Auto-Scaling and Load-Aware SDNFV Architecture for 5G User Plane Functions

Tse-Han Wang*†, Jian-Yu Li*, 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, liyf0404, lhyen, cctseng}@cs.nycu.edu.tw
†Network Management Laboratory, Chunghwa Telecom Laboratories, Taoyuan, Taiwan.

***Abstract*—The fifth-generation (5G) core network requires high scalability and performance to support rapidly increasing traffic demands and latency-sensitive applications. However, existing implementations often lack mechanisms for seamless and real-time scaling of User Plane Functions (UPFs), which leads to inefficient resource utilization and potential service disruption. This paper presents a scalable and session-aware UPF deployment framework that leverages Software-Defined Networking (SDN) and Network Function Virtualization (NFV). The proposed architecture integrates Kubernetes-based orchestration, Prometheus-driven monitoring, and P4-enabled traffic steering to dynamically adjust UPF instances while preserving session continuity. A Packet Forwarding Control Protocol (PFCP) proxy and a migration-aware load balancer are designed to ensure consistent state synchronization and resource-aware traffic distribution. Experimental evaluation demonstrates near-linear throughput scalability, seamless session continuity during scale-in, and effective overload mitigation. These results confirm the feasibility of dynamic UPF scaling in cloud-native 5G core networks.**

***Index Terms*—5G Core, UPF, SDN, NFV, auto-scaling, load balancing**

I. Introduction

The continuous evolution of telecommunication networks has accelerated the development of fifth-generation (5G) mobile communication systems. Beyond enhancing transmission efficiency and improving user experience, 5G must also support diverse requirements such as ultra-reliable low-latency communications (URLLC) for autonomous driving and remote healthcare, as well as massive machine-type communications (mMTC) for industrial automation.

To address these heterogeneous demands, the 3rd Generation Partnership Project (3GPP) introduced Control and User Plane Separation (CUPS) and the Service-Based Architecture (SBA). These paradigms modularize core network functions, reducing inter-component coupling and improving deployment flexibility.

Although the 5G Core (5GC) network leverages Network Function Virtualization (NFV) to achieve elasticity and scalability, existing implementations remain constrained by rigid software architectures. In particular, User Plane Functions (UPFs) are often statically pre-configured, preventing their dynamic instantiation or removal without restarting core components. This limitation restricts scalability and leads to inefficient utilization of resources.

To address these challenges, this paper proposes a virtualized 5GC deployment framework that supports dynamic UPF scaling and intelligent load balancing. The key contributions are summarized as follows:

- **Elastic scaling:** Automatic adjustment of the number of UPF instances based on real-time resource utilization.
- **Seamless migration:** Preservation of session continuity during scale-in and scale-out events.
- **High-performance load balancing:** Session-aware traffic steering in the data plane to ensure efficiency under varying traffic loads.
- **Practical applicability:** Compatibility with open-source 5GC implementations (e.g., free5GC) without source code modifications. The proposed framework operates without modifying existing 5GC control-plane functions.

II. Background and Related Work

A. UPF and Packet Forwarding Control Protocol

The UPF is the primary data plane element of the 5GC and is responsible for handling traffic across the N3, N6, and N9 interfaces [1]. It manages tunneling, Quality of Service (QoS) enforcement, and packet forwarding. The Packet Forwarding Control Protocol (PFCP) is the standardized signaling protocol between the Session Management Function (SMF) and the UPF over the N4 interface, allowing the SMF to create, modify, and delete session contexts [2]. Consistent PFCP state synchronization across distributed UPFs is essential for maintaining session continuity during scaling, motivating the use of PFCP proxies and state management mechanisms.

Definition of Session: In this paper, the term *session* refers to a Protocol Data Unit (PDU) session, which represents an end-to-end logical connection between User Equipment (UE) and a Data Network (DN). It is managed by the SMF and enforced at the UPF via PFCP rules. Each session is uniquely identified by the Session Endpoint Identifier (SEID) on the control plane between the SMF and UPF, and by the Tunnel Endpoint Identifier (TEID) on the user plane to distinguish GTP-U tunnels, typically per QoS flow, across the N3, N6, and N9 interfaces. Throughout this paper, the term *session* is used interchangeably with PDU session.

B. Scalability and Load Balancing in Virtualized Mobile Cores

NFV-Based Scaling in LTE EPC: Early work explored auto-scaling of data-plane Virtual Network Functions (VNFs). In [3], the authors implemented a non-standard slicing mechanism in NFV-based LTE EPC, introducing per slice throughput limits for the Serving Gateway (SGW) and Packet Data Network Gateway (PGW). Scaling decisions were based on current load, but scale-in only selected idle instances without session reallocation, leading to potential overload or resource underutilization. An adaptive framework for LTE PGW using Software-Defined Networking (SDN) and NFV [4] proposed scaling based on flow table utilization and traffic load, with load balancing per UE IP address via OpenFlow switches. However, it relied on non-standard extensions to parse GTP-U headers, limiting portability.

Container-Based Mobile Core Networks: Cloud-native designs were later introduced to improve elasticity. The work in [5] containerized the softEPC and employed Kubernetes/Mesos to scale SGW instances based on CPU utilization. Stateless service pools and relay servers were adopted for state synchronization, although load balancing remained static (round-robin or random) without overload awareness. A complementary approach [6] introduced runtime session offloading between PGWs to prevent overload and enable seamless scale-in migration, but lacked integration with monitoring and scaling triggers. Recently, Kubernetes Horizontal Pod Autoscaler (HPA) [7] has been extended with Prometheus custom metrics to enable traffic-aware scaling.

Programmable Data Plane Acceleration: Advances in P4 and programmable hardware have improved UPF throughput and flexibility. The UP4 project [8] [9] by the Open Networking Foundation (ONF) demonstrated a P4-based UPF integrated with the SD-Fabric platform. HiP4-UPF [10] optimized P4 pipelines to achieve near-line rate performance for 3GPP-defined UPFs. Hybrid P4 pipelines for gNodeB and UPF deployments have also been proposed [11]. Load-aware balancing schemes, such as Charon [12] showcased line-rate distribution using programmable switches. Beyond P4, kernel-level acceleration frameworks such as Cable [13] and eBPF-based SBA enhancements [14] have improved both user-plane and control-plane efficiency.

Summary of Gaps: Despite these advances, three key gaps remain: (i) lack of session-aware allocation and migration strategies, (ii) limited real-time PFCP state synchronization across distributed UPFs, and (iii) insufficient cloud-native designs that combine autoscaling with programmable data plane acceleration. These gaps motivate the architecture proposed in this work.

III. Scalable 5G User Plane Function with SDNFV-Enabled Load Balancing

A. System Overview

This work presents a scalable, load-aware 5G UPF deployment framework based on SDN and NFV. The framework operates without requiring modifications to existing 5GC software. At the center of the design is the concept of a *UPF Service*, which encapsulates multiple UPF instances, an automatic scaling component (Scaler), and a Load Balancer. A virtual IP address maintains communication with other 5GC components, enabling transparent integration. The PFCP Proxy manages control-plane signaling, and both the Load Balancer and PFCP Proxy can be shared across multiple UPF Services.

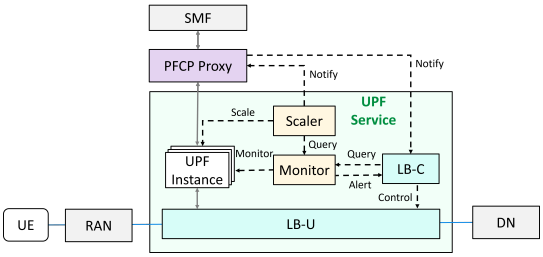

Fig. 1. System Architecture Overview

The framework introduces two complementary mechanisms: the *Scaler*, which governs service-level elasticity by adjusting the number of UPF instances, and the *Monitor*, which supervises individual instances to detect and mitigate overload. Together, these mechanisms achieve dynamic scalability while preserving session continuity.

B. System Architecture

The proposed architecture consists of several interdependent components that collectively enable automated scaling and load-aware balancing of UPFs in a virtualized 5GC environment. Fig. 1 illustrates the overall system architecture.

1) Control and Scaling Components:

- **Scaler**: Adjusts the number of UPF instances according to aggregate service load. It coordinates with the PFCP Proxy to synchronize control-plane state during scale-out and scale-in operations, after which the proxy notifies the Load Balancer Control Plane (LB-C) to update forwarding behavior.
- **Monitor**: Continuously tracks per-instance utilization and generates overload alerts when predefined thresholds are exceeded. These alerts trigger session migration toward less-loaded UPF instances.

2) Control-Plane Mediation:

- **PFCP Proxy**: Deployed on the N4 interface, the PFCP Proxy mediates PFCP signaling between the SMF and the UPF Service. Its primary role is to maintain synchronized association and session states across UPF instances. In our implementation, it is containerized and co-located with the SMF within the same pod to reduce signaling latency.

3) Load Balancing Subsystem:

- **Load Balancer Control Plane (LB-C)**: Processes session events, scaling notifications after PFCP synchronization, and overload alerts, and dynamically updates flow rules in the Load Balancer User Plane (LB-U).

- **Load Balancer User Plane (LB-U)**: Enforces LB-C decisions through session-aware packet forwarding and load balancing. It is implemented on a P4-programmable switch to achieve line-rate performance.

C. PFCP State Management

The Scaler, PFCP Proxy, and Monitor cooperate to enable elastic UPF scaling while preserving session continuity.

1) PFCP Message Handling: The PFCP Proxy intercepts and distributes PFCP messages to synchronize control-plane state across UPF instances within a UPF Service.

Heartbeat messages: The PFCP Proxy responds to Heartbeat Requests on behalf of UPF instances, abstracting the internal multi-instance structure from the SMF. The Monitor detects instance-level failures, and LB-C excludes unhealthy UPF instances from session allocation and dataplane steering.

Association and Session messages: These messages are forwarded to all UPF instances after the address fields are rewritten. For session messages, the PFCP Proxy extracts identifiers, including the UPF Service IP address, SEID, TEID, and UE IP address, and notifies LB-C to allocate or release the corresponding UPF instance.

2) PFCP State Synchronization: PFCP state synchronization is triggered by state-related messages and scale-out events.

State distribution: Association and session requests are rewritten for individual UPF instances and broadcast to all of them. Only the first valid response is returned to the SMF, with IP address fields rewritten to the UPF Service IP address.

State initialization: After scale-out, the PFCP Proxy replays stored association and session state to the newly added UPF instance. After synchronization completes, the proxy notifies LB-C to include the instance in session allocation.

Design trade-off: This broadcast-style synchronization simplifies correctness and state consistency across UPF instances, but increases control-plane overhead as the number of sessions grows.

D. Load Balancing Mechanism

The Load Balancer consists of LB-C and LB-U, which jointly provide high-performance, session-aware traffic distribution. LB-C performs event-driven control decisions, while LB-U enforces them through programmable flow tables.

1) LB-C: LB-C is implemented as an ONOS application and updates LB-U flow rules based on events reported by the PFCP Proxy and the Monitor. After PFCP synchronization, the PFCP Proxy notifies LB-C of scaling operations. LB-C then executes resource-aware allocation and migration strategies.

Flow rule updates fall into two categories: (i) instance-level changes, where Source Network Address Translation (SNAT) and Forward rules are added or removed during scale-out or scale-in, and (ii) session-level changes, where Destination Network Address Translation (DNAT) and Forward rules are updated when session-to-UPF mappings change.

Event Handling:

- **Scale-out:** After PFCP synchronization, LB-C installs SNAT and Forward rules to activate the new UPF instance.
- **Scale-in:** LB-C stops assigning new sessions to the retiring instance and invokes UPF Instance Allocation to redistribute active sessions, updating DNAT and Forward rules accordingly.
- **Session Creation/Deletion:** DNAT and Forward rules are installed or removed based on the allocation policy (Section III-D2).
- **Overload:** Overload alerts trigger Migrated Session Selection (Section III-D2) to migrate low-traffic sessions.

2) Decision Strategies of LB-C: LB-C applies two complementary strategies: *Migrated Session Selection* and *UPF Instance Allocation*.

a) Migrated Session Selection: During overload handling, LB-C selects sessions to offload from an overloaded UPF instance. Per-session throughput T_{s_i} is estimated from UPF pod network counters collected at a fixed sampling interval and correlated with session identifiers in LB-C.

Let UPF_{over} denote the overloaded instance with throughput T_u and threshold $threshold$. The excess load is:

$$T_m = T_u - threshold. \tag{1}$$

LB-C sorts sessions s_i by throughput T_{s_i} and selects the minimum set such that:

$$\min_n \left(\sum_{i=1}^{n} T_{s_i} > T_m \right). \tag{2}$$

This greedy strategy minimizes disruption by migrating multiple low-traffic sessions rather than a single high-traffic one.

b) UPF Instance Allocation: For new sessions and scale-in events, LB-C assigns sessions to the least-loaded UPF instance j with available capacity:

$$c_j = threshold - T_{u_j}. \tag{3}$$

Sessions s_i with demand T_{s_i} are allocated sequentially while ensuring $\sum T_{s_i} < c_j$. Unallocated sessions are then placed on the next least-loaded instance.

3) LB-U: LB-U is implemented on a P4 switch and enforces LB-C decisions through flow tables. ARP requests are forwarded to LB-C for Proxy ARP, while data packets are processed using three tables:

- **SNAT table:** Rewrites the source IP addresses from UPF instances to the UPF Service virtual IP address.
- **DNAT table:** Rewrites destination IP addresses and TEIDs to the assigned UPF instance.
- **Forward table:** Determines the output port and rewrites the MAC addresses.

TEID consistency: TEIDs are not modified by the framework. LB-U matches packets using destination IP and TEID, and rewrites only the destination IP to steer traffic to the selected UPF instance.

The P4 pipeline is shown in Fig. 2, and rule updates are summarized in Table I.

E. Scaler and Monitor

The Scaler and Monitor provide elastic resource control at the service and instance levels.

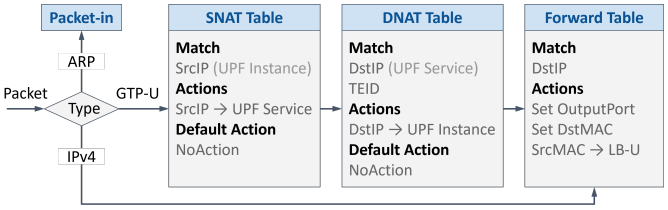

Fig. 2. LB-U P4 Pipeline Overview

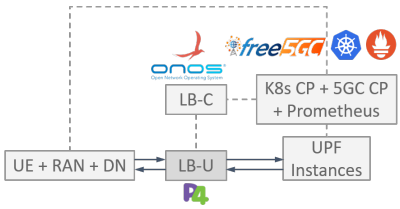

Fig. 3. Single UPF Service testbed, including Kubernetes control, UPF worker node, LB-C server, and P4-based LB-U.

TABLE I
FLOW RULE UPDATES IN LB-U UNDER DIFFERENT EVENTS

Event	SNAT	DNAT	Forward
Scale-out	Add (new inst.)	–	Add (new inst.)
Scale-in	Del (retired inst.)	Update (reallocated sess.)	Update (reallocated sess.); Del (retired inst.)
Session create	–	Add (new sess.)	Add (new sess.)
Session delete	–	Del (ended sess.)	Del (ended sess.)
Overload	–	Update (migrated sess.)	Update (migrated sess.)

1) Scaler: Dynamic scaling is realized using the Kubernetes HPA, with the scaling policy defined as:

$$N_{\text{des}} = \left\lceil N_{\text{cur}} \cdot \frac{M_{\text{cur}}}{M_{\text{tar}}} \right\rceil, \tag{4}$$

where N_{des} and N_{cur} denote the desired and current number of pods, respectively, M_{cur} is the measured average throughput, and M_{tar} is the target throughput. A Prometheus adapter is used to expose throughput-based custom metrics, enabling traffic-aware scaling decisions.

During scale-in, HPA does not control which UPF instance is terminated. To prevent disruption, the pod termination lifecycle is extended with a `preStop` hook and a sufficient grace period, ensuring that active sessions are migrated before termination.

2) Monitor: Prometheus periodically collects pod-level throughput metrics from kubelets. When utilization exceeds a predefined threshold, an alert is sent via webhook to LB-C, which triggers session migration to mitigate overload.

Metric selection rationale: Throughput is used as the primary metric because UPF processing cost is dominated by GTP-U encapsulation and forwarding volume in the evaluated deployment. Although additional metrics could refine decisions, throughput provides a stable and readily available indicator of load.

IV. EVALUATION

This section evaluates the proposed framework under single- and multi-UPF Service scenarios. The goals are to validate functional correctness, scalability, and robustness under dynamic traffic conditions.

A. Single UPF Service Scenario

1) Experimental Setup: The experimental testbed consists of four servers and one P4 switch, as illustrated in Fig. 3. One server acts as the Kubernetes control-plane node, which hosts the 5GC control-plane functions (e.g., AMF, SMF) and the monitoring stack (Prometheus). A second server functions as a Kubernetes worker node, on which all UPF service pods are deployed. A third server emulates the UE, the Radio Access Network (RAN), and the DN. A dedicated controller server runs the LB-C, while an Inventec D10056 P4 switch implements the LB-U.

TABLE II
EXPERIMENTAL TESTBED CONFIGURATION

Node	Hardware / OS	Software / Functions
K8s control-plane (5GC + Prometheus)	Intel i7-10700, 16GB RAM Ubuntu 18.04.5 (Kernel 5.4.0-87)	Kubernetes v1.19.0 Prometheus v2.28.1 free5GC v3.0.4 5GC control-plane and monitoring
K8s worker node (UPF service)	Intel i7-10700, 16GB RAM, Intel 82599ES 10Gb NIC Ubuntu 18.04.4 (Kernel 5.4.0-87)	Executes UPF Service pods
UE + RAN + DN emulator	Intel i7-10700, 16GB RAM, Intel 82599ES 10Gb NIC Ubuntu 18.04.4 (Kernel 5.4.0-89)	UERANSIM, libgtp5gnl + gtp5g, iPerf3 UE registration, PDU session setup, traffic generation
LB-C server	Intel Xeon E5-2620, 64GB RAM Ubuntu 18.04.5 (Kernel 5.0.0-23)	ONOS v2.2.2 Runs LB-C application
LB-U (P4 switch)	Inventec D10056, Barefoot SDE v9.1.0	Implements LB-U with P4 pipeline

For the RAN emulator, `libgtp5gnl` controls the `gtp5g` kernel module to execute GTP-U encapsulation and decapsulation in the kernel space, which reduces the processing overhead of the user space during high-throughput traffic generation.

Table II summarizes the detailed hardware and software configurations. This structured presentation highlights reproducibility and clarifies the role of each component.

2) Auto-Scaling Test: **Objective:** Validate auto-scaling behavior under dynamic traffic variation.

Setup: HPA was configured to maintain one to four UPF instances, with a target average inbound traffic of 1.5 Gbps, a measurement interval of 1 s, and a 10 s scale-in stabilization window.

Results and Analysis: Traffic was increased by establishing a new PDU session every 20 s, each generating 2 Gbps for 100 s. The scaling decisions were computed according to the

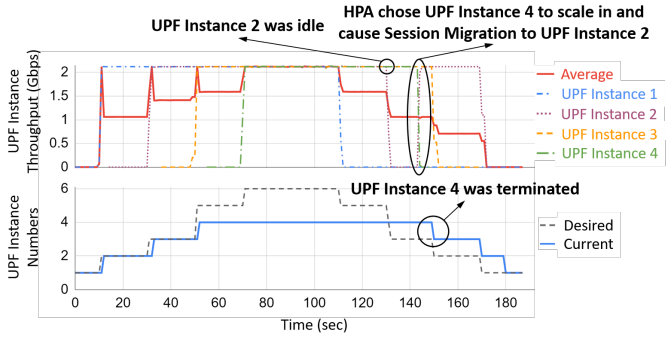

Fig. 4. Scaling dynamics in the single UPF Service scenario, showing scale-out and scale-in actions with seamless migration.

HPA formulas in Eq. 4. During the first 80 s, each new session caused the per-instance average to exceed the threshold, triggering scale-out to a maximum of four instances. After 100 s, sessions terminated sequentially, leading to scale-in actions.

Between 140 s and 160 s, HPA attempted to remove UPF instance 4. Since it was still handling active sessions, seamless migration was triggered, and all sessions were reassigned to idle instance 2. This prevented packet loss.

The stabilization window successfully suppressed oscillations, eliminating the "ping-pong effect." Overall, the integration of HPA with PFCP state synchronization achieved elasticity and service continuity, as illustrated in Fig. 4.

3) Overload Handling Test: **Objective:** Evaluate LB-C's overload mitigation mechanism.

Setup: Three UPF instances were deployed with an overload threshold of 2.5 Gbps. Sessions were distributed with one overloaded instance carrying 3.1 Gbps.

Results and Analysis: LB-C applied its migrated session selection policy, which sorts active sessions s_i on the overloaded UPF in ascending order of throughput T_{s_i} and selects the minimum number of sessions according to Eq. 2. In the evaluated scenario, Session 2 and Session 3 (each 0.5,Gbps) were chosen for migration.

During reallocation, UPF instance 2 had the lowest utilization, but only 0.8,Gbps residual capacity, so it accepted one 0.5,Gbps session, while the other 0.5,Gbps session was assigned to UPF instance 3, which had sufficient capacity. This experiment confirms that LB-C can effectively detect and mitigate overload conditions by migrating sessions in a resource-aware manner.

The strategy effectively alleviated overload on UPF instance 1 while maintaining balanced utilization across UPF instances. As a result, the service operated stably, as illustrated in Fig. 5.

4) Seamless Migration Test: **Objective:** Assess seamless migration in scale-in events.

Setup: Two UPF instances, each handling 600 Mbps, were deployed. One instance was deleted to emulate scale-in. UDP packet loss and TCP retransmissions were measured.

Results and Analysis: Without migration, abrupt termination caused disruption. With seamless migration enabled, UDP packet loss decreased by about 20% and TCP retransmissions

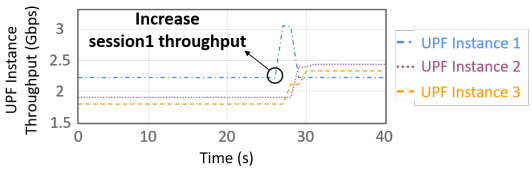

Fig. 5. Overload handling through Migrated Session Selection, balancing sessions across UPF instances.

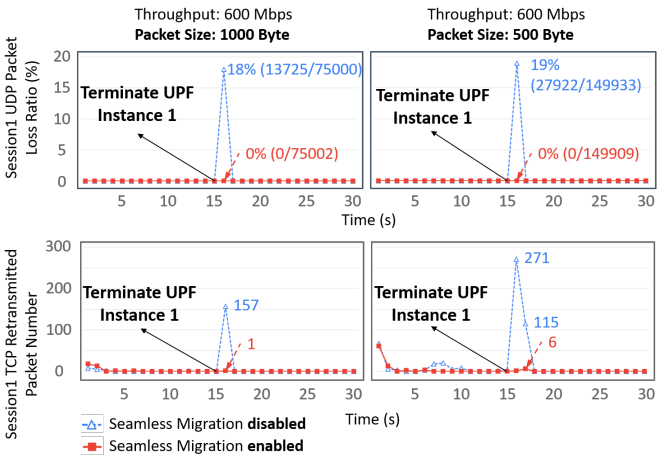

Fig. 6. Impact of seamless migration on UDP packet loss and TCP retransmissions during scale-in.

were significantly reduced. This demonstrates improved reliability and continuity, as illustrated in Fig. 6.

B. Multiple UPF Services Scenario

1) Experimental Setup: This scenario extended the setup by adding two servers hosting PDU Session Anchor (PSA) Services, PSA1 and PSA2. The three services (Uplink Classifier (ULCL), PSA1, and PSA2) shared the same LB-U. The two servers hosting PSA1 and PSA2 are each equipped with an Intel Xeon E5-2630 v4 CPU and 128 GB of memory. The ULCL is deployed on the same server as the original UPF service in the Single UPF Service scenario. Fig. 7 shows the configuration.

2) Auto-Scaling Test: **Objective:** Validate scaling in multiple UPF Services.

Setup: ULCL supported one to four instances, PSA1 and PSA2 each one to two instances, with a 1.5 Gbps target. Four sessions were established, with Sessions 1-2 using ULCL and PSA1, and Sessions 3-4 using ULCL and PSA2.

Results and Analysis: During the first 40 seconds, the establishment of Sessions 1 and 2 triggered scale-out of both ULCL and PSA1 services. Between 40 and 80 seconds, additional traffic from Sessions 3 and 4 led to scale-out of ULCL and PSA2 services. After 100 seconds, as sessions were sequentially terminated, the system scaled in by reducing the number of UPF instances.

The observed behavior demonstrates that each UPF service independently scaled with its traffic load while sharing the

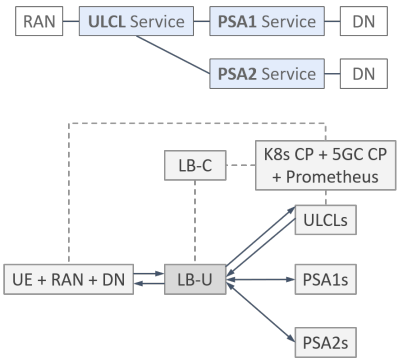

Fig. 7. Multiple UPF Service testbed with ULCL, PSA1, and PSA2 sharing one LB-U.

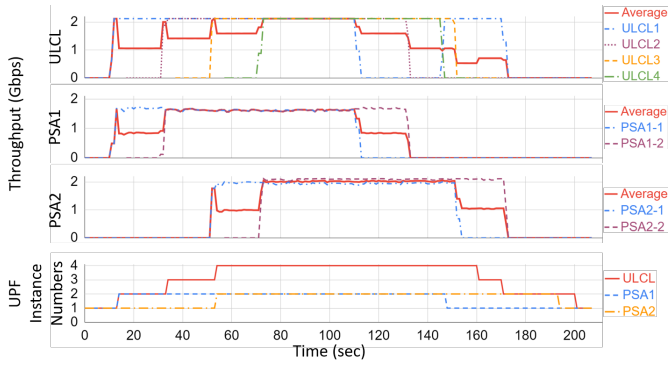

Fig. 8. Autoscaling in multiple UPF Services, showing independent scaling of ULCL, PSA1, and PSA2.

same LB-U hardware. This confirms that the framework operated correctly in a multi-service environment (Fig. 8).

V. Conclusion

This paper has presented the design, implementation, and evaluation of a scalable and load-aware UPF deployment framework for the 5GC. The contributions of this work are threefold: (i) the introduction of the *UPF Service* abstraction, which integrates multiple UPF instances with a PFCP Proxy, a Scaler, a Monitor, and a load balancing subsystem; (ii) the design of dual-level control mechanisms, where the Scaler governs service-level elasticity while the Monitor enforces instance-level overload control through session migration; and (iii) the implementation of resource-aware decision strategies in the LB-C, combined with a P4-based LB-U that offloads packet forwarding for high-throughput performance.

Experimental validation demonstrated that the framework achieves near-linear throughput scalability within the capacity bounds of the experimental testbed, preserves session continuity during scale-in through seamless migration, and mitigates overload by redistributing sessions in a load-aware manner. These results confirm that combining Kubernetes-based orchestration, P4-enabled data-plane programmability, and PFCP-driven synchronization enhances elasticity, efficiency, and robustness in virtualized mobile core networks.

Future work will optimize migration latency, leverage historical monitoring data for predictive scaling, and incorporate machine learning for proactive resource orchestration. Extending the framework to beyond 5G and sixth-generation (6G) scenarios, such as immersive and massive communications, represents a promising research direction.

Acknowledgement

This work was supported in part by the Center for Intelligent Team Robotics and Human-Robot Collaboration under the "Top Research Centers in Taiwan Key Fields Program" of the Ministry of Education (MOE), Taiwan, and in part by the National Science and Technology Council, Taiwan, under grants NSTC 114-2218-E-011-003 and 114-2221-E-A49-009.

References

[1] "5G; System architecture for the 5G System (5GS)," 3GPP, TS 23.501, V17.9.0, Release 17, Jul. 2023.

[2] "LTE; 5G; Interface between the Control Plane and the User Plane nodes," 3GPP, TS 29.244, V17.9.0, Release 17, Jul. 2023.

[3] T. V. K. Buyakar, A. K. Rangisetti, A. A. Franklin, and B. R. Tamma, "Auto scaling of data plane VNFs in 5G networks," in *13th International Conference on Network and Service Management (CNSM)*, 2017, pp. 1–4.

[4] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "An adaptive mechanism for LTE P-GW virtualization using SDN and NFV," in *13th International Conference on Network and Service Management (CNSM)*, 2017, pp. 1–9.

[5] D.-H. Luong, H.-T. Thieu, A. Outtagarts, and Y. Ghamri-Doudane, "Cloudification and Autoscaling Orchestration for Container-Based Mobile Networks toward 5G: Experimentation, Challenges and Perspectives," in *IEEE 87th Vehicular Technology Conference (VTC Spring)*, 2018, pp. 1–7.

[6] M. Liebsch and F. Z. Yousaf, "Virtualized EPC — Runtime offload for fast data-plane scaling," in *IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2016, pp. 1–6.

[7] Kubernetes SIG Autoscaling, "Horizontal Pod Autoscaler v2 API Specification," https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, accessed: Aug. 14, 2025.

[8] Open Networking Foundation, "SD-Fabric and UP4: High-Performance 5G User Plane on P4," https://docs.sd-fabric.org/master/advanced/p4-upf.html, 2021, accessed: Aug. 14, 2025.

[9] R. MacDavid, C. Cascone, P. Lin, B. Padmanabhan, A. Thakur, L. Peterson, J. Rexford, and O. Sunay, "A P4-based 5G User Plane Function," in *ACM SIGCOMM Symposium on SDN Research (SOSR)*, 2021, pp. 162–168.

[10] Z. Wen and G. Yan, "HiP4-UPF: Towards High-Performance Comprehensive 5G User Plane Function on P4 Programmable Switches," in *USENIX Annual Technical Conference (USENIX ATC)*, Jul. 2024, pp. 303–320.

[11] 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, 2023.

[12] C. Rizzi, Z. Yao, Y. Desmouceaux, M. Townsley, and T. Clausen, "Charon: Load-Aware Load-Balancing in P4," in *17th International Conference on Network and Service Management (CNSM)*, Oct. 2021, pp. 91–97.

[13] J. Zhou, Z. Ma, W. Tu, X. Qiu, J. Duan, Z. Li, Q. Li, X. Zhang, and W. Li, "Cable: A framework for accelerating 5G UPF based on eBPF," *Computer Networks*, vol. 222, 2023.

[14] Y. X. Huang, K. C. Wang, and B. S. Ke, "Accelerating 5G Service-Based Architecture with eBPF," in *12th International Conference on Networks, Communication and Computing (ICNCC)*, 2023, pp. 200–209.