L²5GC+: An Improved, 3GPP-compliant 5G Core for Low-latency Control Plane Operations

Yu-Sheng Liu†, Shixiong Qi*, Po-Yi Lin†, Han-Sing Tsai†, K. K. Ramakrishnan*, Jyh-Cheng Chen†
†National Yang Ming Chiao Tung University, *University of California, Riverside

***Abstract*—While 5G offers fast access networks and a high-performance data plane, the control plane in 5G core (5GC) still presents challenges due to inefficiencies in handling control plane operations (including session establishment, handovers and idle-to-active state-transitions) of 5G User Equipment (UE). The Service-based Interface (SBI) used for communication between 5G control plane functions introduces substantial overheads that impact latency. Typical 5GCs are supported in the cloud on containers, to support the disaggregated Control and User Plane Separation (CUPS) framework of 3GPP. L²5GC is a state-of-the-art 5G control plane design utilizing shared memory processing to reduce the control plane latency. However, L²5GC has limitations in supporting multiple user sessions and has programming language incompatibilities with 5GC implementations, *e.g.,* free5GC, using modern languages such as GoLang. To address these challenges, we develop L²5GC+, a significant enhancement to L²5GC. L²5GC+ re-designs the shared-memory-based networking stack to support synchronous I/O between control plane functions. L²5GC+ distinguishes different user sessions and maintains strict 3GPP compliance. L²5GC+ also offers seamless integration with existing 5GC microservice implementations through equivalent SBI APIs, reducing code refactoring and porting efforts. By leveraging shared memory I/O and overcoming L²5GC's limitations, L²5GC+ provides an improved solution to optimize the 5G control plane, enhancing latency, scalability, and overall user experience. We demonstrate the improved performance of L²5GC+ on a 5G testbed with commercial basestations and multiple UEs.**

***Index Terms*—5G, control plane, low latency, shared memory**

I. Introduction

The demand for 5G and beyond technologies is being driven by the emergence of applications like the Internet of Things (IoT) and connected vehicles, which rely heavily on cellular networks for ubiquitous access and low latency. Further, the deployment of 5G, especially the 5G Core (5GC) and (soon) the Radio Access Network (RAN), in cloud infrastructure has been instrumental in its widespread implementation as well as its scalability. Cloud-based 5G core networks allow for flexible and efficient resource provisioning, using seamless scaling to accommodate the diverse demands of connected User Equipment (UE) and applications.

For a seamless end-to-end low-latency user experience, both the radio access and as well as the core components of 5G cellular networks have to improve. Advancements in radio access technology, such as millimeter wave, have reduced access network latency to approximately the order of a few milliseconds (possibly 1 ms [1]). The recent effort shows that electronic mmWave beam alignment and link acquisition can be completed within 1-10 ms, allowing a UE's connection establishment with the gNodeB to be completed quickly [2]. In addition, the advent of disaggregated 5GC has spurred significant efforts to re-architect the data plane to meet the stringent requirements of performance and scalability. A variety of optimizations have been explored to enhance the 5GC data plane, including DPDK [3], eBPF [4], SmartNIC [5], and offloading to hardware switches using P4 [6].

However, the *control plane* still contributes substantially to the overall high latency observed in the 5GC. One major contributor is the potential for increased mobility handovers, driven by the wide adoption of millimeter-wave cells [7], which have smaller cell sizes as well as limited coverage, leading to more frequent handover events. These handovers have to be handled by the 5G control plane. Additionally, with the need to conserve energy in batteries on UEs like mobile phones as well as IoT devices, there will likely be much more idle-active transitions among the UEs. The proliferation of 5G UEs (*e.g.,* mobile phones, IoT devices, autonomous vehicles) further increases the load on the 5G control plane. The completion times of control plane events, for instance, a handover process taking 1.9 seconds [8], directly influence the delay and packet loss encountered by the data packets transmitted to an end-user device.

The disaggregated 5GC architecture represents a transformative approach to implementing 5G core networks, moving away from the monolithic, tightly integrated network elements of previous approaches to build the cellular core to a flexible and scalable approach based on microservices. The various components of the 5GC are implemented as software-based Network Functions (NFs), interconnected as a chain to accomplish the required functionality. Each NF is implemented as an individual microservice, focusing on a specific task, such as Access and Mobility Management, Session Management, Authentication, *etc.* The use of microservices enables fine-grained control and allows for rapid deployment and upgrades of individual components without affecting the entire 5G system. Additionally, this disaggregated approach enables resource optimization, since NFs can be dynamically scaled based on traffic demand.

A crucial implementation feature of the 5G control plane is the Service-based Interface (SBI) recommended by 3GPP, which has been the de-facto communication standard used for communication between disaggregated 5G control plane NFs. However, the use of SBI introduces a number of overheads, such as data copies, protocol processing, and user-kernel space boundary crossings [3]. These overheads can result in increased latency, apart from the penalty due to the traditional

cellular control plane core procedures (details in §IV).

To address the challenge of achieving low-latency communication in the 5GC control plane, efforts have been made to explore innovative approaches that harness high-performance shared memory I/O (*i.e.,* data exchange) for information exchange between NFs implementing microservices in the 5GC. By leveraging shared memory processing, the 5G control plane can significantly reduce the delays caused by data copies and protocol processing, resulting in improved response times and a more seamless user experience. In [3], we discussed L^{2}5GC, a state-of-the-art 5G control plane design developed on top of OpenNetVM [9], a high-performance shared-memory NFV platform. L^{2}5GC utilizes shared memory processing among the 5G control plane components, which reduces the completion time of control plane events (*e.g.,* UE registration, handover, paging) by almost 50% on average [3].

However, L^{2}5GC's control plane only supports a limited number of user sessions due to a rather limited implementation of the shared memory I/O to replace the SBI.[1] L^{2}5GC chose to use raw shared memory I/O provided by DPDK, which operates *asynchronously* between caller (source) and callee (destination). For asynchronous data exchange, the caller typically continues with other tasks without being blocked and does not wait for a response from the callee. However, this is incompatible with the HTTP/REST-based SBI, which primarily operates *synchronously* between caller and callee, *i.e.,* the caller sends a request to the callee and waits until a response is returned.

The mismatch between L^{2}5GC's asynchronous shared memory I/O and synchronous SBI makes it hard to harmonize them, unfortunately increasing the complexity of code development and the difficulty of code maintenance and updates of L^{2}5GC. Further, L^{2}5GC's shared memory I/O only supports stateless processing. This lack of capability to preserve connection context makes L^{2}5GC's shared memory I/O connection-agnostic and not able to distinguish between different user sessions. The implementation complexity and the statelessness of L^{2}5GC's shared memory I/O eventually impede L^{2}5GC's ability to scale up, supporting multiple user sessions.

In addition to the mismatch between synchronous and asynchronous I/O, another challenge comes from programming language incompatibility. L^{2}5GC is adapted from our earlier work on a 3GPP-compliant 5GC implementation, free5GC [11]. For the purpose of functionality and development velocity, free5GC chose to use Golang, a high-level programming language, in its implementation. On the other hand, L^{2}5GC's asynchronous shared memory I/O is developed with the C-based DPDK libraries for high-performance networking. This leads to a need for substantial re-factoring of code when porting the 3GPP-compliant free5GC to L^{2}5GC to reduce the control plane latency.

We propose **L^{2}5GC+**, an enhancement to L^{2}5GC. **L^{2}5GC+** takes advantage of our newly designed shared memory I/O interface, X-IO [12], and tackles the pain points of L^{2}5GC we have outlined above, including limited user session support and the need for complex code refactoring when porting free5GC's (or other traditional SBI-based) control plane implementation, while retaining the performance benefits of L^{2}5GC's shared memory processing. To achieve this, we re-design the shared-memory-based networking stack in L^{2}5GC to support *synchronous* I/O between control plane NFs. This avoids heavy-weight kernel-based networking used in HTTP/REST-based SBI, while being strictly 3GPP-compliant. To support multiple user sessions simultaneously in shared memory processing, **L^{2}5GC+** introduces necessary connection establishment and teardown procedures. Important connection states, such as caller and callee ID[2] are kept in a state map maintained in **L^{2}5GC+**'s shared memory networking stack. This enables **L^{2}5GC+**'s shared memory I/O to be aware of distinct connections, on top of which **L^{2}5GC+** can distinguish different user sessions, unlike L^{2}5GC.

To speed up the development velocity when porting free5GC to **L^{2}5GC+**, we expose the equivalent SBI APIs from **L^{2}5GC+**'s networking stack. By leveraging the cross-language support offered by the CGo interface [13], we mitigate the programming language incompatibility between the lower-layer shared memory transport (developed with C-based DPDK libraries) and upper-layer Golang-based SBI APIs. This allows us to seamlessly replace the kernel-based SBI APIs for existing free5GC control plane NFs, while keeping the NF implementation *unchanged*, thus greatly reducing porting efforts.

To gain a solid understanding of how **L^{2}5GC+** actually performs, we evaluate **L^{2}5GC+** on a commercial testbed with an increasing number of UEs. We select several representative control plane events, including UE registration, PDU session establishment, to evaluate **L^{2}5GC+** against a popular 5GC implementation, free5GC [11], which uses kernel-based SBI in the control plane. Results demonstrate the performance improvement of **L^{2}5GC+**'s shared memory SBI, especially when there are multiple user sessions operating concurrently in the 5GC control plane.

II. Background and Related work

A. 5G core control plane

The 5G cellular network is typically divided into the RAN and the 5G core network. The RAN incorporates the wireless channel, cellular base stations, and the backhaul network, all working together to establish connections between UEs (*i.e.,* typically mobile client devices) and the 5GC. On the other hand, the 5GC plays a crucial role in connecting UEs to the Data Network (DN) to access internet services.

The 5GC is further split into the data plane and control plane. User Plane Function (UPF) is the key NF in the 5GC data plane, which interconnects the RAN and DN. Fig. 1 depicts the architecture of the 5GC control plane. Unlike previous generations of cellular core networks, the 5GC control

[1] Based on examining the source code [10] of L^{2}5GC at the latest commit hash `74cb035`.

[2] Similar to kernel-based TCP/IP stack, **L^{2}5GC+** uses IP and port numbers to differentiate between NFs using shared memory communication.

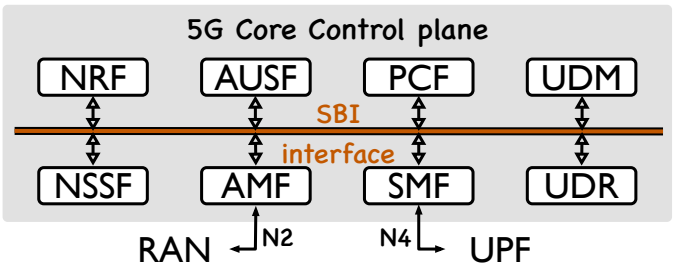


Fig. 1. The architecture of 5G core control plane

plane has undergone significant evolution, transitioning to disaggregated NFs. Several crucial control plane NFs play distinct roles. These include the Access and Mobility Function (AMF), Network Repository Function (NRF), Service Management Function (SMF), and Authentication Server Function (AUSF). This transformation facilitates the implementation of control plane NFs as cloud-native services. This shift towards a cloud-native, service-based approach enhances flexibility and scalability in the 5G Core, facilitating more efficient and agile network management. On the other hand, such disaggregated design requires networking to provide interconnectivity between NFs, forming a service-based architecture (SBA). These NFs offer their functionality through the 3GPP-compliant service-based interface (SBI), which essentially utilizes HTTP/REST APIs for seamless inter-service communication.

B. Related work

Optimizing 3GPP SBI in 5GC control plane: There has been a focus on how to reduce the latency of 3GPP SBI in the 5GC control plane. L^{2}5GC [3] is the state-of-the-art 5GC control plane optimization that seeks to use shared memory processing to reduce the control plane messaging latency incurred by kernel-based 3GPP SBI, which is commonly adopted in existing 5GC implementation, such as our earlier work free5GC [11]. Although L^{2}5GC achieves considerable latency reduction of various control plane events, its imperfect design of shared memory I/O, *e.g.,* lack of synchronous data exchange support, unawareness of connections, making it ill-suited for a 3GPP-compliant 5GC control plane and fail to scale up to multiple user sessions. Buyakar *et al.* [14] propose to replace the HTTP/REST APIs with gRPC to construct 3GPP SBI, since gRPC shows better scalability, in terms of CPU utilization and data transmission latency, compared to the HTTP/REST APIs when dealing with an increasing number of UEs. However, gRPC still suffers from kernel networking overhead as HTTP/REST-based SBI, making it less competent compared to **L^{2}5GC+**.

Optimizing other aspects of the cellular core control plane: Apart from optimization on 3GPP SBI, there are many other efforts on optimizing the cellular core control plane. Neutrino [15] is a 5GC control plane design that also seeks to reduce control plane latency. However, unlike **L^{2}5GC+** that focuses on reducing the messaging latency within the 5GC control plane, Neutrino attempts to reduce the messaging latency between the RAN and the 5GC control plane by minimizing the data serialization overhead, while being 3GPP-compliant. This could be a good complimentary to **L^{2}5GC+**.

CleanG [16] and DPCM [8] reduce the latency of 5GC control plane by redesigning control plane procedures. CleanG primarily focuses on creating a new control plane protocol that can simplify the control plane interactions in cellular networks, thus reducing latency [16]. Another approach, DPCM [8], reuses the UE-side state to skip unnecessary control plane procedures (*i.e.,* those used to generate the UE-side state which is already there). However, both of these proposals are not 3GPP-compliant, which makes them less complimentary to **L^{2}5GC+**, as our faith of **L^{2}5GC+** is to keep 3GPP compliance.

Besides latency optimization, [17] seeks to characterize and model the control plane traffic in cellular cores, which may facilitate the testing and evaluation of **L^{2}5GC+**'s control plane design when real traffic is not available due to regulatory compliance.

III. Design of L^{2}5GC+

We begin with an overview of the **L^{2}5GC+** and describe the key building blocks for developing a high-performance communication paradigm using shared-memory processing, while providing the necessary synchronous I/O primitives to replace the kernel-based 3GPP SBI. We then discuss in detail how to build an SBI on shared memory from the bottom-up. This includes asynchronous shared-memory processing over the DPDK, a POSIX-like synchronous I/O interface, and how we can use the POSIX-like APIs to build a shared-memory-based SBI. We then implement a seamless port of the 5GC control plane NFs from the baseline free5GC code-base to **L^{2}5GC+**.

We describe concurrent user session support in the **L^{2}5GC+**, including connection establishment, connection tear down, and user session management during data transfer between control plane NFs in **L^{2}5GC+**.

A. Overview of L^{2}5GC+

Fig. 2 depicts the architecture of **L^{2}5GC+**, which takes advantage of shared memory processing in userspace for data sharing between control plane NFs. This avoids expensive CPU data copy overheads, protocol processing, context switch, serialization, and deserialization, which are all incurred by the currently recommended 3GPP SBI. In the userspace of each worker node, **L^{2}5GC+** dedicates a shared memory pool and adopts an NF manager to support shared memory processing. Information exchange is performed by message descriptor delivery between NFs (§III-C). The per-node NF manager is responsible for managing shared memory (*e.g.,* initialization and removal) and interacts with the protocol stack to provide a "one-time", consolidated protocol processing when inter-node communication is needed. Our current implementation utilizes the kernel protocol stack for inter-node communication as it is robust and proven. However, high-performance inter-node transport protocols, such as RDMA, may be desirable and is the subject of our future work.

Each **L^{2}5GC+** NF uses our newly developed I/O stack [12] for shared memory communication between other NFs that

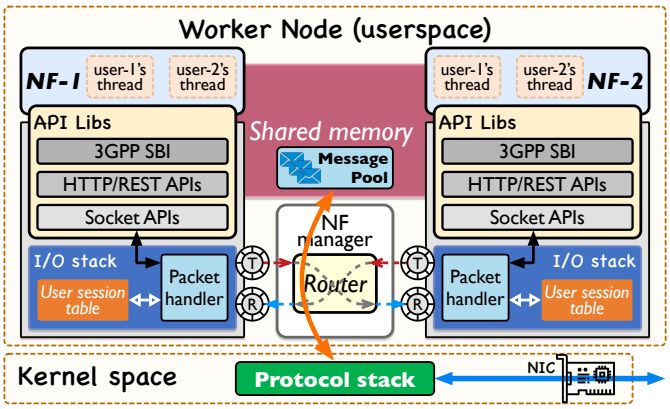


Fig. 2. An architectural overview of **L²5GC+**. Each NF can support multiple user sessions (each represented as a distinct thread) concurrently.

are located on the same node. The stack provides a shared memory I/O interface with a set of *asynchronous* communication primitives, utilizing DPDK [18]. A packet handler in the I/O stack deals with incoming/outgoing messages (essentially descriptor exchanges). **L²5GC+** also exploits the *lock-free communication* of X-IO [12] to avoid the need for any potential locks for multiple-producer, multiple-consumer communication (§III-C). Such a communication pattern commonly exists in 5GC control plane data exchanges.

Connection handling (*e.g.,* establishment/teardown) messages are processed by the packet handler for connection management tasks. These are important extensions in **L²5GC+** that help us to support scalable user sessions going beyond the capability of the previous L²5GC implementation. A local connection table in the I/O stack maintains the connection state. The connection related to a data message is identified by looking up the connection table based on the IP 4-tuple (source IP and port number, destination IP and port number). The packet handler directs the message to the right connection endpoint (*i.e.,* the corresponding user thread) in this I/O stack (details in §III-E).

The primitives exposed for *asynchronous* shared memory I/O by the I/O stack in L²5GC [3] are not 3GPP-compliant. Therefore, it requires extensive refactoring of the baseline free5GC [11] implementation. Thus, we seek to overcome this deficiency. **L²5GC+** introduces an API library (API lib for short), to provide the necessary *synchronous* I/O primitives, enabling the interface to be compliant with 3GPP.

As shown in Fig. 2, the top layer NF code performs several control plane tasks across the SBI, using *synchronous* I/O. This then interacts with the *asynchronous* I/O stack below using the API lib, which includes a socket interface to directly interact with the underlying I/O stack for shared memory communication. A set of HTTP/REST APIs are provided on top of the socket interface. **L²5GC+** utilizes these APIs to create a 3GPP-compliant SBI. **L²5GC+**'s SBI using shared memory has the same semantics as the 3GPP's SBI, just like free5GC [11], for easy portability to **L²5GC+**.

B. Shared memory management in L²5GC+

L²5GC+ depends on the NF manager to manage the shared memory pool. During the initialization of the **L²5GC+** environment, an NF manager is created on a designated worker node. The NF manager then creates a certain number of buffers within the shared memory pool to be utilized as shareable backends for exchanging control plane messages between **L²5GC+** NFs.

We extensively use DPDK's libraries [18] to implement shared memory management in **L²5GC+**. For lifecycle management (*i.e.,* creation/recycle/destroy) of the shared memory buffer, we utilize DPDK's Mempool Library [19]. To enforce access control of the shared memory pool and to prohibit unauthorized access, we leverage the security domain design that is widely adopted in DPDK-based shared memory frameworks [3], [20], [21], depending on DPDK's Environment Abstraction Layer [22] and multi-process support [23] to provide the necessary memory isolation.

C. Message descriptor delivery in L²5GC+

A *lock-free* descriptor delivery mechanism is the key element to derive the value of shared memory communication in **L²5GC+**. As shown in the overview figure (Fig. 2), each NF is assigned a pair of producer/consumer rings in its I/O stack. The producer/consumer rings of the NF are only shared with the NF manager, thus ensuring a strict single-producer, single-consumer communication pattern, avoiding the need for locks. On the other side, the NF manager forwards the descriptor (based on IP 4-tuples) between the I/O stacks of different NFs.

D. Building the SBI over shared memory: detailed design

We establish **L²5GC+**'s 3GPP-compliant SBI starting from the *asynchronous* shared memory I/O adopted by L²5GC, which is neither 3GPP-compliant nor scalable. We build an *asynchronous* shared memory I/O interface associated with a lock-free descriptor delivery mechanism into the I/O stack, thus offering these as raw I/O primitives to leverage shared memory processing.

We first introduce **L²5GC+**'s API libs that add synchronous I/O support on top of the asynchronous I/O stack. The API libs adopt a layered design: (1) the bottom-layer of the library provides a set of POSIX-like socket APIs that directly interact with **L²5GC+**'s I/O stack to leverage shared memory processing, while also providing basic synchronous I/O primitives; (2) the middle-layer library abstracts HTTP/REST APIs from the socket APIs, (3) these are then leveraged by the top-layer library to construct a 3GPP-compliant SBI.

This design choice was made primarily to facilitate ease of implementation and avoids re-implementing the entire stack — the implementation of upper-layer HTTP/REST APIs and 3GPP SBI can be ported from existing solutions (*e.g.,* free5GC) by simply replacing the lower-layer socket APIs, without being re-implemented from scratch.

Asynchronous shared memory I/O over DPDK: We construct the asynchronous shared memory I/O in **L²5GC+** (also in L²5GC) using DPDK's RTE RING [24] and Mempool APIs [19]. The basic I/O primitives that we use from DPDK to

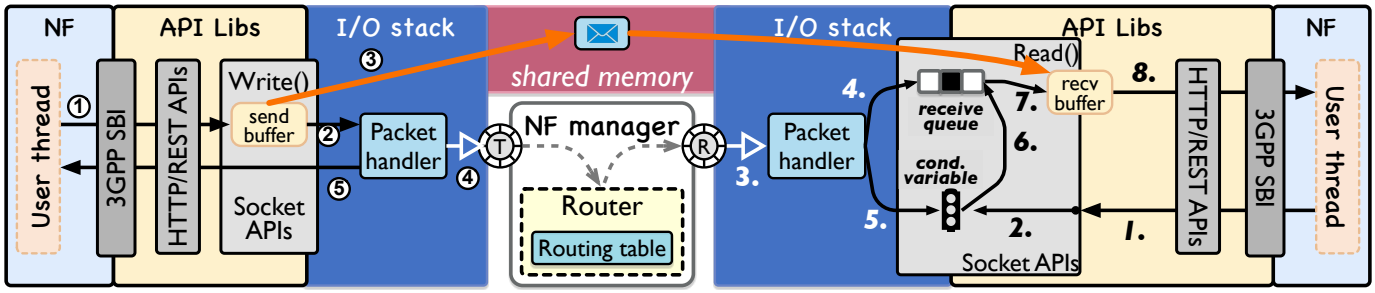


Fig. 3. Synchronous I/O primitives from **L²5GC+**'s socket APIs: Read() and Write(). Note that the circled number (*e.g.,* ①) represents the steps of Write() procedure (left half). Otherwise, it represents the steps of Read() procedure (right half).

enable asynchronous shared memory processing include *rte_mempool_get()*, *rte_mempool_put()*, *rte_ring_enqueue()*, and *rte_ring_dequeue()*.

rte_mempool_get() and *rte_mempool_put()* are obtained from DPDK's Mempool lib. We use *rte_mempool_get()* to retrieve an empty memory buffer from the shared memory pool and return a descriptor (pointing to the retrieved buffer) to the caller NF. *rte_mempool_put()*, on the other hand, is used for recycling the buffer back to the shared memory pool.

rte_ring_enqueue() and *rte_ring_dequeue()* are obtained from DPDK's RTE Ring lib. *rte_ring_enqueue()* is specifically used for enqueuing the descriptor into the producer (TX) ring, while *rte_ring_dequeue()* is used for retrieving the descriptor from the consumer (RX) ring.

The asynchronous access mainly comes from the **non-blocking** nature of *rte_ring_enqueue()* and *rte_ring_dequeue()* APIs. The call to these APIs immediately returns, leading to a mismatch in the synchronous communication required by the upper-layer SBI.

Synchronous shared memory I/O: Following the design of X-IO [12], we abstract the synchronous I/O primitives of **L²5GC+** into two socket APIs, Write() and Read(), maintaining strict alignment with the POSIX-like socket APIs. We further add synchronous access by enforcing a blocking call to Write() and Read(), *i.e.,* the caller of the Write() and Read() API is blocked until the requested I/O task is accomplished.

The synchronous nature of the Write() API is achieved by blocking the caller thread until the message is moved from the send buffer (provided by caller thread) into the shared memory buffer. Fig. 3 (left half) shows how the Write() API interacts with the I/O stack to accomplish the transmission of a message: ① The caller thread initiates the Write() call with a send buffer input; ② The Write() call passes the send buffer to the packet handler in the I/O stack; ③ the packet handler then copies the message from the send buffer to the shared memory buffer. Note that, beforehand, the packet handler obtains an empty memory buffer and associated descriptor using *rte_mempool_get()* API; ④ The packet handler enqueues the descriptor to the producer (TX) ring; and then the packet handler ⑤ unblocks the Write() call to return control to the caller thread.

The synchronous Read() API is achieved by blocking the caller thread until the message is moved from the shared memory buffer to the receive buffer of the caller thread. We take advantage of the approach designed by X-IO [12] to enable the blocking Read() over the asynchronous shared memory I/O. There are two essential elements to block the caller of the Read(), including a condition variable [25] and a receive queue. Note that *each* user thread owns a dedicated condition variable as well as a receive queue for the sake of concurrent user session operations (§III-E).

The condition variable is utilized to suspend the caller thread of the Read() until its state is updated. Note that the condition variable is in effect only when its state is `TRUE`. We use the receive queue to buffer descriptors whose message payload has not yet been transferred from the shared memory buffer to the receive buffer of the caller thread.

These two elements in the Read() interact with the asynchronous packet handler in the I/O stack to accomplish the blocking receive, as depicted in Fig. 3 (right half): (1.) When the caller thread initiates the Read() call, (2.) it is blocked on the condition variable (now set as `TRUE`). Subsequently, (3.) the I/O stack receives a descriptor from the NF on the other side, and (4.) enqueues the descriptor into the receive queue of the caller thread. (5.) The I/O stack updates the condition variable to `FALSE` to unblock the Read() call. (6.) The Read() call then dequeues the descriptor from the receive queue and it (7.) copies the message from the shared memory buffer to the receive buffer of caller thread. Finally, (8.) control returns to the caller thread. Note that Fig. 3 (right half) shows only the case when there is a single user session. Details of message reception for concurrent user sessions are given in §III-E.

Connection management: Similar to a POSIX-like socket interface (*e.g.,* socket APIs, HTTP/REST APIs, 3GPP SBI), the synchronous I/O interface in **L²5GC+** also requires pre-established connections to facilitate data transmission. We adopt the approach from X-IO [12] to manage the lifecycle of connections, including their establishment and teardown. Each NF's I/O stack maintains a local connection table, as shown in Fig. 2, which records essential connection-specific information, such as the IP 4-tuple of the source and destination NFs. Introducing the notion of a connection is a key extension of **L²5GC+** beyond its predecessor, L²5GC, which allows **L²5GC+** to distinguish different user sessions, as described

in §III-E.

Implementation of HTTP/REST APIs and 3GPP SBI in L²5GC+: Our intention is to port the implementation of a 3GPP-compliant implementation like free5GC to **L²5GC+**, to ensure that **L²5GC+** also conforms to the specifications, leveraging free5GC's development effort. As such, we choose Golang [26] to develop the API libs in **L²5GC+**. Golang was the primary programming language that free5GC's control plane NFs are written in.

Since we take a layered design approach for our API libs, we only re-implement the POSIX-like socket interface in order to interact with our asynchronous shared memory I/O stack. We keep the upper layer HTTP/REST APIs and 3GPP SBI *unchanged*. This is achieved by replacing the imported Golang's "net" [27] package[3] with **L²5GC+**'s socket API package. Since our socket APIs keep the same semantics as Golang, the porting is seamless. The cross-language support is described next.

Cross-language support: The I/O stack in each **L²5GC+** NF is developed in C language, ensuring optimal performance and reliability. Seamless interaction between the C-based I/O stack and the higher-layer API libs (in Golang) is achieved through Golang's built-in CGo interface [13]. The C-to-Go boundary crossing incurs minimal additional latency (approximately 70ns in our testbed), showcasing negligible impact on data exchange performance.

E. Concurrent user session support

Thread-based concurrency is commonly used in representative implementations of 5GC control plane NFs that support multiple simultaneous user sessions. Each user session is handled by a dedicated thread in the control plane NF instance (typically deployed as a Linux process within a virtualized sandbox, *e.g.,* container) to accomplish certain control plane procedures (*e.g.,* UE registration, handover).

In order to differentiate user sessions during control plane messaging between NFs, we bind a specific connection for each user session. As shown in Fig. 4, (1) when the destination NF receives a message descriptor, (2) it looks up its local connection table in the I/O stack and finds the correct connection, *i.e.,* user session, to refer to. Subsequently, (3) the I/O stack can enqueue the descriptor to the connection's receive queue, thus ensuring that the message is correctly directed to the appropriate user session thread. This multiplexing/demultiplexing allows seamless concurrent processing of messages in **L²5GC+**'s control plane.

Concurrency control: **L²5GC+** adopts the implementation of 5GC from free5GC [11], which relies heavily on Golang as the primary programming language for the control plane NFs. The concurrency support in Golang is implemented by *goroutines* [28], which are essentially lightweight threads managed by the Go runtime. With the connection abstraction in **L²5GC+**, we can support multiple user sessions using thread-based concurrency.

[3]Golang's "net" [27] package is the official package that offers various POSIX interface for network I/O.

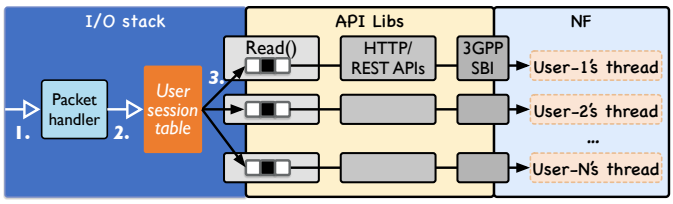


Fig. 4. Concurrent user session support in **L²5GC+**. Control plane messages to different user sessions are de-multiplexed at the I/O stack after user session table lookup.

However, we observe that certain 5GC control plane events (*e.g.,* PDU session establishment) incur very frequent context switches. This is caused by the CPU 'thrashing' between multiple user sessions when they complete the same 5GC control plane events concurrently using a limited number of CPU cores on the node. Since each user session requires a dedicated thread for each of the 5GC NFs to accomplish the control plane event-related tasks, it results in frequent thread context switches. This adds additional delay to the event completion time. As **L²5GC+** and free5GC share the same control plane NF implementation, they both suffer from this performance loss.

To overcome the effect of 'thrashing', we introduce a concurrency control mechanism to limit the number of concurrent execution of certain events (*e.g.,* PDU session establishment) that are processed simultaneously by the 5GC control plane using the available CPU cores on the node. We implement a rate limiter at the AMF, which is the ingress point of the 5GC control plane. After the concurrency threshold is reached, additional PDU session establishment requests are queued at the AMF. We note that the threshold will likely depend on the number of available CPU cores and their capability, the complexity of the operations, and likely the mix of operations. We experimentally determined the suitable concurrency level of *16* in our current testbed.

F. Deployment strategy of L²5GC+

L²5GC+ adopts the same placement and scaling strategy as its predecessor, L²5GC [3]. To harness the benefits of intra-node shared memory processing to minimize the latency of the 5GC control plane, it is important for **L²5GC+** NFs to be co-located on the same node.

In scenarios where inter-node communication becomes necessary (*e.g.,* when resource constraints prevent NF consolidation on a single node), **L²5GC+** relies on NF managers to facilitate communication, by offering consolidated protocol processing. This can be achieved by using a kernel-based protocol stack or by employing a high-performance inter-node communication solution, such as RDMA with zero-copy data transfer primitives. This is part of our ongoing effort. This approach helps minimize the impact of kernel-based networking when inter-node communication is required.

It is important to note that the hybrid communication mode of **L²5GC+** (combining intra-node shared memory processing and inter-node kernel-based networking) ensures that the lower

Fig. 5. (Top) The logical topology, and (Bottom) a snapshot of commercial testbed setup in **L²5GC+**. RU: Radio Unit; DU: Distributed Unit; CU: Central Unit; CN: Core Network; PoE: Power over Ethernet; 5 laptops as UEs with 5G dongles

bound of performance remains no worse than a complete kernel-based networking solution. On the other hand, the typical as well as the upper bound performance of the hybrid communication mode of **L²5GC+** far outperforms kernel-based networking, especially when we consider node affinity to strategically place NFs, thus fully exploiting the benefits of shared memory processing. This hybrid approach offers a flexible and efficient solution that optimizes performance based on the specific deployment scenario and resource availability.

IV. Evaluation

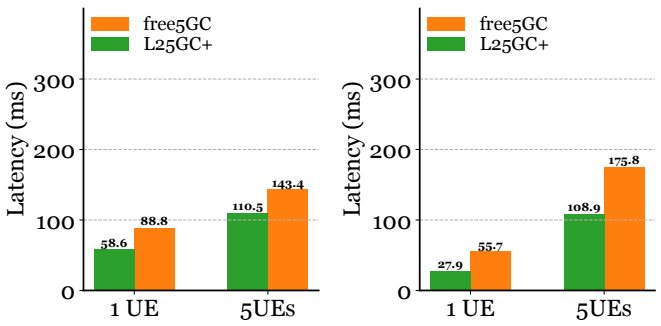
We evaluate the improved performance of **L²5GC+** using a real 5G testbed built using 3GPP-compliant commercial base stations, a variety of UEs (laptops with 5G dongles) and the **L²5GC+** running on commercial off-the-shelf (COTS) servers. We also consider a simulated 5G UE & RAN to evaluate the performance of **L²5GC+** as we scale up to more UEs. We compare **L²5GC+** with free5GC [11], an open-source, 3GPP-compliant 5GC implementation that has been used in many consortium-based 5G frameworks, *e.g.,* Magma [29], SD-CORE [30], and Aether [31].

A. Analysis with *commercial* UEs and RAN

Commercial testbed setup: Fig. 5 shows the experimental setup of our real testbed, including the UE, RAN, and 5G core network (CN). Our 5G RAN contains an RU and a commercial CU/DU system built on an off-the-shelf computer system. We use a 'bare-metal' server to run the 5G core network, including both the control plane and data plane. The CU/DU system connects to the 5GC control plane via the 3GPP N2 interface. The CU/DU system connects to the 5GC data plane (*i.e.,* UPF) via the 3GPP-specified N3 interface. The UPF of the 5GC connects to the data network via the N6 interface.

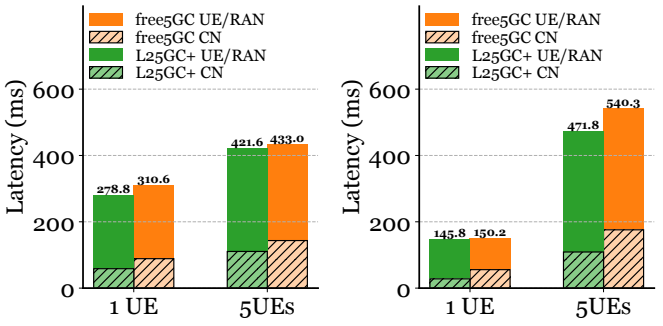
We choose the 5G small cell from Alpha Networks Inc. [32] as the RU. We use the commercially available CU/DU from AEWIN Technologies [33]. The server running the 5GC contains a 16-cores Intel Core i7 13700 CPU and 16G memory. We install two NICs on the 5GC server: a 10Gbps Intel X520-DA2 NIC used for N3 and N6 interface in the data plane, and a 2.5Gbps Realtek RTL8125BG NIC for the N2 interface in the control plane. We choose the 10Gbps NIC for the data plane for its higher bandwidth. We use a total of 5 laptops to emulate multiple UEs. Each laptop is equipped with an Apal 5G Dongle [34] to communicate with the RU in 5G RAN.

Tested control plane events: We select a pair of representative control plane events to evaluate, including UE registration and the PDU (Packet Data Unit) session establishment that follows. During registration, the UE establishes its presence and identity on the 5GC before accessing 5G services. A PDU session represents a logical connection between the UE and the 5GC for data communication. The PDU session establishment creates a dedicated data path between the UE and the 5GC data plane (*i.e.,* UPF) for handling data traffic. We evaluate **L²5GC+** with a single UE and also with 5 UEs running concurrently on the testbed to understand the ability of **L²5GC+** to support



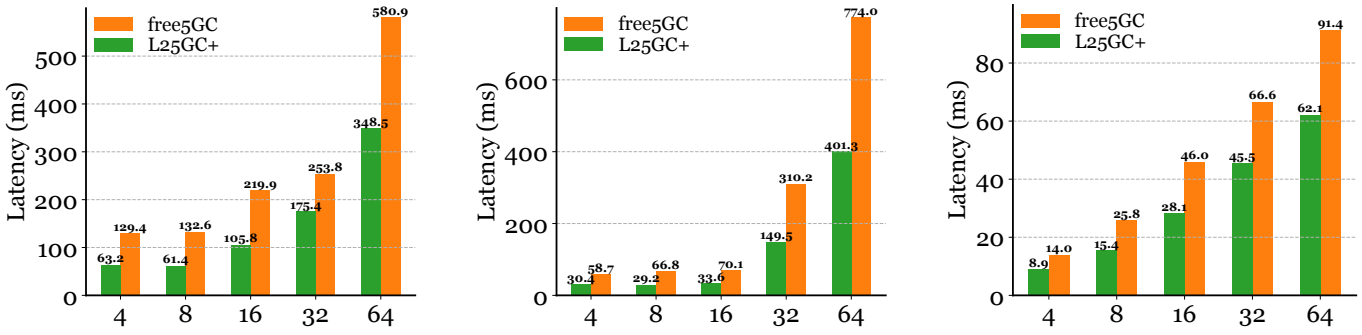
(a) UE registration (b) PDU session estab.

Fig. 6. Latency from 5G CN, tested with ***commercial*** UEs and RAN.



(a) UE registration (b) PDU session estab.

Fig. 7. Total latency (including core network and UE/RAN) of control plane events tested with ***commercial*** UEs and RAN.



(a) UE registration (b) PDU session establishment (c) Paging (idle-active transtion)

Fig. 8. Total latency of various control plane events with ***simulated*** UEs and RAN. *x-axis*: number of UEs.

multiple user sessions in the 5G control plane (unlike L^{2}5GC which had limited support).

Fig. 6 shows the contribution to end-end latency by the 5G core network. The result demonstrates the performance benefits of **L^{2}5GC+**'s shared-memory-based SBI in a commercial testbed: When handling a single UE, **L^{2}5GC+** has 1.51× lower latency than free5GC to complete a UE registration. **L^{2}5GC+** also achieves 2× latency improvement compared to free5GC for PDU session establishment. When 5 UEs register simultaneously, **L^{2}5GC+** saves 1.29× latency on average. **L^{2}5GC+** lowers latency by 1.61× on average to complete 5 PDU session establishment events concurrently. These improvements come mainly from the use of shared memory processing in **L^{2}5GC+**, which incurs much lower communication overheads for control plane NFs to exchange messages, compared to the kernel-based SBI of free5GC.

In addition, we measure the "total" latency for different control plane events (in Fig. 7). This includes the latency contributed by the core network (named "CN") and the part contributed by the commercial UE/RAN. The somewhat slower UE and the disaggregated RAN system in our testbed reduces the relative impact of **L^{2}5GC+**'s improvements to this "total" latency. However, with higher-speed UEs (e.g., smartphones) and improved RAN implementations, the significant reduction of the 5GC latency (the "CN" latency) due to our improvements will lower the overall cellular control plane latency.

*B. Analysis with **simulated** UEs and RAN*

We use the UE & RAN simulator from L^{2}5GC [3] to simulate user events, which allows us to scale up testing with more UEs. Throughout, we seek to understand the improved scalability of **L^{2}5GC+** compared to the kernel-based SBI in free5GC, in handling multiple user sessions. We additionally evaluate the latency for paging events in the 5G control plane, where a UE transitions from idle to active state. A UE typically moves into an idle state to conserve (battery) energy, which is important for mobile devices and UEs such as IoT devices. When either a packet arrives at the 5GC or the UE has to transmit a (first) packet, the UE is "paged" to wake up the UE. The time it takes for the 5GC to complete this task and have the UE transition from idle to active has a direct impact on the latency experienced by that first packet. **L^{2}5GC+** seeks to improve this latency in its control plane implementation. We vary the number of UEs from 4 to 64 and report the total latency (as in Fig. 7) that includes the total latency contributed by both the core network and the simulated UE/RAN.

Fig. 8 shows the total completion latency of different control plane events when multiple user sessions operate concurrently in the 5GC control plane. Compared to the kernel-based SBI in free5GC, **L^{2}5GC+**'s shared memory SBI shows a consistent reduction in latency as the number of UEs increases up to 64. **L^{2}5GC+** reduces UE registration latency by 1.87× on average. **L^{2}5GC+** also reduces the average PDU session establishment latency by 2.1× and average paging latency by 1.6×.

V. Conclusion

We presented **L^{2}5GC+**, a low-latency 5G control plane implementation. **L^{2}5GC+** is an enhancement to our previous effort L^{2}5GC developed on top of OpenNetVM, a high-performance shared-memory NFV platform. **L^{2}5GC+** makes several key extensions to L^{2}5GC, including support for concurrent user sessions using a 3GPP-compliant shared memory SBI. These two capabilities significantly improve the applicability of shared memory processing of the 5GC control plane, allowing 3GPP-compliant 5GC implementations such as free5GC to be seamlessly ported to **L^{2}5GC+**, while retaining the performance benefits of shared memory processing. Our evaluation using a testbed with commercial 5G UE and RAN components shows that **L^{2}5GC+**, with the help of its shared memory SBI, outperforms a kernel-based SBI implementation. **L^{2}5GC+** significantly reduces the control plane latency for UE registration and PDU session establishment by 1.29× and 1.61×, respectively, with 5 commercial UEs operating concurrently.

Acknowledgment

We thank the US NSF for their generous support through grant CRI-1823270. This work was also supported in part by the National Science and Technology Council of Taiwan under grant numbers 112-2218-E-A49-021, 112-2218-E-A49-023, and 111-2221-E-A49-093-MY3.

REFERENCES

[1] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3098–3130, 2018.

[2] H. Hassanieh, O. Abari, M. Rodriguez, M. Abdelghany, D. Katabi, and P. Indyk, "Fast Millimeter Wave Beam Alignment," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 432–445. [Online]. Available: https://doi.org/10.1145/3230543.3230581

[3] V. Jain, H.-T. Chu, S. Qi, C.-A. Lee, H.-C. Chang, C.-Y. Hsieh, K. K. Ramakrishnan, and J.-C. Chen, "L25GC: A Low Latency 5G Core Network based on High-performance NFV Platforms," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 143–157.

[4] F. Parola, F. Risso, and S. Miano, "Providing telco-oriented network services with ebpf: the case for a 5g mobile gateway," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, 2021, pp. 221–225.

[5] S. Panda, K. K. Ramakrishnan, and L. N. Bhuyan, "Synergy: A Smart-NIC Accelerated 5G Dataplane and Monitor for Mobility Prediction," in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, 2022, pp. 1–12.

[6] 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 *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, ser. SOSR '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 162–168. [Online]. Available: https://doi.org/10.1145/3482898.3483358

[7] S. Wang, J. Huang, X. Zhang, H. Kim, and S. Dey, "X-array: Approximating omnidirectional millimeter-wave coverage using an array of phased arrays," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[8] Y. Li, Z. Yuan, and C. Peng, "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*, 2017, pp. 56–69.

[9] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. K. Ramakrishnan, and T. Wood, "OpenNetVM: A Platform for High Performance Network Service Chains," in *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, 2016, pp. 26–31.

[10] https://github.com/nycu-ucr/l25gc, 2023, [ONLINE].

[11] "free5GC," https://github.com/free5gc/free5gc, 2023, [ONLINE].

[12] S. Qi, H.-S. Tsai, Y.-S. Liu, K. Ramakrishnan, and J.-C. Chen, "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, 2023, pp. 107–115.

[13] "Cgo," https://pkg.go.dev/cmd/cgo, 2023, [ONLINE].

[14] T. V. Kiran Buyakar, H. Agarwal, B. R. Tamma, and A. A. Franklin, "Prototyping and Load Balancing the Service Based Architecture of 5G Core Using NFV," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 228–232.

[15] M. Ahmad, S. U. Jafri, A. Ikram, W. N. A. Qasmi, M. A. Nawazish, Z. A. Uzmi, and Z. A. Qazi, "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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 648–661. [Online]. Available: https://doi.org/10.1145/3387514.3406218

[16] A. Mohammadkhan, K. K. Ramakrishnan, and V. A. Jain, "CleanG—Improving the Architecture and Protocols for Future Cellular Networks With NFV," *IEEE/ACM Transactions on Networking*, vol. 28, no. 6, pp. 2559–2572, 2020.

[17] J. Meng, J. Huang, Y. C. Hu, Y. Koral, X. Lin, M. Shahbaz, and A. Sharma, "Characterizing and Modeling Control-Plane Traffic for Mobile Core Network," *arXiv preprint arXiv:2212.13248*, 2022.

[18] "Data Plane Development Kit," https://www.dpdk.org/, 2023, [ONLINE].

[19] "DPDK Mempool Library," https://doc.dpdk.org/guides/prog_guide/mempool_lib.html, 2023, [ONLINE].

[20] S. Qi, L. Monis, Z. Zeng, I.-c. Wang, and K. K. Ramakrishnan, "SPRIGHT: Extracting the Server from Serverless Computing! High-performance eBPF-based Event-driven, Shared-memory Processing," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 780–794.

[21] S. Qi, Z. Zeng, L. Monis, and K. K. Ramakrishnan, "MiddleNet: A Unified, High-Performance NFV and Middlebox Framework with eBPF and DPDK," *IEEE Transactions on Network and Service Management*, 2023.

[22] "Environment Abstraction Layer," https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html, 2023, [ONLINE].

[23] "DPDK Multi-process Support," https://doc.dpdk.org/guides/prog_guide/multi_proc_support.html, 2023, [ONLINE].

[24] "DPDK Ring Library," https://doc.dpdk.org/guides/prog_guide/ring_lib.html, 2023, [ONLINE].

[25] "Using condition variables," https://www.ibm.com/docs/en/aix/7.2?topic=programming-using-condition-variables, 2023, [ONLINE].

[26] "The Go Programming Language," https://go.dev/, 2023, [ONLINE].

[27] "net," https://pkg.go.dev/net, 2023, [ONLINE].

[28] "How To Run Multiple Functions Concurrently in Go," https://www.digitalocean.com/community/tutorials/how-to-run-multiple-functions-concurrently-in-go, 2023, [ONLINE].

[29] S. Hasan, A. Padmanabhan, B. Davie, J. Rexford, U. Kozat, H. Gatewood, S. Sanadhya, N. Yurchenko, T. Al-Khasib, O. Batalla, M. Bremner, A. Lee, E. Makeev, S. Moeller, A. Rodriguez, P. Shelar, K. Subraveti, S. Kandi, A. Xoconostle, P. K. Ramakrishnan, X. Tian, and A. Tomar, "Building Flexible, Low-Cost Wireless Access Networks With Magma," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 1667–1681. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/hasan

[30] "SD-Core," https://opennetworking.org/sd-core/, 2023, [ONLINE].

[31] "Aether," https://opennetworking.org/aether/, 2023, [ONLINE].

[32] "Products: 5G Small Cell - Alpha Networks Inc." https://www.alphanetworks.com/index.php/en/product_detail/a3caef706423d652, 2023, [ONLINE].

[33] "Performance Network System - SCB-1921B," https://www.alphanetworks.com/index.php/en/product_detail/a3caef706423d652, 2023, [ONLINE].

[34] "Dongle - APAL," https://www.apaltec.com/dongle/, 2023, [ONLINE].