Towards an eBPF+XDP based Framework for Open, Programmable and Scalable NextG RANs

Udhaya Kumar Dayalan, Ziyan Wu, Gaurav Gautam, Feng Tian, Zhi-Li Zhang
Department of Computer Science & Engineering, University of Minnesota – Twin Cities, U.S.A.
{dayal007, wu000598, gauta044, tianx399}@umn.edu, zhzhang@cs.umn.edu

2023 IEEE Future Networks World Forum (FNWF) | 979-8-3503-2458-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/FNWF58287.2023.10520475

***Abstract*—Starting with 5G, radio access networks (RANs) are moving towards an disaggregated architecture, with most of its functionality (except for the low-level PHY layer) implemented in software. While software affords the benefits of programmability and scale-out, it is also far slower than hardware. This is further compounded by the needs for more complex, dynamic and intelligent features in Next-Generation (NextG) RANs. In this work, we advocate an eBPF (extended Berkeley Packet Filter)+XDP (eXpress Data Path) based framework for scaling and accelerating software packet processing in (O-RAN compliant) NextG RANs. Using 5G Central Unit User Plane (CU-UP) as a key case study, we present an initial design of our proposed framework, dubbed PRANAVAM, and it's the key components. We also discuss additional options for further improvements. Our preliminary evaluation results shows that PRANAVAM improves the 5G CU-UP throughput by 22-26%, compared with the existing (open-source) 5G RAN implementation.**

***Index Terms*—5G, RAN, 5G Throughput, O-RAN Central Unit, eBPF, XDP**

I. Introduction

With the needs for more flexibility, openness and programmability, not only cellular core networks but also radio access networks (RANs) are moving towards virtualization and cloudification. Both 3GPP and the Open-RAN (O-RAN) Alliance have introduced new *disaggregated* RAN architectures that divide 5G RANs into, for example, *Central Unit* (CU) and *Distributed Unit* (DU). CU is further split into CU-UP (CU user plane) and CU-CP (CU control plane), see §II for more details. In particular, the O-RAN Alliance has introduced intelligent RAN controllers (RICs) and defined open interfaces for communications among the disaggregated units and RICs. Softwarization or "cloudification" of 5G and Next-Generation (NextG) RANs and core networks are especially appealing to many industrial use cases, as it makes it easier to support industrial verticals and *private* 5G [1] and NextG networks For example, RAN functionality can be tailored to the specific bandwidth, latency and reliability requirements of these use cases, and existing features may be upgraded or new features can be readily rolled out as the requirements or use cases change over time.

While software affords the benefits of programmability and scale-out, software implementation of RAN is in generally far slower than dedicated hardware appliances. This is further compounded by the needs for more complex, dynamic and intelligent features in NextG RANs. This is particular the case for applications that that require a large number of simultaneous connections, high bandwidth, low latency and stringent reliability such as many industrial IoT (Internet of Things) and Industrial 4.0 Digital Twins use cases. Therefore, *scaling the NextG RAN software architecture while maintaining its programmability and openness is a key challenge in future RAN development*.

In this work we advocate an eBPF+XDP-based framework for scaling and accelerating software packet processing in NextG RANs. As a concrete example, we focus on 5G CU-UP as a key case study. On the one hand, CU-UP performs the upper layers of the 5G RAN protocol stack – Service Data Adaptation Protocol (SDAP) and Packet Data Convergence Protocol (PDCP) – and does not require specialized radio signal processing hardware (see §II). On the other hand, CU-UP often connects with several UPFs (User Plane Functions) in the 5G core as well as multiple DUs. As it lies on the critical path between the users and service endpoints, it must be capable of processing 10s or 100s millions of downlink packets from the core network to user equipment (UE) and uplink packets from UE to the core network per second in order to meet the bandwidth demands and minimize latency. Taking advantage that connections between CU-DU and CU-UPF are Ethernet-based, we exploit exploit eBPF and XDP for kernel extension, kernel bypassing and software packet processing optimization. We summarize the **key contributions** of our paper below.

- We present an eBPF+XDP-based framework, dubbed PRANAVAM, for (O-RAN compliant) future RAN architecture development. Using 5G CU-UP as a key case study, we outline the initial design of our proposed PRANAVAM.
- Using eBPF+XDP for kernel extension/bypassing, our preliminary evaluation shows that PRANAVAM improves the throughput by 22-26% over existing 5G RAN implementations. We will make our code publicly available.
- We also discuss additional options to further accelerate software packet processing to scale 5G RAN implementation to meet bandwidth and latency demands.

While our initial design focuses on 5G RAN CU-UP, the ultimate goal is to apply PRANAVAM as a general framework for future RAN architecture development to meet the needs for openness, programmability, scalability and evolvability.

II. Background

We provide a quick overview of the 5G RAN protocol stack and 3GPP/O-RAN disaggregated RAN architecture. We end by a brief discussion of eBPF/XDP.

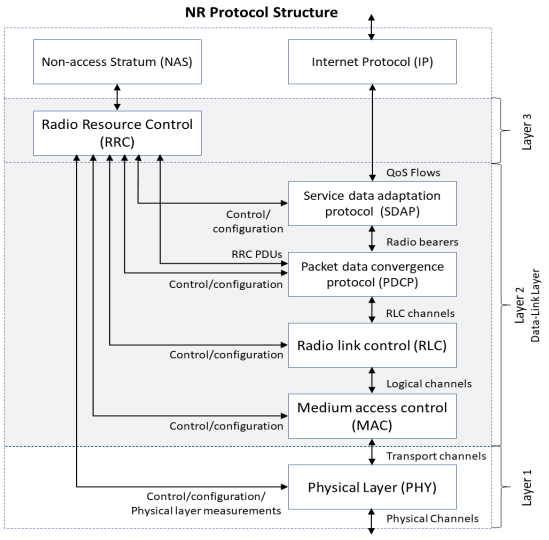

Figure 1: 5G Radio Access Network Protocol Stack.

A. 5G RAN Protocol Stack and O-RAN

Fig. 1 depicts the 5G RAN protocol stack specified by 3GPP, which resides below the OSI network layer ("IP layer"). 5G RAN functions are traditionally performed by *dedicated* (and *closed*) physical appliances (5G "base stations"), i.e., 5G nodeB (gNB) (see Fig. 3), that are supplied by cellular equipment vendors. 3GPP also introduces a CU-DU split RAN architecture which is adopted by O-RAN: under this split, CU performs the upper layer functions of the RAN protocol stack, namely, SDAP, PDCP and RRC (radio resource control) layers; whereas DU performs the lower layer functions, namely, RLC (reliable link control), MAC (media access control) and PHY (physical) layers. While 3GPP also discussed multiple options for possibly further splitting the lower layer functions of DU, e.g., between MAC and PHY or upper and lower parts of the PHY layer, they were not pursued further by 3GPP. Instead, O-RAN adopts a version of the latter option and standardizes it. Under the so-called *7.2x split* specified by O-RAN, DU performs the RLC, MAC and the upper part of the PHY layer functions, whereas RU (radio unit) performs the lower part of the PHY layer functions. O-RAN further split CU along the control and user (data) plane separation, and introduces two components: CU-CP which performs RRC and PDCP control plane functions, and CU-UP which performs SDAP and PDCP user plane functions. We refer the reader to 3GPP specifications [2], [3], [4] and O-RAN specifications [5] for details. In this paper, we assume a *disaggregated* RAN architecture that follows the O-RAN standard, and thus the disaggregated units are O-CU (O-CU-CP/O-CU-UP), O-DU and O-RU (see Fig. 4, where we indicate *select* standardized interfaces *between the key units of interest*[1]). Subsequently, we will drop the prefix "O-" for clarity.

[1]In the figure we have also included the additional O-RAN components such as SMO (service and management orchestration), non-real-time RAN intelligent controller (non-RT RIC), and near-real-time RIC (nRT RIC). Since these components are irrelevant to this paper, so we will not elaborate here.

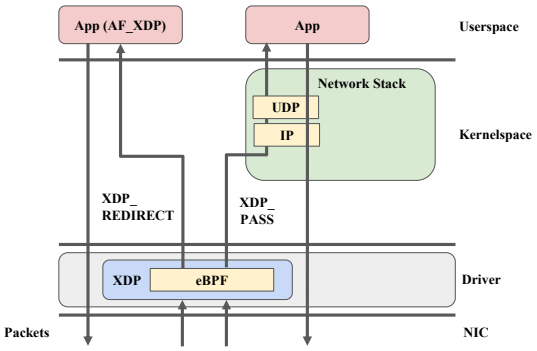

Figure 2: eBPF/XDP Sockets

As depicted in Fig. 4, CU-UP typically connects to multiple UPFs (via the NG interface) on the 5G core side, and may connect to multiple DUs (via the F1 interface) on the RAN side (the suffixes "-U" and "-C" in the interface names distinguish the user plane and control plane versions of the standardized interfaces). Hence it may become a bottleneck in processing the downlink and uplink traffic between UPFs and DUs. We note that both the 3GPP/O-RAN NG-U and F1-U interfaces are implemented using the GTP (GPRS Tunnelling Protocol [6]) tunnels, more specifically, GTP-U tunnels, which run on top of UDP/IP over Ethernet. Hence CU-UP can be implemented entirely using commodity servers with conventional Ethernet-based network interfaces (NICs) (and possibly also Ethernet-based smartNICs). In contrast, while DU connects to CU via the F1 interface, the connection between DU and RU requires a specialized *radio fronthaul* interface, the extended Common Public Radio Interface (eCPRI) [7].

While we can incorporate eBPF+XDP to optimize the packet processing in DU for its F1-U interface with CU-UP, in this paper we will focus on CU-UP due to its critical role in the user plane data path between DU and UPF. The main SDAP function in CU-UP involves adding or removing QFIs (quality-of-service flow identifiers) for downlink data packets from UPF to DU or uplink data packets from DU to UPF, based on (pre-defined) user data's QCI (QoS class identifier) profiles (QCI tables). The PDCP-U functions are more involved: besides integrity protection and ciphering, the PDCP-U layer is also responsible for reliable data transfer by adding sequencing numbers, buffering data, and performing retransmissions if needed. After adding/removing the SDAP and PDCP headers, CU-UP routes the user data packets using appropriate GTP-U tunnels to DUs/UPFs.

B. eBPF/XDP

Extended Berkeley Packet Filter (eBPF) [8] and eXpress Data Path (XDP) [9] are (relatively) recent innovations in the Linux kernel that allow safe kernel extension and kernel bypassing for more efficient network processing, among other usages. While eBPF can be used for both the transmit and receive side operations, XDP operates only at the receive side, residing within the NIC driver (see Fig. 2 for an illustration). We assume the reader has some familiarity with eBPF & XDP, thus will not elaborate further.

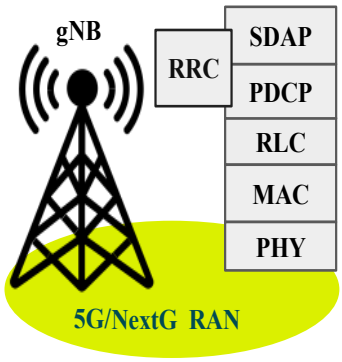

Figure 3: Monolithic RAN

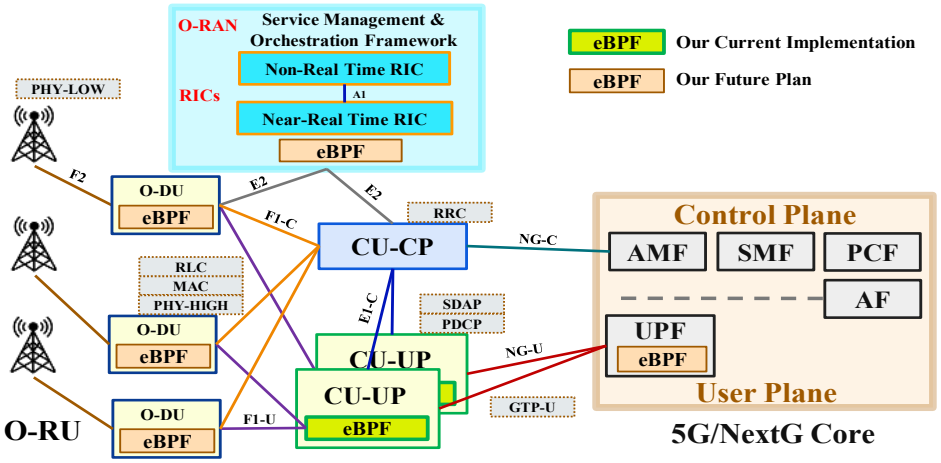

Figure 4: **O**-RAN Disaggregated RAN Architecture and 5G Core

III. Design

In this section, we present PRANAVAM's architecture and design for fast processing of user plane packets in CU-UP of the RAN. The overall design of PRANAVAM is schematically sketched in Fig. 5.

Features. The key feature of CU-UP is *routing* and *forwarding* packets to respective DUs in downlink or UPFs in uplink. The main features supported by PRANAVAM are listed below:

- *Connect* to the configured CU-CP through E1-C interface in-order to get configuration information. The configuration information includes the AMF and UPF information.
- *Update* the QFI to DRB (Data Radio Bearer) mapping table and setup respective DRBs for each connected UEs.
- *Maintain* GTP-U tunnels as PDU (Packet Data Unit) Sessions with UPFs for each connected UE.
- *Fast packet processing* of downlink user plane data to DUs and fast packet processing of uplink data to UPFs.

eBPF Maps. There are several types of eBPF Maps [10] available and each map is used for a particular purpose. In our design, we use two types of eBPF Maps. First, the *"eBPF XSKMAP Map"*, which is used to redirects raw XDP frames to AF_XDP sockets (XSKs) and it has two ring buffers, *"RX_Ring"* and *"TX_Ring"*. Second, *"eBPF PERCPU_ARRAY Map"*, which is used to store and retrieve traffic statistics between the kernel and user space. More details about how these eBPF Maps are used are discussed in detail in the below sections.

Design. As shown in Fig. 5, PRANAVAM is divided into three layers: (i) Management Layer, (ii) Data Path Kernel Layer (DPKL), and (iii) Data Path User Layer (DPUL). During uplink or downlink, the user plane data passes through both DPKL and DPUL. The packets are processed differently during uplink and downlink. We will discuss the detailed design of each of these layers below.

A. Management Layer

This is one of the user space layer which manages the control plane part of PRANAVAM. The Management Layer consists of three main components as listed below:

(i) **E1 Session Manager.** It actively manages the E1 Session of CU-UP with CU-CP. During start-up, the CU-UP connects to the configured CU-CP. The CU-UP can connect to only one CU-CP. For each connected UE, the CU-CP instructs CU-UP with necessary information about the UE which includes QFI to DRB mapping and along with the ciphering and integrity protection for each DRBs. The E1 Session Manager instructs the eBPF Program Manager to dynamically load the eBPF program into the kernel space.

(ii) **eBPF Program Manager.** It is responsible to manage the lifecycle of the eBPF program. Based on the instruction from the *E1 Session Manager*, it loads the eBPF bytecode in the kernel space. There will be only one eBPF program per NIC. If the CU-UP machine has multiple NIC, then the *eBPF Program Manager* loads a eBPF program for each NIC.

(iii) **Report Manager.** It manages the status and statistical information of the components of PRANAVAM. It interacts with the *"eBPF PERCPU_ARRAY Map"* to get the sent and received packets information in the DPKL periodically. It also gets the sent and received packets information in the DPUL as well using an API.

B. Data Path Kernel Layer

This is the kernel space layer which process the user plane traffic inside the eBPF/XDP and is one of the Data Path layers. The DPKL uses *"eBPF XSKMAP Map"* to send (downlink) and receive (uplink) packets directly to and from the DPUL.
Downlink During downlink, the CU-UP receives the user plane data from the UPFs. Following are the key components of DPKL for downlink:

(i) **Parser.** This is the first component which gets called for each packet (from the UPF) received in the NIC. The main functionality of the *Parser* is to parse each packet and

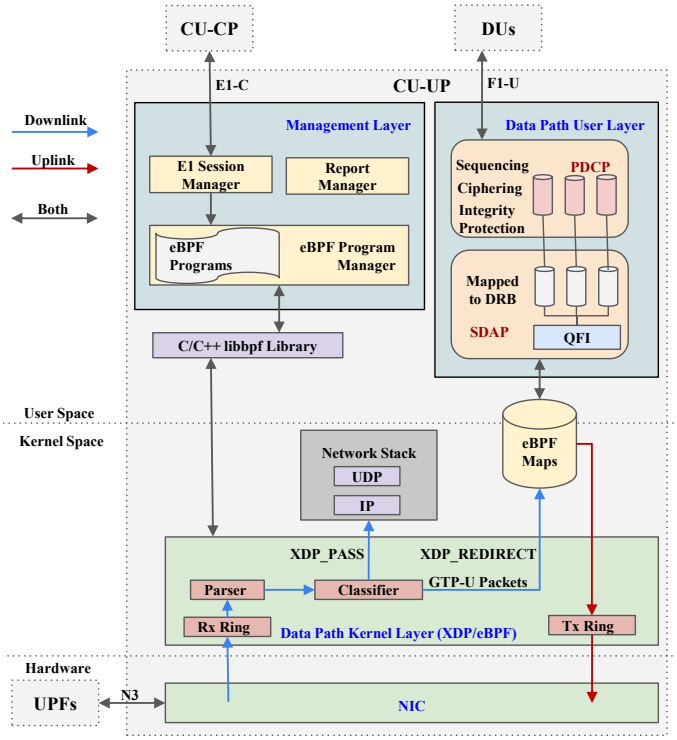

Figure 5: PRANAVAM- eBPF/XDP Socket Based CU-UP

validate it for valid *"ethernet header"* structure. If the packet doesn't have valid structure, then the packets are dropped using *"XDP_DROP"*. The *Parser* component passes the valid packets to the *Classifier*.

(ii) **Classifier.** It classifies and send downlink GTP-U packets to DPUL using *"XDP_REDIRECT"* through *"eBPF XSKMAP Map"* and pass other packets to the network stack using *"XDP_PASS"*. The classifier uses two fields each GTP-U packet to classify downlink GTP-U packets. The first field is *"Message Type"* and the value should be "255" for GTP-U PDU Sessions. And the next field is *"PDU Type"* in the *"PDU Session Container"* extension header of the GTP-U packet which should be "0" for downlink. The value will be "1" for uplink GTP-U packets. When a GTP-U packet contains any other value in either of these fields, then the packet will be sent to the network stack rather than "eBPF XSKMAP Map".

Uplink There is no work in kernel space for uplink because the GTP-U packets are already parsed, classified and inserted directly into the *"eBPF XSKMAP Map"*s *"TX_Ring"* and assigned to the kernel space in the user space itself, which sends the packets to the UPF through the NIC.

C. Data Path User Layer

DPUL is the other user space layer and the other part of the Data Path layer. This layer consists of the higher layers of the RAN protocol stack such as PDCP and SDAP.

Downlink During downlink, the packets from the DPKL are available in *"eBPF XSKMAP Map"*s *"RX_Ring"*. DPUL polls the *"eBPF XSKMAP Map"*s *"RX_Ring"* for packets. First, each packet is processed for SDAP function in which the QFI value is retrieved and respective DRB is assigned in the PDCP function based on the QFI to DRB mapping. Next, the data undergoes the configured PDCP function processing such as integrity protection and ciphering. Finally, the data is routed to the respective DU based on the established PDU session.

Uplink In the current design, *eBPF* is not used for data data traffic between CU-UP and DUs. During uplink, DPUL receives the user plane data from the DUs using regular socket, then process the PDCP function and assign the data to the respective DRBs in which integrity protection and ciphering are done. Next, the SDAP function is processed and the respective DRB to QFI mapping is assigned. Finally, instead of routing the packet to UPF using regular socket, the data is inserted into the *"eBPF XSKMAP Map"*s *"TX_Ring"*. In order to send the data to the UPF, the kernel need to be explicitly notified using the *"sendto()"* call.

IV. Implementation

In this section, we briefly discuss the implementation of PRANAVAM's design discussed in §III. For CU-UP, we considered to leverage either srsRAN [11] or Open Air Interface (OAI) RAN [12]. We decided to leverage OAI because OAI supports the O-RAN split using multiple network interfaces. Even though srsRAN claims that they support O-RAN support, their CU-CP, CU-UP and DU are integrated with tightly-coupled function calls and not network interfaces. The Management Layer and DPUL are developed in C and the DPKL is developed using restrict C. As discussed in the design section, we implemented two types of eBPF Maps and the *"eBPF XSKMAP Map"* has two ring buffers. Not all network interfaces support XDP (native mode), so we added support for both generic and native mode. In generic mode, PRANAVAM will work continue to work without any issues but there won't be any improved performance as seen in native mode. In order to avoid packet loss during heavy traffic and for better performance since we run the application and the driver on the same core, PRANAVAM supports *poll* mode. Even if we run the CU-UP and the kernel driver in different cores, *poll* mode reduces the number of *syscalls* needed for TX path.

V. Preliminary Evaluation

A. Test Setup

The test setup for PRANAVAM's performance evaluation is shown in Fig. 6. We conducted the experiments in a 6 CPU and 8GB RAM Ubuntu 20.04.6 OS virtual machine created on top of 11th Gen Intel(R) Core(TM) i3, 4 Cores, 8 logical processors machine. The end-to-end 5G system which consists of 5G core, RAN and UE was emulated in a single machine. We used OAIs 5G core, CU-CP, DU, UE along with our modified CU-UP for our experiments. Each component (AMF, SMF, NRF, UPF) of the OAIs 5G core were running in its own docker container inside the virtual machine. The CU-CP, CU-UP, DU and UE were ran as processes inside the virtual machine. We didn't isolate the system under test on purpose because we want to show the end-to-end performance results under close to realistic conditions.

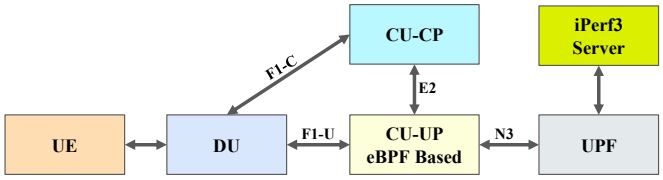

Figure 6: Test Setup

B. Data Traffic Generation

For evaluation, we used iperf3 [13] as a data generator. Our focus was on comparing the performance between regular socket and eBPF/XDP socket. We didn't use high-speed traffic generators because the other OAI components used in the end-to-end tests don't achieve higher throughput. We ran the iperf3 server in a server machine behind UPF and the client in the OAI UE. In the client side, we used -P option to generate multiple parallel flows for generating diverse traffic, -u for UDP packets and -R option to conduct performance evaluation in the downlink direction.

C. Results

Fig. 7 shows the throughput per number of parallel connections from a single UE. The results shows a performance improvement of around 22-26 percent in PRANAVAM in the downlink direction from the 5G core to UE. There is a linearity between the number of parallel connections and the throughput. We also noticed that there is a performance degrade in both regular socket and XDP as the number of parallel connection increases. We have shown the results only for the downlink direction as the results are clearly visible when heavy traffic is going from 5G core to UE than the uplink traffic from the UE to the 5G core. We would like to provide additional clarification on the throughput shown in Fig. 7. The downlink throughput for 1 connection from a single UE was around 40 Mbps because of the throughput limitation of several others components in the end-to-end system used for evaluation. As mentioned in the §IV, we built our solution on top of OAIs 5G core, RAN and UE reference implementation.

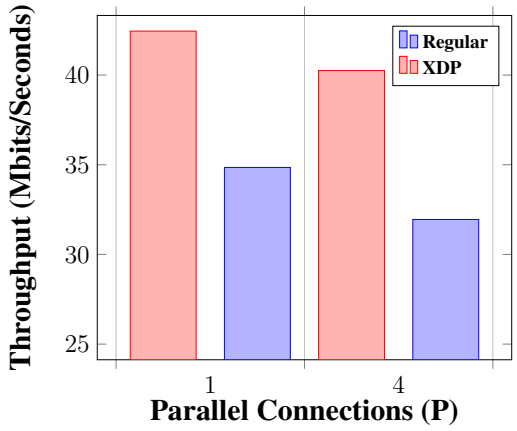

Figure 7: Performance Comparison Between Regular and XDP Socket - Downlink

VI. Additional Design Options

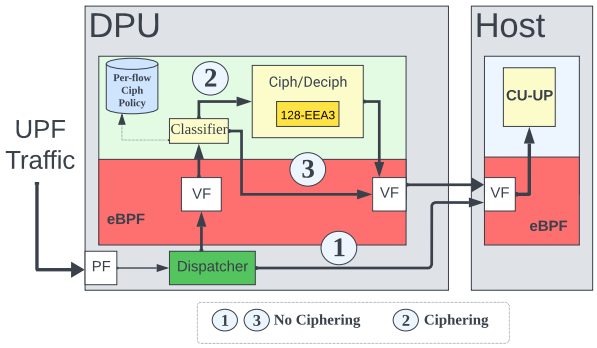

Figure 8: Ciphering Offloading Design

To improve performance and better utilize computational resources, we explore dynamically offloading operations to the DPU, particularly *ciphering*. As it needs to perform operations on every bit of the packet payload, ciphering is CPU intensive [14] and can pollute the cache (which is crucial for the performance of stateful packet processing on the host [15], [16], [17], and mitigate interference on other cores [18]). To improve CPU utilization, cache efficiency, and leverage cost-efficiency cores on DPU, we design a ciphering operations offloading policy and dynamically offload the ciphering/deciphering operations to the SmartNIC. The dynamically offloading policy can be based on, but not limited to, the following factors: the high-level policy associated with the QoS flows on the types of traffic (whether it is sensitive information), the movement of the users (whether user move from a secure environment to an insecure environment), and the load current hardware infrastructure can handle.

Fig. 8 illustrates the design of the offloading of ciphering operations of the downlink traffic path on 5G CU-UP. When traffic arrives at the DPU, the hardware can provide coarse-grained classification based on whether the traffic needs ciphering based on the installed rules (such as OVS rules on Bluefield SmartNIC). If it does not, the traffic will be sent to the host directly for further processing. And if not sure, the traffic will be processed by the eBPF/DPDK based ciphering/deciphering network functions running in the general-purpose processing unit (core) on the SmartNIC. The network function consists of a stateful classifier that classifies packets based on the per-flow ciphering policy, which is dynamically changed based on the high-level policy and current environ-

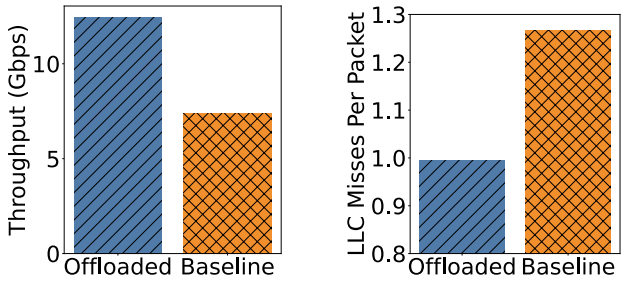

Figure 9: Preliminary Evaluation on Offloading Ciphering.

ment. Based on the policy, the packet will be sent directly to the host or ciphered by the ciphering module. Using this design, we allow for a fine-grained policy to improve both the security and efficiency of the RAN system.

We have performed a preliminary evaluation (Fig. 9) of the potential benefits on a server to show its effectiveness. We play a synthetic traffic of 1024-byte packets with 50% of the traffic that requires ciphering operations. With the offloading feature on, traffic is instead sent to the ciphering component in SmartNIC for ciphering as designed. From the result we show that offloading can improve the performance of the data plane and improve the utilization of the host cache.

VII. Related Work

While eBPF and eBPF/XDP have been widely used in various (wired) networking and cloud computing systems, for example, to optimize service mesh/serverless computing [19], their application to 5G networks are rather limited. We are aware of only two recent papers, both apply eBPF/XDP to 5G core networks. In [20], eBPF/XDP based 5G UPF is implemented and they deploy it in a restrictive environment such as MEC (multi-access edge computing). In another paper [21], the focus is again on eBPF/XDP kernel-based 5G UPF with fallback on user-space for more complex packet handling. In [22], the authors proposed a 5G Mobile Gateway based on eBPF/XDP, but the solution is completely implemented in the 5G core. None of these works consider applying eBPF/XDP to 5G RAN. In a two-page poster paper [23], we outlined an initial design of a purely kernel-based CU-UP design. Building on this, in this paper we advance, implement and evaluate the combination of user-space+kernel-based CU-UP framework.

VIII. Conclusion & Future Work

In this paper, we designed, and evaluated PRANAVAM, an eBPF+XDP based framework for open, programmable and scalable NextG RANs. Our evaluation shows that there is more than 22% improvement in PRANAVAM when compared to using regular sockets. We also discuss the design of offloading the ciphering operations to the SmartNIC using fine-grained, dynamic per-flow ciphering policy to improve the efficiency of the host and the security of the 5G RAN. As a future work, we are planning to implement the entire data plane layers of each components in the NextG system using eBPF and evaluate our system using real-world workload.

IX. Acknowledgement

This research was supported in part by the NSF under Grants CNS-1814322, CNS-1836772, CNS-1831140, CNS-1901103, CNS-2106771 and CNS-2128489, and Seed Grants from University of Minnesota MnRI, CTS and CSE and an unrestricted gift from InterDigital.

References

[1] Cisco. (2023) Cisco private 5g solution overview. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/wireless/private-5g/private-5g-service-so.html

[2] 3rd Generation Partnership Project. (2020, April) Release 15. https://www.3gpp.org/release-15.

[3] ——. (2020, July) Release 16. https://www.3gpp.org/release-16.

[4] ——. (2021, March) Release 17. https://www.3gpp.org/release-17.

[5] O-RAN Alliance. (2023) O-ran alliance. [Online]. Available: https://www.o-ran.org/

[6] Wikipedia. (2022) Gprs tunnelling protocol. [Online]. Available: https://en.wikipedia.org/wiki/GPRS_Tunnelling_Protocol

[7] C. P. R. Interface. (2023) Specification overview. [Online]. Available: http://www.cpri.info/spec.html

[8] (2023) ebpf - introduction, tutorials community resources. [Online]. Available: https://ebpf.io/

[9] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 54–66. [Online]. Available: https://doi.org/10.1145/3281411.3281443

[10] P. Kernel. (2023) ebpf maps. [Online]. Available: https://prototype-kernel.readthedocs.io/en/latest/bpf/ebpf_maps.html

[11] srsRAN Project. (2023) srsRAN. [Online]. Available: https://www.srslte.com/

[12] Open Air Interface. (2022) Open air interface. [Online]. Available: https://openairinterface.org/

[13] iPerf3. (2023, June) iperf3. https://iperf.fr/.

[14] R. Avanzi and B. B. Brumley, "Faster 128-eea3 and 128-eia3 software," in *Information Security: 16th International Conference, ISC 2013, Dallas, Texas, November 13-15, 2013, Proceedings*. Springer, 2015, pp. 199–208.

[15] P. Zheng, W. Feng, A. Narayanan, and Z.-L. Zhang, "Nfv performance profiling on multi-core servers," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 91–99.

[16] Z. Wu, T. Cui, A. Narayanan, Y. Zhang, K. Lu, A. Zhai, and Z.-L. Zhang, "Granularnf: Granular decomposition of stateful nfv at 100 gbps line speed and beyond," *ACM SIGMETRICS Performance Evaluation Review*, vol. 50, no. 2, pp. 46–51, 2022.

[17] H. Ghasemirahni, T. Barbette, G. P. Katsikas, A. Farshin, A. Roozbeh, M. Girondi, M. Chiesa, G. Q. Maguire Jr, and D. Kostić, "Packet order matters! improving application performance by deliberately delaying packets," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 807–827.

[18] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker, "{ResQ}: Enabling {SLOs} in network function virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 283–297.

[19] S. Qi, L. Monis, Z. Zeng, I.-c. Wang, and K. K. Ramakrishnan, "Spright: Extracting the server from serverless computing! high-performance ebpf-based event-driven, shared-memory processing," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 780–794. [Online]. Available: https://doi.org/10.1145/3544216.3544259

[20] T. A. N. do Amaral, R. V. Rosa, D. F. C. Moura, and C. E. Rothenberg, "An in-kernel solution based on xdp for 5g upf: Design, prototype and performance evaluation," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 146–152.

[21] C. Scheich, M. Corici, H. Buhr, and T. Magedanz, "Express data path extensions for high-capacity 5g user plane functions," in *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, ser. eBPF '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 86–88. [Online]. Available: https://doi.org/10.1145/3609021.3609298

[22] F. Parola, S. Miano, and F. Risso, "A proof-of-concept 5g mobile gateway with ebpf," in *Proceedings of the SIGCOMM '20 Poster and Demo Sessions*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2021, p. 68–69. [Online]. Available: https://doi.org/10.1145/3405837.3411395

[23] U. K. Dayalan, Z. Wu, G. Gautam, F. Tian, and Z.-L. Zhang, "Pravega: Scaling private 5g ran via ebpf/xdp," in *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, ser. eBPF '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 89–91. [Online]. Available: https://doi.org/10.1145/3609021.3609303