Heterogeneous UPF Integration Framework and 5G User Plane Acceleration

Tse-Han Wang*†, Min-Chih Hu*, Li-Hsing Yen*, and Chien-Chao Tseng*
*Department of Computer Science, College of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.
Email: {wangth, mzhu1129, lhyen, cctseng}@cs.nycu.edu.tw
†Network Management Laboratory, Chunghwa Telecom Laboratories, Taoyuan, Taiwan.

***Abstract*—The 3rd Generation Partnership Project (3GPP) proposes User Plane Function (UPF) in the fifth generation (5G) mobile networks to handle user data between radio access network (RAN) and data network (DN). Since pure software-based UPF does not fulfill the diversified service requirements of 5G networks, researchers have leveraged various hardware acceleration techniques to enhance the performance of UPF. However, different hardware implementations require different installations and adaptations. This paper proposes a framework to integrate various UPF implementations. It is compliant with standard UPF but decouples the control plane function (UPF-CP) from the user plane function (UPF-UP). We implemented UPF-CP by Packet Forwarding Control Protocol (PFCP) Agent with two implementations of UPF-UP: one with Intel Data Plane Development Kit (DPDK) and the other with Smart Network Interface Card (SmartNIC). We integrated the framework with an open-source 5G core network, free5GC, and evaluated the framework by experiments. The results confirm the interoperability of our work with free5GC and demonstrate the superiority of the hardware-accelerated UPF over pure software approach in terms of packet processing speed on the user plane.**

I. Introduction

The core network in the fifth generation (5G) mobile networks provides service connection, device authentication and accounting for mobile users. To meet diverse application service requirements and scenarios, the 3rd Generation Partnership Project (3GPP) has restructured the 5G core network to adopt Service Based Architecture (SBA) [1] and Control and User Plane Separation (CUPS) [2]. SBA decomposes applications and services in 5G core network into fine-grained network functions, leveraging Network Function Virtualization (NFV) to enable flexible network deployment [3]. CUPS decouples the control plane function from the user plane function. Consequently, the processing and forwarding of user packets are designated to User Plane Function (UPF) while the management and control functionalities are designated to Session Management Function (SMF). These two functions together handle user data between radio access network (RAN) and data network (DN).

We may apply the same notion of Software-Defined Networking (SDN) [4] to further divide the functionalities of UPF into control plane and user plane [5]. On the control plane, UPF needs to communicate with SMF via reference interface N4 using Packet Forwarding Control Protocol (PFCP) [6]. This task can be done by an application running on an SDN controller. On the user plane, UPF needs to perform packet routing, forwarding, and inspection. This task can be done by a high-speed programmable hardware such as P4 switch [7], Smart Network Interface Card (SmartNIC), or Intel Data Plane Development Kit (DPDK) [8]. However, most UPF implementations tie up the control plane and user plane functions of UPF, and are hardware-specific because the user plane is critical for the overall performance. Consequently, these implementations cannot be easily adapted to other environments even if the control plane part of UPF is in fact hardware-independent.

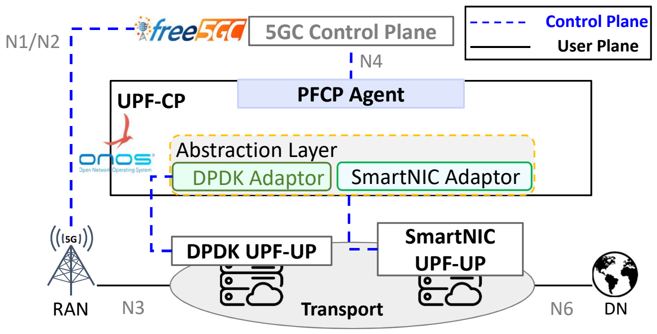


Fig. 1: Heterogeneous UPF Integration System Architecture

This study proposes an implementation framework for UPF that exercise the control/user plane separation. On one hand, the separation needs only a unified, hardware-independent UPF control plane function (UPF-CP), for which we run PFCP Agent to communicate with SMF using PFCP. On the other hand, the separation also enables the hosting of different UPF user plane (UPF-UP) implementations. The main challenge of this framework is to make the PFCP Agent 5G-compatible and hardware-independent while allowing easy plug-in of diverse hardware-boosted UPF-UP. Our key idea is to introduce an Abstraction Layer which provides a unified management interface for the PFCP Agent to configure and manage abstract UPF rules, which are then converted by an implementation-specific adaptor to physical UPF rules that are executable by the physical hardware. We showcase real implementations with performance evaluation in this paper. UPF-CP in our implementation is an application running on ONOS [9] SDN controller. Two physical UPF-UPs are implemented: one is based on SmartNIC and the other is based on DPDK. A unique feature of the latter implementation is the support of downlink packet buffering for idle user equipment (UE). Fig. 1 shows the architecture of the proposed framework.

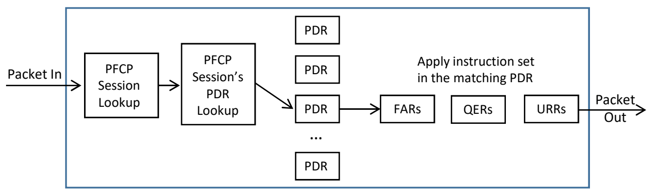

Packet In
PFCP Session Lookup
PFCP Session's PDR Lookup
PDR
PDR
PDR
...
PDR
Apply instruction set in the matching PDR
FARs
QERs
URRs
Packet Out

Fig. 2: UPF Packet Forwarding Model

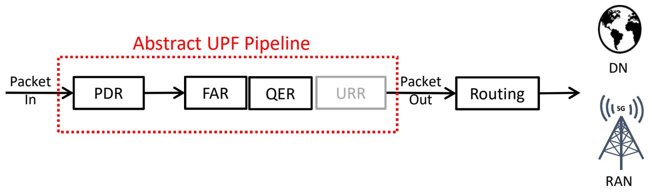


Fig. 3: Abstract UPF pipeline

The remainder of this paper is structured as follows. Sec. II briefs the backgrounds and related work. Sec. III presents the proposed framework and associated implementations. Sec. IV presents the experimental results and the last section concludes the paper.

II. Background and Related Work

A. Control Plane of User Plane Function (UPF)

The main tasks in the control plane of UPF are to communicate with SMF using PFCP and to set up rules for packet processing and forwarding. SMF should first establish an association with UPF and exchange the supported capabilities and Node ID (IP address) of each other. After that, SMF may send PFCP messages to create, modify, and release a PFCP session on UPF, and instruct UPF to report the user plane information to SMF. The creation and modification of a PFCP session are associated with relevant packet processing and forwarding rules. These rules are used in the packet forwarding model on UPF [6] (Fig. 2), including Packet detection rule (PDR), forwarding action rule (FAR), QoS enhancement rule (QER), and usage reporting rule (URR).

The packet forwarding model is used by the user plane of UPF to handle incoming packets. When UPF-UP receives a user packet, it first matches this packet with an existing PFCP session. If there is a match, the UPF-UP then finds the matching PDR of the PFCP session with the highest precedence. The UPF-UP then applies FARs, QERs, and URRs on the packet. These rules can be pipelines if SDN switches are used for UPF-UP.

B. Smart Network Interface Card (SmartNIC)

SmartNIC is a technology that leverages network interface card (NIC) equipped with power CPU and memory to offload packet processing tasks from server CPU. Doing so not only accelerates packet processing tasks as packets can be processed entirely by the NIC, but also liberates precious server CPU power from performing repetitive packet handling routines, saving the CPU power for other computation-intensive processes. The packet processing logic of SmartNIC is programmable so that we could handle different packet formats and meet diverse packet processing needs.

C. Data Plane Development Kit (DPDK)

The Data Plane Development Kit (DPDK) [8] is an open-source software project that provides a development platform and interface for packet processing enhancement. It works by two key techniques: kernel bypass and poll-mode driver. Kernel bypass offloads packet processing from the operating system kernel to processes running in user space, which eliminates context-switching time. Poll-mode driver allows the driver to actively poll NIC for possible incoming packet rather than passively wait for the interrupt triggered by packets coming into the NIC. DPDK comes with a set of data plane libraries and network interface controller polling-mode drivers.

D. Related Work

The concept of CUPS was in 3GPP R14 for the function decoupling of Serving Gateway (SGW), PDN Gateway (PGW), and Traffic Detection Function (TDF) in the Evolved Packet Core (EPC). The same concept has been applied to the decoupled SMF and UPF in 5G and this work also follows the same concept to further decompose UPF.

Our work here is closely related to ONF OMEC UPF [5]. Both opt to an abstract user plane. The difference lies in the implementations. The OMEC UPF demonstrated two implementations: one based on a software switch and the other based on P4 switch. By contrast, our implementations are based on SmartNIC and DPDK. Moreover, OMEC UPF configures the data plane only once at start-up, which makes hot swapping impossible. By contrast, our framework allows an on-line switch to another UPF-UP implementation without rebooting the whole UPF.

III. Our Design and Implementation

A. UPF Control Plane: PFCP Agent

The control plane of UPF centers on PFCP Agent, which needs to interpret PFCP messages as defined in 3GPP TS 29.244 [6]. We implemented PFCP Agent as an application on ONOS controller [9]. When the PFCP Agent receives an Association Setup request message from SMF, it responds with the Node ID and supported capabilities of this UPF. When the PFCP Agent receives a message pertaining to a PFCP session, the message may describe the packet processing rules required for the session. A straightforward design for the PFCP Agent is to convert the rules associated with the message into corresponding physical rules to be enforced and executed by the underlying UPF-UP entity. However, to make the PFCP Agent independent of physical UPF-UP implementations, the PFCP Agent in the proposed framework converts session-related rules to corresponding abstract rules.

The abstract rules are embodied by an abstract UPF pipeline, which includes three essential tables in the UPF packet forwarding model (Fig. 3). The abstract rules for a PFCP session, once created, could be modified or deleted subsequently. Moreover, more abstract rules for the same session could be added later. Therefore, we propose an abstract rule management interface in the Abstraction Layer for the PFCP Agent to manage the abstract rules.

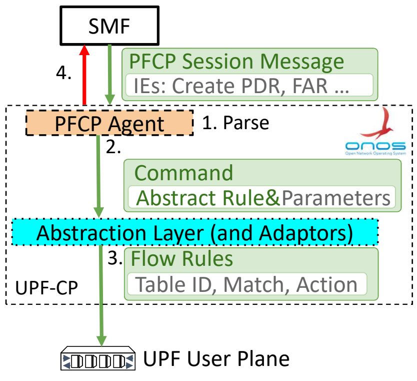


Fig. 4: PFCP session-related message processing flow

TABLE I: Supported UPF User Plane Pipelines

Function	SmartNIC UPF	DPDK UPF
PDR	N3, N6	N3, N6
FAR	Forwarding	Forwarding Buffering Notify SMF
QER	No	No

Since the abstract rules cannot be directly executed by the underlying UPF-UP entity, we propose an implementation-dependent Adaptor to convert the abstract rules into executable UPF rules. Fig. 5 shows the conversion of abstract rules.

We currently support two implementations of the UPF-UP: SmartNIC UPF and DPDK UPF. Table I shows some features of these two implementations. The following two sections will detail these implementations.

B. UPF User Plane with SmartNIC

The SmartNIC UPF consists of hardware and software parts. For the hardware part, we used the SmartNIC from Netronome [10], which supports expressing packet processing logic in C and P4 languages. We used P4 language to describe the packet parsing and processing logic and implemented a P4 pipeline that can run on the SmartNIC. The P4 pipeline supports the PDR and FAR functions of UPF, through which we could provide basic routing functionality for user packets in the transport network.

For the software part, we used the software development kit provided by the vendor to manage the P4 pipeline and implement SmartNIC Adaptor. The Adaptor communicates with the UPF-CP through the Thrift API and sends packet forwarding rules to the SmartNIC hardware.

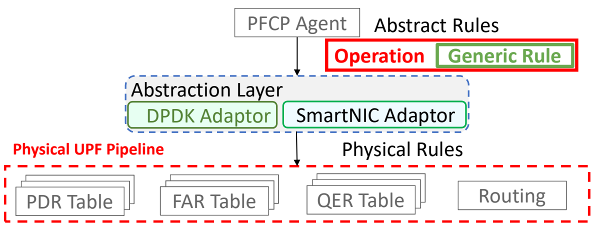


Fig. 5: Abstract UPF rules transformation

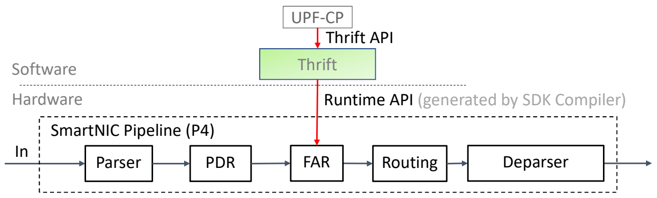
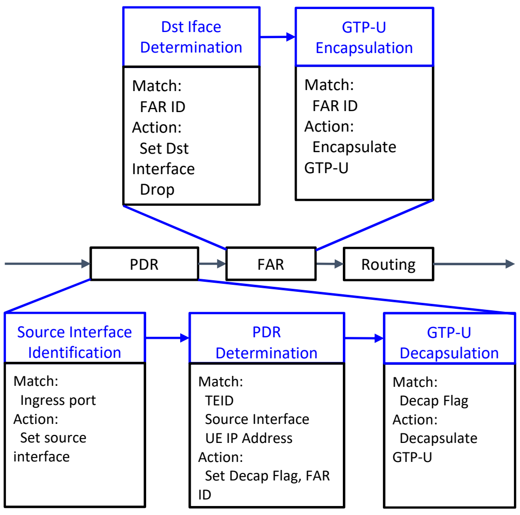


Fig. 6: SmartNIC UPF architecture



Dst Iface Determination
Match: FAR ID Action: Set Dst Interface Drop
GTP-U Encapsulation
Match: FAR ID Action: Encapsulate GTP-U
PDR
FAR
Routing
Source Interface Identification
Match: Ingress port Action: Set source interface
PDR Determination
Match: TEID Source Interface UE IP Address Action: Set Decap Flag, FAR ID
GTP-U Decapsulation
Match: Decap Flag Action: Decapsulate GTP-U

Fig. 7: Pipeline tables for PDR and FAR

Fig. 6 shows the SmartNIC pipeline. It consists of five main parts: Parser, PDR, FAR, Routing, and Deparser. The Parser handles user packets by analyzing the headers of different protocol stacks and storing them as metadata for subsequent pipeline tables. The PDR contains three tables. The first two tables classify 5G user packets and the third table performs GTP-U packet decapsulation based on instructions from the UPF-CP. The FAR contains two tables. The first table determines which FAR action to apply for a given packet. Currently supported FAR actions include forwarding the packet to a specific interface, discarding the packet, and whether or not to encapsulate the packet with a GTP-U header. The second table is for packets that require the installation of a GTP-U header. Based on instructions from the UPF-CP, it encapsulates the used Tunnel Endpoint ID (TEID) and destination IP address of the GTP-U header into the packet. Detailed PDR and FAR tables can be found in Fig. 7.

The Routing is performed by one table, which provides basic routing functionality for transmitting packets in the transport network. It is worth noting that PFCP messages only indicate the destination interface for a packet. Based on the indicated destination interface, the routing table replaces the destination MAC address with the MAC address of the corresponding gateway. This replacement makes packets to be transmitted to the DN or RAN as intended.

The last part of the SmartNIC pipeline, Deparser, reassembles the packet processed by the packet parser back into a packet according to the protocol stack. Once the reassembly is completed, the packet will be sent to the NIC to complete

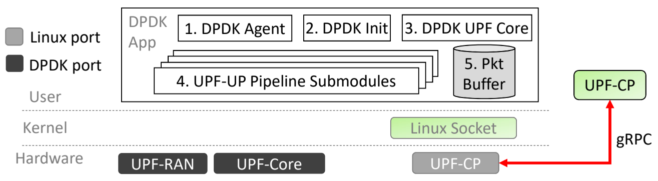


Fig. 8: DPDK UPF architecture

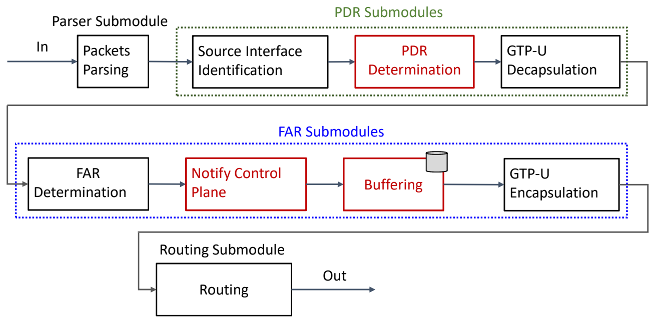


Fig. 9: DPDK UPF full pipeline

the processing procedure.

All actions in these tables are to match packet contents and to configure metadata. The only two exceptions are to encapsulate and decapsulate GPU-T packets.

C. UPF User Plane with DPDK

Our implementation of DPDK UPF uses two NICs managed by DPDK [8] for communication with the DN and the RAN, respectively. Another NIC managed by the Linux kernel is used for the communication with the UPF-CP. The DPDK UPF architecture is shown in Fig. 8.

We implemented DPDK UPF using C++ language and DPDK-provided APIs. DPDK UPF consists of five major components.

- DPDK Agent, which is responsible for communication with UPF-CP and managing packet processing modules.
- DPDK Init, which is responsible for initializing environment resources using the APIs provided by DPDK, such as NIC information and memory allocation.
- UPF DP Core, which is responsible for obtaining packets from physical NICs using the poll-mode driver and processing them. This module works with the packet processing submodules to form a packet processing pipeline.
- UPF-UP Pipeline Submodules, which implements different packet processing logic functions and provides APIs to be invoked by DPDK Agent for managing rule entries in this submodule.
- Packets Buffer, which is the storage space for temporary downstream traffic.

Fig. 9 shows the complete DPDK UPF pipeline, which contains Parser, PDR, FAR, and routing submodules. The logic of packet parser and routing is similar to those used in SmartNIC UPF. PDR-related tables use the incoming NIC as the source interface for the classification of user plane packets in the PDR rule determination stage. The decap flag, FAR ID, Session Endpoint Identifier (SEID), and PDR ID

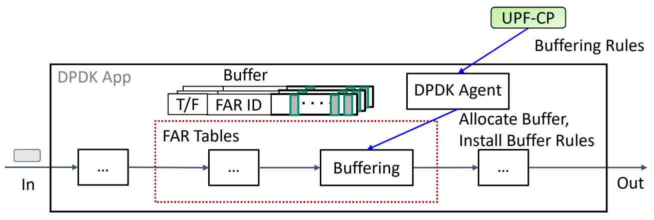


Fig. 10: Downlink buffer preparation

metadata are set in this table for use by subsequent tables (the processing of buffered downlink packets may require this information). The third table decides whether to decapsulate the GTP-U header on the packet based on the decap flag. FAR-related tables use the FAR ID to decide whether to use forwarding, buffering, or other functions.

D. Buffering Downlink Data for Idle User Equipment

DPDK UPF additionally supports downlink data buffering for UE in idle mode. When a UE has not sent or received packets for an extended time, the UE may enter idle mode. The system should buffer downlink packets for an idle UE, and release the buffered packets to the UE when the UE becomes active. Note that this functionality is not supported by the SmartNIC UPF.

Three tasks are needed to buffer downlink traffic for an idle UE. First, a buffer should be prepared somewhere in the pipeline for each idle UE. Second, the pipeline should inform the control plane of the presence of downlink packets destined for an idle UE besides buffering these packets. Finally, the data plane should release all packets buffered for an idle UE when later the UE becomes active.

It is the Buffering table (an FAR submodule) that prepares a buffer space for an idle UE. When the AMF in the core network detects an idle UE, it notifies the SMF to change the packet processing rules of the PFCP session pertaining to the idle UE. The SMF then instructs UPF to prepare downlink data buffering for the idle UE (Fig. 10). The UPF CP issues an Allocate Buffer Storage command to the Buffering table to allocate a memory block to keep future downlink traffic for this UE. All the buffer spaces kept by the Buffering table are indexed by FAR ID. When a future packet with a matched FAR ID comes to the Buffering table, it will be stored in the corresponding buffer space, waiting to be released later. Each buffer space has a flag indicating whether the Buffering table is releasing it.

Besides buffering the packet, the UPF should also notify the AMF when a downlink packet for an idle UE comes to the data plane. This is to avoid possible buffer overflow and to wake up the UE. The Notify Control Plane table (also an FAR submodule right before the Buffering table) is responsible for the notification. When detecting such a packet, the action in this table sends a PFCP session report to the UPF-CP through a UDP socket. This message carries the SEID and PDR ID of the packet. When the UPF-CP receives the PFCP session report from the DPDK UPF, it forwards this message to the SMF.

To avoid buffer overflow, all packets buffered for an idle UE should be released after the UE has been woken up.

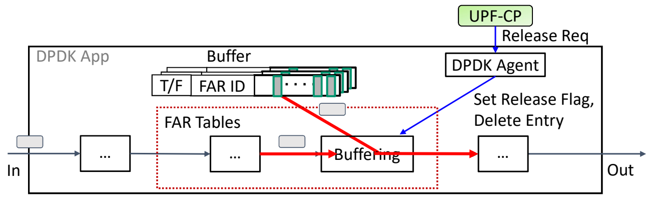


Fig. 11: Releasing donwlink packets

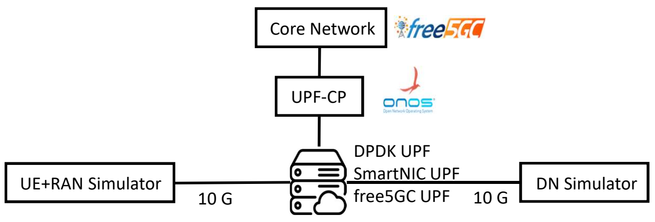


Fig. 12: Topology of the experiments

This is initiated by the SMF with a request to UPF-CP to update the FAR action of a PFCP session from buffering to forwarding. The UPF-CP then sends a packet release request to DPDK UPF. The DPDK UPF performs two actions for the request: to set the release flag of the corresponding buffer space and to delete the buffering rule associated with this FAR in the Buffering table so that subsequent packets will not enter the buffer space.

After that, the Buffering table must handle two different data streams from two sources (Fig. 11). One is packets waiting to be released in the buffer space. The other is packets sent over by the previous table. To maintain packet sequencing, the Buffering table will prioritize releasing the packets in the buffer space. Only when no packets are left in the buffer space can the Buffering table process the packets sent over by the previous table. During this period, packets sent over by the previous table will be temporarily stored in memory allocated by DPDK, waiting to be processed later. In this sense, the packet buffering space and DPDK buffering space form a two-level priority queue for the Buffering table to process.

IV. Experimental Results

We set up the experimental environment as shown in Fig. 12. The core network (free5GC [11]) was placed in Edgecore SAU5081-2X with Intel Xeon E5-2620 v4 CPU and Intel X520-DA2 10 Gbps. UPF-CP was an application of ONOS controller running on a Ubuntu PC with Intel Core i7-6700 CPU. Three Ubuntu PCs each with Intel Core i7-10700 CPU (2.90 GHz) and Intel X710-DA4 10Gbps SFP+ were used for DPDK/free5GC UPF, UE/RAN Simulator (UERANSIM [12]), and DN Simulator, respectively. SmartNIC UPF was implemented in a Ubuntu PC with Intel Core i7-10700 CPU and Netronome Agilio CX Dual-Port 10 GbE SmartNIC.

A. *PFCP Message Processing*

We first tested the interoperability between heterogeneous UPF and the 5G core network. This was done by making PDU session registrations from the free5GC core network.

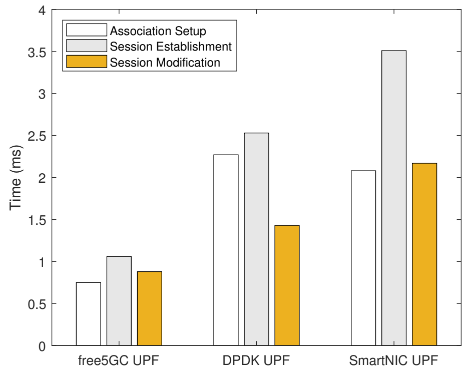


Fig. 13: PFCP Message Processing Time

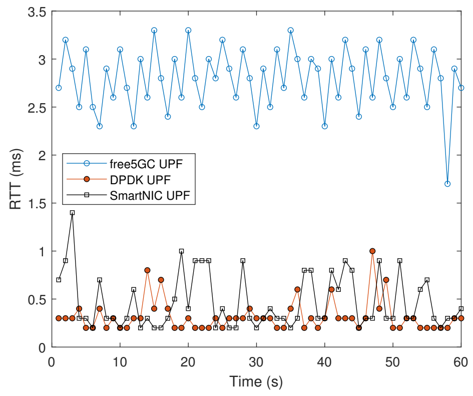


Fig. 14: TCP RTT Comparison

We took the registration testing script provided by free5GC to measure the processing time of control messages for free5GC, UPF-CP, and individual UPF-UP. The results are shown in Fig. 13.

First of all, we have verified the interoperability of the proposed framework with the free5GC core network. However, the message processing time of DPDK or SmartNIC UPF was longer than that of the original free5GC UPF. The reason is that we implemented UPF-CP in Java and additionally implemented abstract rules generations and conversions. By contrast, the free5GC UPF was implemented in C language and need not handle abstract rules. Since 5G control messages are processed only when the UE connection status changes, we do not expect too much volume of control messages. Therefore, the processing delay is acceptable.

B. *UPF User Plane Performance*

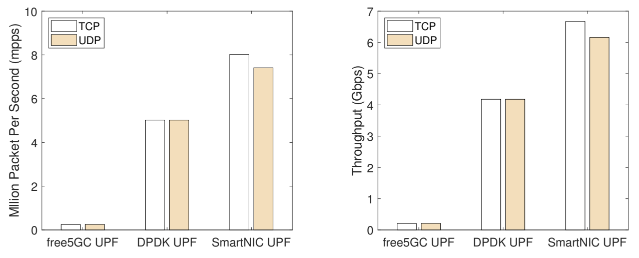
1) TCP RTT Comparison: We used flowgrind [13] to measure the Round-Trip Time (RTT) of TCP connections between UE and DN. We took the results as the packet processing delays of each UPF-DP implementation. We tested the SmartNIC UPF, DPDK UPF, and free5GC UPF. Fig. 14 shows the experimental results. It is observed that SmartNIC UPF and DPDK UPF exhibited lower latency compared to the VNF-based free5GC UPF.

2) TCP/UDP Throughput Comparison: We used TRex [14] to generate a large number of TCP/UDP short packets (payload size set to 64 bytes) to evaluate the throughput

TABLE II: First Packet RTT and TCP Retransmissions Comparison

Function	free5GC UPF	DPDK UPF
First Packets RTT	18.464 ms	21.697 ms
TCP RETRs (100M bits/sec)	0	0
TCP RETRs (1G bits/sec)	140–160	0
TCP RETRs (2G bits/sec)	1k	0
TCP RETRs (3G bits/sec)	2k	90–120

of the SmartNIC UPF and DPDK UPF. The goal is to study the packet processing capacity of each implementation. The free5GC UPF was also tested for comparison. Fig. 15 displays the results in terms of Million Packets Per Second (MPPS) and throughput (Gbps). The results indicate that the hardware-accelerated UPFs (SmartNIC UPF and DPDK UPF) significantly outperformed the VNF-based UPF.



(a) (b)

Fig. 15: TCP/UDP Throughput Comparison by (a) MPPS (b) Gbps

C. Impact of Downlink Data Buffering

We evaluate implemented downlink packet buffering feature in DPDK UPF. We used UERANSIM to emulate an idle UE and used free5GC core network to wake up an idle UE (when the core network receives a PFCP session report sent by DPDK UPF upon receiving a downlink packet). We also tested the downlink packet buffering feature built in free5GC UPF for comparison. The testing steps are as follows.

- Register an idle PDU session with free5GC using UERANSIM. This action shall cause the free5GC SMF to send PFCP messages to inform DPDK UPF or free5GC UPF to prepare for buffering downlink packets and set up the corresponding buffering rules within the UPF.
- Use iperf2 [15] to generate TCP traffic from the DN to the idle UE, which shall be handled by DPDK UPF or free5GC UPF.
- Measure the RTT of the first packet and the number of TCP retransmissions for free5GC UPF and DPDK UPF.

The test results are presented in Table II. The first-packet RTT was 3 ms higher in DPDK UPF than in free5GC UPF. This was due to the handling of considerable PFCP messages for awakening the idle UE. However, DPDK had a lower number of TCP retransmissions compared with free5GC UPF, especially when the traffic rate was high. The ability to alleviate TCP retransmissions also demonstrates the superiority of DPDK UPF.

We also measured the packet processing performance of DPDK UPF with and without processing buffered packets. Table III shows the difference in terms of MPPS. The performance impact caused by the handling of buffered packets was somehow significant.

TABLE III: Million Packets Per Second of DPDK UPF

Protocol	With Buffering	Without Buffering
TCP	3.41	5.02
UDP	3.42	5.02

V. Conclusions

This study proposes a heterogeneous UPF integration framework that allows for rapid development of new 5G UPF user plane implementation and is managed by a unified UPF control plane. The messages between UPF and other 5G network functions conform to the 3GPP standards. Additionally, the SDN data plane offloading technology is applied to implement the functions of the heterogeneous 5G UPF user planes. Compared to traditional VNF-based UPF, it significantly reduces packet processing latency and improves network throughput.

Acknowledgement

This work was supported in part by the National Science and Technology Council, Taiwan, under Grants MOST 110-2221-E-A49-044-MY3, MOST 110-2221-E-A49-064-MY3, NSTC 112-2218-E-011-004 and NSTC 112-2218-E-011-006.

References

[1] "System architecture for the 5G system (5GS)," 3GPP, TS 23.501, V16.4.0, Release 16, Mar. 2020.
[2] P. Schmitt, B. Landais, and F. Y. Yang, "Control and user plane separation of epc nodes (CUPS)," https://www.3gpp.org/news-events/3gpp-news/cups, accessed: 2023-05-10.
[3] P. Rost, A. Banchs, I. Berberana, M. Breitbach, M. Doll, H. Droste, C. Mannweiler, M. A. Puente, K. Samdanis, and B. Sayadi, "Mobile network architecture evolution toward 5G," *IEEE Commun. Mag.*, pp. 84–91, May 2016.
[4] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[5] "Upf-epc: A 4g/5g user plane function (upf) compliant with 3gpp ts 23.501," https://github.com/omec-project/upf, accessed: 2023-05-10.
[6] "Interface between the control plane and the user plane nodes," 3GPP, TS 29.244, V16.6.0, Release 16, Jan. 2021.
[7] 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," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, Jul. 2014.
[8] "Data plane development kit," https://www.dpdk.org/, accessed: 2023-05-10.
[9] "ONOS: Open Network Operating System," https://opennetworking.org/onos, accessed: 2023-05-10.
[10] "SmartNIC Overview: Netronome," https://www.netronome.com/products/smartnic/overview, accessed: 2023-05-10.
[11] "free5gc," https://www.free5gc.org, accessed: 2023-05-10.
[12] "GitHub: aligungr/UERANSIM: Open source 5G UE and RAN (gNodeB) implementation," https://github.com/aligungr/UERANSIM, accessed: 2023-05-10.
[13] "Flowgrind by flowgrind," https://flowgrind.github.io, accessed: 2023-05-10.
[14] "Trex realistic traffic generator," https://trex-tgn.cisco.com, accessed: 2023-05-10.
[15] "Iperf 2: A means to measure network responsiveness and throughput," https://sourceforge.net/projects/iperf2/, accessed: 2023-05-10.