Offloading Virtual Evolved Packet Gateway User Plane Functions to a Programmable ASIC

Suneet Kumar Singh
University of Campinas
Campinas, Brazil
ssingh@dca.fee.unicamp.br

Christian Esteve Rothenberg
University of Campinas
Campinas, Brazil
chesteve@dca.fee.unicamp.br

Gyanesh Patra
Ericsson Research
Stockholm, Sweden
gyanesh.patra@ericsson.com

Gergely Pongracz
Ericsson Research
Budapest, Hungary
Gergely.Pongracz@ericsson.com

ABSTRACT

Roll-outs of 5G Mobile Packet Core (MPC) rely on principles and technologies of Software-Defined Networking (SDN) and Network Function Virtualization (NFV). While the benefits of SDN and NFV in terms of flexibility are well-known, how to guarantee data plane performance for critical 5G services is less clear. Advances in programmable switch ASICs render an opportunity to offload data plane virtual network functions (VNFs) running on x86 servers to programmable hardware featuring strict performance guarantees. In this work, we present the design and performance evaluation of a critical element of 5G MPC, namely the virtual Evolved Packet Gateway (vEPG). We describe the P4-based uplink and downlink pipelines and evaluate a software and hardware implementation based on a Barefoot Tofino hardware switch, the ONOS controller, and P4Runtime support to manage match-action tables. The obtained results show that vEPG hardware implementation runs at line rate with low latency (<2 μs) and jitter, scaling up to 1.7 millions active users.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **Network performance analysis**; **Mobile networks**;

KEYWORDS

Mobile Core Network; Programmable Switches; P4

ACM Reference Format:
Suneet Kumar Singh, Christian Esteve Rothenberg, Gyanesh Patra, and Gergely Pongracz. 2019. Offloading Virtual Evolved Packet Gateway User Plane Functions to a Programmable ASIC. In *1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms (ENCP '19), December 9, 2019, Orlando, FL, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3359993.3366645

ENCP '19, December 9, 2019, Orlando, FL, USA

ACM ISBN 978-1-4503-7000-4/19/12…$15.00
https://doi.org/10.1145/3359993.3366645

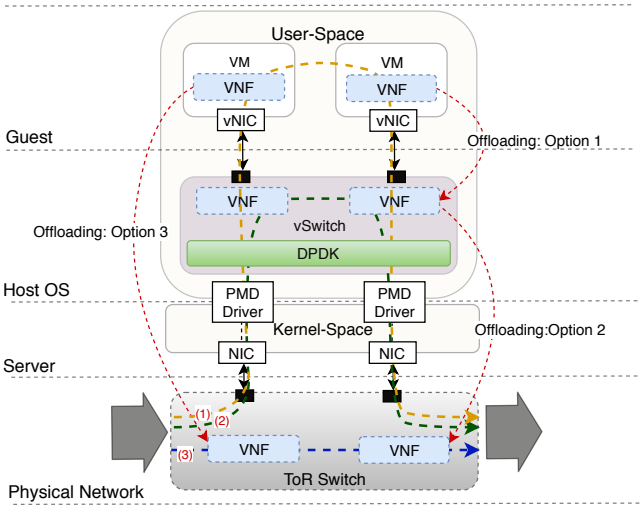


Figure 1: Performance bottlenecks in NFV infrastructure and VNF offloading options in multi-tenant environment to accelerate packet-processing functions

1 INTRODUCTION

Next-generation mobile network (NGMN) describes twenty five use cases for 5G [1], many of which require extremely low latency and high data rates. To satisfy such performance targets and the rapid traffic growth, next generation Mobile Packet Core (MPC) calls for flexible, scalable, and cost-effective approaches [2]. To this end, mobile operators are leveraging key technologies such as Software-Defined Networking (SDN) [3] and Network Function Virtualization (NFV) [4].

A network softwarization approach to the MPC provides agility and flexibility [5] by decoupling the control and data planes, and cost-effectively moving MPC network functions to computational pools at strategic data centers [6]. As shown in Fig. 2, user plane entities of MPC are being deployed in telco cloud infrastructures as Virtual Network Functions (VNFs) on x86 servers, proving adequate performance for some 5G services. However, for time-sensitive applications like autonomous driving, robot control and

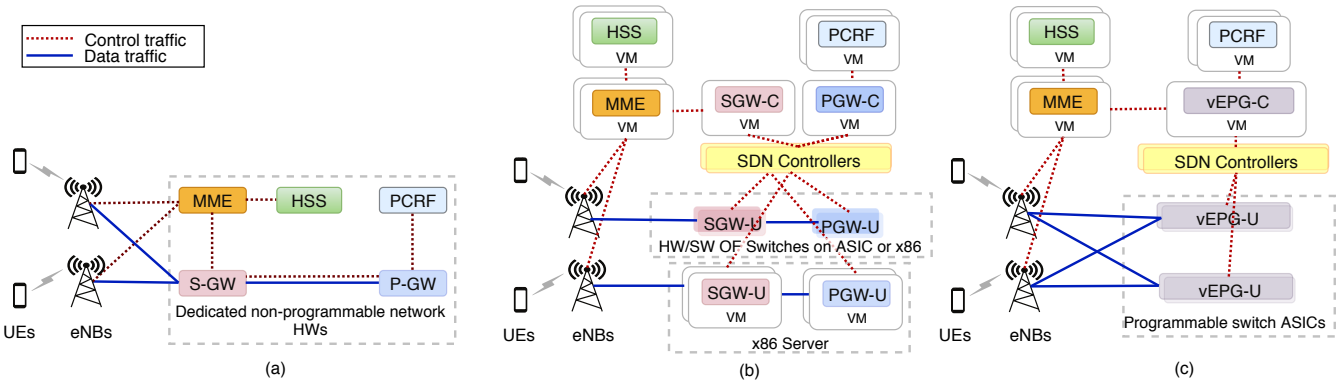


Figure 2: MPC architecture implementation alternatives: (a) Traditional MPC appliances, (b) MPC using SDN and NFV with full and partial virtualization, and (c) MPC user plane functions offloaded to programmable ASICs.

virtual/augmented reality (VR/AR), there is a need to avoid performance bottlenecks [7] of NFV infrastructures and the non-deterministic behavior of x86 software stacks [8, 9].

The advent of programmable switch ASICs and high-level network-specific languages like P4 (Programming Protocol Independent Packet Processors) [10] opens the door to offloading user plane VNFs from x86 servers to the network infrastructure. Figure 1 illustrates VNF offloading options to accelerate packet processing functions in programmable software (SW) or hardware (HW) switches. Performance bottlenecks due to memory read/writes and SW/HW resource sharing [9] can be avoided when directly running network functions in a programmable Top-of-rack (ToR) switch.

In this paper, we investigate offloading MPC user plane functions to P4-capable programmable switch ASICs. More specifically, we present a P4 implementation of a virtual Evolved Packet Gateway (vEPG) that compiles in a Barefoot Tofino switch and uses the ONOS[1] controller to manage the match-action table entries through P4 Runtime[2]. To analyze the performance gains when offloading the vEPG to a P4 switch ASIC, we also compile the vEPG P4 pipeline to a high-end x86 using the MACSAD [11] compiler, which leverages OpenDataPlane (ODP)[3] APIs to run P4 applications on different target platforms (e.g., x86, ARM) at high-speed (DPDK acceleration). Altogether, the contributions of this paper are summarized as follows:

(1) vEPG user plane functions design in P4[4] featuring: (*i*) GPRS tunneling protocol (GTP) (de)encapsulation; (*ii*) VxLAN (de)encapsulation of GTP flows; (*iii*) IP routing towards the Internet and eNodeB nodes; (*iv*) Management of Tunnel Endpoint Identifier (TEID) for flow multiplexing in GTP sessions; (*v*) Stateless firewall; (*vi*) Eth/IP forwarding to/from datacenter gateways.

(2) Implementation of vEPG on Barefoot Tofino hardware switch using P4 Runtime APIs and the ONOS controller to manage table entries.

(3) Experimental evaluation of (*i*) performance of ASIC offloaded vEPG compared to x86 server, (*ii*) scalability for increasing active mobile users, and (*iii*) Barefoot Tofino hardware resource utilization.

(4) Memory efficient P4 match+action mapping methods delivering ~8x increase in table entry capacity (active users).

The remainder of this paper is organized as follows: Sec. 2 describes the background. Sec. 3 presents the vEPG design in P4 along the target use case, and Sec. 4 discusses the implementation setup and performance evaluation. Sec. 5 provides an overview of related work. Finally, Sec. 6 offers concluding remarks and our future work activities.

2 BACKGROUND

2.1 Mobile Packet Core (MPC) 101

MPC refers to the core functional components of modern mobile networks. Figure 2(a) illustrates a traditional Long-Term Evolution (LTE) MPC network, where the signaling gateway (SGW) routes uplink and downlink IP traffic to radio network evolved Node B (eNB) stations. SGWs handle multiple eNodeBs and provide handover between eNodeBs or between LTE and other 3GPP technologies. The Packet Data Network Gateway (PGW) connects the MPC with external IP networks and performs packet filtering, charging, and Quality of Service (QoS) enforcement as instructed by the Policy and Charging Rules Function (PCRF).

Unlike the SGW, the PGW acts as anchor point for mobility between 3GPP and non-3GPP networks. Both SGW and PGW use GTP for the communication between eNodeB and PGW via SGW, which maintains the GTP sessions between eNodeBs and the PGW. The Mobility Management Entity (MME) is responsible for security procedure such as end-user authentication with the support of Home Subscriber Server (HSS), terminal-to-network session handling, and user tracking across the network.

The Evolved Packet Gateway (EPG) merges the functions of both SGW and PGW and can be implemented as a physical (bare-metal appliance) or a virtual node (vEPG VNF).

[1]https://github.com/opennetworkinglab/onos
[2]https://github.com/p4lang/p4runtime
[3]https://opendataplane.org/
[4]https://github.com/intrig-unicamp/macsad-usecases/tree/master/p4-16

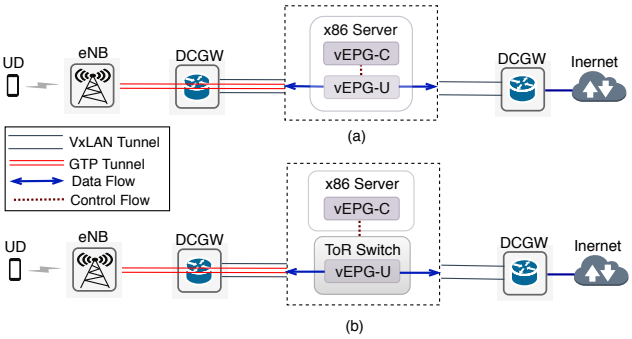


Figure 3: vEPG offloading use case : vEPG-U running on (a) x86 server and (b) Barefoot Tofino ToR switch.

2.2 Programmable ASICs and P4

Programmable ASICs such as Intel Flexpipe, Cisco Doppler, Cavium (Xpliant) and Barefoot Tofino are emerging networking technologies that allow developers to add new protocols and custom packet de-parsing and match-action pipelines.

With growing support, P4 [10] simplifies the programmability of datapath pipelines through high-level language constructs. Programming ASICs with P4 allows implementing feature-rich datapaths supporting high data rates at deterministic ultra-low latency with less power consumption compared to x86 approaches. Therefore, the rationale of offloading MPC user plane functions to programmable ASICs.

3 VIRTUAL EPG P4 PIPELINE & USE CASE

Figures 2 (b) and 2 (c) show SDN- and NFV-based MPC architectural approaches with different candidate targets to run MPC gateway functions. Our goal is implementing in P4 a vEPG combining both SGW and PGW user plane functions. The target vEPG use cases are presented in Fig. 3, where (a) vEPG user-plane (vEPG-U) and vEPG control-plane (vEPG-C) run on the x86 server, and (b) vEPG-U is offloaded to a programmable ASIC.

We divide the vEPG datapath in uplink and downlink sub-pipelines where vEPG-UL indicates the traffic coming from user devices (UD) and vEPG-DL refers to traffic from the Internet towards the users. All implemented user plane functions are given in Tab. 1. Fig. 4 presents all vEPG implemented functions in terms of P4 match-action tables. The total size of the P4 program is around 750 lines of code.

Effectively occupying 100% of the available memory footprint to maximize the amount of active UDs ultimately depends on the P4 program structure in addition to potential program- and target-specific compiler optimizations – a topic we scratch the surface of later in the evaluation section. Next, we provide further details on the vEPG pipeline.

Infrastructure Virtualization. VxLAN provides the isolation capabilities among VNFs in multi-tenant environments. VxLAN (de-)encapsulation of L2 frame within UDP header and handling outer L2 forwarding and L3 routing towards data center gateways (DCGWs) are common for both vEPG-UL and vEPG-DL. Outer and inner IP and MAC addresses represent underlay and overlay network, respectively.

Table 1: User Plane Functions of vEPG Uplink (vEPG-UL) and vEPG Downlink (vEPG-DL)

vEPG-UL	vEPG-DL
Match Outer L2 and L3	
VxLAN Decapsulation	
Validate IPv4/GTP	
Match vEPG 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	

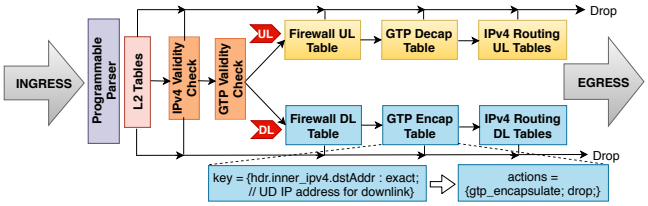


Figure 4: P4-defined vEPG pipelines featuring the main Uplink (UL) and Downlink (DL) tables

L2/Ethernet. Incoming packets are parsed and then passed through the L2 tables to execute two separate lookups for checking source and destination outer MAC addresses.

Firewall. After successful key matches in L2 tables, outer IPv4 and GTP headers validity check is performed to select the packet flow towards UL or DL pipeline. If GTP headers are valid, packets follow UL, otherwise they pass through the DL pipeline. In both the cases, the global firewall rule is applied to mitigate the unauthorized access of servers or EPC network connected to the internet.

GTP Decap/Encap. In GTP (de-)encapsulation , the vEPG removes or adds GTP headers. GTP encapsulation table lookup is responsible for checking UDs IP addresses, while GTP decapsulation table matches vEPG IP address to apply the actions accordingly. Also, this table manages inner L2/L3 MAC, the TEID, and routes inner IP packets towards the eNodeB or DCGW.

IPv4 Routing. Outer IP routing towards DCGWs.

4 EVALUATION

Like any HW-SW system design, offloading MPC gateway functions to programmable hardware is subject to certain trade-offs. The fixed memory resources of ASICs provide performance guarantees but pose hard limits on the number of active UDs, which could be 'arbitrarily' large (e.g., 10^6 - 10^9) in vSwitch or VM/containers on high-end x86 servers. Furthermore, in programmable ASICs, match-action tables are assigned to stages containing limited amount of resources following specific (fixed) allocation rules.

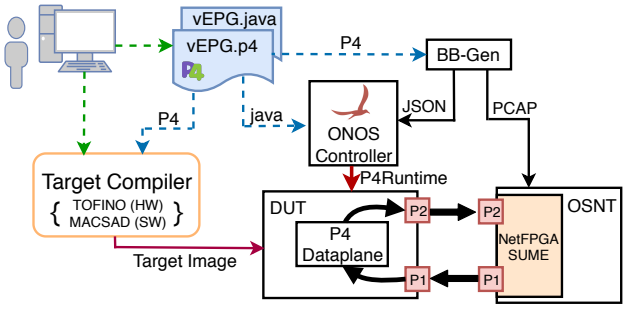


Figure 5: Testbed for vEPG evaluation on HW and SW.

After describing testbed, we evaluate the performance of vEPG in HW and SW targets. The objective is not to directly compare both targets but to profile their performance behaviour. Finally, we introduce two P4 code optimization methods to re-design match-action tables to reach larger active UD capacity.

4.1 Testbed

Figure 5 shows the testbed featuring Open Source Network Tester (OSNT) [12] as the traffic generator using NetFPGA-SUME with 10G SFP+ interfaces connected to the device under test (DUT), in our case (1) Edgecore Wedge 100BF-32X with Barefoot Tofino ASIC[5], or (2) x86 server with Intel Xeon D-1518 processor (4 CPU cores, Cache 6M, 2,20 GHz) for the vEPG application generated by the MACSAD compiler for ODP-DPDK x86 targets (cf. [13]). Moreover, forwarding error correction (FEC) setting is disabled in the Tofino interfaces.

The same vEPG pipeline functions (Tab. 1) written in $P4_{16}$[6] are used as an input P4 code for the Barefoot and MACSAD compilers to generate the target images for the Tofino HW and x86 SW platforms, respectively. P4Runtime through ONOS controller (version 2.1.0) REST APIs is used to manage the table entries in the Barefoot Tofino DUT. From the vEPG P4 code, we create the required java files such as the pipeline interpreter for mapping between ONOS known headers/actions to P4 specific entities among other files to create an ONOS application.

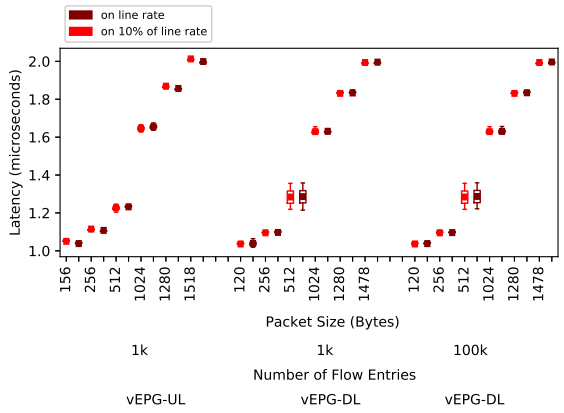
The BB-Gen [14] tool is used to generate the JSON file for the required table entries to be populated by ONOS and the PCAP packet traces to feed OSNT to generate the test traffic.

4.2 Performance

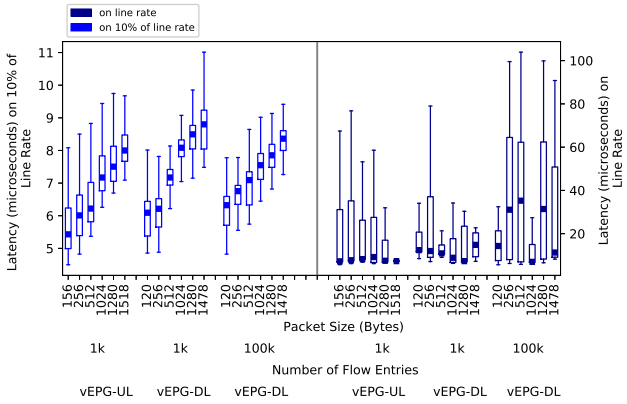
We analyze the vEPG performance in terms of latency and throughput on the hardware switch (Barefoot Tofino HW) and the x86 server (MACSAD SW) for various traffic patterns and packet sizes. We measure latency under two different load scenarios. Firstly, OSNT sends packets at 10% of line rate, and, secondly, packets are transmitted at line rate.

Latency is measured one way end-to-end by OSNT (see Fig. 5) and includes propagation, transmission, queuing, and processing delays (cf. [12]). OSNT hardware measurements add 16 bytes headers to the original packet for TX/RX timestamps.

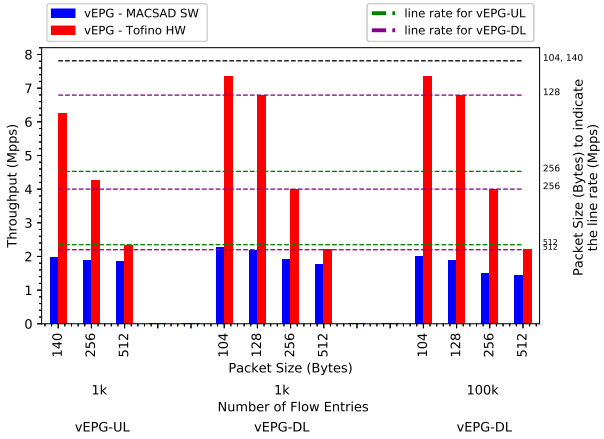
[5]https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=335
[6]Minor differences in $P4_{16}$ code are due to target architecture specificities



(a) Latency for Barefoot Tofino HW



(b) Latency for x86 using MACSAD SW



(c) Throughput

Figure 6: vEPG performance on HW and SW targets.

Throughput is calculated in million packets per second (mpps). Minimum packet sizes are 140 and 104 bytes for vEPG UL and DL, respectively. Since 36-byte headers are added/removed during GTP (de-)encapsulation, the received packet size is modified accordingly.

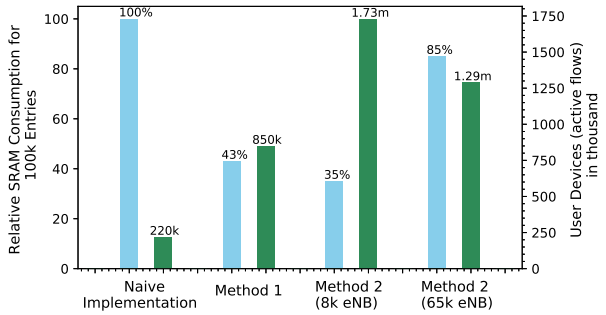


Figure 7: Resource consumption in Tofino hardware using different optimization methods.

Table 2: L2 and vEPG on Tofino HW (highlighted rows) and x86 (PS: Packet Size in Bytes; 10% of Line Rate; 1k Entries)

PS	Average Latency (μs)			Throughput (Mpps)		
	L2	vEPG-DL	vEPG-UL	L2	vEPG-DL	vEPG-UL
156	5.01	6.17	5.42	5.24	1.98	1.87
156	0.97	1.04	1.04	7.10	5.89	6.02
256	5.52	6.23	6.10	4.20	1.92	1.80
256	1.04	1.10	1.09	4.52	4.00	4.25
512	5.90	7.10	6.23	2.26	1.76	1.85
512	1.24	1.25	1.25	2.34	2.20	2.34

For table key matching, we consider 1k unique IP addresses to apply firewall rules for each Firewall UL and DL table. Since the number of table entries in the GTP Encap table depends on the number of UDs, we use 1k, 10k, and 100k unique table entries (UD IP). A single entry is considered for each of the remaining tables in the vEPG pipelines.

Performance for different workloads. Due to the Tofino HW line rate packet processing, the observed performance of vEPG-UL and vEPG-DL is not influenced by the traffic rate. The latency of the vEPG on Tofino HW (Fig. 6(a)) remains below 2μs with the packet size being responsible for the transmission and serialization delays [15]. Compared to x86 (see Fig. 6(b)), the observed latency is about 2-4x lower than in case of 10% line rate and 20-40x at line rate – an evidence of the non-deterministic behavior (cf. large jitter) of x86 packet processing due to cache misses, branch mis-predicts, long latency instructions and exceptions, etc. [16].

Increasing active user devices. We now measure latency and throughput as number of active user devices increases. As shown in Fig. 6, except for smallest packet sizes (104 bytes), vEPG achieves line rate on the Tofino HW for all packet sizes when increasing user devices from 1k to 100k. The throughput is 2-8x higher on the Tofino hardware compared to x86. Increasing user devices does not affect the vEPG performance on either platforms due to the deterministic nature of the hardware ASIC, and the use of hugepages by DPDK to efficiently handle the TLB (Translation Lookaside Buffers) in case of x86 servers [9].

Pipeline complexity. As a baseline performance experiment, we run a simple L2 use case on both platforms and analyze the impact of the vEPG pipeline complexity (Tab. 2). As one would expect from the HW implementation, we do not see any significant performance difference between L2 and vEPG. In contrast, latency and throughput in the x86 are impacted by the increased pipeline complexity. These outcomes confirm that, in addition to overall total packet processing capacity requirements (e.g., 3.2 Tbps of 32x100G interfaces), for increasing user plane functions complexity and whenever strict performance guarantees are required, programmable ASICs become the candidate design choice.

4.3 Scalability / HW Resource Utilization

We now turn the attention to the scalability properties by analysing the average ASIC hardware resource utilization for an increasing amount of active user devices. We focus the HW resource consumption analysis on the vEPG-DL where the UD IP exact-match lookup is performed and SRAM becomes the critical resource. We analyze the resource utilization of different implementation alternatives, from our initial naive one to more optimized table designs.

Naive Implementation. We use our very first P4-based vEPG pipeline design for 100k UDs as a baseline to evaluate the SRAM consumption. As shown in Fig. 7, up to 220k entries could be compiled. We observed that large amounts of memory were occupied by GTP Encap action data (including constant values – hence the naiveness of the implementation). Under these circumstances, to scale up the total number of entries, we identified some P4 code optimization methods based on minimizing the action data per table entry and memory efficient match+action table splits. The basic idea is to execute the UD IP lookup on a table with less action data, reducing the memory consumption and, therefore, allowing for an increased number of table entries.

Method 1. The UD IP lookup is performed with an action to set the TEID and destination IP (i.e. eNB address) for downlink sessions, whereas the remaining actions are executed independently. Under this arrangement, the GTP Encap table becomes narrower and can scale up to 850k UDs. For 100k UDs, this method consumes 43% of SRAM when compared to the naive implementation.

Method 2. The GTP Encap table is divided into two tables, one for UD's IP lookup with an action to set the corresponding eNB ID, and a second table to execute the actions using the eNB ID as a key to index the eNB 32-bit IP to forward the packets. This method allows up to 1.73 million(m) table entries for 8k eNBs (13-bit eNB ID), and 1.29 m entries for 65k eNBs (16-bit eNB ID) – an arguably over-provisioned design of eNBs in scope of a single vEPG node. For 100k UDs, compared to the naive approach, 35% and 85% of SRAM is occupied for 8k and 65k eNBs, respectively. The increased scale obtained through such a table split method is explained by the action data bits (16 or 13) saved per entry in the larger GTP Encap table at the cost of an extra table with fewer entries (65k or 8k) of 32-bit action data (eNB IP) each.

More details about both the methods can be find in the given reference P4 code.[4] Lessons learned from these initial P4 pipeline design optimization methods suggest that different levels and types of code optimization techniques should be investigated to achieve higher scalability by effectively using the target HW resources.

5 RELATED WORK

Approaches to re-design MPC [17] include (*i*) NFV-based [18], (*ii*) SDN-based [19], and (*iii*) SDN/NFV-based-MPC architectures [20]. MPC approaches based on SDN/NFV are prevailing due to flexibility [5] and backward compatibility among other factors [17]. Decoupling the control and user planes of MPC allow them to scale separately in a cost-effective manner.

In [2, 21, 22], authors propose MPC architectures implementing SGW and PGW user-plane functions in OpenFlow switches with GTP tunneling support and MPC control functions executing in the cloud. [13] and [15] point to P4 programmable switches as capable of defining complex forwarding pipelines with high performance.

Efforts in [23, 24] are closest to our work on MPC data planes embracing P4 to overcome well-known limitations from OpenFlow on protocol flexibility, extensibility, and full hardware support. [23] presents a P4 implementation of a Broadband Network Gateway (BNG) compiled into P4-NetFPGA, Netronome P4-SmartNIC, and Barefoot Tofino switch. The performance evaluation against x86 targets provide similar insights compared to our work sustaining the potential gains of hardware offloading approaches. [24] outlines a VNF offloading approach of S/PGW functionality into a data center fabric solution for MPC where GTP headers are encapsulated in HW switches. The S/PGW pipeline design in P4[7] follows the Method 1 implementation described in our work.

6 CONCLUSIONS AND FUTURE WORK

Our research on hardware offloading of MPC user plane gateway functions running at line rate suggests a promising outlook, featuring low and bounded latency, with the potential to scale to over 1.5 millions of active users in a commercial P4 switch. P4 programs, as code written in any other programming language, are quite amenable to optimization. The design of the P4 pipeline tables may have a very high impact on the final system scalability, as we observed in our alternative designs and optimization methods.

In future work, we will devote efforts to implement some missing features of vEPG, such as charging functionality and rate limiting, in addition to more elaborated traffic management features per user or service type in spirit of network slicing. In parallel, different P4 code optimization techniques will be studied and generalized to scale the resource usage of ASIC for stateful applications. Another research strand towards a highly scalable vEPG approach includes a hybrid design that dynamically spreads P4 match action tables and user state across multiple HW switch ASICs and SW servers.

ACKNOWLEDGEMENTS

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreement UNI.63. We would like to thank the Barefoot FASTER program and Vladimir Gurevich for the insightful discussions that contributed to our understanding of memory allocation in the P4 code optimization methods.

REFERENCES

[1] NGMN Alliance 5G Initiative team. Ngmn 5g white paper. *White paper*, pages 1–125, 17 February, 2015.

[2] M. Ylianttila. O.L.Perez M.U. Itzazelaia E.M. de Oca J. Costa-Requena, I.A.M. Liyanage. SDN and NFV Integration in Generalized Mobile Network Architecture. *IEEE European Conference on Networks and Communications (EuCNC)*, June 2015.

[3] D. Kreutz, F. M. V. Ramos, P. E. VerÃŋssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[4] Yong Li and Min Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[5] M. He, A. M. Alba, A. Basta, A. Blenk, and W. Kellerer. Flexibility in softwarized networks: Classifications and research challenges. *IEEE Communications Surveys Tutorials*, 21(3):2600–2636, thirdquarter 2019.

[6] S. Pettersson H. Luning J. Kempf, B. Johansson and T. Nilsson. Moving the mobile Evolved Packet Core to the cloud. *Wireless and Mobile Computing, Networking and Communications (WiMob)*, 103:784–791, Oct, 2012.

[7] C. Maciocco K. B. Ramia S. Kapury A. Singhy J. Ermanz V. Gopalakrishnanz A. S. Rajan, S. Gobriel and R. Janaz. Understanding the bottlenecks in virtualizing cellular core network functions. *Proc. 21st IEEE Int. Workshop Local Metropolitan Area Netw. (LANMAN)*, pages 1–6, 2015.

[8] A. S. Rajan A. Mohammadkhan, K. K. Ramakrishnan and C. Maciocco. Considerations for re-designing the cellular infrastructure exploiting software- based networks. *In ICNP*, pages 397–413, 2016.

[9] L. Linguaglossa, S. Lange, S. Pontarelli, G. RÃľtvÃąri, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi. Survey of performance acceleration techniques for network function virtualization. *Proceedings of the IEEE*, 107(4):746–764, April 2019.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[11] G. Patra, C. E. Rothenberg, and G. Pongracz. Macsad: High performance dataplane applications on the move. In *2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, June 2017.

[12] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. Mckeown, N. Feamster, B. Felderman, M. Blott, A. W. Moore, and P. Owezarski. OSNT: Open Source Network Tester. *IEEE Network*, 28(5):6–12, Sep. 2014.

[13] G. Patra, F. E. Rodriguez, J. S. Mejia, D. L. Feferman, C. E. Rothenberg, and G. Pongracz. Towards a sweet spot of dataplane programmability, portability and performance: on the scalability of multi-architecture P4 pipelines. *IEEE Journal on Selected Areas in Communications*, 36(6):3–14, 2018.

[14] F. Rodriguez, G. Patra, L. Csikor, C. Rothenberg, P. Vörös S. Laki, and G. Pongrácz. Bb-gen: A packet crafter for p4 target evaluation. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, pages 111–113, New York, NY, USA, 2018. ACM.

[15] R. Joshi P. G. Kannan and M. C. Chan. Precise time-synchronization in the data-plane using programmable switching ASICs. *ACM Symp. SDN Res*, (2):8–20, 2019.

[16] S. Moore V. M. Weaver, D. Terpstra. Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations. *IntâĂŹl Symp. on Performance Analysis of Systems and Software*, 2013.

[17] K. Grinnemo V. Nguyen, A. Brunstrom and J. Taheri. SDN/NFV- based Mobile Packet Core Network Architectures: A Survey. *IEEE Communications Surveys & Tutorials*, 19(3):1567–1602, 2017.

[18] P. Loureiro F. Z. Yousaf, J. Lessmann and S. Schmid. SoftEPC: Dynamic instantiation of mobile core network entities for efficient resource utilization. *Proc. IEEE Int. Conf. Commun. (ICC)*, pages 3602–3606, 2013.

[19] Z. Mao L. Li and J. Rexford. Toward software-defined cellular networks. *Software Defined Networking (EWSDN)*, pages 07–12, 2012.

[20] A. S. Rajan A. Mohammadkhan, K. K. Ramakrishnan and C. Maciocco. CleanG: A clean-slate EPC architecture and control plane protocol for next generation cellular networks. *ACM CoNEXT Workshop Cloud Assist. Netw. (CAN)*, pages 31–36, 2016.

[21] G. Hasegawa and M. Murata. Joint bearer aggregation and control- data plane separation in LTE EPC for increasing M2M communica- tion capacity. *IEEE Glob. Commun. Conf. (GLOBECOM)*, pages 1–6, 2015.

[22] J. Kaippallimalil and H. A. Chan. Network virtualization and direct Ethernet transport for packet data network connections in 5G wireless. *IEEE Glob. Commun. Conf. (GLOBECOM)*, pages 1836–1841, 2014.

[23] R. Kundel, L. Nobach, J. Blendin, H.J. Kolbe, G. Schyguda, V. Gurevich, B. Koldehofe, and R. Steinmetz. P4-BNG: Central Office Network Functions on Programmable Packet Pipelines. *IEEE International Conference on Network and Server Management (CNSM)*, pages 21–25, October 2019.

[24] C. Cascone and U. Chau. Offloading VNFs to programmable switches using P4, March 2018.

[7] https://github.com/opennetworkinglab/onos/blob/master/pipelines/fabric/src/main/resources/include/spgw.p4