the host requires an additional table lookup in the SmartNIC. This allows a policy-based offloading for prioritized users requiring, e.g., low latency services implemented through the control plane by populating proper table entries.

5.6. Hybrid-gNodeB Design and Implementation

Hybrid gNodeB implementation is a mix of two separate targets. Conversely, the direction choice and header operations are programmed in P4 for TNA, while the other not hardware supported operations (such as encryption/decryption, buffering, and time-based retransmission) are implemented using DPDK.

Packet processing in the gNodeB branches along with the upstream and downstream directions, as shown in Figure 6. The downstream direction receives GTP encapsulated packets. It saves the GTP TEID field, decapsulates the GTP header, and creates the RLC and PDCP headers with the PDCP serial number (SN) calculated from the TEID value. The newly constructed PDCP packet is sent to the encryption service, from where the encrypted packet is received

Algorithm 1 Buffering

```
1: i := 0; n := resubmission interval in ms
2: for j IN 1 ... n do
3:     bucket[j] := empty bucket
4: end for
5: while True do
6:     if 1 ms passed then
7:         i := (i + 1)%n
8:         for pkt IN bucket[j] do
9:             if hashtable.contains(pkt.id) then
10:                pkt2 = clone(pkt)
11:                send_out(pkt2)
12:            else
13:                bucket[j].remove(pkt)
14:                free(pkt)
15:            end if
16:        end for
17:    end if
18:    pkt := receive_packet()
19:    if pkt is PDCP_STATUS then
20:        hashtable.remove(pkt.id)
21:    else
22:        bucket[j].add(p)
23:        hashtable.insert(pkt.id)
24:    end if
25: end while
```

asynchronously later. As a next step, the encrypted packet is cloned. One copy goes through the appropriate downstream port, while the other is transmitted to the buffering service.

The upstream direction handles incoming PDCP packets. PDCP protocol distinguishes status messages (acknowledgments) and data packets. Acknowledgments are forwarded to the buffering service without modification. While for data packets, the gNodeB has to rebuild the GTP header. After the PDCP and RLC header removal and the proper GTP encapsulation, the packet goes to the upstream port.

5.6.1. Buffer as a Service (BaaS). RLC protocol requires reliable communication, with guarantees that packets can not be lost. Packets have to be constantly resubmitted until acknowledged by the UE. To achieve that, we have to implement a buffering and automatic packet resubmission service. However, as mentioned in Sec 4.2, several limitations in the current P4 language do not allow us to implement such services. The key for buffering is to have a large amount of memory to buffer packets long enough to hold at least an RTT-long stream of downlink packets. What, as discussed above, is not possible in current hardware switches. Automatic repeat request (ARQ) is an error-control technique that uses acknowledgments and timeouts to achieve reliable communication. It requires clock-based actions to be able to resend unacknowledged packets. P4 currently only supports packet-based actions, so there is no way to use timers; therefore, to implement such a mechanism, we either need to send a flow with a consistent rate just to keep track of the time or outsource the solution to an external service. To overcome every limitation, we chose to implement the buffering service, including the ARQ as an external DPDK service on x86, as shown in Figure 6.

The implementation details of the buffer service component are given in Algorithm 1. We use n buckets, where n is the number of milliseconds for which packets should be buffered. Every millisecond, we move to the next bucket and process every packet contained in the bucket. If the packet id is found in the hash table, it is resubmitted; otherwise, it is dropped. When the bucket is fully processed, we start handling incoming packets. Upon a packet arrival, we first check to see if it is a status message or not. Data packets must be buffered, the id of each packet is stored in the hash table, and the packets are placed in the current bucket. For each incoming ACK, the packet ids are removed from the hash table.

TABLE 4: Resource usage in Tofino switch ASIC

Resource	UPF-Tofino.p4		HH-IPG.p4	H-UPF.p4	
	10K (%)	430K (%)	NA (%)	10K (%)	430K (%)
Exact Match Input Xbar	0.8	1.6	5.0	5.9	6.5
Hash Bit	1.8	4.5	2.8	8.6	12.8
Hash Distribution Unit	0.0	0.0	29.2	29.2	29.2
Meter ALU	0.0	0.0	10.4	10.4	10.4
SRAM	1.9	32.0	1.5	3.9	34.4
TCAM	0.0	0.0	4.2	6.3	2.1
VLIW Instruction	1.8	3.4	4.7	6.8	7.8
Exact Match Result Bus	1.6	3.1	8.3	10.4	12.0

6. Experimental Evaluation

In this section, we experimentally evaluate each hybrid pipeline design presented in the previous section using real-time traces and artificial workloads. We conclude by showcasing two use case scenarios. Altogether, we aim to answer the following questions:

- How do hybrid approaches improve the scalability of P4-based 5G data planes?
- Which and how much are the P4 Tofino hardware resources consumed by our hybrid 5G UPF pipelines?
- What is the performance (throughput and latency) of our hybrid UPF for different target combinations, varying UE sessions, and HH/offloading configurations?
- What is the throughput performance of the hybrid gNodeB buffer service for a varying amount of CPU cores, and what are the practicality implications?
- Which QoS improvements in terms of Flow Completion Time are possible in our hybrid UPF implementations?

6.1. Hybrid-UPF (Tofino + x86)

6.1.1. Methodology. We analyze the performance using a *Barefoot Tofino switch ASIC* (Edgecore Wedge 100BF-32X) and x86 server with Intel Xeon D-1518 processor (4 CPU cores, 2.20 GHz) for the UPF application generated by the MACSAD compiler for ODP-DPDK x86 targets [30]. Tofino is connected with an x86 server with 10G SFP+ interfaces, as shown in Figure 5(a), for offloading traffic based on the HH detector running on Tofino. We use the NetFPGA-SUME based OSNT [6] as a traffic generator running on

TABLE 5: Scalability analysis with UPF-Tofino and H-UPF

	UPF-Tofino	**H-UPF(Tofino+x86)**
SRAM	60%	34.4%(Tofino)
UEs	850K	430K(Tofino)+15M(x86)

an x86 server (Intel Xeon D-1518) with 10G SFP+ interfaces connected to the Tofino switch under test (DUT). For pushing the required table entries related to the UPF and HH detector, we use the ONOS controller connected with Tofino switch supported to P4Runtime. In the case of UPF, running on x86, we use the MACSAD controller to push the required entries.

We use CAIDA-2016 [9] ISP backbone link traces for evaluation. CAIDA 2016 contains the anonymized passive traffic traces from the equinix-chicago high speed monitor. We replay these traces on line rate (i.e., 10 Gbps) for our hybrid design evaluation. The traffic traces are available for four different days, and each is around 60 Secs long and contains around 20 Million packets and 800K flows. To analyze the performance of different flows or UEs, we split or concatenate the traces and prepare 100K, 800K, 1M, and 2M flow traces. Since the DDR3 memories of OSNT NetFPGA-SUME are 4 Gbytes, we can load a maximum of around 2M flows to replay at a 10 Gbps rate.

6.1.2. Resource Utilization. For the programmable switch ASIC, the P4 pipeline of hybrid-UPF (H-UPF.p4) is around 1000 lines of code. We add IPG based heavy-hitter (HH-IPG.p4) functional block to the UPF P4 pipeline for hybrid design. H-UPF.p4, running on the Tofino switch, classifies the incoming packets based on HH and routes the non-HH flow packets to the x86 server to match the UE IPv4 address and GTP encapsulation for achieving higher scalability.

To analyze the resource consumption on the Tofino switch, we focus on different resources shown in Table 4. The exact match input crossbar is used for choosing input keys for exact match lookup, while the hash bit is for table lookup. Hash distribution unit is utilized to map hash output to Packet Header Vector containers without using any table lookup. Meter ALU and SRAM are used for stateful memory and exact match lookups and action tables, respectively. As shown in Table 4, we notice that all resources are used less than 5% for UPF-Tofino, but SRAM is around 32% for exact match lookups of 430K flows. For HH-IPG, 29.2% of the hash distribution unit is used because ~~of~~ hashing used to index the register and table and compute the flow ID. Therefore, adding the HH functional block to the UPF-Tofino for hybrid design shows a significant difference only for the Hash distribution Unit compared with UPF-Tofino.

6.1.3. Scalability. Offloading 5G UPF to the Tofino HW provides high performance; however, it is hard to scale the number of UEs because of the HW resource limitations. In our proposed hybrid design, we can significantly scale the number of UEs with high performance. For evaluation, we increase the table size, where we perform the matches on UE IPv4 address in Tofino and try to hit the maximum possible SRAM utilization. However, to evaluate the maximum number of UEs in Tofino, we must carefully optimize the H-UPF P4 code to manage the table dependencies to utilize maximum SRAM. When we add the HH-IPG functional block to the UPF-Tofino, the conditional statement can be a cause to keep HH-IPG and GTP Encap table in separate stages, which reduces the SRAM utilization. We optimize the code and try to keep both HH-IPG and GTP Encap table without any conditions, which allows the UE IPv4 matches and action in parallel to HH-IPG.

Table 5 presents the maximum SRAM usage with the number of UEs for UPF-Tofino and H-UPF. The UPF-Tofino uses 60% of SRAM with a maximum of 850K UEs, while H-UPF for Tofino utilizes 34.4% of SRAM with a maximum of 430K UEs treated as HH. For practical cases, 430K HH flows would be sufficient for 10-100Gbps links. As shown in Table 6, the number of HH flows is less than 700 in all the given threshold ranges for CAIDA-2016 traces captured on the 10G link. For non-HH flows to keep on x86, we insert up to 15M flows to H-UPF running on x86 using the MACSAD compiler. However, we can keep more than 15M (e.g., 64M, 5 tuple keys) that fits in the CPU cache depending on the server memory specifications.

6.1.4. Performance. The performance of UPF is evaluated in terms of throughput and ~~end-to-end~~ latency. We analyze the throughput in the worst-case scenario where packets are smaller (payload is removed from all packets because of the anonymized dataset). The switch receives the packet from the data network and encapsulates it with the GTP header, so the packet size is around 104 Bytes used for performance evaluation. To evaluate throughput, we explore 3 cases: (1) UPF is running entirely on the x86 server (UPF-x86), (2) UPF offloaded to Tofino switch (UPF-Tofino), and (3) HH flows on Tofino, and non-HH flows go through the Tofino and then x86 server to perform UE address match and encapsulation (H-UPF). All three cases are evaluated for a different number of UEs and 4 Days of CAIDA-16 traces. In addition, H-UPF is also analyzed for different HH threshold ranges.

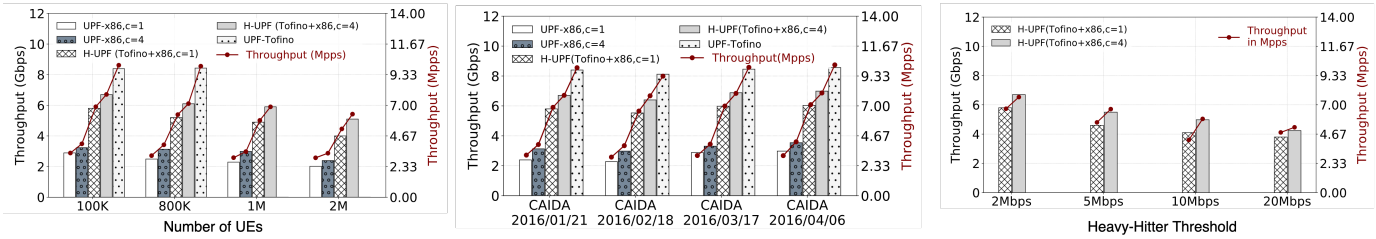
As shown in Figure 7(a), as expected, UPF achieves a maximum of 2-3 Mpps, while UPF-Tofino is 3xtimes faster than x86 for 100K and 800K UEs. Also, we notice that the UPF-Tofino performance does not reach the line rate (i.e., 10 Gbps) because Tofino drops the IPv6 traffic in CAIDA traces. On the other hand, H-UPF performance lies between them and reaches up to 6-7 Mpps. Since UPF-Tofino hits the limitation to run a maximum of 850K UEs discussed in the previous section, we evaluate the performance of UPF-x86 and H-UPF for 1M and 2M UEs. H-UPF performs 2xtimes better than UPF-x86. We also note that the number of entries to perform exact matches impacts the performance.

Figure 7(b) shows the performance using CAIDA-2016 traces of 4 different days. However, we do not see any significant changes in the performance. In Figure 7(c), H-UPF is analyzed for different HH threshold ranges. For better understanding, we make Table 6, showing the number of

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

TABLE 6: The number of Heavy-Hitter per second for different threshold ranges, where 'f' represents the total number of flows and 'n' is the total number of packets. We get around the same number of HH flows per second for f $\geq$ 500K.

	HHs/sec (f=100K, n=1M)				HHs/sec (f=500K, n=9M)			
ISP Trace	**$\geq$2 Mbps**	**$\geq$5 Mbps**	**$\geq$10 Mbps**	**$\geq$20 Mbps**	**$\geq$2 Mpbs**	**$\geq$5 Mbps**	**$\geq$10 Mbps**	**$\geq$20 Mbps**
CAIDA 2016/01/21	632	186	56	23	513	107	26	13
CAIDA 2016/02/18	612	162	46	16	509	102	19	10
CAIDA 2016/03/17	646	193	62	27	528	119	35	16
CAIDA 2016/04/06	648	196	60	26	518	117	32	17

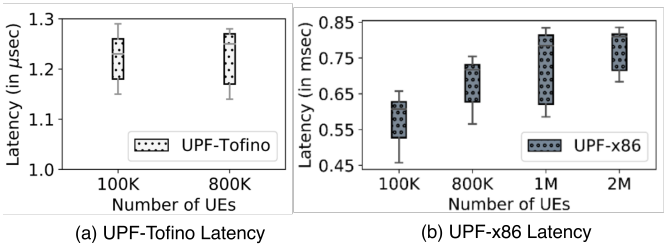


(a) TP for the number of UEs (b) TP for 4 days of CAIDA trace data (c) TP based on different HH thresholds

Figure 7: Throughput (TP) comparison among UPF on x86 (UPF-x86), Tofino (UPF-Tofino) and hybrid UPF on Toinfo+x86 (H-UPF). For H-UPF in Figure(a) and (b), the Heavy-Hitter threshold is set to 2 Mbps. The first y-axis shows the TP in Gbps and the second y-axis represents the TP in Mpps (in red color).

HH flows per second for different ranges of HH thresholds (i.e., 2, 5, 10, and 20 Mbps). For 2 Mbps, Tofino keeps around 600 HH flows, while other flows go through the x86 server. Since 600 flows can have two or more than 2 Mbps flow speed, H-UPF improves throughput by around 3 Mpps. Also, we notice that H-UPF has less improvement in throughput for higher HH thresholds because of less number of HH flows (i.e., 10-25 flows of 20 Mbps threshold).

For UPF latency, we use OSNT, connected with Tofino HW with 10G interfaces, to send packets at the line rate and calculate latency at the OSNT receiver end when the link is saturated. As shown in Figure 8(a) and (b), as expected, UPF latency is around 1.1 to 1.23 microseconds on Tofino HW. However, for x86, latency is between 0.45 to 0.85 msec. We also notice that latency is affected by increasing the number of flows because of the exact match lookup. For H-UPF, we analyze the impact on latency for non-HH flows passing through Tofino and then x86 to match the UE IPv4 address. As shown in Figure 9, offloading around 600 HH flows to the Tofino in case of 2Mbps threshold, we observe lower latency of non-HH flows because of reduction in the x86 bottleneck, but for other threshold values, we do not see any changes compared to UPF-x86.



(a) UPF-Tofino Latency (b) UPF-x86 Latency

Figure 8: Latency for UPF-Tofino and UPF-x86.

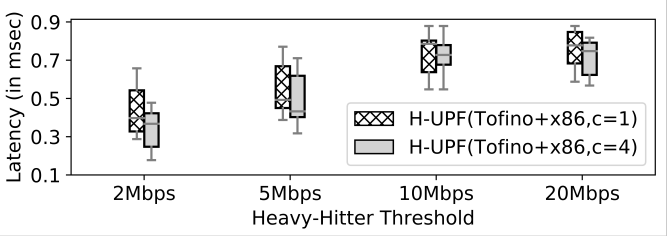


Figure 9: Latency of H-UPF for non-HH flows at different Heavy-Hitter threshold ranges.

6.1.5. Runtime Flow Classification and HW Offloading. As discussed in Section 5.4, we classify the incoming traffic based on the HH algorithm running on the Tofino hardware and then re-route the packet depending on the algorithm decision. To understand the percentage (%) of HH flows offloading on runtime and their overall bandwidth occupancy, we use CAIDA-2016 ISP backbone link traces [9] for analysis on a microsecond scale. We replay the CAIDA traces on line rate (i.e., 10 Gbps) and analyze the number of HHs and their bandwidth occupancy.

As shown in Figure 10, the analysis is performed on the two different HH thresholds values, i.e., 2 Mbps and 20 Mbps. Table 6 provides more insights on the number of HH values based on the total number of flows per sec for other threshold values. In Figure 10, for both the threshold values, only a few HHs utilize 10-40% of the link bandwidth. This confirms that handling heavy flows within the programmable

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

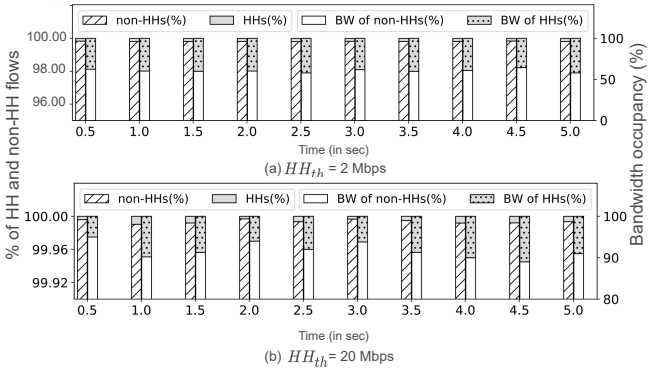


Figure 10: Percentage of HH and non-HH flows and their impact on link bandwidth using CAIDA-16 traces for HH threshold 2Mbps and 20Mbps. The same behaviour of flows has been seen for long duration interval.

switch ASIC, where the resources are limited, would be a better choice to make the network more scalable with high performance.

6.2. Hybrid-UPF (SmartNic + x86)

6.2.1. Methodology. To evaluate the hybrid UPF design with smartNIC offloading, we set up a testbed consisting of the Netronome Agilio CX 2x40G SmartNIC (split into 8x10G links) mounted inside a server with one Intel(R) Xeon(R) Silver CPU with Hyperthreading (HT) enabled for 20 threads at 2.2 GHz each.

We use NetFPGA running the OSNT traffic generator to replay synthetic packet traces while measuring latencies and aggregate throughput. These packet traces are static and pre-generated according to test parameters. Statefulness is pre-populated in the NIC and x86 to match the generated traces, including offloading decisions and tunnel encapsulation metadata for downlink traffic.

The non-deterministic performance of the NIC- and x86 architectures necessitates a wide range of tests to identify performance anomalies and ensure system stability. We have therefore performed thorough tests with varying numbers of active UEs, traffic intensities, offloaded UEs, allocated x86 cores, and overall traffic characteristics for both uplink and downlink flows. Packet sizes range from 128B-1024B, and the packet emission rate aims to reach the target aggregate throughput on the downlink link (where encapsulation headers are included).

The SmartNIC is responsible for performing UPF processing of offloaded traffic and delivering non-offloaded packets down to one of the T4P4S containers running the entire UPF pipeline. Non-offloaded traffic is load-balanced among docker containers through RSS-like hashing of the IPv4 address of the UE, avoiding potential reordering of per-flow packets. The offloaded flows are explicitly stated in an exact-match lookup table in the SmartNIC pipeline, which the controller updates.

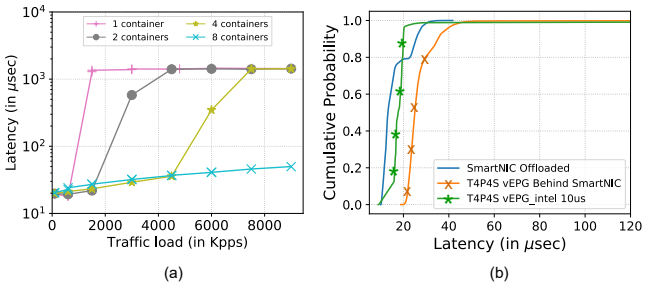


Figure 11: Latency measurements with 100K UEs and 128B packets. (a) Varying number of UPF processing containers/cores, showing UPF capacity scaling. (b) Comparison of offloaded and non-offloaded tunnels, and the imposed latency for non-offloaded traffic by the SmartNIC @1.5MPPS.

6.2.2. Latency Performance. Figure 11 shows the base latency of non-offloaded traffic, where all packets are processed in T4P4S containers on the x86 host machine. The maximum capacity increases near-linearly as we deploy more T4P4S containers (i.e., performing UPF functionality on more cores), showing that the smartNIC is capable of performing decent load balancing while assigning traffic to specific containers. When we use one container at a target rate of 1.2 Mpps for processing 100K UEs, offloading all traffic (blue) significantly reduces the experienced latencies compared to traditional processing in x86 (orange, smartNIC in pass-through mode) as the additional PCIe transfer contributes additional latency as well as a cache miss inside DPDK T4P4S container. Interestingly, when replacing the smartNIC with standard Intel 10G interface (green), lower latencies for host-only based processing is achievable compared to using smartNIC in pass-through mode (orange). This is attributed to the ASIC-based X710 intel card, which has lower and more predictable performance than the Netronome smartNIC, based on 800 MHz flow processing cores. Also, the SmartNIC passthrough pipeline requires an additional table lookup to identify if the packet is to be processed in smartNIC or not.

We can also see in Figure 12 how SmartNIC offloaded traffic experiences a much more stable performance, with a near-nonexistent latency tail. Non-offloaded traffic, processed in T4P4S, does suffer from a long latency tail which can be problematic in several scenarios. We can also see the latency impact on non-offloaded traffic by still passing through the offloading program on the SmartNIC. However, the SmartNIC achieves lower latency than T4P4S even while T4P4S is bound to a fixed-function Intel x710 network card.

6.2.3. Throughput Performance. Figure 13 shows the throughput of non-offloaded traffic in the 8-container T4P4S setup, compared with offloaded traffic processed directly in the SmartNIC. We can see how both of these targets are capable of processing traffic in parallel, and we see in Figure 12 how the SmartNIC-offloaded traffic outperforms non-offloaded traffic in terms of latency. Also, note how the performance of offloaded traffic is highest when fewer

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

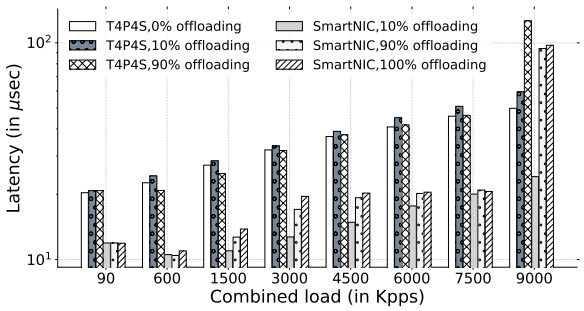


Figure 12: Performance comparison of offloaded vs non-offloaded tunnels in the hybrid UPF

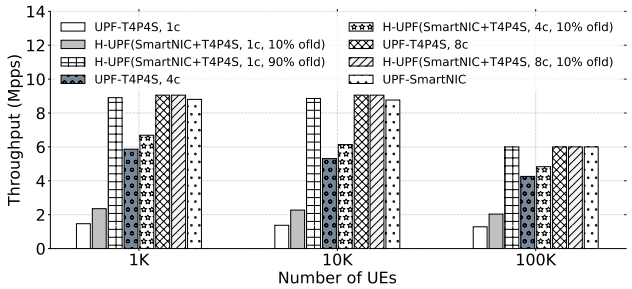


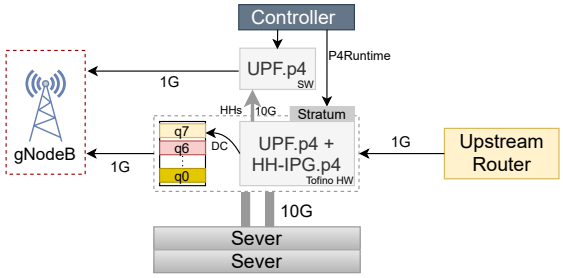
Figure 13: Performance comparison of UPF running on different targets by varying the number of UEs (c: containers, ofld: offloading, H-UPF: hybrid-UPF).

flows are offloaded due to less traffic competing for on-NIC processing. Both offloaded *and non-offloaded* traffic pass through the SmartNIC, which imposes a higher latency on the non-offloaded traffic. When the SmartNIC is under a higher load as P4 packet processing cores seem to be the limiting factor due to the complex pipeline, delaying packet forwarding towards the host. We, therefore, recommend allocating a manageable load for offloading on the smartNIC, depending on the complexity of the pipeline operations.

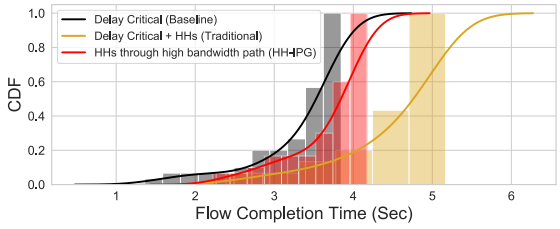
6.3. Use Case Scenarios

In this section, we focus on showcasing and evaluating the performance of the proposed hybrid designs in two use case scenarios: (*i*) UPF QoS application, (*ii*) gNodeB buffering service and (*iii*) gNodeB cryptographic service.

6.3.1. UPF-QoS. To meet the Quality of Service (QoS) objectives, it is required to assign the queues of incoming packets based on their priority levels. In practice, however, due to some misbehaving flows (i.e., Heavy-Hitters - HH), the queue buffer fills quickly, and packets start to drop or causes delay for other flows. For a solution, we can keep the normal flows in the P4 switch ASIC and re-route the misbehaving or heavy-hitters to a high bandwidth path to improve the QoS. Specifically, we consider delay-critical (DC) flows and push them to the high priority queue. If the switch detects any DC flow misbehaving (i.e., heavy-hitter)



(a) QoS-HH Use-case Scenario



(b) Flow completion time with and without HHs offloading

Figure 14: 5G mobile network scenario (a) and distribution of Flow Completion Time (b) of delay critical flows, when HHs IPG are re-routed to the high bandwidth link (red), compared to a traditional pipeline (yellow).

in the same queue, affecting the flow completion time of other DC flows, it will be re-routed to the high bandwidth path.

As shown in Figure 14, we consider a mobile network where the Tofino switch ASIC running the UPF pipeline with HH functional block is connected with an x86 server running with the same UPF pipeline. The Tofino switch and x86 server are connected to three different hosts acting as gNodeB, Application Servers, and Upstream Router facing the Internet. Stratum [40] is used to configure the port shaping rate from 10G to 1G interface to establish a QoS analysis congestion environment. The required entries to configure the HH algorithm and other match-action tables, as are necessary for forwarding packets, are pushed using P4 Runtime [28]. We used 20 TCP flows as delay critical to download the 15 Mbytes data from the Internet for analysis. These flows share the available link bandwidth.

We analyze three different test-case scenarios. First, as a baseline, we analyze the Flow Completing Time (FCT) for each delay critical TCP flow without any background traffic. The second scenario represents a traditional setup, where the FCT of DC flows are calculated with concurrent HH DC flows (occupied 40% of link bandwidth with around 80 Mbps flow rate). Finally, in the third case, we show the FCT gains of DC flows when the detected HHs are re-routed to the high bandwidth path where the UPF pipeline is running on the x86 server. Adapting the hybrid design with the HH algorithm, we can see the difference in FCT for the different scenarios, as shown in Figure 14(b). In the most realistic traffic scenario (DC + HHs), the FCT distributions increase around 1-2 Seconds, compared to baseline (DC). On the

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3201512

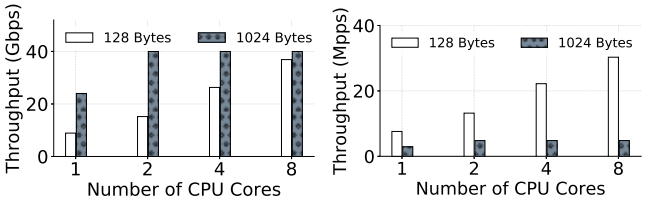


Figure 15: Performance of the buffer service showing packet processing rate and throughput scaling with the number of cores for different packet sizes.

other hand, after offloading HHs to the SW path, the FCT reaches around the same as the baseline.

6.3.2. Hybrid-gNB buffering service. As described in section 5.6, the packet processing component of a base station can be built using a P4 switch with a few generic CPUs running the buffer as a service (BaaS) module. The bottleneck at the BaaS is the focus of the analysis in the next section.

Performance benchmarks. Our buffering service benchmarks ran on three nodes. Node A handles the upstream and the downstream (Intel XL710 40GbE NIC, 32GB RAM, Intel(R) Xeon(R) CPU E5-2630 Processor); Node B, a Stordis BF2556X-1T a P4 hardware switch; Node C, the buffering service (Intel XL710 40GbE NIC, 64GB RAM, Intel(R) Xeon(R) CPU E5-2680 v3 Processor).

As shown in Figure 15, the buffering performance scales well with the number of cores. The one-core performance is 7.6 million packets per second using 128-byte packets while scaling up to 8 cores 30 Mpps is reached. We can achieve line-rate processing with just two cores with more realistic packet sizes (e.g., 1024 bytes).

Calculations on gNodeB performance. The performance of the BaaS module is mainly limited by the memory architecture of the given CPU. The above measurements show that on a reasonably low-performance machine with four cores, this can hit 40 Gbps with average-sized packets on the Internet (IMIX avg. = 353 bytes).

Note that the buffer modules only need a copy of each downlink frame, meaning that its latency does not increase the overall latency of the solution: the main packet path only uses the Tofino switch. To sustain a peak rate of 200 Gbps that a typical 5G base station handles, one will need five low-end servers or two high-end servers. This would require five or two ports on the switch. Additional 2+2 ports would be used for the up and downlink directions.

It is clearly visible that in its current form, Tofino is not suitable for such a network function - its performance is severely underutilized: even the smaller version has 32x100GbE ports and 3.2 Tbps switching capacity. The maximum utilization of the gNodeB use case is below 20%, which shows that such edge network functions would require a different trade-off between capacity and memory size. Sacrificing ports and switching capacity for some additional RAM would lead to more suitable hardware for implementing the packet processing functions of a gNodeB.

6.3.3. Hybrid-gNB cryptographic service. The cryptographic service can be implemented using a purely software-based approach or a cryptographic hardware accelerator like Intel QAT. In environments where encryption is mandatory, the cryptographic service could be the main bottleneck since all packets must be passed through. To measure its effect, we have created a simple DPDK-based application implementing encryption and decryption operation (SNOW 3G) as a software device (no HW acceleration). The evaluation was done on the same machine as the buffer service. The obtained single-core encryption performance was 4.6 MPPS in the case of 64 Byte packet size, and it scaled linearly with the number of cores where the scaling factor was about 0.87. The average latency of the encryption step (64B packet) on a single core was 0.212 microseconds; the DPDK-based component resulted in 8-10 microseconds latency in normal operation (non-overloaded). The software implementation of cryptographic operations could be the main bottlenecks in such a system, but there are cryptographic coprocessors and accelerator cards like Intel QAT that can significantly increase the performance of such computational heavy operations. We had no access to such a card, but according to the available performance tests [2], more than 40 Gbps with Internet-average packet size can easily be reached with HW acceleration.

7. Conclusion and Future Work

This paper explores hybrid design models for 5G UPF and gNodeB, combining the features of different P4 targets to make the user plane more powerful and flexible. Each proposed hybrid approach has its trade-offs and seeks to overcome P4 language limitations and challenges towards realizing softwarized 5G data planes. Functions unsupported in the native P4 language, such as ARQ and ciphering/deciphering, are implemented in DPDK as P4 extern functions. We vertically and horizontally split the entire pipeline and compiled it on different P4 targets, namely x86, SmartNIC, and Tofino ASIC. The extensive experimental evaluation shows that hybrid user planes are an attractive approach to meet feature requirements and improve overall scalability with high throughput and low latency.

Besides acting as an infrastructure component, Tofino ASICs are suitable for offloading heavy traffic flows if the number of rules is relatively small. This way, programmable HW pipelines are suitable for running a simplified user plane (e.g., for user equipment not requiring complex treatment) or for offloading specific users or flows selectively based mainly on their intensity. SmartNICs are in-between smart switches and CPUs when it comes to performance. So although they are less limited in terms of memory and thus the number of rules that could be handled, they can handle much fewer heavy users. Their main tasks would be to act as a protection and load sharing layer in front of the CPUs. Besides that functionality, they could also run some user plane functions roughly similarly to Tofino. Still, due to their more limited bandwidth, the number of users or flows could be smaller. Another option could be to run more complex

user plane rules in SmartNICs, this way entirely offloading those users from the CPUs - this option was not covered by the current study as the SmartNIC used for the tests was not capable of handling that high complexity. As new SmartNICs are expected to be available on the market in the coming years, the offload architecture might also evolve.

To be able to carry out the proper placement of functions, a critical aspect has adequate strategies to select which flows or users are handled by hardware pipelines and by the software counterparts. We leverage a heavy hitter technique based on inter-packet gap metrics that runs entirely in the P4 data plane at affordable resource costs with high accuracy and timeliness. In the future, refinements of the HH detection module along with other candidates HW/SW splitting techniques and slicing mechanisms should be explored. More specifically, we are already working on an intelligent P4 'chained load balancer' from P4 Tofino hardware over SmartNIC to specific x86 cores handling Kubernetes workloads for 5G applications. Latency-aware load balancing capability is another research sub-track on this front.

Based on our experiences, including knowledge and technology transfer to business units of telecommunication equipment providers, and analyzing the observed trends of the networking community, we believe that hybrid data planes are likely to become common in P4-like programmable data planes for different 5G flavors and other networking domains. One final concluding remark would be to expect scale-down (smaller fan-out) versions of Tofino ASICs to fit better the needs of distributed edge and radio gNodeB systems in addition to edge-optimized servers equipped with SmartNIC as programmable networking and compute targets for 5G and beyond.

Acknowledgments

This work was partially supported by the Innovation Center, Ericsson S.A., Brazil, grants UNI.64 and UNI.66. S. Laki and P. Vörös thank the support of the "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided from the NRDI Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme. The views expressed are solely those of the authors and do not necessary represent Ericsson's official standpoint.

References

[1] Cisco. Internet of Things, 2016. https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf.

[2] DPDK Intel Cryptodev Performance Report, 2021. http://fast.dpdk.org/doc/perf/DPDK_21_02_Intel_crypto_performance_report.pdf.

[3] NPL 1.3. Specification, 2021.

[4] Bose Abhik, Maji Diptyaroop, Agarwal Prateek, Unhale Nilesh, Shah Rinku, and Vutukuru Mythili. Leveraging programmable dataplanes for a high performance 5g user plane function. In *5th Asia-Pacific Workshop on Networking*, APNet '21, page 1âĂŞ8, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Alveo SN1000, fully software defined, fully hardware accelerated SmartNIC. [Available]:https://www.xilinx.com/applications/data-center/network-acceleration/alveo-sn1000.html.

[6] Gianni Antichi, Muhammad Shahbaz, Yilong Geng, Noa Zilberman, Adam Covington, Marc Bruyere, Nick McKeown, Nick Feamster, Bob Felderman, Michaela Blott, et al. Osnt: Open source network tester. *IEEE Network*, 28(5):6–12, 2014.

[7] Giulia Attanasio, Claudio Fiandrino, Marco Fiore, and Joerg Widmer. *Characterizing RNTI Allocation and Management in Mobile Networks*, page 189âĂŞ197. Association for Computing Machinery, New York, NY, USA, 2021.

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87âĂŞ95, July 2014.

[9] CAIDA. Anonymized Internet Traces. http://www.caida.org/data/passive/passive_dataset.xml.

[10] Huynh Tu Dang, Han Wang, Theo Jepsen, Gordon Brebner, Changhoon Kim, Jennifer Rexford, Robert Soulé, and Hakim Weatherspoon. Whippersnapper: A p4 language benchmark suite. In *Proceedings of the Symposium on SDN Research*, SOSR '17, page 95âĂŞ101, New York, NY, USA, 2017. Association for Computing Machinery.

[11] DPDK. Home - DPDK. https://www.dpdk.org [Online; accessed 7-June-2021].

[12] R Hattachi, E. Next Generation Mobile Networks, NGMN, 2015. https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf.

[13] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. The p4-> netfpga workflow for line-rate packet processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, page 1âĂŞ9, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Enio Kaljic, Almir Maric, Pamela Njemcevic, and Mesud Hadzialic. A survey on data plane flexibility and programmability in software-defined networking. *IEEE Access*, 7:47804–47840, 2019.

[15] Naga Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. Cacheflow: Dependency-aware rule-caching for software-defined networks. In *Proceedings of the Symposium on SDN Research*, SOSR '16, New York, NY, USA, 2016. Association for Computing Machinery.

[16] James Kempf, Bengt Johansson, Sten Pettersson, Harald LÃijning, and Tord Nilsson. Moving the mobile evolved packet core to the cloud. In *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 784–791, 2012.

[17] Elie F. Kfoury, Jorge Crichigno, and Elias Bou-Harb. An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, 9:87094–87155, 2021.

[18] Ralf Kundel, Tobias Meuser, Timo Koppe, Rhaban Hark, and Ralf Steinmetz. User Plane Hardware Acceleration in Access Networks: Experiences in Offloading Network Functions in Real 5G Deployments. In *55th Hawaii International Conference on System Sciences*, 2022.

[19] Ralf Kundel, Leonhard Nobach, Jeremias Blendin, Wilfried Maas, Andreas Zimber, HansâĂŘJoerg Kolbe, Georg Schyguda, Vladimir Gurevich, Rhaban Hark, Boris Koldehofe, and Ralf Steinmetz. Openbng: Central office network functions on programmable data plane hardware. *Int. J. Netw. Manag.*, 31(1), January 2021.

[20] David Lake, Ning Wang, Rahim Tafazolli, and Louis Samuel. Softwarization of 5G Networks âĂŞ Implications to Open Platforms and Standardizations. *IEEE Access*, PP:1–1, 04 2021.

[21] Yi-Bing Lin, Tse-Jui Huang, and Shi-Chun Tsai. Enhancing 5g/iot transport security through content permutation. *IEEE Access*, 7:94293–94299, 2019.

[22] Robert MacDavid, Carmelo Cascone, Lin Pingping, Badhrinath Padmanabhan, Ajay Thakur, Larry Peterson, Jennifer Rexford, and Oguz Sunay. A P4-based 5G User Plane Function. In *Symposium on SDN Research (SOSR)*. ACM, 2021.

[23] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, Feng Qian, and Zhi-Li Zhang. A variegated look at 5g in the wild: Performance, power, and qoe implications. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 610âĂŞ625, New York, NY, USA, 2021. Association for Computing Machinery.

[24] Katayoun Neshatpour, Maria Malik, Mohammad Ali Ghodrat, Avesta Sasan, and Houman Homayoun. Energy-efficient acceleration of big data analytics applications using fpgas. In *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*, BIG DATA '15, page 115âĂŞ123, USA, 2015. IEEE Computer Society.

[25] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. Understanding pcie performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 327âĂŞ341, New York, NY, USA, 2018. Association for Computing Machinery.

[26] Van-Giang Nguyen, Anna Brunstrom, Karl-Johan Grinnemo, and Javid Taheri. Sdn/nfv-based mobile packet core network architectures: A survey. *IEEE Communications Surveys Tutorials*, 19(3):1567–1602, 2017.

[27] ONF. Aether enterprise-5g/lte-edge-cloud-as-a-service. In *Aether White Paper*, 2020.

[28] P4Rntime. A control plane specification for controlling the data plane elements of a device or program defined by a p4 program.

[29] White Paper. Intel, kaloom create p4-programmable network solutions, 2020. https://builders.intel.com/docs/networkbuilders/intel-kaloom-create-p4-programmable-network-solutions.pdf.

[30] P Gyanesh Patra, Christian Esteve Rothenberg, and Gergely Pongrácz. Macsad: Multi-architecture compiler system for abstract dataplanes (aka partnering p4 with odp). SIGCOMM '16, page 623âĂŞ624, New York, NY, USA, 2016. Association for Computing Machinery.

[31] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. A high performance packet core for next generation cellular networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 348âĂŞ361, New York, NY, USA, 2017. Association for Computing Machinery.

[32] Ruben Ricart-Sanchez, Pedro Malagon, Jose M. Alcaraz-Calero, and Qi Wang. P4-netfpga-based network slicing solution for 5g mec architectures. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 1–2, 2019.

[33] Ruben Ricart-Sanchez, Pedro Malagon, Pablo Salva-Garcia, Enrique Chirivella Perez, Qi Wang, and Jose M. Alcaraz Calero. Towards an fpga-accelerated programmable data path for edge-to-core communications in 5g networks. *Journal of Network and Computer Applications*, 124:80–93, 2018.

[34] Ruben RicartâĂŘSanchez, Pedro Malagon, Antonio MatencioâĂŘEscolar, Jose M. Alcaraz Calero, and Qi Wang. Toward hardwareâĂŘaccelerated qosâĂŘaware 5g network slicing based on data plane programmability. *Trans. Emerg. Telecommun. Technol.*, 31(1), jan 2020.

[35] Jeferson Santiago da Silva, Thibaut Stimpfling, Thomas Luinaud, Bachir Fradj, and Bochra Boughzala. One for all, all for one: A heterogeneous data plane for flexible p4 processing. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 440–441, 2018.

[36] Rinku Shah, Vikas Kumar, Mythili Vutukuru, and Purushottam Kulkarni. Turboepc: Leveraging dataplane programmability to accelerate the mobile packet core. In *Proceedings of the Symposium on SDN Research*, SOSR '20, page 83âĂŞ95, New York, NY, USA, 2020. Association for Computing Machinery.

[37] Harminder Singh, Changcheng Huang, Mathieu Sicard-Gagne, Gauravdeep Shami, Marc Lyonnais, Dmitri Fedorov, and Rodney Wilson. Int-sdn: Evaluation of various p4 parameters using optical telemetry having reconfigurable data plane on 40 gbps line rate. In *2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 1–6, 2019.

[38] Suneet Kumar Singh, Christian Rothenberg, Marcelo Caggiani Luizelli, Gianni Antichi, and Gergely Pongracz. Revisiting heavy-hitters: Don't count packets, compute flow inter-packet metrics in the data plane. In *Proceedings of the SIGCOMM '20 Poster and Demo Sessions*, SIGCOMM '20, page 49âĂŞ51, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Suneet Kumar Singh, Christian Esteve Rothenberg, Gyanesh Patra, and Gergely Pongracz. Offloading virtual evolved packet gateway user plane functions to a programmable asic. In *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms*, ENCP '19, page 9âĂŞ14, New York, NY, USA, 2019. Association for Computing Machinery.

[40] Stratum. An open source silicon-independent switch operating system for software defined networks.

[41] Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo. Flightplan: Dataplane Disaggregation and Placement for P4 Programs. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 571–592. USENIX Association, April 2021.

[42] The P4 Source Code of IPG based Heavy-Hitter detection for Tofino. [Available]:https://github.com/intrig-unicamp/P4-HH.

[43] Marcos A. M. Vieira, Matheus S. Castanho, Racyus D. G. Pacífico, Elerson R. S. Santos, Eduardo P. M. Câmara Júnior, and Luiz F. M. Vieira. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Comput. Surv.*, 53(1), February 2020.

[44] Péter Vörös, Gergely Pongrácz, and Sándor Laki. Towards a hybrid next generation nodeb. In *Proceedings of the 3rd P4 Workshop in Europe*, EuroP4'20, page 56âĂŞ58, New York, NY, USA, 2020. Association for Computing Machinery.

[45] PÃl'ter VÃűrÃűs, DÃąniel HorpÃącsi, RÃşbert Kitlei, DÃąniel LeskÃş, MÃątÃl' Tejfel, and SÃąndor Laki. T4p4s: A target-independent compiler for protocol-independent packet processors. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8, 2018.

[46] Qi Wang, Jose Alcaraz-Calero, Ruben Ricart-Sanchez, Maria Barros Weiss, Anastasius Gavras, Navid Nikaein, Xenofon Vasilakos, Bernini Giacomo, Giardina Pietro, Mark Roddy, Michael Healy, Paul Walsh, Thuy Truong, Zdravko Bozakov, Konstantinos Koutsopoulos, Pedro Neves, Cristian Patachia-Sultanoiu, Marius Iordache, Elena Oproiu, Imen Grida Ben Yahia, Ciriaco Angelo, Cosimo Zotti, Giuseppe Celozzi, Donal Morris, Ricardo Figueiredo, Dean Lorenz, Salvatore Spadaro, George Agapiou, Ana Aleixo, and Cipriano Lomba. Enable advanced qos-aware network slicing in 5g networks for slice-based media use cases. *IEEE Transactions on Broadcasting*, 65(2):444–453, 2019.

[47] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic Sketch: Adaptive and Fast Network-wide Measurements. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2018.

[48] Tong Yang, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. HeavyKeeper: An Accurate Algorithm for Finding Top-k Elephant Flows. In *Transactions on Networking, Volume:27, Issue:5*. IEEE/ACM, 2019.