Rethinking 5G Core for Performance, Fairness, and Flexibility

Yang-Zhe Lin
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
linyangzhe0508@gmail.com

Chun-Ting Lin
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
timlin891207@gmail.com

Po-Yi Lin
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
linpoyi53@gmail.com

Shixiong Qi
University of Kentucky
Lexington, USA
shixiong.qi@uky.edu

Jyh-Cheng Chen
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
jcc@nycu.edu.tw

K. K. Ramakrishnan
University of California, Riverside
Riverside, USA
kk@cs.ucr.edu

Abstract

The 5G Core (5GC) adopts a service-based architecture that enables flexible deployment of network functions (NFs) across the control and user planes. Despite its advantages, building a 3GPP-compliant, high-performance 5GC remains a significant challenge. In this paper, we present L^{2}5GC+, a 3GPP-compliant 5GC platform designed for low-latency 5GC operations. L^{2}5GC+ advances our previous prototype, L25GC, by replacing the kernel-based Service-Based Interface (SBI) with X-IO, a shared-memory-based SBI that maintains HTTP-style semantics while significantly reducing inter-NF communication latency. The platform integrates Golang-based NFs from free5GC, supports seamless upgrades to 3GPP Release 17, and introduces several key features: (1) enhanced QoS enforcement in the UPF with fairness guarantees, (2) rate limiting for hotspot control plane NFs (*e.g.,* AMF) to prevent thrashing, (3) network slicing to enable multi-tenant isolation and amortize polling overhead.

We evaluate L^{2}5GC+ on the NSF FABRIC testbed in a geographically distributed deployment, integrating the RAN, core, and data network across multiple sites. L^{2}5GC+ is being made available as an open Core-as-a-Service to support the broader 5G research community.

CCS Concepts

• **Networks** → **Mobile networks**; • **Computer systems organization** → **Cellular architectures**.

Keywords

5G core, shared memory, low latency, 3GPP, QoS, fairness, slicing

ACM Reference Format:
Yang-Zhe Lin, Chun-Ting Lin, Po-Yi Lin, Shixiong Qi, Jyh-Cheng Chen, K. K. Ramakrishnan. 2025. Rethinking 5G Core for Performance, Fairness, and Flexibility. In *Proceedings of the 2025 free5GC World Forum (free5GC '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3733814.3765491

*Corresponding author: shixiong.qi@uky.edu.

free5GC '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1903-5/25/10
https://doi.org/10.1145/3733814.3765491

1 Introduction

The 5G core (5GC) network represents a fundamental shift in mobile core architecture, moving beyond the legacy monolithic 4G Evolved Packet Core (EPC) towards a microservice-based architecture [14, 17]. The 5GC decouples the network functions (NFs) in the mobile core control plane into individual microservices that communicate over a 3GPP-recommended Service-Based Interface (SBI) [1]. This enables flexible scaling and chaining of individual NFs to facilitate cloud-native deployment to lower the operational cost of the 5GC.

Such a decoupled design results in frequent message exchanges between control plane NFs, given the complexity of 3GPP-defined control plane protocols: a PDU session establishment event takes up to 16 messages between control plane NFs to complete; completing a paging event involves up to 10 messages between control plane NFs [7]. In addition, the existing 3GPP-recommended SBI typically chooses kernel-based HTTP/REST APIs as the underlying implementation, which introduces significant overhead when control plane messages are exchanged between NFs [14]. Furthermore, the move to smaller cells is an important consideration as well. For example, 5G seeks to utilize millimeter wave (mmWave) radio links for higher bandwidth and ultra-low latency, with the goal of also supporting massive device densities. MmWave-oriented base stations have smaller coverage area, leading to more frequent handovers when users switch between smaller cells. This in turn introduces more control plane messages exchanges, magnifying the latency penalties for the control plane with the decoupled 5GC.

Although a lot of existing work focuses on reducing latency in 5GC user plane (*i.e.,* UPF) [12, 20, 27, 34, 36], reducing control plane latency is equally critical. The control plane is responsible for user session setup, mobility management, *etc.* , which are necessary operations before the user data can begin to flow. Thus, a slow control plane also impacts user place performance in 5GC [14].

To enable low-latency control plane operations in 5GC, we developed L^{2}5GC+ in [19], a system designed around a shared-memory-based SBI built on top of OpenNetVM [39]. This design facilitates low-latency communication between the decoupled control plane NFs. In addition, L^{2}5GC+ directly adopts Golang-based NFs from free5GC [4], allowing us to inherit its latest advances, such as 3GPP Release 17 compatibility and New Radio Dual Connectivity

(NR-DC [35]) support. While OpenNetVM offers high-performance shared memory communication, it is fundamentally incompatible with the synchronous, HTTP/REST-based semantics required by the 3GPP SBI. This mismatch, compounded by language differences between DPDK-based OpenNetVM (C) and free5GC (Golang), prevents direct integration. To address this, we developed X-IO [30], a compatibility I/O layer that emulates SBI semantics over shared memory and bridges the language gap via Golang's CGO interface. Enabled by X-IO, L^{2}5GC+ has been seamlessly upgraded to 3GPP Release 17 using the free5GC codebase for NFs with only minimal dependency adjustments.

In this work, we further enhance L^{2}5GC+'s control plane performance by introducing a rate limiter for control plane NFs. Our analysis in §2.2.1 reveals that bursts of control plane load, often triggered by a large number of concurrent user sessions, can easily overwhelm certain "hotspot" NFs, such as AMF, thereby increasing the completion time of control plane events. Existing 5GC NF implementations typically employ a thread-based concurrency model to handle events from multiple concurrent user sessions, where each user session spawns a separate thread (or goroutine, in the case of Golang). While this model enables parallelism, it also incurs frequent context switches and increased scheduling latency under high load, which exacerbates control plane latency. The dynamic rate limiter in L^{2}5GC+ adaptively adjusts the concurrency limit in the control plane based on runtime performance indicators within the NF, preventing the control plane overload.

In addition to the aforementioned optimizations, our latest update to L^{2}5GC+ introduces several key features: fairness-enhanced QoS support in the user plane and network slicing. These capabilities are increasingly important in cellular networks for serving a diverse mix of applications, such as video streaming, cloud gaming, and autonomous driving, each of which has distinct requirements for QoS guarantees and resource isolation.

The QoS mechanism in L^{2}5GC+'s UPF adheres to the Two Rate Three Color Marker (trTCM) model recommended by 3GPP [9], ensuring prioritization of designated user flows. Our implementation further extends beyond standard 3GPP support by incorporating a token bucket to ensure the fairness across flows, addressing the known limitation of the traditional leaky bucket approach. This enhancement ensures not only strict compliance with flow-level QoS guarantees in 3GPP but also fair bandwidth allocation between QoS and non-QoS flows, particularly under congestion.

L^{2}5GC+ supports network slicing to enable the creation of isolated virtual 5GCs (called slices), which allows operators to offer differentiated services to distinct user groups. Moreover, slicing in L^{2}5GC+ can be extend to be end-to-end, including both the RAN as well as the backhaul, enabling advanced use cases such as QoS-assured VPNs. This enhancement also creates the opportunity to improve resource utilization of L^{2}5GC+. Resources (*e.g.,* CPU) can be multiplexed across slices to amortize the polling overhead of DPDK-based packet processing in the underlying OpenNetVM framework.

Beyond system design and implementation, we validate the capabilities of L^{2}5GC+ in a geographically distributed setup on the US NSF Fabric testbed [11]. Our deployment spans L^{2}5GC+ core network, simulated UE/RAN and data networks (DN) across multiple locations, providing a more realistic evaluation environment. We will make the L^{2}5GC+ publicly deployable as a "Core-as-a-Service" on the NSF Fabric testbed to facilitate open 5G research. L^{2}5GC+ is publicly available at https://github.com/nycu-ucr/L25GC-plus.

2 Background and Challenges

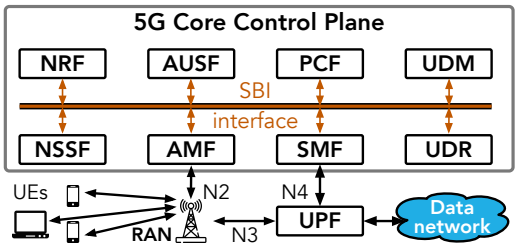

Figure 1: Architecture of 5GC.

2.1 Architecture of 5G Core (5GC) Network

Fig. 1 depicts the architecture of the 5G Core (5GC) network with the separate control plane and user plane. The control plane is responsible for managing user sessions, such as the registration, authentication, mobility handling (*e.g.,* handover), billing, *etc.* The user plane consists of one or more User Plane Function (UPF) instances that handle user data forwarding between the radio access network (RAN) and the data network (DN). In addition, the UPF is the key NF responsible for enforcing QoS policies and traffic shaping. These UPF instances may also be distributed, while logically sharing a centralized control plane, which enables flexible deployment of 5GC.

The 5GC control plane follows the service-based architecture as specified by 3GPP, where each NF operates as an independently running microservice. We list the key control plane NFs in Fig. 1, including Access and Mobility Management Function (AMF), Session Management Function (SMF), Network Slice Selection Function (NSSF), Authentication Server Function (AUSF), Network Repository Function (NRF), Policy Control Function (PCF) and Unified Data Management (UDM). Among these, the AMF serves as the entry point for UEs to access the 5GC (through by RAN). The AMF acts as a frontend to orchestrate UE registration and mobility procedures (*e.g.,* handover) by coordinating with other control plane NFs. The SMF serves as the bridge between the control plane and user plane. It configures user sessions on the UPF based on instructions received from the rest of the control plane. Communication between control plane NFs is standardized through the 3GPP SBI, which typically uses HTTP/REST APIs over a kernel-based networking stack that however could introduce substantial overhead. Every inter-NF message passing involves copying, context switches, protocol processing, serialization and deserialization. These overheads compound over time and become significant due to the complexity of 5GC control plane procedures. As shown in Fig. 2, representative procedures such as PDU (Protocol Data Unit) session establishment, handover, and paging (idle-to-active transition) involve substantial inter-NF communication. Each procedure requires the exchange of dozens of messages, with each message incurring the full overhead

of kernel-based networking. This repeated cost becomes a critical bottleneck in the latency-sensitive 5GC.

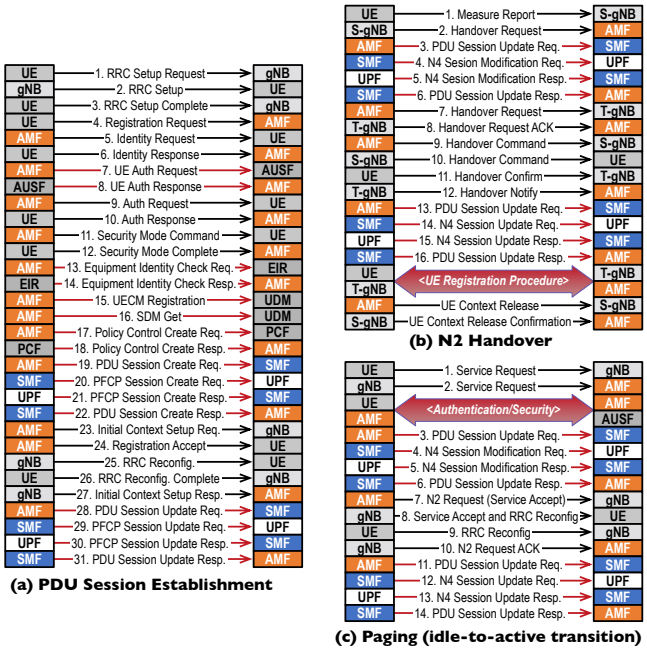

Figure 2: Procedures of representative 5GC control plane events: (a) PDU session establishment; (b) N2 Handover; (c) Paging. Red arrows depict communication within 5GC.

2.2 Challenges

We introduced X-IO [30] in L^{2}5GC+ to replace the kernel-based SBI, which significantly reduces the latency overhead associated with inter-NF communication (details in §3.1). However, several important challenges remain unaddressed in L^{2}5GC+, including the need to manage NF concurrency under load to prevent thrashing (§2.2.1), lack of fairness in QoS handling across user flows (§2.2.2), and lack of network slicing support (§2.2.3).

2.2.1 Thrashing of hotspot control plane NFs. AMF and SMF are typically the "hotspot" (most frequently involved) NFs in the 5GC, heavily involved in frequent control plane procedures, such as PDU session establishment, handover, and paging, as shown in Fig. 2. In most 5GC implementations, each user session is served by a dedicated thread. This also applies to Golang-based 5GC platforms (*e.g.,* free5GC, SD-core, Magma Core), where threads are abstracted as goroutines. Golang is often chosen for developing 5GC NFs due to its concise programming syntax and built-in concurrency support via lightweight goroutines. Upon receiving a control plane message associated with a new user session, the NF immediately spawns a new goroutine to process the message, enabling concurrent event processing for multiple user sessions.

While this per-session threading model simplifies concurrency, it introduces scalability bottlenecks and may lead to thrashing under heavy traffic (observed in the past for general kernel network protocol processing [22]), a condition where excessive context switching overwhelms CPU resources. The purely load-driven creation of goroutines for each request leads to a rapid buildup of concurrent execution contexts, and thus to frequent context switching between threads under high load.

To evaluate the impact of excessive concurrent processing of events on the overall event completion time, we conducted offline profiling. We vary the maximum number of events (*i.e.,* concurrency limits) in the AMF: 2, 4, 8, 12, 16, 20, and also an unlimited number, as shown in Fig. 3. Details of the experiment setup are provided in §4.1. Our results from the PDU session establishment procedure confirm that beyond a point, the increase in the concurrency also increases the completion time for this control plane event. As the number of active goroutines grows, overhead from scheduling and context switching goes up non-linearly, resulting in an excessive completion time for the control plane operation. The lowest event completion time for the establishment of the PDU session is achieved when the limit is set between 8 and 16. Setting the limit too low also results in a longer event completion time. This is due to insufficient use of concurrency, which in turn leads to increased queueing delay.

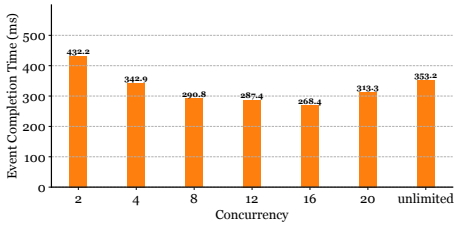

Figure 3: Completion time of PDU session establishment under different concurrency limits.

2.2.2 "Leaky-bucket" QoS support in 3GPP-based UPF. Effective QoS support in UPF is essential to ensure differentiated treatment across user data flows. The 3GPP-recommended design, as outlined in TS 23.501 [9], employs the Two Rate Three Color Marker (trTCM [13]) to classify traffic into green, yellow, and red based on compliance with Guaranteed Bit Rate (GBR) and Maximum Bit Rate (MBR). While this mechanism provides a basis for prioritizing traffic, 3GPP lacks clear guidance on how to manage fairness across concurrent flows, particularly in scenarios where aggregated bandwidth of flows exceeds available capacity.

In fact, it is common for non-QoS flows and QoS flows operate above their committed rate, *i.e.,* sharing a portion of excess bandwidth. Without explicit fairness enforcement, the non-QoS flows may experience undesirable starvation, even if it originates from legitimate users with ongoing tasks to be completed in reasonable time. Thus, enforcing fair allocation is important to ensure the proper use of that excess bandwidth across all flows (Qos or non-QoS). This is addressed by our fairness-enhanced QoS support using additional token bucket shaper (§3.4).

2.2.3 Lack of 5GC network slicing support. Network slicing enables the coexistence of multiple virtual 5GC networks (each called a network slice) on a shared physical infrastructure. Each network slice is defined by a unique ID, called a Single Network Slice Selection Assistance Information (S-NSSAI). Each network slice is customized to

serve different application classes, *e.g.,* enhanced mobile broadband (eMBB), for high-throughput applications such as video streaming; ultra-reliable low-latency communications (URLLC), for latency-sensitive scenarios industrial automation; and massive machine-type communications (mMTC) for large-scale IoT deployments with low per-device data rates.

Despite its improvements in reducing inter-NF communication latency, L^{2}5GC+ lacked the support for network slicing. This limits its ability to support multiple virtual core networks to enable multi-tenant services (*e.g.,* MVNO) and enforce quality guarantees across distinct use cases.

In addition, each L^{2}5GC+ NF instance uses a dedicated CPU core for busy polling (*e.g.,* using OpenNetVM's base design in §3.2) to enable shared-memory-based inter-NF communication. This design avoids interrupt overhead and enables low-latency communication. However, it simplistically provisions for peak loads, leading to wasted overhead with busy polling under light load. Naively scaling the number of used CPU cores (each operating at 100% regardless of the load) linearly with the number of NF instances leads to poor resource utilization. The lack of network slicing support in L^{2}5GC+ further exacerbates this issue. Without slicing support, it is not feasible to share NF instances across multiple slices, making it necessary to allocate separate polling cores for each slice-specific NF. This rigid one-core-per-NF model (even when a slice uses a tiny amount of resources) leads to inefficient resource usage of L^{2}5GC+. Adding network slicing support will enable more flexible resource sharing in L^{2}5GC+, as we discuss in §3.5.

3 Design

We begin with an architecture overview of L^{2}5GC+, introducing its core components (§3.1). Next, we analyze the fundamental incompatibility between the 3GPP SBI and OpenNetVM's shared memory interface, and how X-IO addresses it (§3.2). We then discuss the dynamic rate limiter deployed in the AMF in L^{2}5GC+ to prevent thrashing (§3.3). We then discuss our fairness-enhanced QoS support in L^{2}5GC+'s UPF (§3.4). We discuss support for network slicing and how to amortize the polling overhead caused by DPDK (§3.5). We finally discuss the roadmap and ongoing development of L^{2}5GC+, including the NR-DC support in (§3.6).

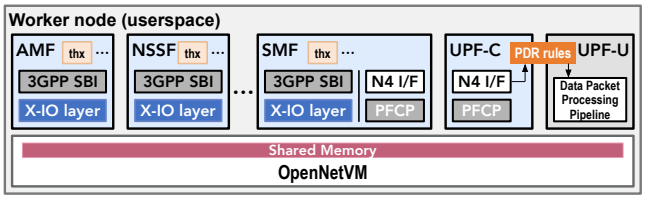

Figure 4: Overview of L^{2}5GC+. This figure shows the consolidated deployment of L^{2}5GC+ NFs on a single node, including both control and user planes. L^{2}5GC+ remains a 3GPP-compliant design through careful layering of different stacks.

3.1 Overview of L^{2}5GC+

Fig. 4 shows the overall architecture of L^{2}5GC+, which uses OpenNetVM [39] for some key functionality. OpenNetVM is a high-performance kernel-bypass NFV platform based on DPDK. It supports zero-copy NF chaining using shared memory processing, making it a suitable foundation for L^{2}5GC+. We leverage OpenNetVM to support both low-latency control plane operations and user plane packet processing. By bypassing the kernel, shared memory processing eliminates expensive overheads associated with data copies, protocol processing, context switches, serialization, and deserialization, which are resource-intensive operations commonly seen in the kernel-based 3GPP SBI.

We show the deployment of L^{2}5GC+ NFs in Fig. 4, where control plane NFs and user plane NFs are co-located on the same worker node to maximize the benefits of shared memory processing. In each worker node's userspace, L^{2}5GC+ allocates a shared memory pool accessible to all co-located NFs. Messages are exchanged through lightweight descriptors delivered between NFs, while the actual message payload remains in-place within the shared memory. To enable seamless 3GPP SBI interactions between control plane NFs, L^{2}5GC+ integrates X-IO within each NF. X-IO acts as a bridge between the HTTP-based SBI and OpenNetVM's shared memory APIs, enabling smooth interaction with the Golang-based NF implementation from free5GC.

Furthermore, the N4 interface between the control plane and user plane is also realized over shared memory in this consolidated deployment. As shown in Fig. 4, the SMF communicates PDR rules and session contexts to the UPF over the N4 interface using PFCP (Packet Forwarding Control Protocol), as specified by 3GPP. In L^{2}5GC+, we replace the kernel-based UDP socket with shared memory processing to enable PFCP over shared memory.

Disaggregated user plane: As shown in Fig. 4, L^{2}5GC+ further disaggregates the user plane (*i.e.,* UPF) into two complementary NFs: UPF-C and UPF-U. The UPF-C handles control plane operations, acting as the N4 interface endpoint that communicates with the SMF to receive and update PDR rules and user session contexts. This disaggregation allows for rapid updates to PDRs and session state without impacting the user plane performance. Meanwhile, it maintains flexibility after offloading the UPF-U to a SmartNIC, where we can fully customize the interface between UPF-C and UPF-U for optimal performance without being constrained by 3GPP implementation recommendations.

With this deployment, the UPF-C and UPF-U still share the PDRs and user session state, residing in OpenNetVM's shared memory. This eliminates extra state updates between UPF-C and UPF-U during events such as paging or N2 handover. As a result, the N4 communication overhead is minimized and UPF-U can access PDRs at minimal cost.

3.2 X-IO Layer

Shared memory communication in OpenNetVM: Following OpenNetVM's zero-copy packet processing design, each control plane NF in L^{2}5GC+ has a pair of producer/consumer rings for receiving (RX) and transmitting (TX) descriptors, as shown in Fig. 5. Each NF includes a dedicated packet handler implemented in *C* using the NFLib in OpenNetVM, which provides low-level primitives

for interacting with producer/consumer rings. Descriptor routing between NFs is coordinated by OpenNetVM's NF manager, which is also responsible for creation of the shared memory pool.

When an NF sends a message to another NF, it writes a descriptor containing the target NF ID and a pointer to the message payload residing in shared memory. This descriptor is put into the source NF's TX ring. The NF manager polls the TX ring, retrieves the descriptor, and puts it into the RX ring of the target NF. The target NF then polls its RX ring and addresses the message using the pointer in the descriptor. This design ensures each ring has only a single producer and a single consumer, offering lock-free descriptor exchange between NFs.

Incompatibility between shared memory processing and 3GPP SBI: Despite its performance advantages, OpenNetVM's shared memory API is incompatible with the 3GPP SBI, which relies on synchronous HTTP/REST-based request-response semantics. OpenNetVM provides an asynchronous, non-blocking messaging interface built on lock-free ring buffers and polling-based communication, optimized for high-throughput NFV workloads. In contrast, the 3GPP SBI requires different semantics, such as such as blocking calls. It also requires timeout handling, concurrent connections, and structured response codes. These essential abstractions are absent in OpenNetVM's shared memory API, creating a semantic mismatch that prevents direct support for 3GPP-compliant NFs.

This incompatibility is further compounded by a language-level mismatch: OpenNetVM is implemented in C and tightly coupled with DPDK [3], while the NFs from free5GC integrated into L^{2}5GC+ are written in Golang. Bridging both the semantic and language-level gaps is critical for enabling seamless integration of 3GPP-compliant NFs with a shared memory communication interface.

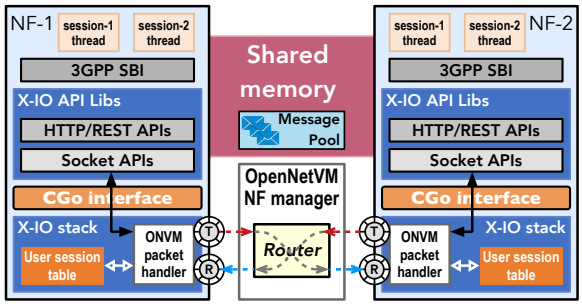

Figure 5: The X-IO [30] layer in L^{2}5GC+. X-IO offers an unified interface to mitigate the incompatibility between synchronous, Golang-based 3GPP SBI and asynchronous C-based shared memory communication API from OpenNetVM [39].

X-IO to unify shared memory processing with 3GPP SBI. To address these incompatibilities, we designed X-IO [30], a compatibility I/O layer that bridges OpenNetVM's shared memory I/O with the 3GPP SBI. As shown in Fig. 5, we wrap OpenNetVM's packet handler into the X-IO stack (in C), which introduces several necessary abstractions (condition variable for blocking calls, user session table for concurrent connections *etc.*) to emulate HTTP-style semantics over a high-performance shared memory substrate. On top of the X-IO stack, we provide the X-IO API library (in Golang), which exposes a socket interface to directly interact with the underlying X-IO stack for shared memory communication. A full set of HTTP/REST APIs, which preserve the same semantics as the kernel-based one, are provided on top of the socket interface. The user-space NF code can operate with the 3GPP SBI directly on top of these APIs. X-IO leverages Golang's Foreign Function Interface (FFI), or CGO, to allow Golang-based 3GPP SBI to directly access low-level shared memory primitives in C, including raw pointers and ring buffers. This integration ensures that L^{2}5GC+ retains full 3GPP compliance while benefiting from the performance advantages of shared-memory-based communication [19].

3.3 Dynamic Rate Limiter for Control Plane NFs

After observing that an unbounded number of goroutines leads to a sharp decline in control plane performance (Fig. 3 in §2.2.1), one intuitive mitigation is to cap the total number of goroutines, *i.e.,* prevent spawning new goroutines once a certain limit is reached. However, a static, offline-profiled rate limiter may be insufficient in practice due the complexity of the 5GC deployment. Factors such as NF scaling, placement, and slice-specific resource allocations and adjustments introduce runtime variability and inter-NF interference, making fixed limits suboptimal. Instead, such limit should be dynamically adjusted at runtime based on real-time performance metrics to accurately reflect the NF's current state.

We introduce a dynamic rate limiter in the L^{2}5GC+ control plane to cap the number of in-progress events, thereby bounding the total number of goroutines across control plane NFs to prevent thrashing. We deploy the rate limiter primarily at the AMF as it is the entry point for all control plane events (via the N2 interface). Thus, it is more susceptible to overload than other downstream NFs. When the AMF reaches its concurrency limit, it defers processing of new incoming messages until in-progress events complete. This mechanism implicitly limits the rate at which downstream NFs (*e.g.,* SMF, AUSF) receive and process events, thereby protecting the rest of the control plane from overload. The concern of insufficient utilization of downstream NFs can be addressed by carefully multiplexing underutilized NFs across different slices (§3.5).

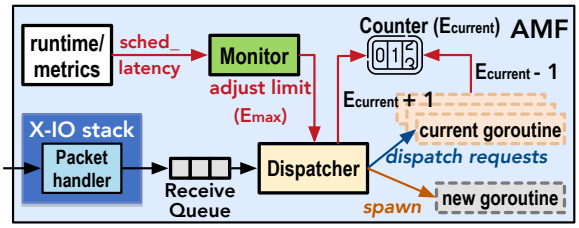

Figure 6: Rate limiter in L^{2}5GC+'s AMF.

Fig. 6 depicts the design of the rate limiter in L^{2}5GC+'s AMF. Key components include a receive queue for buffering incoming messages, a dispatcher for distributing messages to idle goroutines or spawning new goroutines as needed, a counter for tracking concurrent events in progress and a monitor for recording goroutines scheduling latency. We describe the details of these components next.

Receive queue: When the AMF receives a message that initiates a new control plane event, the message is first placed into a receive queue. For example, in a PDU session establishment procedure triggered by the UE, the event begins with a *PDU Session Establishment Request* message. When this message arrives at the AMF via the N2 interface, it is enqueued and waits to be scheduled for execution by the dispatcher.

Dispatcher: Within the AMF, we implement a lightweight goroutine called the dispatcher. When the number of in-progress events (E_{current}) is below the configured limit, *i.e.,* maximum number of concurrent events (E_{max}), it retrieves arrival messages from the receive queue and distributes messages to goroutines for processing. When E_{current} reaches the E_{max} limit, the AMF enters the blocking state, waiting for some in-progress events to finish.

Concurrent event counter: To track the number of in-progress events (E_{current}), we introduce a counter in the AMF. As shown in Fig. 6, this counter is incremented when a message initiating an event begins processing and decremented when the corresponding completion message is processed. For example, in the PDU session establishment procedure (Fig. 2 (a)), the counter is incremented upon receiving the *PDU Session Establishment Request* and decremented upon processing the *PDU Session Resource Setup Response.*

Scheduling latency monitor: We monitor the scheduling latency that goroutine spends waiting in the scheduler, *i.e.,* time spent in a runnable state before actually running. This is a reliable indicator of scheduler contention and goroutine queuing delays. Increased scheduling latency indicates potential thrashing, suggesting a reduction of the number of goroutines, thus determining the maximum number of concurrent events (E_{max}) in the concurrent event counter. The monitor collects this metric every 500 ms, via '*/sched/latencies:seconds*' exposed by Golang's built-in runtime/metrics module. To reduce sensitivity to transient fluctuations on measured scheduling latency, we apply an exponential weighted moving average (EWMA) to the monitored latency to smooth out short-term variations (Line#1-2 in Algorithm-1). We set the EWMA coefficient to 0.7 as it yielded better results in our testing.

Algorithm 1 Adjustment on E_{max} based on scheduler latency

1: **every** 500 ms:
2: $\text{EWMA}_t = 0.3 \times \text{EWMA}_{t\text{-}1} + 0.7 \times \text{sched\_latency}_t$ ▷ Update EWMA of scheduling latency measured at interval t
3: **if** $\text{E}_{\text{current}} < \text{E}_{\text{max}}$ **then**
4: **continue** ▷ Skip adjustment when the number of in-progress events is lower than the configured limit
5: **end if**
6: **if** $\text{EWMA}_t >$ `sched_latency_thresh` **then**
7: $E_{\text{max}} = E_{\text{max}} - 1$ ▷ Reduce one goroutine
8: **else if** $\text{EWMA}_t \leq$ `sched_latency_thresh` **then**
9: $E_{\text{max}} = E_{\text{max}} + 1$ ▷ Increase one goroutine
10: **end if**

As shown in Fig. 6, the monitor dynamically adjusts the maximum number of concurrent events (E_{max}), based on the monitored scheduling latency. If the monitored scheduling latency is greater than the pre-configured threshold (`sched_latency_thresh`[1]), we decrement E_{max} by one (Line#6-7 in Algorithm-1). Otherwise, we increment E_{max} by one (Line#8-9 in Algorithm-1). To ensure that these adjustments are meaningful, the rate limiter only considers increasing or decreasing the E_{max} when the number of currently in-progress events reaches the configured limit (Line#3-5 in Algorithm-1).

3.4 QoS Support in UPF with Fair Allocation of Additional Bandwidth

Based on 3GPP specification R17 [2] and the need for fairness in managing available bandwidth (beyond the guaranteed rate) between QoS flows and non-QoS flows (discussed in §2.2.2), the QoS support in our UPF is implemented as a combination of trTCM [13] and a token-bucket-based QoS shaper. The trTCM classifies QoS flows into red, yellow, and green based on traffic characteristics. The token bucket controls token consumption rate to shape/smooth traffic.

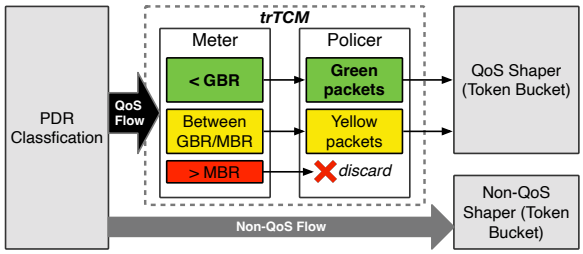

Figure 7: Fairness-enhanced QoS Support in L^{2}5GC+'s UPF: trTCM + Token Bucket.

Fig. 7 shows the QoS workflow in the UPF: (1) **PDR classification:** Upon packet arrival, the UPF classifies the traffic as QoS or non-QoS based on the PDRs; (2) **Traffic metering:** Packets belonging to the QoS flows are passed to the trTCM, which marks them with color labels based on the metered flow rate relative to two thresholds: the Guaranteed Bit Rate (GBR) and the Maximum Bit Rate (MBR). Packets metered with a rate below GBR (fully compliant) is marked as green. Yellow is used for marking packets with a rate between GBR and MBR (partially compliant), and red for packets exceeding the MBR (non-compliant). (3) **Policing:** Red packets are dropped by the policer, while green and yellow packets are allowed to proceed. Yellow packets are treated with lower priority than green packets in scheduling in the QoS shaper. (4) **Traffic shaping:** The QoS shaper uses a token bucket to smooth traffic bursts and ensures that the outgoing traffic flow complies with desired flow bandwidth contracts, which is subject to the session's aggregate MBR (AMBR).

For non-QoS flows, packets bypass the trTCM and are directed to a separate shaper for non-QoS flows. Although these packets do not receive differentiated (preferntial) treatment, the shaper still applies a token bucket to regulate the flow rate in accordance with the session's AMBR to ensure fairness. To enforce the correct rate and achieve fairness, the token update rate is computed separately

[1]We set `sched_latency_thresh` to 50ms, which delivers optimal performance in our testing.

for QoS and non-QoS flows. For QoS flows, the token update rate R_{QoS} is defined as:

$$R_{\text{QoS}} = \min\left\{\text{GBR} + \frac{\text{AMBR} - \text{GBR}}{2}, \text{MBR}\right\}. \quad (1)$$

For non-QoS flows, the token update rate $R_{\text{non-QoS}}$ is the residual rate:

$$R_{\text{non-QoS}} = \text{AMBR} - R_{\text{QoS}}. \quad (2)$$

This ensures that compliant QoS flows are prioritized, non-compliant flows are penalized, and best-effort traffic is fairly shaped, while maintaining adherence to the session's flow bandwidth contracts (*i.e.,* AMBR).

3.5 L²5GC+ Network Slicing

L²5GC+ incorporates the 3GPP-compliant network slicing directly using free5GC's implementation[2]. The NSSF in the control plane (Fig. 4) is responsible for assigning UEs to the appropriate network slice based on their subscription profiles, service requirements, or other operator-specific criteria. Upon receiving a UE request, the AMF determines the slice by querying the NSSF for the mapping between the S-NSSAI and the slice. Based on the result, the AMF dispatches the request to slice-specific instances of the control plane NFs.

Resource multiplexing across 5GC slices: Beyond enabling service differentiation, L²5GC+ also enables resource multiplexing across slices[3] to reduce polling cost. Instances of non-security-critical NFs, such as the AMF, and SMF can be directly shared across slices to amortize the CPU cost of busy polling in the underlying shared memory processing. When a shared instance becomes saturated, as detected by our dynamic rate limiter (§3.3), L²5GC+ triggers the scaling of additional instances. Specifically, when the measured goroutine scheduling latency (EWMA_t)) exceeds the pre-configured threshold (`sched_latency_thresh`), while the number of in-progress ($\text{E}_{\text{current}}$) is also higher than the concurrency limit of the instance (E_{max}), it indicates the overload of current instance. At this point, L²5GC+ instantiates additional instances. L²5GC+ also coordinates with gNB to re-balance the load when the AMF instances are scaled up.

This coordination follows the guidance of 3GPP TS 23.501 [9], where AMF instances share the relative capacity (referred to as "weight factor" by 3GPP) with the gNB via NGAP (Next Generation Application Protocol) messages. While 3GPP leaves the exact form of the weight factor open, L²5GC+ defines it as the ratio $E_{\text{current}}/E_{\text{max}}$ to accurately reflect the current load of an AMF instance. The gNB uses this information to distribute UE requests proportionally across available AMF instances, ensuring load balancing and avoiding overload. This design allows L²5GC+ to minimize the polling cost in shared NFs and exploits the elasticity of a decoupled 5GC. A comprehensive evaluation of this slicing-based multiplexing strategy is part of our ongoing work.

[2]https://github.com/free5gc/nssf.git

[3]To clarify, we focus on the slicing of the core network, rather than end-to-end network slicing which additionally includes RAN and backhaul. However, the discussion is generally applicable as part of an end-to-end network slicing design.

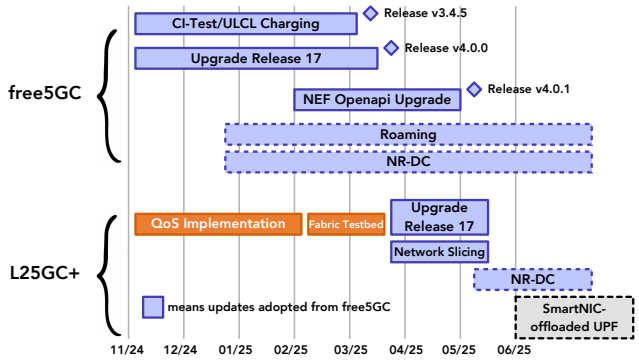

Figure 8: Roadmap of L²5GC+. L²5GC+ keeps adopting updates from free5GC [4].

3.6 Roadmap of L²5GC+

As shown in the Fig. 8, the development of L²5GC+ closely aligns with the free5GC's releases. We continuously adapt free5GC's latest features, such as 3GPP Release 17 compliance, network slicing to make sure L²5GC+ remains up-to-date and feature-rich. The development of X-IO [30] enables the rapid evolution of L²5GC+ by seamlessly incorporating updates from free5GC.

Following the upcoming official release of NR-DC in free5GC, we plan to release NR-DC in L²5GC+ as well. The NR-DC extension improves both throughput and mobility resilience by allowing User Equipments (UEs) to simultaneously connect to multiple base stations (gNBs in 5G), facilitating seamless handovers under high mobility scenarios. Looking forward, our next milestone is the development of a production-ready, SmartNIC-offloaded UPF to enhance the user plane performance of L²5GC+.

4 Evaluation

4.1 Microbenchmarking the Rate Limiter

Concurrent event generation: We select the PDU session establishment procedure to evaluate the effect of the rate limiter, as this UE task places the heaviest load on the control plane. To apply maximum stress to the core network, we generate concurrent PDU session establishment events. We first register a batch of 32 UEs with the 5GC. Once all these registrations complete, we instruct all of these UEs to simultaneously issue *PDU Session Establishment Requests* to the core network. This ensures that multiple *PDU Session Establishment Requests* arrive at the AMF within a short time interval. After the current batch is completed, we issue another batch of 32 *PDU Session Establishment Requests*. We developed a customized UE/RAN simulation script to achieve precise control over the timing for generating the concurrent PDU session establishment events.

Analysis: To analyze the change on the completion time of PDU session establishment events as the rate limiter adjusts the E_{max} (shown in Fig. 9 (a)), we order each event based on its issue time and collect the corresponding completion time in Fig. 9 (b). Note that the *x-axis* in Fig. 9 (a) shows the absolute time in seconds, while the *x-axis* in Fig. 9 (b) shows the relative order between distinct PDU session establishment events, and doesn't reflect the absolute time.

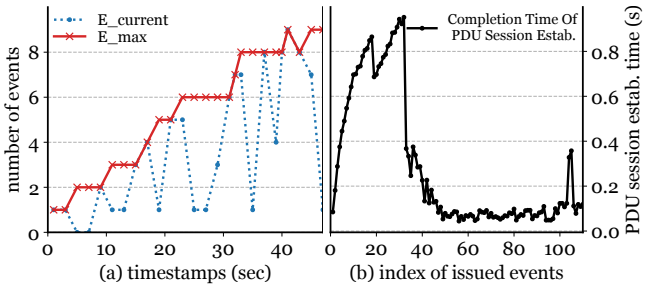

Figure 9: Microbenchmark results on rate limiter: (a) The rate limiter adapts the number of goroutines (E_{max}) to the actual in-progress events ($E_{current}$); (b) The completion time of PDU session establishment reduces as the E_{max} scales up.

In Fig. 9 (a), the rate limiter adjusts the E_{max} following Algorithm 1, every 500 milliseconds.

As shown in Fig. 9 (b), the first batch (1-32) of PDU session establishment events experience a sharp increase in latency: the completion time increases from 84 milliseconds for the first event to 952 milliseconds by the 32nd event. This is because the maximum number of concurrent events E_{max} was initially limited to 1, resulting in constrained processing capacity and increased queue delay.

This mismatch between E_{max} and $E_{current}$ triggers the adjustments in the rate limiter, as described in Line#6-9 in Algorithm 1. Since the E_{max} remains below 6 during the first 23 seconds (see Fig. 9 (a)), the scheduler latency remains low, repeatedly activating Line#8-9 in Algorithm 1 to incrementally increase E_{max}, *i.e.,* more goroutines are spawned to process the events. Consequently, the completion time is reduced to approximately 100 milliseconds after processing around 50 events.

Eventually, E_{max} stabilizes around 8, after multiple rounds of adjustment, which aligns with our offline profiling in Fig. 3, *i.e.,* optimal latency performance occurs when the number of goroutines is limited to a range between 8 and 16.

Table 1 compares the average completion time of 32 concurrent PDU session establishment events. With the rate limiter enabled, L^{2}5GC+ achieves a latency reduction of 1.5× compared to L^{2}5GC+ without the rate limiter and 1.96× compared to free5GC. The unbounded goroutine creation in L^{2}5GC+ without the rate limiter leads to excessive context switching and scheduling overhead, while free5GC incurs additional latency from kernel-based networking.

Table 1: Average completion time of 32 concurrent PDU session establishment event.

Platforms	**L^{2}5GC+ w/ rate limiter**	**L^{2}5GC+ w/o rate limiter**	**free5GC [4]**
	0.23 second	0.35 second	0.45 second

4.2 Impact of Fairness in QoS Support

We focus on verifying the fairness provided by the UPF in L^{2}5GC+ to enforce fair sharing of excess bandwidth in the user plane between a QoS flow and a non-QoS flow. The test setup involves two concurrent *UDP* flows, generated by two pairs of iperf clients/servers running on the data network and the UE/RAN simulator. We evaluate two cases with distinct configurations: (1) Case 1 uses a GBR of 2 Mbps and the MBR of 5 Mbps for the QoS flow; (2) Case 2 uses the GBR of 4 Mbps and the MBR of 7 Mbps for the QoS flow. In both cases, the AMBR of 10 Mbps is set for the overall session, limiting the total throughput that flows in the same session can utilize. Both the QoS flow and the non-QoS flow attempt to transmit at 10 Mbps.

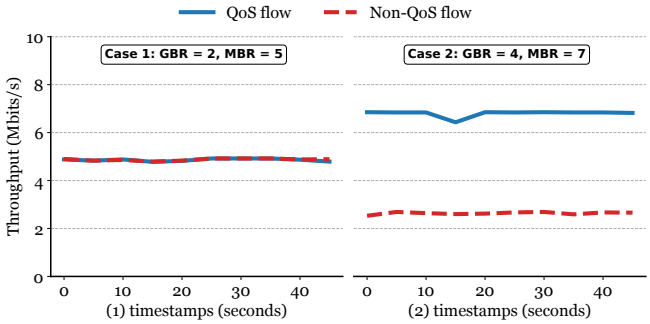

Figure 10: L^{2}5GC+ provides fairness in sharing the *excess bandwidth* between QoS and non-QoS flows: In Case 1, QoS flow receives 5 Mbps (2 Mbps guaranteed + 3 Mbps excess) and non-QoS flow receives 5 Mbps (entirely from excess). In Case 2, QoS flow receives 7 Mbps (4 Mbps guaranteed + 3 Mbps excess) and non-QoS flow receives 3 Mbps (entirely from excess).

Based on the design in §3.4, the token update rate for the QoS flow in Case 1 is calculated as:

$$R_{\text{QoS}} = \min\left(2\text{Mbps} + \frac{10\text{Mbps} - 2\text{Mbps}}{2}, 5\text{Mbps}\right) = 5\text{Mbps} \quad (3)$$

The token update rate for the non-QoS flow in Case 1 is calculated as:

$$R_{\text{non-QoS}} = \text{AMBR} - R_{\text{QoS}} = 10\text{Mbps} - 5\text{Mbps} = 5\text{Mbps} \quad (4)$$

Similarly, the token update rates for the QoS flow and non-QoS flow in Case 2 are calculated as 7Mbps and 3Mbps, respectively.

Fairness analysis of Case 1: Given the parameters (GBR = 2 Mbps, MBR = 5 Mbps, and AMBR = 10 Mbps), the excess bandwidth available for allocation is AMBR − GBR = 8Mbps. Ideally, the QoS flow and non-QoS flow receive half of this excess (8/2 = 4Mbps), resulting in QoS flow get GBR + 4Mbps = 6Mbps in total and non-QoS flow get 4Mbps. However, since the QoS flow is capped by its MBR at 5 Mbps, it cannot utilize the full 6 Mbps. The unused 1 Mbps is thus reallocated to the non-QoS flow, which increases its share to 5 Mbps. In total, QoS flow receives 5 Mbps (2 Mbps guaranteed + 3 Mbps excess) and non-QoS flow receives 5 Mbps (entirely from excess). This allocation achieves max-min fairness in sharing of the excess bandwidth.

Fairness analysis of Case 2: Given the parameters (GBR = 4 Mbps, MBR = 7 Mbps, and AMBR = 10 Mbps), the excess bandwidth available for allocation is AMBR − GBR = 6Mbps. Ideally, the QoS flow and non-QoS flow receive half of this excess (6/2 = 3Mbps), resulting in QoS flow get GBR + 3Mbps = 7Mbps in total (which doesn't violate MBR) and non-QoS flow get 3Mbps. This allocation also achieves fairness in sharing the excess bandwidth.

Both observations are consistent with the throughput measurements of QoS and non-QoS flows in L^{2}5GC+, as shown in Fig. 10 (1) and (2). The session's AMBR limits the total available bandwidth to 10 Mbps and the UPF in L^{2}5GC+ fairly allocates excess bandwidth to the non-QoS flows. Throughput achieved by the QoS flow complies with both GBR and MBR for the flow. Both flows are shaped to remain within their session constraints without causing any starvation.

4.3 Holistic System Evaluation on NSF Fabric Testbed [11]

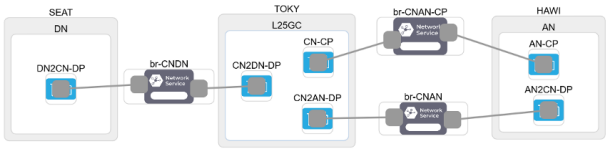

Figure 11: The topology setup on NSF Fabric.

Fabric testbed setup: We deployed a geographically distributed testbed on the NSF Fabric testbed. As shown in Fig. 11, the testbed spans three nodes located across multiple Fabric sites, including Tokyo, Hawaii, and Seattle. Each node is allocated with 16 CPU cores and 64GB RAM. The Tokyo node hosts the core network. We deployed the control plane and user plane on the same node. The Hawaii node runs the simulated UEs and RAN to generate user events and data traffic. We use the UE & RAN simulator from L^{2}5GC [14]. The Seattle node is used to deploy the data network. This distributed deployment helps to having realistic propagation delays to emulate a large-scale WAN-like environment.

We quantify the latency reduction and scalability improvement offered by L^{2}5GC+'s shared memory SBI compared to the kernel-based SBI in free5GC, particularly in handling concurrent user sessions (from 4 to 64). We measure the end-to-end latency for key events in the 5G control plane, including UE registration, PDU session establishment, and paging (transitioning UEs from idle to active state).

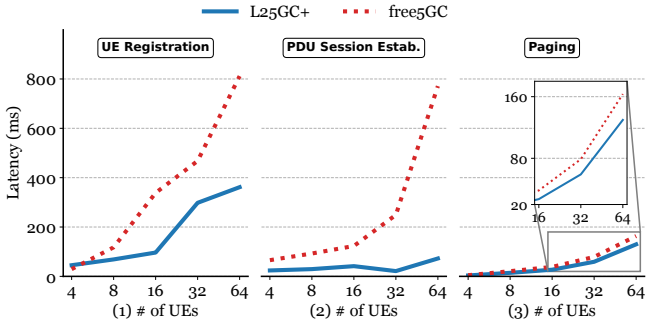

Figure 12: End-to-end latency of various control plane events on NSF Fabric testbed [11].

Analysis: As shown in Fig. 12, L^{2}5GC+ consistently outperforms free5GC for all measured events, maintaining lower latency as the number of concurrent UEs increases. Specifically, L^{2}5GC+ reduces UE registration latency by up to 3.5×, PDU session establishment latency by up to 10×, and paging latency by 1.3×. These results demonstrate the benefit of L^{2}5GC+'s shared memory SBI and the feasibility of NSF Fabric as a testbed for a wide-area deployment.

5 Related Work

Open-source 5GC Implementations. Popular open-source 5GC offerings are free5GC [4], OAI-core [5], Open5GS [6], and SD-core [8]. Table 2 compares L^{2}5GC+ with other open-source 5GCs on important features. Among them, L^{2}5GC+ maintains close alignment with free5GC, actively integrating its latest features, while introducing a range of advanced capabilities beyond existing offerings, such as fairness-enhanced QoS support in the user plane, dynamic rate limiter for control plane NFs, and low-latency shared-memory-based SBI. These enhancements make L^{2}5GC+ more feature-rich than other open-source 5GCs. The full-fledged network slicing support and NR-DC support make L^{2}5GC+ viable for a broad range of 5G use cases, which require differentiated service treatment and better mobility resilience.

Table 2: Comparison of Existing Open-source 5GC Implementations.

Features	Network Slicing	NR-DC Support	*Fair* QoS Support	Control Plane Rate Limiter	Low-latency SBI
L^{2}5GC+	✓	✓	✓	✓	✓
free5GC [4]	✓	✓	✗	✗	✗
OAI-core [5]	✓	✗	✗	✗	✗
Open5GS [6]	✓	✗	✗	✗	✗
SD-core [8]	✓	✗	✗	✗	✗

User plane optimizations. As 5G aims to deliver low-latency Internet access to end users, a significant body of work has focused on optimizing the UPF. [28] implements the UPF using eBPF/XDP in the Linux kernel, however, the performance of eBPF/XDP is significantly inferior to DPDK under high load [31]. Prior efforts such as [20] and [36] propose offloading the UPF to P4 switches to benefit programmable data plane acceleration. However, these switch-based approaches faces practical limitations due to limited buffering capacity of P4 switches, which can worsen the packet loss and retransmissions, hurting user plane performance. Instead, Synergy [27], Hybrid-UPF [34], and AccelUPF [12] propose using P4-capable SmartNICs as an alternative offloading target. SmartNICs offer a more viable option by combining programmable data plane acceleration with deeper buffering, making them better suited for production-grade UPF deployments.

Control plane optimizations. In addition to user plane, substantial work has focused on reducing control plane latency, especially following the softwarization of cellular cores beginning with 4G EPC [18, 23–25, 29, 32]. This trend has continued into the 5G era, where the decoupled architecture of the 5GC control plane further amplified the need for low-latency control plane operations.

Several efforts have attempted to replace or improve the 3GPP SBI, *e.g.,* [15] uses gRPC to replace the default HTTP/REST APIs, but still incurring expensive costs related to serialization and kernel sockets. L^{2}5GC [14] takes a more advanced approach by replacing the kernel-based SBI with low-latency shared memory communication. However, its brute-force implementation exposes shared

memory primitives directly to Golang-based NFs, violating 3GPP compliance and failing to scale across multiple user sessions. This limitation is addressed in L^{2}5GC+ through the introduction of X-IO [30].

Other efforts have explored adjacent aspects of control plane latency. Neutrino [10] reduces latency between the RAN and the core by minimizing the serialization overheads and replicating user state across broader geographical area to accelerate handovers. This could be a good complimentary to L^{2}5GC+, which targets latency reduction within the 5GC. FlexCore [33] implements an XDP-based SCTP load balancer between RAN and AMF, which could complement to L^{2}5GC+'s rate limiter by improving load distribution across AMF instances to reduce thrashing. CoreKube[17] consolidates control plane NFs (*e.g.,* AMF, SMF) into a single stateless worker and relies on external storage function (UDSF) for state management. However, this model introduces extra state synchronization overheads and undermines the decoupled nature of 5GC, making it harder to independently scale NFs.

Characterization on cellular cores. There has been work on characterizing the behavior of cellular cores. Prior efforts includes end-to-end performance measurements on operational 5G systems using both Non-Standalone (NSA) [37] and Standalone (SA) architectures [26, 38]. Other studies, such as [21] and [16], characterized and modeled the control plane traffic within cellular cores, which provides synthetic workloads that are useful for testing and evaluating L^{2}5GC+ when real user traffic is unavailable due to regulatory constraints.

6 Conclusion

We presented L^{2}5GC+, a high-performance 3GPP-compliant 5GC designed especially to deliver low-latency control plane operations. L^{2}5GC+ introduces the X-IO interface to bridge the semantic gap between OpenNetVM's shared memory operation and Golang-based 3GPP NFs. L^{2}5GC+ incorporates a range of novel features, including a dynamic rate limiter at the AMF to prevent goroutine thrashing under load, a fairness-enhanced QoS capability in the UPF to balance bandwidth between QoS and non-QoS flows, and provides full support for network slicing.

Future work includes the release of NR-DC support and the planned public deployment on NSF Fabric testbed. This deployment offers a promising opportunity for the research community to perform detailed control plane analysis under realistic conditions. Such fine-grained, internal analysis of 5GC control plane at scale remains limited due to the black-box nature of commercial 5GC deployment, where core internals are not exposed for analysis. Instead, L^{2}5GC+ provides an open-source, high-performance, and comprehensive 5GC solution, enabling visibility into NF interactions and bottlenecks.

Acknowledgments

We thank the U.S. NSF for their generous support with JUNO-3 grant 2210379. This work is also supported in part by the National Science and Technology Council of Taiwan under grant numbers 114-2218-E-A49-018 and 114-2218-E-A49-017.

References

[1] 2023. 3GPP TS29.500: Technical Realization of Service Based Architecture. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3338. [ONLINE].
[2] 2025. 3GPP Release 17. https://www.3gpp.org/specifications-technologies/releases/release-17. [ONLINE].
[3] 2025. Data Plane Development Kit. https://www.dpdk.org/. [ONLINE].
[4] 2025. free5GC. https://github.com/free5gc/free5gc. [ONLINE].
[5] 2025. OAI-5G Core Network. https://openairinterface.org/oai-5g-core-network-project/. [ONLINE].
[6] 2025. Open5GS. https://open5gs.org/. [ONLINE].
[7] 2025. Procedures for the 5G System (5GS) (3GPP TS 23.502 version 17.4.0 Release 17). https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/17.04.00_60/ts_123502v170400p.pdf. [ONLINE].
[8] 2025. SD-Core. https://opennetworking.org/sd-core/. [ONLINE].
[9] 2025. System architecture for the 5G System (5GS) (3GPP TS 23.501 version 17.5.0 Release 17). https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/17.05.00_60/ts_123501v170500p.pdf. [ONLINE].
[10] Mukhtiar Ahmad, Syed Usman Jafri, Azam Ikram, Wasiq Noor Ahmad Qasmi, Muhammad Ali Nawazish, Zartash Afzal Uzmi, and Zafar Ayyub Qazi. 2020. A Low Latency and Consistent Cellular Control Plane. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 648–661. doi:10.1145/3387514.3406218
[11] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. 2019. FABRIC: A National-Scale Programmable Experimental Network Infrastructure. *IEEE Internet Computing* 23, 6 (2019), 38–47. doi:10.1109/MIC.2019.2958545
[12] Abhik Bose, Shailendra Kirtikar, Shivaji Chirumamilla, Rinku Shah, and Mythili Vutukuru. 2022. AccelUPF: accelerating the 5G user plane using programmable hardware. In *Proceedings of the Symposium on SDN Research* (Virtual Event) *(SOSR '22)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3563647.3563651
[13] Juha Heinanen and Roch Guerin. 1999. RFC2698: A two rate three color marker.
[14] Vivek Jain, Hao-Tse Chu, Shixiong Qi, Chia-An Lee, Hung-Cheng Chang, Cheng-Ying Hsieh, K. K. Ramakrishnan, and Jyh-Cheng Chen. 2022. L25GC: A Low Latency 5G Core Network based on High-performance NFV Platforms. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 143–157.
[15] Tulja Vamshi Kiran Buyakar, Harsh Agarwal, Bheemarjuna Reddy Tamma, and Antony A. Franklin. 2019. Prototyping and Load Balancing the Service Based Architecture of 5G Core Using NFV. In *2019 IEEE Conference on Network Softwarization (NetSoft)*. 228–232. doi:10.1109/NETSOFT.2019.8806648
[16] Z. Jonny Kong, Nathan Hu, Y. Charlie Hu, Jiayi Meng, and Yaron Koral. 2024. High-Fidelity Cellular Network Control-Plane Traffic Generation without Domain Knowledge. In *Proceedings of the 2024 ACM on Internet Measurement Conference* (Madrid, Spain) *(IMC '24)*. Association for Computing Machinery, New York, NY, USA, 530–544. doi:10.1145/3646547.3688422
[17] Jon Larrea, Andrew E. Ferguson, and Mahesh K. Marina. 2023. CoreKube: An Efficient, Autoscaling and Resilient Mobile Core System. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking* (Madrid, Spain) *(ACM MobiCom '23)*. Association for Computing Machinery, New York, NY, USA, Article 25, 15 pages. doi:10.1145/3570361.3592522
[18] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. 2017. A Control-plane Perspective on Reducing Data Access Latency in LTE Networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. 56–69.
[19] Yu-Sheng Liu, Shixiong Qi, Po-Yi Lin, Han-Sing Tsai, K. K. Ramakrishnan, and Jyh-Cheng Chen. 2023. L25GC+: An Improved, 3GPP-compliant 5G Core for Low-latency Control Plane Operations. In *2023 IEEE 12th International Conference on Cloud Networking (CloudNet)*. 203–211. doi:10.1109/CloudNet59005.2023.10490024
[20] Robert MacDavid, Carmelo Cascone, Pingping Lin, Badhrinath Padmanabhan, Ajay ThakuR, Larry Peterson, Jennifer Rexford, and Oguz Sunay. 2021. A P4-Based 5G User Plane Function. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)* (Virtual Event, USA) *(SOSR '21)*. Association for Computing Machinery, New York, NY, USA, 162–168. doi:10.1145/3482898.3483358
[21] Jiayi Meng, Jingqi Huang, Y Charlie Hu, Yaron Koral, Xiaojun Lin, Muhammad Shahbaz, and Abhigyan Sharma. 2022. Characterizing and Modeling Control-Plane Traffic for Mobile Core Network. *arXiv preprint arXiv:2212.13248* (2022).
[22] Jeffrey C. Mogul and K. K. Ramakrishnan. 1997. Eliminating receive livelock in an interrupt-driven kernel. *ACM Trans. Comput. Syst.* 15, 3 (Aug. 1997), 217–252. doi:10.1145/263326.263335
[23] Ali Mohammadkhan, K. K. Ramakrishnan, and Vivek A. Jain. 2020. CleanG—Improving the Architecture and Protocols for Future Cellular Networks With NFV. *IEEE/ACM Transactions on Networking* 28, 6 (2020), 2559–2572. doi:10.1109/TNET.2020.3015946
[24] Mehrdad Moradi, Yikai Lin, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2018. SoftBox: A Customizable, Low-Latency, and Scalable 5G Core

Network Architecture. *IEEE Journal on Selected Areas in Communications* 36, 3 (2018), 438–456. doi:10.1109/JSAC.2018.2815429

[25] Mehrdad Moradi, Karthikeyan Sundaresan, Eugene Chai, Sampath Rangarajan, and Z. Morley Mao. 2018. SkyCore: Moving Core to the Edge for Untethered and Reliable UAV-based LTE Networks. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (New Delhi, India) *(MobiCom '18)*. Association for Computing Machinery, New York, NY, USA, 35–49. doi:10.1145/3241539.3241549

[26] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, Feng Qian, and Zhi-Li Zhang. 2021. A variegated look at 5G in the wild: performance, power, and QoE implications. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 610–625. doi:10.1145/3452296.3472923

[27] Sourav Panda, K. K. Ramakrishnan, and Laxmi N. Bhuyan. 2022. Synergy: A SmartNIC Accelerated 5G Dataplane and Monitor for Mobility Prediction. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. 1–12. doi:10.1109/ICNP55882.2022.9940261

[28] Federico Parola, Fulvio Risso, and Sebastiano Miano. 2021. Providing Telco-oriented Network Services with eBPF: the Case for a 5G Mobile Gateway. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. 221–225. doi:10.1109/NetSoft51509.2021.9492571

[29] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. 2017. A High Performance Packet Core for Next Generation Cellular Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 348–361. doi:10.1145/3098822.3098848

[30] Shixiong Qi, Han-Sing Tsai, Yu-Sheng Liu, KK Ramakrishnan, and Jyh-Cheng Chen. 2023. X-IO: A High-performance Unified I/O Interface using Lock-free Shared Memory Processing. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE, 107–115.

[31] Shixiong Qi, Ziteng Zeng, Leslie Monis, and K. K. Ramakrishnan. 2023. MiddleNet: A Unified, High-Performance NFV and Middlebox Framework with eBPF and DPDK. *IEEE Transactions on Network and Service Management* (2023).

[32] Rinku Shah, Vikas Kumar, Mythili Vutukuru, and Purushottam Kulkarni. 2020. TurboEPC: Leveraging Dataplane Programmability to Accelerate the Mobile Packet Core. In *Proceedings of the Symposium on SDN Research* (San Jose, CA, USA) *(SOSR '20)*. Association for Computing Machinery, New York, NY, USA, 83–95. doi:10.1145/3373360.3380839

[33] Bhavishya Sharma, Shwetha Vittal, and A Antony Franklin. 2023. FlexCore: Leveraging XDP-SCTP for Scalable and Resilient Network Slice Service in Future 5G Core. In *Proceedings of the 7th Asia-Pacific Workshop on Networking* (Hong Kong, China) *(APNet '23)*. Association for Computing Machinery, New York, NY, USA, 61–66. doi:10.1145/3600061.3600073

[34] Suneet Kumar Singh, Christian Esteve Rothenberg, Jonatan Langlet, Andreas Kassler, Péter Vörös, Sándor Laki, and Gergely Pongrácz. 2023. Hybrid P4 Programmable Pipelines for 5G gNodeB and User Plane Functions. *IEEE Transactions on Mobile Computing* 22, 12 (2023), 6921–6937. doi:10.1109/TMC.2022.3201512

[35] Alonza Tu. 2025. Introducing NR-DC: Dual Connectivity for Next-Gen 5G Capabilities. https://free5gc.org/blog/20250219/20250219/. [online].

[36] Zhixin Wen and Guanhua Yan. 2024. HiP4-UPF: Towards High-Performance Comprehensive 5G User Plane Function on P4 Programmable Switches. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 303–320. https://www.usenix.org/conference/atc24/presentation/wen

[37] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. 2020. Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 479–494. doi:10.1145/3387514.3405882

[38] Xinjie Yuan, Mingzhou Wu, Zhi Wang, Yifei Zhu, Ming Ma, Junjian Guo, Zhi-Li Zhang, and Wenwu Zhu. 2022. Understanding 5G performance for real-world services: a content provider's perspective. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 101–113. doi:10.1145/3544216.3544219

[39] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K. K. Ramakrishnan, and Timothy Wood. 2016. OpenNetVM: A Platform for High Performance Network Service Chains. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*. 26–31.