

6GUPF: A DPU-based Programmable User Plane Function for Enhanced Flow-based QoS

Rana Abu Bakar^{*†}, Ahmed Salah Tawfik Ibrahim[†], Francesco Paolucci[†], Andrea Sgambelluri^{*},
Piero Castoldi^{*}, Filippo Cugini[†], Juan Jose Vegas Olmos[‡]

^{*}Scuola Superiore Sant'Anna, Pisa, Italy

Emails: {rana.abubakar, andrea.sgambelluri, piero.castoldi}@santannapisa.it

[†]CNIT, Pisa, Italy

Emails: {ahmed.ibrahim, filippo.cugini, francesco.paolucci}@cnit.it

[‡]NVIDIA, Copenhagen, Denmark

Email: juanj@nvidia.com

Abstract—Traditional 5G user plane function (UPF) architectures are software-based implementations that struggle to maintain performance. To meet the stringent quality of service (QoS) and scalability requirements of 5G under high session and data plane loads, a highly efficient next-generation UPF is required. In this paper, we present a next-generation 6GUPF that fully leverages the hardware acceleration of SmartNICs/DPUs. The 6GUPF is developed using DOCA Flow API, which enables fast, programmable packet processing directly in the data path, specifically for the N3 gNB network and N6 interfaces of datanet. The architecture integrates flow-based acceleration for stateful flow tracking and symmetric Receive Side Scaling (RSS) to utilize cores and manage non-blocking states efficiently. This will help minimize latency and avoid contention in multi-core environments. Our experiments show that hardware-offloaded UPF achieves an aggregate line rate of 400 Gbit/s, supports 1 million concurrent data streams, and scales up to 500,000 active User Equipment (UE) instances while maintaining QoS and session isolation. It also enables 40% lower latency compared to traditional software UPFs. Unlike Tofino-based P4 switch UPF designs, which are limited by SRAM and TCAM constraints when storing large-scale data streams, our DPU-based solution is ideal for high-density Protocol Data Unit (PDU) deployments without sacrificing programmability and performance. It creates a path for cloud-native, 6G UPF deployments that can scale to a wide range of workloads, from ultra-low-latency applications to large-scale IoT backhaul.

Index Terms—Network, 6G, QoS, Flow, UPF, Traffic Analysis

I. INTRODUCTION

Mobile networks have significantly enhanced communication capabilities, enabling higher data rates, lower latency, and support for billions of connected devices. These developments increase the demand for highly efficient next-generation mobile networks, particularly for video streaming, online gaming, and the Internet of Things (IoT). The development of smarter cities and enhanced industrial automation is also driving the need for these networks. However, emerging use cases, such as autonomous driving, medical robotics, and real-time operational industrial control, demand ultra-reliable, low-latency, and deterministic communication, which are beyond the capabilities of current 5G infrastructures [1], [2].

To meet these evolving demands, 5G architectures have adopted technologies such as Network Function Virtualization (NFV) and Software-Defined Networking (SDN) [3]–[6]. NFV enables the virtualization of core network functions, including the User Plane Function (UPF), on general-purpose hardware, allowing for dynamic deployment and scalability [7]–[11]. SDN further decouples the control and data planes, allowing centralized traffic orchestration and flexible resource management [12].

Despite these advances, virtualized UPF implementations often face performance bottlenecks when deployed on commodity servers or traditional NICs. Kernel-based packet processing, high context-switch overheads, and inefficient handling of GTP tunnels limit their ability to meet the high-throughput and low-latency demands of modern mobile networks [13]. Current approaches [14]–[20] have introduced in-network acceleration using Data Plane Development Kit (DPDK) and P4 to improve performance, yet they fall short in supporting scalable, flow-aware UPFs with hardware offload. DPDK is a widely used software framework that accelerates packet processing on CPUs by bypassing the kernel networking stack. While it reduces kernel overhead, DPDK remains CPU-bound, leading to scalability limitations under high traffic loads. High-performance user-space dataplanes, such as DPDK, eliminate kernel overheads while keeping per-packet work on general-purpose CPUs, which limits scalability at high packet-per-second (PPS) rates. In-kernel eBPF/XDP reduces I/O overhead and enables safe packet processing within the kernel's fast path, it still consumes host CPU cycles and is constrained by verifier/program complexity. In contrast, 6GUPF offloads encapsulation/decapsulation and flow lookup into the DPU hardware pipes, freeing the host CPU even under high PPS.

Recent work, such as HiP4-UPF [19], has shown performance improvements in UPF deployment using P4 programmable switches. By introducing a three-tiered optimization strategy, RED, SPT, and CAD, they addressed rule storage inefficiencies and operational latency. Their approach achieved a 2X increase in UE support and 84.6% lower reporting latency compared to baseline systems. However, HiP4-UPF remains tightly coupled to Intel's Tofino architecture and lacks full on-

board buffering capabilities, which requires external servers for packet storage. P4 targets line-rate parsing/match-action in fixed pipelines. P4 switch-based UPFs achieve low latency but are limited by on-chip memory (TCAM/SRAM) and pipeline depth for stateful processing. Our approach differs by maintaining a fully programmable software control plane while offloading data-plane primitives (GTP-U encapsulation/decapsulation, flow steering) to the DPU. This avoids the table-size constraints and supports larger flow tracking states (e.g., TEID/5-tuple) without sacrificing the line-rate throughput of 400 Gbit/s.

In particular, existing UPF solutions struggle with limitations in the number of flow tracking, CPU-bound GTP processing, and a lack of granular QoS enforcement. These challenges are critical for a transition toward 6G, where networks must support billions of devices and different real-time services and applications.

To address these challenges, we propose *6GUPF*, a novel DPU-based programmable UPF that leverages hardware offload, flow-based programmability, and stateful traffic tracking using NVIDIA’s DOCA SDK API. Flow management is achieved using Stateful Flow Table (SFT) offload, which enables efficient per-flow tracking in hardware. While the SmartNIC/DPU supports Deep Packet Inspection (DPI) offload, this work leverages SFT for scalable flow state management. We demonstrate 6GUPF’s ability to emulate and handle over 500,000 concurrent UE IDs (TEIDs), significantly outperforming existing software-based solutions (DPDK, bmv2-P4) and hardware-based solutions, such as P4-switch (Tofino).

Table I compares several UPF solutions and highlights the scalability advantages of hardware offloading.

TABLE I
COMPARISON OF UPF SOLUTIONS WITH HARDWARE OFFLOAD

UPF	UEs	SRAM	TCAM	Flows
SD-Fabric [21]	35.8K	34.2%	17.4%	~100K
SD-Fabric-UPF [21]	55.3K	43.3%	4.2%	~150K
HiP4-UPF [19]	159.7K	85.4%	9.7%	~400K
6GUPF	≥500K	~45% (DRAM)	N/A	~800K

Notably, our proposed Bluefield-3 (BF-3) UPF, leveraging SFT offload, demonstrates support for over 500,000 UEs and 1 million flows, while maintaining moderate SRAM and TCAM usage. This significantly outperforms other state-of-the-art implementations, underscoring the potential of 6GUPF to meet the performance and scalability demands of next-generation mobile networks.

II. PROPOSED 6GUPF METHODOLOGY

In this paper, we propose a novel architecture for 6GUPF that integrates a DPU to offload computationally intensive tasks such as GTP-U encapsulation, decapsulation, and forwarding by leveraging the hardware acceleration capabilities of NVIDIA’s DOCA Flow API. The 6GUPF architecture is depicted in Figure 1.

The ingress pipeline is structured around five primary tables, each performing a distinct function. The *GTP Default Forward*

Table routes GTP-U packets for processing, while the *IP Default Forward Table* manages the forwarding of standard IP packets. The *GTP Encap Table* inserts GTP-U headers for outgoing packets, whereas the *GTP Decap Table* removes the GTP-U headers from incoming flows. Finally, the *L2 Forward Table* ensures correct layer-2 forwarding by modifying the source MAC address as required after encapsulation or decapsulation.

At the initial state, the DOCA Flow pipeline contains only default rules, forwarding all traffic to the GTP/IP Default Forward Tables. During runtime, dynamic rules are installed into the DPU, enabling packets that match specific headers (e.g., TEID, 5-tuple) to be fully processed on the DPU without host intervention. Each table is implemented as a DOCA Flow stage $C_1 \dots C_5$, as summarized in Algorithm 1, which formalizes the C_1 as Ingress/Classify, C_2 as QoS assignment, C_3 as GTP processing, C_4 as L2 Forwarding and C_5 as Egress. If no rule is matched, packets will follow the default path to the host for further processing and potential rule update, ensuring a robust fallback mechanism.

Algorithm 1 DPU-Based GTP-U Flow Handling

Require: DOCA Flow stages $C_1 \dots C_5$: C_1 Ingress/Classify, C_2 QoS assignment, C_3 GTP processing, C_4 L2 Forwarding, C_5 Egress.

Tables \leftrightarrow stages: GTP/IP Default $\rightarrow C_1$, QoS $\rightarrow C_2$, Encap/Decap $\rightarrow C_3$, L2 Fwd $\rightarrow C_4$.

Load initial rule set \mathcal{R} (e.g., TEID/5-tuple) into DPU.

for each packet p **do**

$[C_1]$ Extract headers; lookup $r \in \mathcal{R}$.

if r found **then**

 Assign flow ID f_i and priority q_p .

$[C_2]$ Tag QoS per q_p .

$[C_3]$ **if** GTP-U \Rightarrow decap; **elseif** required \Rightarrow encap.

$[C_4]$ Resolve next-hop MAC; $[C_5]$ enqueue to SR-IOV/VF or DN.

else

 Steer p to host via default rule.

 Control plane update \mathcal{R} (install/learn rule).

end if

end for

A. Flow Initialization and Processing in a DPU-based 6GUPF Network

Figure 2 describes the sequence of interactions involved in flow initiation, acceleration, aging, and de-acceleration within a 6GUPF network using a DPU. This process involves three primary components: the N3/N6 interfaces, the BF-3/DPU, and the 6GUPF. The sequence flow diagram demonstrates how packets are processed, accelerated, monitored, and eventually deaccelerated based on flow context and aging criteria.

When a new flow is initiated, the N3/N6 interface receives a packet and forwards it to the BF-3 DPU (step 1). The DPU then sends the packet to the 6GUPF (step 2), where a flow context is created and the flow is categorized (step 3). Once the flow context is established, the 6GUPF signals the BF-3 to

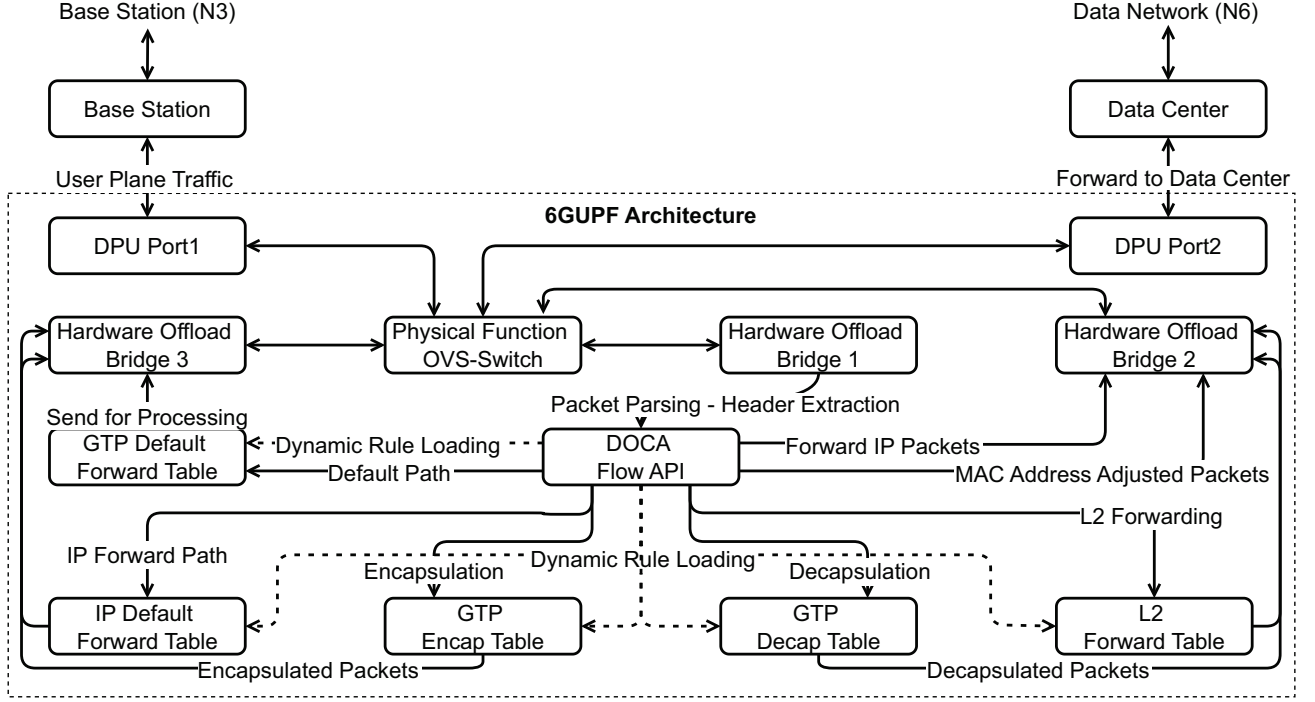


Fig. 1. Packet Processing and Forwarding in DPU-Based 6GUPF

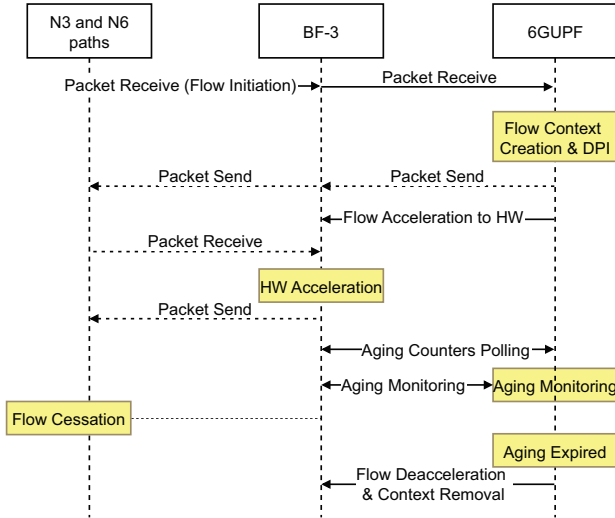


Fig. 2. Flow Initiation, Acceleration, and Deacceleration Process in a DPU-based 6GUPF

accelerate the flow processing to hardware, optimizing packet handling for subsequent transmissions (step 4). The packet is then sent back through the N3/N6 interface, leveraging the hardware-accelerated processing (steps 5-6).

Figure 3 illustrates the DPU-based 6GUPF QoS architecture, where IP flows are first parsed and processed by the DOCA QoS Engine. This engine performs classification, policy

enforcement, load balancing, and scheduling using the DPU offload pipeline. Each uplink/downlink IP flow is encapsulated into GTP-U PDUs and associated with a QoS Flow Identifier (QFI). These QFI flows are mapped to Data Radio Bearers (DRBs) via the gNodeB's Service Data Adaptation Protocol (SDAP) layer. The DRB mapping ensures that the QoS levels defined in the core are considered at the radio interface. Notably, DOCA handles QoS classification, enforcement, and flow state in hardware, allowing the system to scale with minimal CPU intervention.

III. IMPLEMENTATION OF 6GUPF

To evaluate the proposed 6GUPF architecture, we implemented a realistic testbed on a Dell R760 server with an NVIDIA BF-3 DPU, integrating both control- and data-plane functions representative of 6G core deployments. Figure 4 illustrates this setup, which connects a base station emulator to a data center traffic sink. Traffic is generated from the Base Station, which represents the 5G radio access network (RAN), and is forwarded via the N3 interface as GTP-U encapsulated packets. These packets are injected using an emulated VIAVI/TRex traffic generator, which emulates high-throughput and latency-sensitive mobile traffic over L2 links.

At the core of the testbed lies a Dell R760 server equipped with a BF-3 DPU, responsible for the hardware offloading of UPF packet processing workloads. Inside the BF-3 DPU, the DOCA Flow API manages programmable pipelines for flow parsing, GTP-U header encapsulation/decapsulation, and traffic

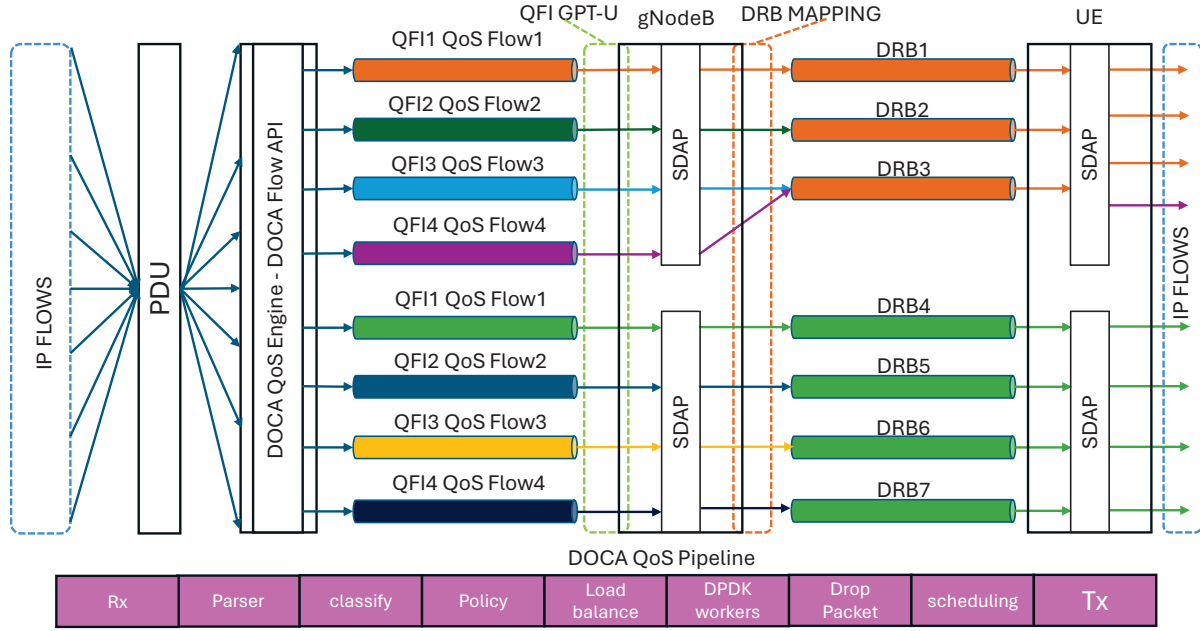


Fig. 3. Flow-Based QoS at UPF-Downlink, gNodeB, and UE

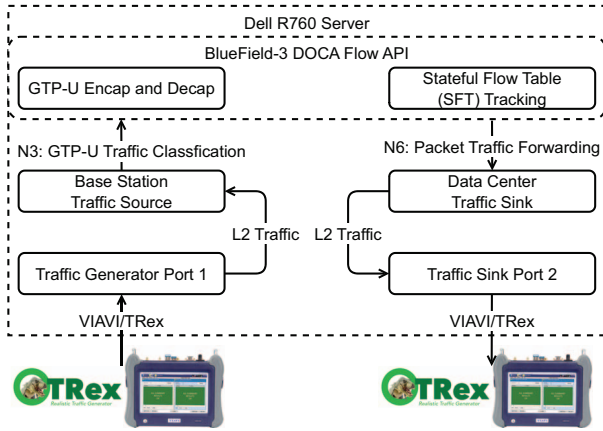


Fig. 4. DPU-accelerated 6GUPF testbed emulating GTP-U traffic for 6G core evaluation.

classification. A key component of the system is the SFT for flow tracking, which maintains per-flow metadata to enable intelligent state-aware forwarding at line rate.

Following classification and processing, the decapsulated packets are forwarded over the N6 interface as standard IP traffic to the Data Center, acting as the traffic sink. Traffic sink validation and replay are facilitated via another port of the DPU, which connects to the VIAVI/TRex traffic generators. Traffic is generated with a Gaussian mixture of 70% elephant flows (long-lived) and 30% burst mice flows, calibrated across MTUs from 128B to 1500B. The traffic generator produces a mix of GTP-U and IP packets to simulate real-world 6G network scenarios.

This architecture encapsulates a production-realistic deploy-

ment scenario where the UPF is embedded directly within the data path of the base station. Packet processing is conducted based on QoS policies, which are dynamically assigned to packets according to their classification and network conditions. High Priority is assigned to packets requiring minimal delay for network applications. Medium Priority is assigned to packets that require high throughput but are not latency-sensitive. Low Priority is assigned to packets that are less sensitive to latency and throughput.

The DOCA Flow API manages the ingress pipeline with five stages (as defined in Sec. II), namely ingress classification, QoS assignment, GTP processing, L2 forwarding, and final forwarding. After ingress classification, the DOCA flow checks for QoS. The encapsulation process involves defining programmable tables for TEID-based matching and forwarding. As shown in Listing 1, a `teid_as_key` table is configured to match on GTP TEIDs and apply forwarding actions accordingly.

Listing 1. TEID-based forwarding table in encapsulation pipeline

```
table teid_as_key {
    key = {
        headers.gtpv1.teid : exact;
    }
    actions = {
        NoAction;
        drop;
        forward;
    }
    size = 128;
    default_action = forward(3);
    const entries = {
        (32w0x01) : forward(1);
        (32w0x02) : forward(2);
    }
}
```


}
}

The complete DOCA Flow configuration scripts, including TEID-based match-action pipelines and rule installation routines, are publicly available in our repository¹. This implementation demonstrates that offloading GTP-U processing into the DPU pipeline significantly reduces CPU overhead, lowers packet-processing latency, and enables wire-speed throughput while preserving programmability through a software-managed control plane. This implementation forms the foundation for the performance evaluation that we present in Sec. IV.

IV. RESULTS

This section presents the performance evaluation of the proposed 6GUPF architecture under varying traffic loads and packet sizes. Key metrics, including throughput, server CPU utilization, latency, and packet drop rate, were measured to demonstrate the superiority of the DPU-based implementation over DPDK-UPF, P4-UPF, and our 6GUPF.

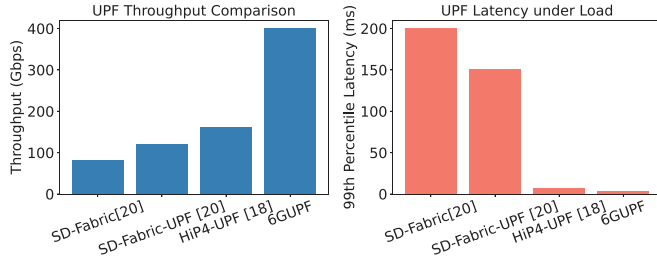


Fig. 5. Comparative analysis of throughput and latency among software-based and DPU-accelerated UPF implementations.

Figure 5 shows the comparative performance of multiple UPF designs, including SD-Fabric [21], HiP4-UPF [19], and our proposed 6GUPF. The DPU-based 6GUPF demonstrates a fourfold increase in throughput, reaching up to 400 Gbps, compared to 80–160 Gbps in software UPF variants. The 6GUPF achieves a drastic reduction in the 99th percentile latency, dropping from 200 ms in conventional platforms to just 0.2 ms.

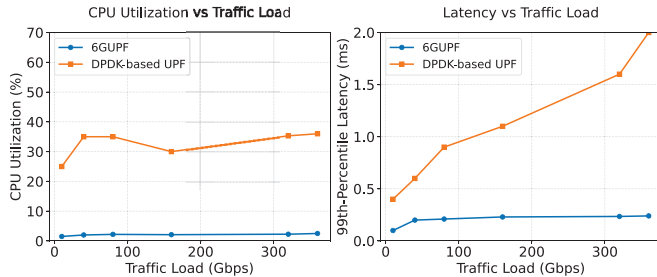


Fig. 6. CPU utilization and latency analysis of DPU-based UPF versus baseline DPDK-based UPF under different load conditions.

Figure 6 shows the performance benefits of DPU offloading in UPF processing. On the left, CPU utilization rises sharply for

the DPDK-based UPF because per-packet operations (parsing, TEID lookup, GTP-U encap/decap) saturate the cores at high PPS. Offloading these primitives to the DPU removes the CPU bottleneck and shortens the processing pipeline, which directly lowers the 99th-percentile latency shown on the right. At a 380 Gbit/s traffic load, CPU utilization is under 5% for the DPU-based design, compared to 50% for the DPDK-based UPF implementation.

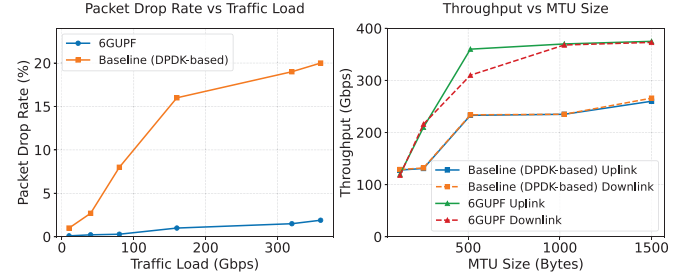


Fig. 7. Performance comparison of the proposed 6GUPF against a DPDK-based baseline.

Figure 7 shows that the DPU-based UPF has a significantly lower packet drop rate than the NIC-based implementation, even under high traffic loads. The packet drop rate was 2.29% with 380 Gbit/s load using the DPU, while the NIC-based system suffered a drop rate of 10%. At the maximum MTU size of 1500 bytes, 6GUPF achieved a throughput of 380 Gbit/s, compared to the baseline DPDK-UPF's throughput of 234 Gbit/s. For an MTU size of 256 bytes, 6GUPF throughput reached 110 Gbit/s, significantly surpassing the baseline DPDK-UPF. For smaller MTU sizes, such as 512 bytes and 256 bytes, 6GUPF demonstrated sustained performance.

The results clearly illustrate the advantages of the DPU-based UPF architecture in achieving higher throughput, reduced server CPU utilization, lower latency, and improved reliability. These results indicate not only raw performance improvements but also practical scalability for next-generation mobile networks. For instance, reducing CPU utilization by over 50% directly translates into cost savings in cloud-native deployments, as fewer servers are required to handle equivalent traffic loads. Similarly, the packet drop reduction (2.29% vs. 20%) ensures service continuity under congestion, which is crucial for applications such as URLLC in 6G. Compared to P4-switch UPFs that suffer from limited flow table memory, our DPU-based design supports large-scale flow states without sacrificing wire-speed processing. This highlights the viability of DPUs as a future-proof platform for UPF.

A. Discussion

The results confirm that CPU-based UPFs quickly saturate, explaining the high drop rates of NIC-only (DPDK-based) implementations even at modest loads. However, the DPU sustains near-zero losses by executing encapsulation and forwarding in dedicated hardware pipes.

The observed ~200–300 μ s latency of the 6GUPF under high load reflects not only GTP-U processing in hardware but also

¹6GUPF implementation: <https://github.com/engranaabubakar/6GUPF>

queuing and flow-handling overheads currently executed on the DPU's ARM cores due to SDK limitations. By comparison, the DPDK-based UPF exhibits ~1 ms delay caused by context switches and cache contention on host CPUs. While DPU latency is higher than bare-metal switching, it remains an order of magnitude lower than CPU-based solutions while sustaining 200–400 Gbps throughput.

Compared with DPDK-UPFs and P4 switches, the DPU provides a unique balance of programmability, hardware acceleration, and scalable stateful flow tracking. Future SDK releases that enable further migration of queuing and flow management into hardware are expected to reduce latency further, making DPUs an increasingly attractive platform for next-generation UPFs. These insights suggest that although DPUs cannot yet match bare-metal forwarding latency, their combination of programmability, scalability, and line-rate performance makes them strong candidates for 6G UPF deployments. A limitation of our current testbed is that it remains server-attached, and fully edge-integrated UPFs will be explored in future work. We also plan to extend the evaluation to multi-DPU clusters to investigate horizontal scalability and to integrate inline security functions, such as DPI and DDoS mitigation, into the UPF pipeline.

V. CONCLUSION

This paper presented a DPU-based programmable UPF for 6G networks, designed to overcome the performance limitations of traditional software-based implementations. By offloading computationally intensive tasks such as QoS management, GTP tunnel handling, and packet encapsulation/decapsulation into the hardware pipeline of NVIDIA BlueField-3 using the DOCA Flow API, the proposed 6GUPF achieves substantial improvements in throughput, latency, and scalability. Our evaluation demonstrates that the system can sustain a throughput of up to 400 Gbps with significantly reduced latency, while maintaining programmability and efficient flow-based QoS enforcement.

These results highlight the potential of DPUs as a practical foundation for future mobile cores, striking a balance between wire-speed performance and flexibility. Future work will extend this approach to fully edge-integrated UPFs, explore multi-DPU scalability, and integrate AI-assisted QoS enforcement and inline security functions, paving the way for autonomous and intelligent 6G core deployments.

ACKNOWLEDGMENT

This work has been funded by the European Commission Horizon Europe SNS JU DESIRE6G (GA 101096466) Project.

REFERENCES

- [1] M. K. Banafaa, Ö. Pepeoğlu, I. Shayea, A. Alhammadi, Z. A. Shamsan, M. A. Razaz, M. Alsagabi, and S. Al-Sowayan, "A comprehensive survey on 5g-and-beyond networks with uavs: Applications, emerging technologies, regulatory aspects, research trends and challenges," *IEEE access*, vol. 12, pp. 7786–7826, 2024.
- [2] S. M. Mohammed, A. Al-Barrak, and N. T. Mahmood, "Enabling technologies for ultra-low latency and high-reliability communication in 6g networks," *Ingénierie des Systèmes d'Information*, vol. 29, no. 3, 2024.
- [3] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, 2020.
- [4] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, programmable, and virtualized 5g networks: State-of-the-art and the road ahead," *Computer Networks*, vol. 182, p. 107516, 2020.
- [5] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "Nfv and sdn—key technology enablers for 5g networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.
- [6] A. K. Alnaim, "Securing 5g virtual networks: a critical analysis of sdn, nfv, and network slicing security," *International Journal of Information Security*, pp. 1–21, 2024.
- [7] D. Lake, N. Wang, R. Tafazolli, and L. Samuel, "Softwarization of 5g networks—implications to open platforms and standardizations," *IEEE access*, vol. 9, pp. 88 902–88 930, 2021.
- [8] S. Malik and S. Bera, "Security-as-a-function in 5g network: Implementation and performance evaluation," in *2024 International Conference on Signal Processing and Communications (SPCOM)*. IEEE, 2024, pp. 1–5.
- [9] L. Wernet, L.-M. Spang, F. Siegmund, and T. Meuser, "Resilient user plane traffic redirection in cellular networks," in *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2024, pp. 1–6.
- [10] U. K. Dayalan, Z. Wu, G. Gautam, F. Tian, and Z.-L. Zhang, "Towards an ebpf+ xdp based framework for open, programmable and scalable nextg rans," in *2023 IEEE Future Networks World Forum (FNWF)*. IEEE, 2023, pp. 1–6.
- [11] L. Dong, N. Hu, W. Deng, X. Xu, H. Ding, and Y. Huang, "Quality of service evolution and enhancement for holographic video communications toward 6g," *IEEE Wireless Communications*, vol. 32, no. 2, pp. 140–146, 2025.
- [12] K. T. Selvi and R. Thamilselvan, "An intelligent traffic prediction framework for 5g network using sdn and fusion learning," *Peer-to-Peer Networking and Applications*, vol. 15, no. 1, pp. 751–767, 2022.
- [13] A. Bhattacharyya, S. Ramanathan, A. Fumagalli, and K. Kondepudi, "An end-to-end dpdk-integrated open-source 5g standalone radio access network: A proof of concept," *Computer Networks*, vol. 250, p. 110533, 2024.
- [14] P. Salva-Garcia, R. Ricart-Sanchez, E. Chirivella-Perez, Q. Wang, and J. M. Alcaraz-Calero, "Xdp-based smartnic hardware performance acceleration for next-generation networks," *Journal of Network and Systems Management*, vol. 30, no. 4, p. 75, 2022.
- [15] Y. Moon, Y. Han, S. Kim, M. S. Sai, A. Deshmukh, and D. Kim, "Data plane acceleration using heterogeneous programmable network devices towards 6g," in *ICC 2024-IEEE International Conference on Communications*. IEEE, 2024, pp. 421–426.
- [16] R. A. Bakar, F. Alhamed, P. Castoldi, A. Sgambelluri, J. J. V. Olmos, F. Cugini, and F. Paolucci, "5gddad: A deep learning approach for ddos attack detection in 5g p4-based upf," in *2024 IEEE 25th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2024, pp. 185–190.
- [17] S. K. Singh, C. E. Rothenberg, J. Langlet, A. Kassler, P. Vörös, S. Laki, and G. Pongrácz, "Hybrid p4 programmable pipelines for 5g gnodeb and user plane functions," *IEEE Transactions on Mobile Computing*, vol. 22, no. 12, pp. 6921–6937, 2022.
- [18] T. Schneider, P. Xu, and T. Hoefler, "Fpspin: An fpga-based open-hardware research platform for processing in the network," *arXiv preprint arXiv:2405.16378*, 2024.
- [19] Z. Wen and G. Yan, "{HiP4-UPF}: Towards {High-Performance} comprehensive 5g user plane function on p4 programmable switches," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 303–320.
- [20] F. Paolucci, D. Scano, F. Cugini, A. Sgambelluri, L. Valcarenghi, C. Cavazzoni, G. Ferraris, and P. Castoldi, "User plane function offloading in p4 switches for enhanced 5g mobile edge computing," in *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2021, pp. 1–3.
- [21] Open Networking Foundation (ONF), "SD-Fabric: Open source full-stack programmable leaf-spine network fabric," Open Networking Foundation (ONF), Tech. Rep., Jun. 2021, white paper, June 2021. [Online]. Available: <https://opennetworking.org/sd-fabric/>