L^{2}5GC: A Low Latency 5G Core Network based on High-Performance NFV Platforms

Vivek Jain[*], Hao-Tse Chu[†], Shixiong Qi[*], Chia-An Lee[†], Hung-Cheng Chang[†], Cheng-Ying Hsieh[†], K. K. Ramakrishnan[*], Jyh-Cheng Chen[†]

[*]University of California, Riverside, [†]National Yang Ming Chiao Tung University

ABSTRACT

Cellular network control procedures (e.g., mobility, idle-active transition to conserve energy) directly influence data plane behavior, impacting user-experienced delay. Recognizing this control-data plane interdependence, L^{2}5GC re-architects the 5G Core (5GC) network, and its processing, to reduce latency of control plane operations and their impact on the data plane. Exploiting shared memory, L^{2}5GC eliminates message serialization and HTTP processing overheads, while being 3GPP-standards compliant. We improve data plane processing by factoring the functions to avoid control-data plane interference, and using scalable, flow-level packet classifiers for forwarding-rule lookups. Utilizing buffers at the 5GC, L^{2}5GC implements paging, and an intelligent handover scheme avoiding 3GPP's hairpin routing, and data loss caused by limited buffering at 5G base stations, reduces delay and unnecessary message processing. L^{2}5GC's integrated failure resiliency transparently recovers from failures of 5GC software network functions and hardware much faster than 3GPP's reattach recovery procedure. L^{2}5GC is built based on free5GC, an open-source kernel-based 5GC implementation. L^{2}5GC reduces event completion time by ∼50% for several control plane events and improves data packet latency (due to improved control plane communication) by ∼2×, during paging and handover events, compared to free5GC. L^{2}5GC's design is general, although current implementation supports a limited number of user sessions.

CCS CONCEPTS

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

KEYWORDS

5G cellular networks, Cellular core, Low latency 5G core, NFV

ACM Reference Format:
Vivek Jain[*], Hao-Tse Chu[†], Shixiong Qi[*], Chia-An Lee[†], Hung-Cheng Chang[†], Cheng-Ying Hsieh[†], K. K. Ramakrishnan[*], Jyh-Cheng Chen[†]. 2022. L^{2}5GC: A Low Latency 5G Core Network based on High-Performance NFV Platforms. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3544216.3544267

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9420-8/22/08.
https://doi.org/10.1145/3544216.3544267

1 INTRODUCTION

Emerging applications such as the Internet of Things (IoT), connected vehicles, *etc.* require low latency, ubiquitous network access [55] and often rely on the cellular network (5G and beyond) for network access. To truly deliver the low latency needed for acceptable user quality of experience (QoE), both the access and core part of cellular networks need to improve. With radio access technology such as millimeter wave, the latency of access network is being reduced (e.g., of the order of 1ms [28, 38]). Recent work [39] shows that mmWave beam alignment and link acquisition can complete within 1-10 ms., allowing a UE's connection establishment with gNodeB to complete quickly. However, the core network still contributes significantly to the overall high latency observed in cellular networks.

In traditional 4G cellular deployments, each cellular core component was implemented on purpose-built hardware, typically distributed in cellular data centers [5]. The resulting complex control plane procedures result in high latency because of the complex protocols needed to ensure a consistent state among these entities. In the 5G cellular ecosystem, cellular components are implemented as software-based cloud-native services for deployment flexibility. However, with control plane procedures in 5G Core (5GC) being similar to the traditional 4G [49], it does not leverage the full benefit of the softwarization of network functions (NFs).

The cellular core is expected to experience more control plane traffic due to: *i)* the massive growth of cellular subscribers, including IoT and machine to machine devices [31], *ii)* frequent handover events because of reduced cell sizes. User event completion times, such as a handover process that takes 1.9 seconds (as shown in [45]) can directly impact the delay and packet loss experienced by the end-user data packets. Apart from the penalty due to traditional cellular control plane core procedures, we recognize several additional issues (details in §2.3) that contribute to latency:
• The adoption of HTTP/REST for inter-NF communication, ostensibly to support the idea of 'dis-aggregation', suffers from overheads of message serialization and TCP processing.
• Current implementations, driven by 3GPP standards (§5.2.1 of [8]), organize forwarding (match-action) rules in a list and perform an inefficient linear search to find the matching rule.
• Users may experience higher packet losses during handover due to the current hairpin and daisy-chain routing, and limited buffering at the target gNB (5G base station).
• During failure, current 3GPP restoration process requires users to re-initiate the connection establishment request and start over. This directly impacts ongoing data connections (*e.g.,* TCP experiences spurious timeouts), affecting QoE.

To achieve the goals of both flexibility and performance beyond current softwarized cellular core, we design L^{2}5GC, an NFV-based, low-latency 5GC network solution, which achieves significant improvement in both control and data plane while still being 3GPP-compliant. We build our open-sourced NFV-based 5GC (details in Appendix E) on top of 'free5GC' [14]. Our current implementation of L^{2}5GC supports a limited number of user sessions, but can be generalized subsequently. Building on free5GC, a complete 3GPP open-source implementation, helps L^{2}5GC have a comprehensive implementation of most of the 3GPP specified [6, 7] control plane protocols and a reasonable implementation of all the 5GC NFs. L^{2}5GC supports the commonly adopted approach of disaggregating the core network NFs. The main distinction is in the core network implementation enhancements we make. By exploiting shared memory, L^{2}5GC constructs an efficient, low latency 5GC service, that retains the advantages of a microservice-based design pattern, while eliminating many of the typical overheads that come from having a general-purpose service interface between individual microservices. L^{2}5GC makes the following contributions:

(1) We exploit the flexibility and scalability of NFV platforms to consolidate the control and data plane NFs on the same node, while retaining the flexibility to have each NF separately implemented. Consolidation on the same node helps to reduce the inter-NF communication overheads.

(2) We rebuild the Service Based Interface (SBI), the N4 interface, and the entire 5GC data plane by using shared memory-based zero-copy mechanism to replace the kernel-based communication channel between NFs, removing the extra processing for serialization and achieving high-speed inter-NF communication.

(3) We optimize the handover procedure to reduce the extra hairpin and daisy chain routing, without changing the protocol specified by 3GPP. L^{2}5GC implements smart buffering that is used both for the optimized handover procedure and buffer packets to User Equipments (UEs) that are idle.

(4) To support evolving 5G use cases that result in significant growth of Packet Detection Rules (PDRs), we implement and compare multiple approaches, including linear search, Tuple Space Search (TSS) [57] and a PartitionSort [59] classifier (which we choose). Thus, we achieve scalability and high data plane throughput.

(5) We mitigate delays from 5GC NF's failure recovery by running lightweight replicas that do not consume any CPU when idle. By leveraging [44, 56], we replicate the state of 5G NFs, ensuring consistency while avoiding the 3GPP-specified UE re-attachment procedure.

L^{2}5GC's evaluation results show significant performance improvement over the non-NFV-based free5GC [14]. We observe a 13× improvement in individual message exchange latency and up to 51% reduction in overall event completion time. Importantly, the reduction in control plane latency has a corresponding impact on data plane performance. Data packet latency (also due to the improved control plane) by ~2×, during paging and handover events, compared to free5GC. Even simple web page's (with many large images) loading time improves by 12.5% with L^{2}5GC compared to free5GC, thus directly improving user QoE. The data forwarding in L^{2}5GC can operate at a line rate for 64 byte packet traffic on a 10 Gbps link, 27× higher than free5GC's kernel-based forwarding.

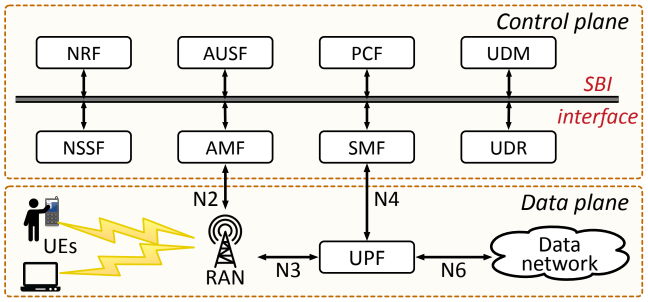


Figure 1: 5G core architecture

2 BACKGROUND AND MOTIVATION

2.1 5G cellular architecture

A typical cellular network consists of the Radio Access Network (RAN), usually comprising the wireless channel, the cellular base station and a backhaul network, all used to connect mobile devices (*i.e.,* UEs) to the core network. The core network is responsible for connecting UEs to the Data Network, (typically IP network), and provides the majority of the 'cellular services'. A 3GPP-compliant 5G architecture is shown in Fig. 1. Unlike previous generation core network, 5GC takes advantage of NFV to implement the core functions in software rather than purpose-built hardware appliances [6]. We list a number of 3GPP abbreviation in Appendix D.

Control plane: 5GC utilizes a service-based architecture in the control plane (Fig. 1). By connecting control plane functions as various cloud-native services in the form of a service mesh, 5GC greatly improves the operability of inter-service communication over a HTTP/REST API compared to inflexible point-to-point connected approach.

Data plane processing: The Session Management Function (SMF) dictates the data plane forwarding at User Plane Function (UPF) by provisioning PDRs, which contain various information, *e.g.,* packet detection information (PDI), priority, and associated actions. The UPF organizes PDRs in a list in descending order of their priority. On arrival of a user packet, UPF performs a lookup to identify the user session, and then the list of PDRs is traversed until the highest precedence rule is found [23]. The data is carried over the GTP tunnel between the gNB and UPF, and requires setting up the unique tunnel endpoint identifier (TEID).

2.2 Related Work

There has been considerable recent research to improve the performance and reliability of cellular networks.

• **Reducing control plane latency:** Mukhtiar *et al.* [27] propose Neutrino, which seeks to optimize the cellular control plane latency by using fast serialization techniques and reduces handover time by maintaining user state replicas in a larger geographical area. This may add overhead of replicating to all neighboring 'regions' in what is already a heavyweight protocol state machine. L^{2}5GC seeks to eliminate the cost incurred due to serialization and it can complement the handover optimizations of [27]. Further, given that cellular operator backhaul networks are based on metro area packet networks, L^{2}5GC's deployment strategy seeks to minimize handovers causing sessions to move to another 5GC through consolidation and processing efficiencies since a 5GC is accessible over the metro network without significant latency difference. With an edge cloud having reasonable scalability in the number of sessions supported, we resort to replication for failure recovery similar to [27, 44, 56]. However, Neutrino [27] focuses only on control events by replicating UE state consistently across multiple nodes by mirroring control plane functions (which unfortunately consumes extra resources) at the multiple nodes. In addition, Neutrino does not account for data packets that get lost during failover, placing the burden on user end-systems to retransmit these data packets, impacting QoE. In contrast, L^{2}5GC addresses *both* control and data plane resiliency.

• **Optimizing 5GC SBI:** Buyakar *et al.* [29] implements the 5GC's SBI with gRPC [2] instead of HTTP/REST. However, gRPC is built on top of HTTP/2 [3] and uses protobuf [1] or JSON as the serializing structure. It still has overheads related to serialization and expensive communication over kernel sockets. Our experiments show that these are major sources of 5G control plane latency, which L^{2}5GC avoids.

• **Cellular core availability:** ECHO [51] proposes a distributed state machine replication protocol for 4G. It replicates the EPC state machines in an NFV environment to provide redundancy against potential failures. L^{2}5GC's failure resiliency is similar to ECHO's design. Our emphasis, however, is on reducing control plane latency.

• **Redesigning control plane procedures:** There have been many proposals [49, 50, 54] to redesign the cellular control plane protocol and the core architecture. CleanG [49] proposes a scalable NFV-based architecture to optimize the control plane and data plane latency in cellular networks. It leverages the shared memory feature of Data Plane Development Kit (DPDK) [12] to improve control plane latency. However, CleanG's focus is on having a substantially new control plane protocol. Our focus here is on developing a 3GPP-compliant high-performance implementation.

• **Leveraging client-side state:** DPCM [45] proposes a client-side solution that initiates and executes control operations in parallel by leveraging the device side user state to reduce control plane latency, and eliminating some 3GPP messages for authenticated UEs. We believe, by incorporating DPCM's client-side modifications, L^{2}5GC may further speed up the control plane processing. However, until their security implications are proved, L^{2}5GC conservatively seeks to retain compatibility with the 3GPP-specified protocol.

Existing Open-Source Implementations:

There are several cellular core network implementations available as open-source [18, 20, 32]. OpenAirInterface [20] is a well known open-source implementation, focused on an open-source RAN. It also provides a 4G EPC implementation. The 5GC version [21] is in the early stages, with a subset of NFs such as AMF, SMF, and UPF being implemented. NextEPC [18] is another, more complete 4G core network implementation. It has been known to work well with commercial 4G small cells. Their subsequent 5GC implementation, Open5GS [19], does not yet have all the features of the 3GPP specifications. Travelping [25] is a Vector Packet Processing based User Plane Gateway (UPG) [26] implementation, a key component of the 5GC to achieve a high-performance data plane. However, it is not yet a full-fledged 5GC implementation.

Our earlier work on a 5GC implementation, free5GC, provides a complete Release 15 [6, 7] 3GPP-compliant 5GC implementation. It includes additional features such as a full-fledged PCF, support for Non-3GPP Interworking Function (N3IWF) [4], EAP-AKA' authentication, and packet buffering. These unique features significantly extend its usage for either research and experimentation or for development into a production environment. The support for N3IWF and EAP-AKA' authentication allows non-3GPP devices to access free5GC, and thus L^{2}5GC which is based on free5GC, without being restricted to the licensed spectrum and production base stations. For example, IoT devices connected to WiFi Access Points can access free5GC via N3IWF, with EAP-AKA' providing the necessary authentication for these devices.

There are several consortium-based 5G frameworks, such as Magma [16], SD-CORE [24], and Aether [11], that utilize open source 5GC implementations. Many of them (especially those 3) have considered using free5GC as their 5G core network.

Although free5GC provides a comprehensive 3GPP-based implementation with substantial flexibility in organizing 5GC NFs, its performance is limited by its kernel-based implementation, as discussed next. L^{2}5GC seeks to overcome them.

2.3 Challenges with existing Framework

We identify the following challenges to implement a low latency 5GC. Several control and data plane components contribute to latency that ultimately impacts the user QoE:

Challenge 1. Control message processing: The number of control messages exchanged in 5G has in fact increased slightly, relative to 4G, to keep the user state consistent across multiple disaggregated NFs [6]. The cost of exchanging these messages is high because of several factors including message serialization, HTTP/TCP overheads. The communication overheads will be even higher if the NFs are placed across the nodes. This becomes a concern as 5G network deployments and applications evolve, *e.g.,* supporting IoT, which are likely to increase control plane traffic [31].

Challenge 2. Complex handover procedure: During a handover process, the user experiences added delay, data loss, and out-of-order delivery. The handover operation can take up to 1.9 seconds [45] to complete. This can affect data plane traffic. For example, TCP-based data traffic can suffer from spurious timeouts, which degrades application throughput. Further, UE handover may be more frequent because of the smaller cell sizes Along with the many control message exchanges, the 5G handover relies on direct (X2) or indirect (S1) forwarding, which involves data packets being buffered at the source gNB. When the UE synchronizes with the target gNB they are re-routed to the target gNB. Unlike 4G base stations, gNBs may be relatively small and have limited buffering capacity. Based on private conversations with major cellular operators and an equipment manufacturer, we estimate the buffering capacity at (macro cell) base stations to be about 2MB (~1300 full MTU packets) per radio resource-connected UE. Thus, both data forwarding techniques (*i.e.,* X2 or S1) may have extra latency and suffer from data loss. Further, the use of X2 handover is relatively small (or nonexistent). With the 3GPP-specified S1 handover, this results in more traffic 'hairpinning' back to 5GC and then forwarded to target gNB.

Challenge 3. Expensive Rule Lookup: The cellular ecosystem has evolved from providing traditional services (e.g., voice, SMS) to enabling various packet-oriented services. Also, with the higher

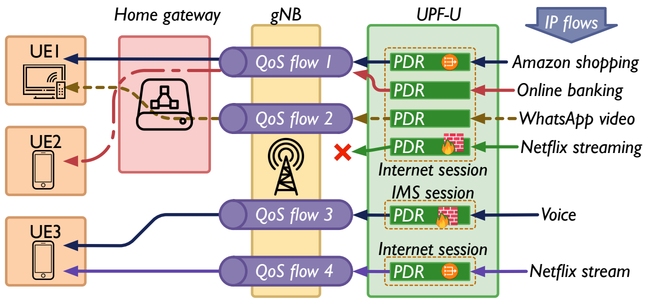


Figure 2: Multiple packet flows within a user session.

bandwidth available, fixed wireless access may be provided by the 5G network. The definition of a UE will evolve from a single user device to a home gateway (Fig. 2) that connects multiple end devices, including cell phones, IoT devices and smart TV, etc. One can think of the 5G home gateway as a single 'virtual UE', which can simultaneously run multiple sessions for different types of services, generating packets with different Quality of Service (QoS). 5GC provides a flexible QoS model where QoS is applied at the granularity of subflows. That is, we have multiple IP-level flows within a session [10]. As a result, the number of PDRs per user session will likely grow much larger than the 2-4 rules currently used primarily for UL and DL classification [43].

Moreover, as 5G networks evolve to become primarily packet-oriented, there is a need for firewall and NAT functionality to be included to secure the flows in a PDU session. Maintaining high performance requires these functions to be tightly integrated into the data plane. Vendors (*e.g.*, Ericsson [36]) are integrating this functionality in the 5G data plane. 3GPP defines a number of Information Elements (IEs) in the PDR for match-action functionality, including the Service Data Flow (SDF) filter, Ethernet Packet Filter, etc [23]. Given the IEs supported by the PDR, NAT and firewall rules can be integrated into the PDRs and maintain data plane performance, as long as the increased number of rules are handled efficiently. Some of the IEs can be further expanded to include additional elements, e.g., the SDF filter can be expanded to IP 5 tuples plus other fields. Thus, the complexity of the PDR's IE structure can potentially result in a large number of PDRs being scanned for each data packet. Therefore, we need better approaches (instead of a linear search of a list, as suggested in §5.2.1 of [8]) for looking up PDRs applied for the flow. This problem is similar to the classical packet classification problem, where studies [57] have shown that linear search does not scale.

Challenge 4. NF resiliency and recovery: To recover from failure, the 3GPP restoration procedure requires additional control messages for restoring the context of 5G sessions in various NFs. Ongoing cellular connections have to wait for this context to be restored before exchanging data. While this re-initiation of the connection is an obvious, simple solution, it adds considerable delay, and potential packet loss. This need for reattachment can be avoided by replicating the primary NFs. Proactive replication of state information is needed to switch over to the active standby, upon failure of the primary. However, maintaining a strongly consistent replica for such NFs can impact normal performance and incur substantial (possibly wasteful) overheads. Although existing techniques (e.g., packet replay with lazy checkpoint and full checkpointing) work well in the NFV environment, 5GC poses significant challenges because of the tight interdependence between the control and data planes, requiring substantial enhancements to their design. Additionally, a constantly running replica consumes system resources. We need a strategy to lower recovery times, coordinate control plane state with the data plane, and ensure consistency without consuming extra resources.

3 L^{2}5GC DESIGN AND IMPLEMENTATION

3.1 Overview: Design Goals and Approach

We seek to reduce control *and* data plane latency and achieve high throughput by optimizing the processing and communication among NFs implementing the 3GPP specified control and data plane protocols. The fundamental idea behind the service-oriented architecture of 3GPP 5G specifications is to permit individual services to be developed and scaled independently, following the microservice design paradigm. However, in our view a service-based interface should not mandate that it must only use HTTP and a REST API. We believe having a straightforward API for communication between microservices is desirable, but should be able to take advantage of alternative ways of exchanging information in the case when the microservices are co-resident on the same node. We believe our design principles will be applicable for 5G and beyond:

Consolidating NFs on the same node: We take advantage of system capabilities, including a shared memory space for data sharing to avoid moving data between NFs, which also avoids the added cost of serialization and de-serialization. This also allows us to mitigate network I/O latency and helps to overcome overheads that occur with the SBI currently recommended. We take a fresh look at deployment and scaling strategies in this context for 5GC (see §4).

Smart Buffering for Handover: A straightforward implementation of the 3GPP specification for handovers involves buffering downlink packets at the source gNB, which may have limited resources, especially with small cells. To address this issue, we utilize the buffering functionality at UPF that is already in-place for paging operations without adding any additional control messages. This has the added benefit of avoiding hairpin routing through the source gNB.

Fast rule lookup: Instead of performing the linear search of PDRs, we explore two alternatives: TSS, and PartitionSort. This reduces the complexity of PDR lookup in the UPF.

Resiliency through state replication: We leverage the idea of external synchrony (as used in [44, 52, 56]) to continue the speculative execution of user events while the state is replicated to a standby. This allows us to avoid synchronous replication and have 5G control and data plane processing progress without reduction in performance, while still providing consistency guarantees on a failure of one or more NFs.

We built our framework on top of free5GC (its implementation details are in Appendix B) and OpenNetVM (ONVM) [41, 60], a high-performance NFV platform based on DPDK. The innovations brought by L^{2}5GC may also be applied to other NFV platforms with similar capabilities.

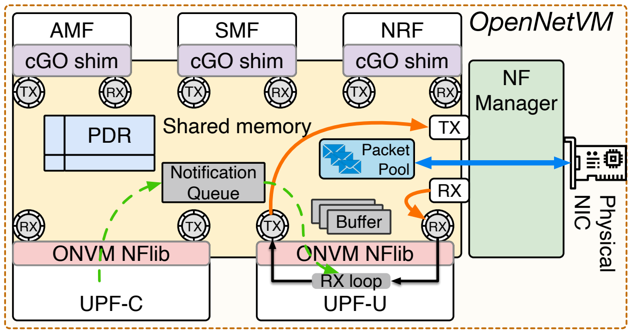


Figure 3: Optimized 5GC architecture through shared-memory communication: data and control plane

3.2 Consolidating NFs on same node

To minimize communication overheads, we leverage software-based 5GC NFs, and run them on a commercial off-the-shelf (COTS) server that has a sufficient number of CPU cores. With the development of high-performance packet processing frameworks (*e.g.,* DPDK, OpenNetVM) and the evolution of virtualization technologies (*e.g.,* single root input/output virtualization [34]), a COTS server can host all 5GC NFs and provide accelerated data packet processing at the same time.

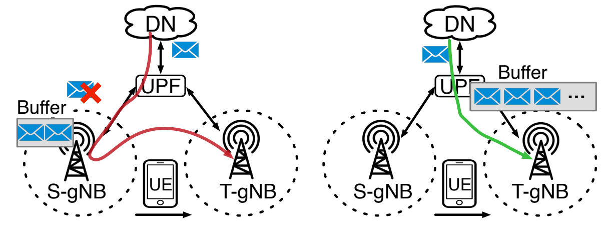
Shared memory communication: To further reduce the packet processing cost and to support fast communication between 5GC NFs, L^{2}5GC uses shared memory to achieve zero-copy packet processing. As shown in Fig. 3, L^{2}5GC runs an NF manager to handle incoming packets, manage the shared memory, and facilitate zero-copy communication between 5GC NFs. Each 5GC NF is assigned a unique service ID and attached receive (Rx) and transmit (Tx) rings. These ring buffers are shared with the manager used for packet descriptors pointing to packets in shared memory. After processing, the NF attaches the metadata to the packet to specify the action (send to port/NF, drop) for that packet and puts it in the Tx ring for the manager to process the action. To send the packets between NFs, the source NF specifies the target NF's id in the metadata. The manager copies the descriptor of flat data structure into the ring buffer of the target NF and thus mitigates serialization and HTTP processing. We replace the SBI and N4 interfaces (shown in Fig. 1) with our descriptor-based shared-memory approach, thus eliminating considerable overhead. We utilize the clean abstraction provided by OpenNetVM's shared-memory [60] to implement serialization and lock-free inter-process communication. To support NFV-based components, we started from the Golang-based free5GC and developed a generic cGO shim layer (shown in Fig. 3). This allows Golang-based NFs to use DPDK functions and use ONVM and shared memory.

Zero cost state update: We divide the data plane (*i.e.,* UPF) into a pair of NFs: UPF-C and UPF-U, to mitigate interference between control and data plane processing. To avoid overheads for state update and propagation across NFs, we ensure that forwarding rules and state for the UPF are in a shared memory. Using shared Hugepages (with DPDK), we maintain two hash tables for storing the pointer to a user session context. The keys for these two tables are TEID and UE IP to differentiate UL and DL traffic, respectively. Each user session context stores a number of different rule sets in shared memory, *e.g.,* PDRs and FARs, to control the packet-forwarding behavior of the data plane.

Currently, our control plane implementation supports two users. The data plane component (*i.e.,* UPF) supports as many users as the available system resources will support.

Security domain in L^{2}5GC: With potentially multiple services (possibly developed by third parties) running in the cloud environment, security concerns such as eavesdropping and data tampering may arise. It is necessary to isolate L^{2}5GC from those other applications. We take advantage of the security domain design of [42].

L^{2}5GC's trust model assumes all of L^{2}5GC's NFs are trusted, managed by the cellular operator. These NFs share a private memory pool not accessible by the other applications running on the same node. At the startup of L^{2}5GC, the NF manager (Fig. 3), which runs as the DPDK primary process, creates a private shared memory pool for NFs in L^{2}5GC. This private shared memory pool is implemented as hugepages in the Linux file system with a unique "shared data file prefix" [13] specified during creation. NFs in L^{2}5GC, which run as DPDK secondary processes, use the same file prefix specified by the NF manager to gain access to the private shared memory pool. For multiple L^{2}5GC instances on the same node managed by different operators, each would have a unique file prefix to keep its shared memory pool isolated from the others. Further enhancements to the security domain design of L^{2}5GC would be to add an admission control mechanism for verifying NFs that seek access to the private memory pool belonging to the cellular operator.



(a) 3GPP-based indirect forwarding and buffering (b) Direct forwarding with 5GC buffering, improving handover

Figure 4: Buffering during Handover: 3GPP vs. L^{2}5GC

3.3 Smart Buffering for Handover

L^{2}5GC redesigns the 3GPP handover behavior. Instead of performing hairpin routing as shown in Fig. 4(a), L^{2}5GC buffers the downlink (DL) packets at the UPF which has more buffering, as in Fig. 4(b). This results in several benefits: *i)* mitigation of packet loss at source gNB, *ii)* avoids unnecessary packet processing (and delay) due to hairpin routing.

L^{2}5GC does not introduce any additional messages to enable the buffering at UPF. However, it modifies the control plane (SMF) behavior to provision the buffering rule at the UPF for a handover event, which already utilizes the buffering capabilities for paging events (idle-active transition). When a UE requests a handover, the SMF sends a Packet Forwarding Control Protocol (PFCP) message to UPF to allocate a new TEID for the target gNB. We leverage this opportunity and piggyback an additional IE along with the original PFCP message to update the PDR and FAR action to buffer the packets. The UPF buffers all the subsequent DL packets for the session and forwards them to the target gNB once the handover completes. To avoid interference from other sessions, L^{2}5GC implements a 3GPP compliant session-based buffering. Our design also ensures in-order delivery of data packets.

3.4 Fast Rule Lookup

Since the UPF-U requires a PDR lookup for each arriving packet to determine its forwarding policy, the lookup speed directly impacts the data plane performance. We examine different alternatives: Linear Search (PDR-LL), TSS (PDR-TSS) and ParititionSort (PDR-PS) for reducing the search complexity of large-scale PDR lookups in UPF-U, which we see as being critical for future-proofing the 5GC.

Both PDR-TSS and PDR-PS have lower search complexity than PDR-LL. PDR-TSS reduces the search complexity by partitioning PDRs into multiple sub-tables based on tuples (*e.g.,* PDI IE fields). PDRs in the same sub-table have the same prefix bits in each tuple, but their values can differ. Each sub-table is organized as a hash table with $O(1)$ complexity for the PDR lookup based on TSS traversal of the tuple space (*i.e.,* a group of sub-tables converted from the PDRs) until a matching PDR is found. Compared to the linked list based PDR-LL, the PDR-TSS search achieves less overhead when there are a large number of PDRs. A linked list with M elements can be converted into N tuples ($N \leq M$). However, PDR-TSS does not guarantee optimal lookup performance since the number of partitioned sub-tables has some variability. In the worst case, PDR-TSS can have the same search complexity as PDR-LL. The variability in the search overhead of PDR-TSS, combined with the need for software hashing to look up a sub-table, can potentially result in high overhead.

PDR-PS reduces the search complexity by leveraging multidimensional binary search. For a set of PDRs, PDR-PS stores sorted PDRs in a multidimensional binary tree based on the values of the tuples. PDR-PS can perform fast binary search among the sorted PDRs compared to the in-order lookup of a linked list. Similar to PDR-TSS, PDR-PS divides the PDRs into multiple groups and then sort these groups to further reduce the complexity of the lookup.

Compared to PDR-TSS, PDR-PS does not rely on software hashing during PDR lookup. With PartitionSort's online ruleset partitioning, PDR-PS eliminates randomness and results in fewer partitioned rule sets, yielding more consistent performance [59].

In addition, L^{2}5GC's UPF-U seeks to meet important requirements of operators, *e.g.,* avoiding DoS attacks [33]. PartitionSort helps to avoid TSS's vulnerability to DoS attack. Given the advantages of PDR-PS, we use it in L^{2}5GC for fast PDR lookup. We studied other state-of-the-art alternatives, *e.g.,* NeuroCuts [46], but lacking production data-sets for learning, we seek to use the option with the highest performance providing the needed flexibility. To accommodate a packetized 5GC, we employ a number of PDI IEs (up to 20) in the PDR to support rich functionality needed, including firewalls, NATs, and per-flow QoS treatment (see Appendix. A for more details).

3.5 Resiliency through state replication

In L^{2}5GC, we have developed a thorough and novel resiliency framework particularly suited for the cellular environment where control and data plane traffic have substantial inter-dependency. We avoid having the UE re-establish the connection after a failure.

3.5.1 Replication and failover. We provide two levels of resiliency to support software failure (local resiliency) and node/link failure (remote resiliency).

• Local resiliency: We maintain a local replica of each NF. Once the replica NFs are initialized, they are kept in 'freezed' state, using cgroup freezer subsystem *consuming no CPU cycles* until the NF Manager issues a signal to wake up the NF. We use a no-replay scheme to synchronize the active and standby NFs and ensure an 'output commit' property once the UE event is completed. The NF (e.g., AMF) does not release any response unless the local replica is synchronized. Since the NFs are on the same system, they take negligible time (less than 5µs) for synchronization. For failover, once the NF Manager detects the failure of an active NF, it 'unfreezes' the standby, which is guaranteed to have a consistent checkpoint of NF state.

• Remote resiliency: While software failures are easier to recover with local resiliency, more importantly we address the case that a serving node becomes unreachable (e.g., link/node failure). The main feature of our resiliency design is to maintain external synchrony during replication, i.e., allow strict state consistency without impeding normal operation. We leverage both checkpointing and packet replay (similar to [27, 44, 56]) but account for both control *and* data packet recovery, to provide complete resiliency from node and link failures. The resiliency procedure is as follows:

(1) The Counter at the load balancer (LB) node (in Fig. 5) attaches a counter value to every outgoing message and maintains its copy in buffer (PacketLogger). To avoid the impact of interference between control and data if the buffer overflows, L^{2}5GC separates the packet logger into four different queues for UL-control, UL-data, DL-control, and DL-data packets. These buffered packets are replayed to reconstruct the state at the replica when a failure occurs and we need to recover the state updates lost between two consecutive checkpoints. The replica node checks the counter value of the packet at the front of each of the four queues and picks from the queue with a lowest counter value, so as to maintain the processing order while replaying the packets.

(2) The local replica at the primary node, which is already in sync with the primary copy, periodically sends the delta of state snapshot (thus reducing the update size) to the remote replica. Both primary and remote replicas maintain a record (counter) of the messages that are processed and synchronized. By utilizing the local replica for synchronization, our design ensures that normal operations are not impeded. We chose periodic state sync over per UE event sync (as done in Neutrino [27]) for two reasons: 1) to recover data packets lost during a failure, and avoid having end-points to retransmit the packets; and 2) the 5GC is expected to have a large number of control plane events, increasing the frequency of event-based checkpointing, which can degrade performance.

(3) Upon receiving the success ACK for state and counter sync from the remote replica, the primary node notifies to the LB to release the processed messages from its buffer.

If a primary node fails, the remote replica is 'unfreezed' with the last checkpointed state, and the remaining state is reconstructed by replaying packets present in the LB node.

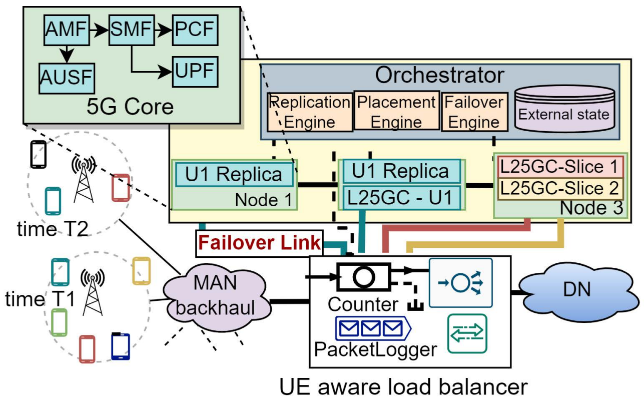


Figure 5: L^{2}5GC's resiliency framework with cloud-native deployment

3.5.2 Failure detector. The NF manager periodically (every few milliseconds) determines the status of all the registered active NFs. L^{2}5GC also leverages REINFORCE's method [44] of detecting node and link failure, using the configuration of the simplified Seamless BFD (S-BFD) [53] that has lower overhead for both link and node failure detection.

4 L^{2}5GC DEPLOYMENT STRATEGY

An important reason to move to software-based NF components for the 5GC is to improve scalability, and ease of deployment by using modern cloud-native frameworks of NFV, Kubernetes and Istio [30]. Our L^{2}5GC design also adopts the same functional goals as sought by 3GPP with the SBI in terms of vendor and implementation flexibility.

Scaling: A user session remains associated with a particular 5GC instance (we term this a 5GC unit) assigned at the establishment, as long as the user remains within the set of gNodeBs supported by the 5GC units instantiated in the same data center. A single 5GC unit can serve a large set of UEs in a serving region. A serving region can have multiple 5GC units. Our design assigns a UE session to a particular 5GC unit. This allows us to limit excessive state migration. Load is balanced by assigning new UE sessions to the appropriate 5GC unit based on its current load. Fig. 5 illustrates the affinity of a UE (*e.g.,* Blue UE) at time T1 and T2 to L^{2}5GC unit on Node 2. The main feature of our approach is to have consolidated 5GC unit instantiated as a service, rather than independent, individual NFs. A 5GC unit serves a number of neighboring gNBs in a metro area. Multiple 5GC units can be provisioned in the same data center to handle increases in user sessions. We employ a UE-aware LB (Fig. 5) to maintain the affinity of a UE to its serving 5GC unit. This allows us to scale 5GC units including control plane NFs without excessive overheads of moving user sessions and avoids the cost incurred for migrating the user state between L^{2}5GC units.

A single server can host multiple 5GC units. Network slices can be supported by logically assigning different service IDs. We leverage Receive Side Scaling [47] offered by modern NICs to segregate incoming packets into different receive queues based on a configurable hash. This allows received packet processing to be load balanced across multiple cores running different 5GC units and network slices.

Supporting Canary Rollouts: Our deployment strategy envisages the adoption of canary deployment of the 5G NFs. This is similar to the Istio service mesh, where Envoy proxies are configured to support the canary rollout of a new version of a service [30]. With the help of the NF manager, L^{2}5GC seamlessly allows the gradual roll out of a new version of a specific NF or a whole 5GC unit. The manager identifies any NF with its NF ID, and when a new version of the same service has started, it uses an instance ID to differentiate between two instances. It can be configured with the percentage of traffic to send to the particular version to support the principles of a canary rollout.

Scheduling: All cellular core functions in our architecture can also be containerized as cloud-native services and still utilize the shared memory communication interface. The containerized service-chain for the 5GC unit can be used with modern orchestration frameworks. Further, by leveraging placement engines (e.g., Kubernetes scheduler) that consider the affinity of the 5GC components, all the containerized NFs of a 5GC unit can be deployed on the same node. The design of such a placement engine is straightforward and only requires knowledge of the available capacity of the system.

5 EVALUATION & ANALYSIS

We measure the performance improvements of L^{2}5GC and compare it with free5GC. We also compare L^{2}5GC's SBI approach with alternatives for sharing data between NFs, such as FlatBuffers (Neutrino [27]) and Protobuf (Buyakar *et al.* [29]). We show the effectiveness of L^{2}5GC's control plane components and overall control plane latency reduction (§5.2); the improvement in data plane latency and throughput (§5.3); the improvements in data plane latency as a result of improved control plane processing (§5.4); finally, we show the improvement due to L^{2}5GC's resiliency framework (§5.5).

5.1 Experiment Setup

Our evaluation testbed consists of three Intel® Xeon® CPU E5-2697 v3 @ 2.60GHz servers running Ubuntu 20.04 with kernel 5.4.0-33-generic. Each server has an Intel 82599ES 10G Dual Port NIC. Server-1 and server-3 are configured as the RAN/UE and DN, respectively. Server-2 runs the L^{2}5GC core, including all the NFs in Fig. 1. We use MoonGen [35] on server-1 and server-3 for generating uplink and downlink traffic.

5.1.1 UE and RAN simulator. We implement a custom UE & RAN simulator for generating user events, based on the New Generation Application Protocol (NGAP) specified by 3GPP. We focus on four common UE events: *i)* UE registration, *ii)* Session request, *iii)* Handover, and *iv)* Paging (Idle-active). Currently, our simulator only implements the N1 & N2 interface of 5GC (*i.e.,* UE to AMF) over an SCTP socket. It does not implement the PHY channel (*e.g.,* mmWave). But, simulating radio characteristics can easily be done

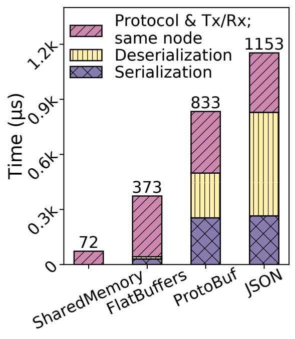


Figure 6: Serialization, Deserialization, protocol overheads

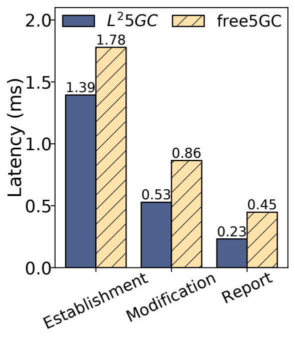


Figure 7: Latency of single control plane message between UPF/SMF

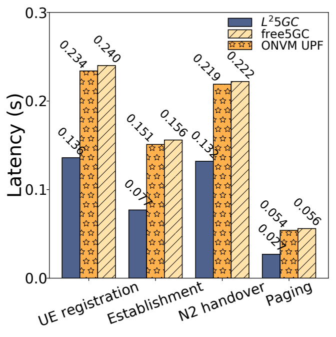


Figure 8: Total control plane latency for different UE events

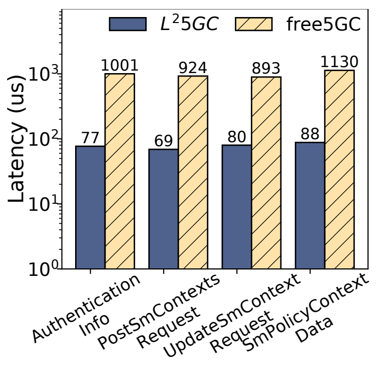


Figure 9: Communication speedup over HTTP

by integrating a model (e.g., ns-3's mmWave [48] model) with AMF over the SCTP socket.

5.2 Control plane Evaluation

In this subsection, we first assess the performance improvement from each control plane enhancement of $L^2$5GC, and then present overall improvement in control plane latency.

Improvement over HTTP: $L^2$5GC's shared memory communication unsurprisingly reduces the latency significantly for messages exchanged over the HTTP/REST interface (*e.g.,* used for AMF-AUSF, AMF-SMF communication) and PFCP (for SMF-UPF). Fig. 6 shows the cost of message serialization during a message exchange over the REST interface. We start two NFs on the same node and exchange a 'PostSmContextsRequest' message. Besides JSON (de facto format for REST API), we use different serializing structures discussed in recent proposals such as FlatBuffers (Neutrino [27]) and Protobuf (Buyakar *et al.* [29]) to compare against $L^2$5GC. We observed that even with optimized serializing structures such as FlatBuffers, there is still significant cost involved in serialization and overhead of message copy, context switch and protocol stack processing when services communicate over kernel sockets. $L^2$5GC completely mitigates these costs to achieve significant speedup for typical control plane messages, as shown in Fig. 9 (log scale) shows the speedup for selected control plane messages. We chose these messages due to their importance and frequent usage. On average, we observe a speedup of 13× over using the HTTP channel. While it is not surprising, the latency saving with $L^2$5GC's shared memory communication is crucial for this application.

Improvement over PFCP: We study the latency of UE-related control plane messages that are frequently exchanged over the PFCP channel between SMF and UPF-C for critical UE events, *e.g.,* the PDU session establishment and modification. The 'SessionModification' message with 'UpdateFAR' IE is used to change forwarding behavior. Similarly, the 'SessionReportRequest' message notifies the SMF of the reception of any DL packet, thus initiating paging when UE is in sleep-mode. These messages influence control plane latency and thereby the data plane latency since they control when data packets can begin to flow after a state change. The comparison between $L^2$5GC and free5GC is shown in Fig. 7. $L^2$5GC achieves 21% ∼ 39% latency reduction compared to free5GC for both the session establishment and modification messages, benefiting from utilizing the shared memory to support messaging over the N4 interface. Compared to in-kernel UDP socket messaging, the shared memory consumes less overhead and helps speed up the control plane messaging between the SMF and UPF-C.

Overall Control Plane Performance: We examine the overall reduction in control-plane latency for user *events* such as UE registration, PDU session request, N2 Handover (both $L^2$5GC and free5GC use our smart buffering and direct handover), and Paging. We begin by running vanilla free5GC that uses the kernel-driver based UPF for the data plane and compare with the "ONVM-UPF". ONVM-UPF runs the OpenNetVM-based data plane function (*i.e.,* UPF) but the rest of the NFs use the original free5GC implementation with the REST-based control plane functions (thus only the SMF-UPF, or N4 interface, runs on OpenNetVM). We evaluated the control plane operation latency with 1 and 2 users performing control plane tasks simultaneously. We see no perceptible difference. We report the average time for specific control procedures for each UE event in Fig. 8 (labeled ONVM-UPF). Compared to vanilla-free5GC, ONVM-UPF shows a slight improvement as it eliminates a data copy in the SMF to UPF communication. Retaining the N4 interface's use of PFCP in our design of $L^2$5GC makes our UPF universally compatible with any SMF implementation. Further, it helps when not all the NFs are on the same node and the SMF and UPF are on different devices. The reduction in latency primarily comes from reduced communication latency, while the handler-processing latency is common (and is a significant part of the latency) for both free5GC and $L^2$5GC. $L^2$5GC has a substantially lower completion time (almost 2×) compared to free5GC, as shown in Fig. 8, because of the efficient processing of the frequent control plane messages. With the improved physical layer access latency (beam alignment and link acquisition can be of the order of 1-10ms [39]), this reduction in handover latency of $L^2$5GC (of the order of 100 ms), is even more significant and provides meaningful QoE improvement to user applications (shown in §5.4.1).

5.3 Data plane Evaluation

We now demonstrate the data plane performance improvement obtained from leveraging the benefits of NFV platform. We use the MoonGen [35] traffic generator on server-1 and server-3 to

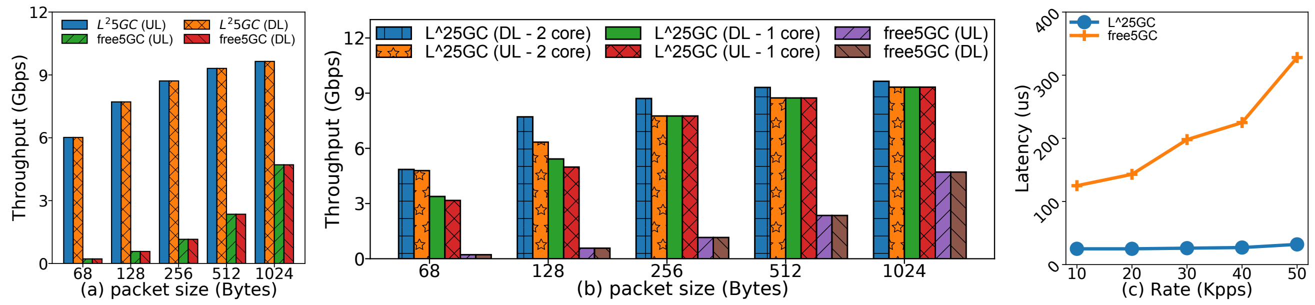


Figure 10: Data plane performance comparison between L^{2}5GC and free5GC with increasing packet sizes: Throughput with (a) uni-direction traffic (UL only and DL only) and (b) bi-direction traffic; (c) Mean end-to-end latency.

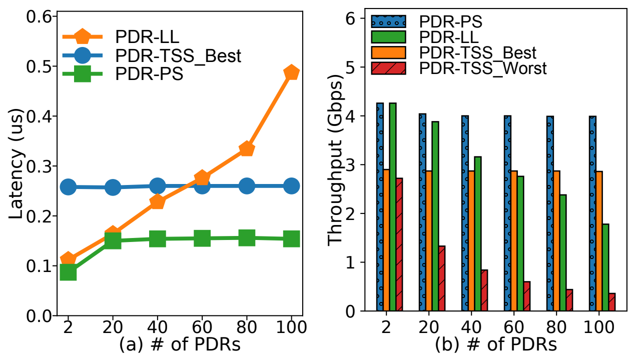


Figure 11: PDR lookup comparison: (a) PDR lookup latency with increasing # of PDR rules; (b) Throughput with increasing # of PDR rules (Packet size 68 bytes)

generate UL and DL traffic. We also demonstrate the benefit of the fast PDR lookup , with an increasing number of PDRs for a single user session in L^{2}5GC.

• **Packet forwarding performance:** Fig. 10(a) and 10(b) compare the throughput between L^{2}5GC and free5GC. With a packet size of 68 bytes, L^{2}5GC achieves 27× throughput improvement for uni-directional traffic in both UL and DL traffic compared to free5GC (Fig. 10(a)), demonstrating the performance benefits of a high performance DPDK-based NFV platform (using one CPU core). The free5GC's throughput improves slightly at large packet sizes due to a reduction (in proportion) in the fixed per-packet processing overhead. When UPF-U is assigned two CPU cores, L^{2}5GC's throughput improves by a factor of 4 over free5GC, even for 1024-byte packets. We expect that by allocating a sufficient number of CPU cores to the UPF-U as needed, the data plane throughput of L^{2}5GC can be further improved. Fig. 10(c) shows the mean end-to-end packet latency. free5GC has high latency due to interrupt-based packet processing in the kernel, even if the packet rate is relatively low. On the other hand, L^{2}5GC leverages DPDK's poll-mode kernel bypass technique to process packets in userspace without any data copies, thus achieving much lower latency. L^{2}5GC's latency remains relatively flat throughout the range we tested.

• **Supporting 40Gbps links:** To support 40 Gbps line rate on the datapath, we need to increase the number of CPUs allocated to the UPF. With 1 core assigned to UPF and NF manager each, the UPF can only achieve a forwarding rate of 10 Gbps with MTU size packets. To saturate a 40Gbps link, we increased the number of CPUs assigned to the UPF and NF manager (Rx and Tx) thread from 1 to 4. With 2 cores each to UPF and NF manager (Rx and Tx), the forwarding rate goes up to 28 Gbps. With 4 cores each to the UPF and NF manager (Rx and Tx) thread, we can comfortably operate at 40 Gbps.

• **PDR lookup comparison:** We compare the lookup performance between PDR-TSS, PDR-PS, and PDR-LL. We extend ClassBench [58] to generate PDRs (including a total of 20 PDI IEs Packet Detection Information (PDI)) for evaluation. Two different scenarios of PDR-TSS are studied: *Best case* (PDR-TSS_Best) and *Worst case* (PDR-TSS_Worst). For PDR-TSS_Best, given N PDR rules, all the N PDRs are inserted into a single sub-table. One hash table lookup finds the target PDR. For PDR-TSS_Worst, each sub-table has only one PDR inserted. Thus, it can take up to N sub-table lookups to find a matching PDR. In PDR-TSS_Worst, we assume the match is in the last sub-table. For PDR-LL, we assume the packet randomly matches a PDRs in the second half of list.

Fig. 11(a) and 11(b) compare the PDR lookup latency and throughput. We omit the case of "PDR-TSS_Worst" from Fig. 11(a) since the latency goes up rapidly to 2.9us, out of the range of the graph for just 100 rules. This poor performance is primarily because of the penalty of software hashing, and having to go through a number of PDR sub-tables. However, despite the increased number of PDRs, PDR-TSS_Best has a constant latency of ∼ 0.26us. It also has a lower latency than PDR-LL, when there are more than 60 PDRs. With 2 PDRs per session, PDR-LL may be acceptable as its throughput and latency are competitive. However, as we evolve to a packet-oriented environment, concerns about throughput degradation can be a significant issue as the number of PDRs per session grows. Finally, PDR-PS achieves the best performance of all, in terms of both latency and throughput.

• **PDR update comparison:** Updating the PDR table is also an important consideration as it impacts events such as session modification, *etc.* We compare the PDR update performance between the alternatives by measuring the average latency for a single PDR update, and repeating it 50 times. PDR-TSS (1.41μs) and PDR-PS (6.14μs) have a higher update latency than PDR-LL (0.38μs), but

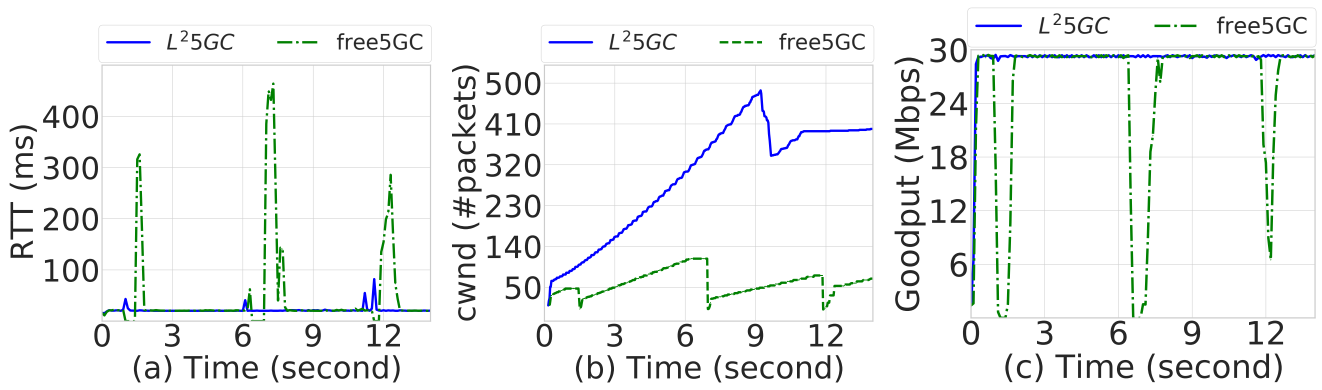


Figure 12: Impact of handovers on application: Round-trip delay (left), congestion window (middle), goodput (right).

the difference is not substantial. Therefore, we choose PDR-PS because of its reduced forwarding overhead and thereby the improved throughput, as shown in Fig. 11(b).

5.4 Impact of control plane on Application Performance (and data latency)

5.4.1 Control Plane Impact on Application Performance. We study the impact of L^{2}5GC's control-plane improvements on the data plane by evaluating the page load time (PLT) for a webpage on a Firefox browser at the UE, as intermittent handovers occur. The webpage contains a few high-resolution images (each ~15MB), javascript libraries, and CSS files, hosted on the DN node. We disabled browser caching to avoid artificial speedup because of our limited web content being cached, and used six parallel TCP connections (Firefox's default). We use the PLT reported by Firefox Developer Tools (Network Monitoring) [17], as it reflects the delay experienced by the user. We set the aggregate bottleneck bandwidth as 30Mbps and round-trip delay (RTT) of 20ms. We use Wireshark to calculate goodput and RTT, and the ss utility to extract congestion window *(cwnd)*.

During a handover, DL packets are buffered at 5GC and experience an additional delay. free5GC incurs up to 463ms delay (Fig. 12(a)), which is higher than the minimum retransmission timeout (200ms) in Linux, leading to ~1500 spurious retransmissions out of ~80K packets with the consequent reduction of each connection's *cwnd*. L^{2}5GC incurs at most 96ms of extra delay due to its faster handover, and there are no timeouts (22 packets are retransmitted, but none were dropped at the 5GC). The total PLT is ~32 seconds with free5GC, while it is ~28 secs. with L^{2}5GC, a 12.5% improvement in the user QoE illustrating the utility of L^{2}5GC.

5.4.2 Impact on UDP streams. We study the impact of the control plane on UDP streams by triggering two UE events (paging and handover) separately, to demonstrate the impact of control plane latency on the data plane's latency. Packets are sent at a constant rate of 10 Kpps for a UE session. To estimate the impact of buffering, we use a buffer of 3K packets at the UPF.
Paging: Fig. 13 (Y-axis in log-scale), and Table 1 show the RTT experienced by packets (we measure RTT of packets sent from and ack'd.

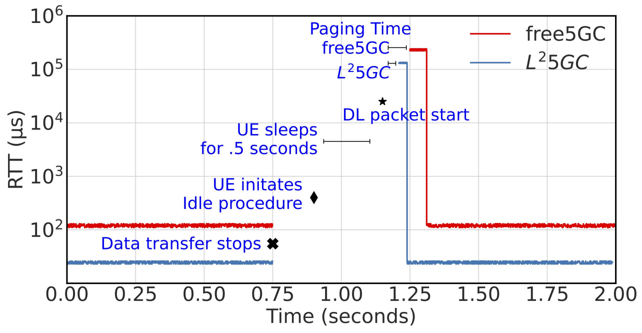


Figure 13: Data plane latency during paging event

Table 1: Control and data plane behavior (paging event)

	Base RTT	Paging time	RTT after paging	# Pkts experience higher RTT
free5GC	116 us	59 ms	63 ms	608
L^{2}5GC	25 us	28 ms	30 ms	294

back to the generator). When data transfer stops, the UE goes into the sleep state to save battery. Once the UPF receives a DL packet, it initiates a paging event to wake up UE. Until the paging event is completed, all the subsequent DL packets have to be buffered in the 5GC. The longer it takes for the event to complete, the higher the queuing and delay experienced by packets. In both free5GC and L25GC, packets experience a higher delay for a period until the queue drains after data starts flowing. The RTT goes up from 116 μsec to 63 ms with free5GC. L^{2}5GC is distinctively better than free5GC both in terms of the base RTT and the RTT after paging. With L^{2}5GC, the base RTT goes from 25 μsec (base RTT is 4× better with L^{2}5GC, because of its kernel bypass zero copy delivery and user space processing) to 30 ms with L^{2}5GC. In addition, paging in L^{2}5GC completes in about half the time, and less than half of the packets experience an increased RTT compared to free5GC.
Handover (HO): Fig. 14 (log-scale Y-axis) and Table 2 show the RTT experienced by data packets during the handover event. We perform

two distinct experiments: expt. (i) uses a single UE session with one flow; expt. (ii) uses multiple UE sessions sending data concurrently, while one UE performs handover. In both experiments, the UE initiates a handover request at 1 second and the UPF starts to buffer packets and the SMF provisions PDR and FAR rules to buffer the incoming packets at the UPF. In expt. (i), with the kernel-based free5GC handover takes longer, resulting in more DL packets being buffered (864 packets more than L^{2}5GC). Packets experience an increased data plane RTT of 242ms for free5GC compared to 132ms for L^{2}5GC. For expt. (ii), the handover time for kernel-based free5GC is quite a bit more than L^{2}5GC. Moreover, since multiple flows are being forwarded by the UPF, it results in an increased RTT of 305ms (free5GC) for all the data packets, compared to 137ms (L^{2}5GC). More packets see this buffering delay (1313 packets more for free5GC than L^{2}5GC). Up to 43 packets are dropped in free5GC, (0 in L^{2}5GC) even with a 3K buffer.

Estimating Smart Buffering benefit: We assess the benefit of our smart handover approach compared to 3GPP's hairpin routing, for packet drops (N_{drop}) and one-way delay (t_{OWD}), UPF to UE:

$$N_{drop} = DL_{rate} \times t_{HO} - Q_{length} \tag{1}$$

$$t_{OWD} = \begin{cases} t_{HO} + t_{UPF,GNB_t} & L^2 5GC \\ t_{HO} + t_{UPF,GNB_s} + t_{GNB_s,UPF} + t_{UPF,GNB_t} & \text{3GPP} \end{cases} \tag{2}$$

•*Packet Drop:* We evaluate two cases: case (i) allocating equal buffers (500 pkts) at gNB (at the GNB_s for 3GPP handover) and UPF (our smart handover), and case (ii) allocating higher buffering at UPF (1500 pkts), with GNB_s (500 pkts) (likely more common). We use the handover time (t_{HO}) measured, 130 ms as in Table 2, and DL data rate of 10 Kpps. In case (i), 3GPP's routing and L^{2}5GC's direct handover both experience a similar packet loss of ~800 packets during the handover (based on Eq. 1). In case (ii), the UPF sees no packet

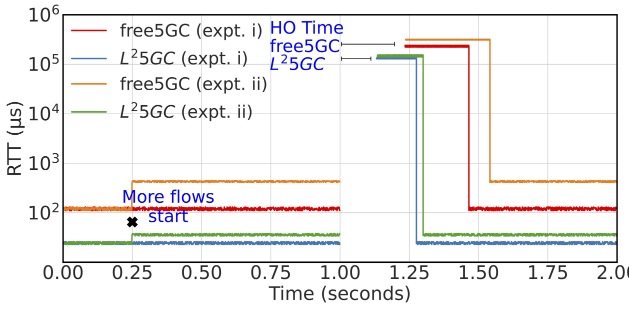


Figure 14: Data plane latency during handover (HO) event (log-scale Y-axis)

Table 2: Control and data plane behavior (HO event)

	Base RTT	**HO time**	**RTT after handover**	**#Pkts experience higher RTT**	**#Pkts Dropped**
free5GC (expt. i)	118 us	227ms	242ms	2301	0
L^{2}5GC (expt. i)	24us	130ms	132ms	1437	0
free5GC (expt. ii)	425us	231ms	305ms	3092	43
L^{2}5GC (expt. ii)	39us	132ms	137ms	1779	0

loss due to increased buffer size, while GNB_s still experiences a similar loss of ~800 packets for 3GPP handover.

•*One-Way Delay:* The 3GPP-based forwarding requires packets to traverse back to 5GC before being forwarded to the target gNB, resulting in additional delay. We assume the propagation delay to be 10*ms* from UPF to gNBs (t_{UPF,GNB_s} *etc.*) and use handover time as in Table 2. Using Eq. 2, the 3GPP handover sees an additional overall delay of 20 ms than L^{2}5GC's smart buffering approach.

The increase in the RTT, during paging and handover events (among others) may affect higher layer protocols, as seen in §5.4.1, (*e.g.,* potential TCP spurious timeouts) degrading the overall user QoE. The latency reduction by 2× with L^{2}5GC mitigates this impact. Further, the gNB is likely to have less buffering (*e.g.,* small cells), resulting in packet drops, thereby wasting processing throughout the entire data path (5GC, gNB). The smart buffering in L^{2}5GC's UPF mitigates this, with more buffering (while being cognizant of buffer-bloat [37]), without additional control plane messaging.

5.5 Impact of Failure recovery

We evaluate the efficiency of the resiliency framework of L^{2}5GC and free5GC by showing the impact of failure, both on the control and data planes (*e.g.,* when the failure occurs as UE undergoes handover). We compare L^{2}5GC's resiliency (failover) with the standard 3GPP approach that requires the UE to reattach to an alternate 5GC through the target gNB, using the free5GC implementation. We use the topology shown in Fig. 5, with three instances of L^{2}5GC (primary, local and remote replicas). We use flent [40] for TCP based data transfer (bandwidth limited to 30Mbps to a single UE). We highlight the penalties from the failure, even with a modest transfer rate. The 3GPP approach to reattach discards all the data packets when 5GC fails. On the other hand, during the failure, L^{2}5GC buffers the data and control packets in its replay buffer (in separate queues, so control packets are not dropped if the replay buffer overflows).

5.5.1 Impact on Control Plane. We consider the case when there is a failure occurring while there is an ongoing handover event. L^{2}5GC potentially makes the failure entirely transparent to the user. With L^{2}5GC, we use a probe agent at the LB node to detect that the 5GC instance is unreachable. It takes less than 0.5ms for the probe agent for the detection of the failure. For the 3GPP case, it is necessary to notify the UE of the failure, and initiate a reattach. For this experiment, we assume that the failure detection time for the free5GC/3GPP option is also 0.5ms before the reattach is initiated.

Even with a failure, the handover procedure in L^{2}5GC only takes a few additional milliseconds, taking 134 ms, instead of 130 ms (without failure). It is substantially faster than 3GPP's reattach-based approach, which takes 401 ms for completing the handover in the presence of a failure. Breaking down the additional operations performed during failure, L^{2}5GC consumes 2 ms for re-routing, and 3 ms for the state re-construction using packet replay, with some overlap between the two.

5.5.2 Impact on Data Plane. With the 3GPP's reattach, all incoming packets (~121 packets) are dropped during failure. With the resulting retransmit timeout (potentially multiple timeouts because of the long delay to reattach), the receiving UE does not receive packets

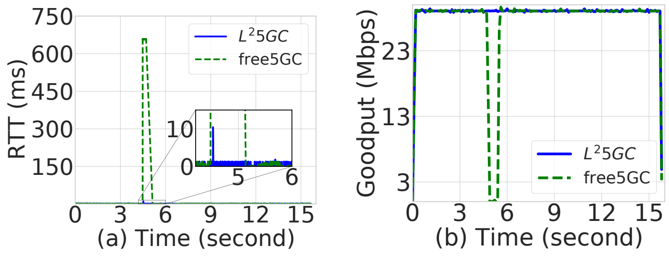


Figure 15: 5GC failover: data plane performance

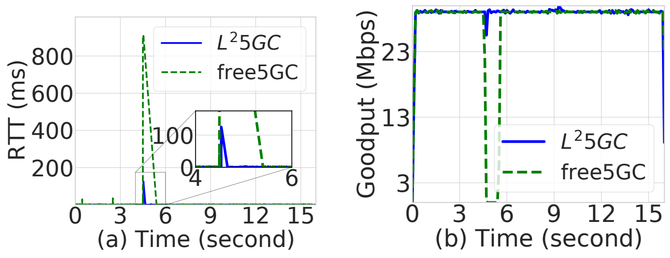


Figure 16: 5GC failover during handover: Control *and* data plane impact

for an interval, and has to reattach to a backup 5GC after notification. TCP also sees a drop in throughput as the sender's congestion window (cwnd) drops. The larger RTT is shown in Fig. 15(a), and the degraded goodput in Fig. 15(b). In contrast, $L^2$5GC seamlessly synchronizes the control and forwarding state with the remote replica. The replay buffer queues incoming packets (in 4 queues: control and data, for UL and DL, separately), thus avoiding data loss. $L^2$5GC's LB node replays the buffered data packets to the replica. TCP maintains its high throughput throughout (Fig. 15(b)). A small number of packets experience slightly increased RTT due to re-routing and packet replay (see zoomed inset in Fig. 15(a)).

5.5.3 5GC Failure Recovery: ongoing control event plus data transfer. We initiate a handover during an ongoing TCP transfer and then fail the links to the primary 5GC unit (at 4.5 seconds in Fig. 16(a)), disrupting both control and data flow to/through it. We assume that half of the handover is executed prior to the failure. The result shows that $L^2$5GC's resiliency framework transparently handles the 5GC failure and still maintain regular control and data plane performance (as in §5.4.1). During handover, both $L^2$5GC and free5GC buffer incoming DL packets. However, $L^2$5GC also buffers control (handover) and data packets in its packet replay buffer (at the LB). When the 5GC fails partway through the handover, $L^2$5GC seamlessly replays the control packets and forward the buffered data packets with minimal overhead. Data packets experience a slightly higher RTT (highlighted in the zoomed inset of Fig. 16(a)). On the other hand, the 3GPP-standard based approach waits to reattach after failure, resulting in all buffered packets being lost and requiring retransmission, thus degrading goodput (see Fig. 16(b)). (as in §5.5.2). Without the control and data plane coupling of $L^2$5GC (*e.g.,* as in Neutrino [27]), even if the control plane state is synchronized, we will continue seeing data packet loss and reduced throughput.

6 CONCLUSION

$L^2$5GC creates a low latency 5GC framework by exploiting the advantages of high-performance NFV-based platforms. Compared to the 3GPP-based free5GC, $L^2$5GC has better performance of both the control plane and data planes. $L^2$5GC consolidates the 5GC NFs on the same node. It simplifies the SBI and N4 interfaces based on the userspace shared memory available in a OpenNetVM-based NFV platform, fundamentally reducing the latency of the 5G control plane. Compared to free5GC, $L^2$5GC reduces the latency of several control plane events by up to 51% on average. Due to tight interdependence between the control and data planes, $L^2$5GC's faster handover completion avoids spurious timeouts and retransmission of *data packets*. Thus, $L^2$5GC improves user QoE (PLT) by 12.5% for a browser accessing even a simple web page (but with many images).

In addition to improving control plane performance, $L^2$5GC improves the data plane performance by leveraging DPDK's userspace packet processing. $L^2$5GC achieves 27× and 15× improvement in the throughput and latency compared to free5GC. Since cellular networks are evolving and will likely have a large number of PDRs, $L^2$5GC accommodates this by implementing an advanced PDR lookup mechanism and speeds up lookup latency by 20× compared to the linear search recommended by 3GPP. Additionally, $L^2$5GC supports smart buffering for handover, avoiding the hairpin routing through the 'old' source base station. The paging latency in $L^2$5GC sees at least a 2× reduction compared to free5GC. Our experiments show seamless continued data plane operation during failure restoration of 5GC NFs. We have released $L^2$5GC as open-source on Github, to further research on high performance NFV-based cellular cores for 5G and beyond. The code is available at: https://github.com/nycu-ucr/l25gc.

This work does not raise any ethical issues.

ACKNOWLEDGMENTS

We thank the US NSF for their generous support through grant CRI-1823270. This research was also sponsored in part by the OUSD(R&E)/RT&L and was accomplished under Cooperative Agreement Number W911NF-20-2-0267. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARL and OUSD(R&E)/RT&L or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This work was also supported in part by the Ministry of Science and Technology of Taiwan under grant numbers MOST 111-2218-E-A49-023, 110-2224-E-A49-002, and 108-2221-E-009-042-MY3. We thank our shepherd, Prof. Barath Raghavan, and the reviewers for their valuable suggestions and comments.

We thank Han-Sing Tsai and Yu-Sheng Liu of National Yang Ming Chiao Tung University, Taiwan, for their tremendous help in setting up $L^2$5GC for the artifact evaluation, and the reviewers of the artifact evaluation committee for their efforts.

REFERENCES

[1] 2008. Protocol Buffers. https://developers.google.com/protocol-buffers. [ONLINE].
[2] 2016. gRPC: A high performance, open source universal RPC framework. https://grpc.io/. [ONLINE].
[3] 2018. gRPC on HTTP/2 Engineering a Robust, High-performance Protocol. https://grpc.io/blog/grpc-on-http2/. [ONLINE].
[4] 2022. 3GPP TS 24.502: Access to the 3GPP 5G Core Network (5GCN) via non-3GPP access networks. https://www.etsi.org/deliver/etsi_ts/124500_124599/124502/15.00.00_60/ts_524502v150000p.pdf. [ONLINE].
[5] 2022. 3GPP TS23.401: General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=849. [ONLINE].
[6] 2022. 3GPP TS23.501 Section 4.2: Architecture reference model. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144. [ONLINE].
[7] 2022. 3GPP TS23.502: System architecture for the 5G System (5GS). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3145. [ONLINE].
[8] 2022. 3GPP TS29.244 Section 5.2.1: Interface between the Control Plane and the User Plane nodes . https://www.etsi.org/deliver/etsi_ts/129200_129299/129244/15.05.00_60/ts_129244v150500p.pdf. [ONLINE].
[9] 2022. 3GPP TS29.501: Principles and Guidelines for Services Definition. https://www.etsi.org/deliver/etsi_ts/129500_129599/129501/15.00.01_60/ts_129501v150001p.pdf. [ONLINE].
[10] 2022. 5G NR QoS Architecture, QoS Attribute and QoS Flow. https://www.techplayon.com/5g-nr-qos-architecture-qos-attribute-and-qos-flow/. [ONLINE].
[11] 2022. Aether. https://opennetworking.org/aether/
[12] 2022. Data Plane Development Kit. https://www.dpdk.org/. [ONLINE].
[13] 2022. DPDK's Multi-process Support. https://doc.dpdk.org/guides/prog_guide/multi_proc_support.html. [ONLINE].
[14] 2022. free5GC. https://www.free5gc.org/. [ONLINE].
[15] 2022. Go Developer Survey. https://blog.golang.org/survey2020-results. [ONLINE].
[16] 2022. Magma: A Modern Mobile Core Network Solution. https://magmacore.org/
[17] 2022. Mozilla firefox: Network Monitor. https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor. [ONLINE].
[18] 2022. NextEPC. https://nextepc.org/. [ONLINE].
[19] 2022. Open5GS. https://open5gs.org/. [ONLINE].
[20] 2022. OpenAirInterface. https://openairinterface.org/. [ONLINE].
[21] 2022. OpenAirInterface 5G CN. https://openairinterface.org/oai-5g-core-network-project/. [ONLINE].
[22] 2022. OpenAPI Generator. https://OpenAPITools.org. [ONLINE].
[23] 2022. Packet detection rule specification. https://www.etsi.org/deliver/etsi_ts/129200_129299/129244/15.05.00_60/ts_129244v150500p.pdf. [ONLINE].
[24] 2022. SD-core. https://opennetworking.org/sd-core/
[25] 2022. Travelping Homepage. https://www.travelping.com/. [ONLINE].
[26] 2022. User Plane Gateway (UPG) based on VPP. https://github.com/travelping/upg-vpp. [ONLINE].
[27] 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.*
[28] NGMN Alliance. 2022. 5G white paper. https://www.ngmn.org/work-programme/5g-white-paper.html. [ONLINE].
[29] 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 *IEEE Conference on Network Softwarization.*
[30] Lee Calcote and Zack Butcher. 2019. *Istio: Up and Running: Using a Service Mesh to Connect, Secure, Control, and Observe.* O'Reilly Media.
[31] Cisco. 2022. Cisco annual internet report (2018–2023) white paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.
[32] Marius Corici, Fabricio Gouveia, Thomas Magedanz, and Dragos Vingarzan. 2010. Openepc: A technical infrastructure for early prototyping of ngmn testbeds. In *International Conference on Testbeds and Research Infrastructures.* Springer.
[33] Levente Csikor, Dinil Mon Divakaran, Min Suk Kang, Attila Kőrösi, Balázs Sonkoly, Dávid Haja, Dimitrios P Pezaros, Stefan Schmid, and Gábor Rétvári. 2019. Tuple space explosion: A denial-of-service attack against a software packet classifier. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies.* 292–304.
[34] Yaozu Dong, Zhao Yu, and Greg Rose. 2008. SR-IOV Networking in Xen: Architecture, Design and Implementation.. In *Workshop on I/O Virtualization*, Vol. 2.
[35] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference.* 275–287.
[36] Telefonaktiebolaget LM Ericsson. 2022. Integrated Packet Core Firewall. https://www.ericsson.com/en/core-network/5g-core/packet-core-firewall. [ONLINE].
[37] Jim Gettys. 2011. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing* 15, 3 (2011), 96–96.
[38] GSMA Intelligence. 2014. Understanding 5G: Perspectives on future technological advancements in mobile. *White paper* (2014), 1–26.
[39] Haitham Hassanieh, Omid Abari, Michael Rodriguez, Mohammed Abdelghany, Dina Katabi, and Piotr Indyk. 2018. Fast millimeter wave beam alignment. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication.* 432–445.
[40] Toke Høiland-Jørgensen, Carlo Augusto Grazia, Per Hurtig, and Anna Brunstrom. 2017. Flent: The flexible network tester. In *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools.* 120–125.
[41] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. 2014. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14).* USENIX Association, Seattle, WA, 445–458. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/hwang
[42] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. 2014. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14).* USENIX Association, Seattle, WA, 445–458. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/hwang
[43] Jain et al. 2022. Evolving to 6G: Improving the Cellular Core to lower control and data plane latency. In *1st International Conference on 6G Networking (6GNet 2022).* IEEE.
[44] Sameer G. Kulkarni, Guyue Liu, K. K. Ramakrishnan, Mayutan Arumaithurai, Timothy Wood, and Xiaoming Fu. 2020. REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization-Based Services. *IEEE/ACM Transactions on Networking* 28, 2 (2020). https://doi.org/10.1109/TNET.2020.2969961
[45] 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.
[46] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural packet classification. In *Proceedings of the ACM Special Interest Group on Data Communication.* 256–269.
[47] Srihari Makineni, Ravi Iyer, Partha Sarangam, Donald Newell, Li Zhao, Ramesh Illikkal, and Jaideep Moses. 2006. Receive side coalescing for accelerating TCP/IP processing. In *International Conference on High-Performance Computing.* Springer.
[48] Marco Mezzavilla, Sourjya Dutta, Menglei Zhang, Mustafa Riza Akdeniz, and Sundeep Rangan. 2015. 5G mmWave module for the ns-3 network simulator. In *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems.* 283–290.
[49] 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).
[50] 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.
[51] Binh Nguyen, Tian Zhang, Bozidar Radunovic, Ryan Stutsman, Thomas Karagiannis, Jakub Kocur, and Jacobus Van der Merwe. 2018. ECHO: A reliable distributed cellular core network for hyper-scale public clouds. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking.* 163–178.
[52] Edmund B Nightingale, Kaushik Veeraraghavan, Peter M Chen, and Jason Flinn. 2008. Rethink the sync. *ACM Transactions on Computer Systems* 26, 3 (2008).
[53] C. Pignataro, D. Ward, and N Akiya. 2016. Seamless Bidirectional Forwarding Detection (S-BFD). RFC 7881, https://datatracker.ietf.org/doc/html/rfc7881.
[54] 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.* 348–361.
[55] Mahadev Satyanarayanan. 2017. The emergence of edge computing. (2017).
[56] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. Rollback-Recovery for Middleboxes. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 227–240. https://doi.org/10.1145/2829988.2787501
[57] Venkatachary Srinivasan, Subhash Suri, and George Varghese. 1999. Packet classification using tuple space search. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication.* 135–146.
[58] David E Taylor and Jonathan S Turner. 2007. Classbench: A packet classification benchmark. *IEEE/ACM transactions on networking* 15, 3 (2007), 499–511.
[59] Sorrachai Yingchareonthawornchai, James Daly, Alex X Liu, and Eric Torng. 2016. A sorted partitioning approach to high-speed and fast-update OpenFlow classification. In *2016 IEEE 24th International Conference on Network Protocols.*
[60] 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.

APPENDICES

Appendices are supporting material that has not been peer-reviewed.

A INFORMATION ELEMENTS IN PACKET DETECTION RULE (PDR)

Table. 3 shows PDI IEs used in packet classification in UPF.

Table 3: Information elements in PDR

Packet Detection Rule ID		
Precedence		
OuterHeaderRemoval		
Forwarding Action Rule ID		
QoS Enforcement Rule ID		
Usage Reporting Rule ID		
Packet Detection Information	Source Interface	
	Local F-TEID	Tunnel Endpoint Identifier
		IPv4
		IPv6
		CHOOSE ID
	UE IP	IPv4
		IPv6
	Network instance	
	Application ID	
	QoS Flow ID	
	SDF Filter	Length of Flow Description
		Source IP
		Destination IP
		Source Port
		Destination Port
		Protocol
		Type of Service
		Security Parameter Index
		Flow Label
		SDF Filter ID

B IMPLEMENTATION DETAILS OF FREE5GC

The 3GPP specifications and several implementations have divided the cellular core network into two separate subsystems, for the control plane and the data plane. We describe the free5GC Kernel-based (non-NFV) implementation.

Control plane functions: All control plane components are implemented as microservices. These microservices interact using a REST API over HTTP/2 on a SBI. 3GPP specifies RESTful interfaces and datatypes for each component as an OpenAPI specification.[1] free5GC uses these OpenAPI specifications and OpenAPI Generator[2] to generate client and server interfaces for each of the 5GC control NFs. Golang [15] is used to implement these services. The RESTful APIs are implemented on top of Golang's inbuilt HTTP/2 server that utilizes the underlying kernel TCP stack. The user event procedure handlers are implemented according to the 3GPP specifications, with all of the essential control plane functions, such as AMF, SMF, AUSF, UDR, PCF. They communicate based on the 3GPP specifications. For example, the source function has to perform service discovery for a target function. Further, the control plane function, SMF, provides UPF the forwarding rules over a PFCP channel, running over a UDP socket. Subscriber information is stored in a MongoDB database, and accessed through the UDR NF.

Data plane functions: The data plane functions have two sub-components: *i)* control plane listener, *ii)* user data handler. The SMF (part of the control plane) provides UPF the forwarding rules over a PFCP channel running on top of a UDP socket. Based on 3GPP specifications, the control function sends the PFCP message, a type-length-value (TLV) encoded message. For user data handling, the kernel-based free5GC leverages the gtp5g kernel module that performs GTP encapsulation and decapsulation to implement the data plane forwarding functionality. This allows us to avoid packet copying to the user space. Further, the rules from PFCP messages are conveyed to the kernel driver using Linux's netlink capabilities.

At first look, this is an intuitive and straightforward 3GPP-based implementation, providing flexibility. But it results in significant performance penalties, as we mentioned in the challenges described in §2. By carefully understanding the causes for performance degradation, we evolved from the strict implementation of the 3GPP specification in a kernel-based environment to the NFV-based 5GC in $L^2$5GC. Our optimizations improve both the control plane communication channel and at the same time substantially improves the data plane throughput.

C IMPACT OF CONTROL PLANE ON ONGOING TCP CONNECTION

We study the impact of the control plane procedure on the data plane by assessing ongoing TCP connections' behavior during handover. This experiment demonstrates that the control plane procedure completion time directly impacts the ongoing data plane and severely impacts QoE. We simulate a scenario where UE launches 10 TCP connections (equivalent to launching a few apps on a Smartphone) and undergo handovers every few seconds (representing mobility, e.g., UE traveling in a bus). These handovers are even more frequent because of small cells. We choose TCP since many user applications are built on reliable transport that require a timely response for continuous operation. We use flent to generate TCP traffic from server to client, with the aggregate bottleneck bandwidth as 100Mbps and RTT as 50ms.

During handover, all incoming DL packets are buffered at 5GC until handover is finished and experience an additional queuing delay. This results in increase in RTT (up to 130ms for $L^2$5GC and 328ms for free5GC), as shown in Fig. 17(a). However, in case of free5GC, upon not increasing the response, TCP senders falsely imply this delay as incipient congestion and result in the expiration of retransmission timeout (RTO), the minimum RTO in Linux is 200ms. As a result, TCP senders spuriously retransmit all the packets (60 packets every handover) after RTO expiration and reduce their sending rate by lowering their congestion window *(cwnd)*, Fig. 17(b). This causes frequent jitters in the application and affect overall goodput (Fig. 17(c)). Consequently, $L^2$5GC allows 442MB of data transfer compared to free5GC, which only transfers 416MB for the runtime of the experiment. These spurious timeouts do not appear in $L^2$5GC because of our fast handover completion time, thus sustaining a good QoE.

[1]The OpenAPI Specification [9] is a set of YAML files that represents a language-agnostic interface to RESTful APIs.
[2]OpenAPI Generator [22] is a tool to create API client libraries and server stubs from OpenAPI 2.0 and 3.x specification.

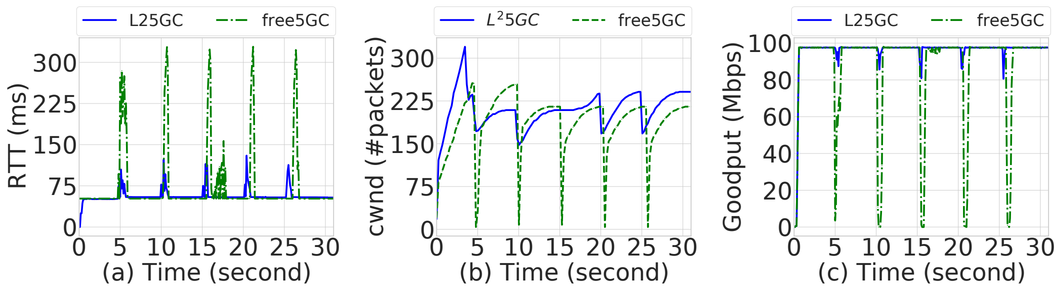


Figure 17: Impact of repeated Handover on data plane

D ABBREVIATIONS

Tables 4 lists various abbreviations based on 3GPP specification.

Table 4: Abbreviations table (general)

Abbreviation	Full form
5GC	5G Core Network
3GPP	3rd Generation Partnership Project
AMF	Access and Mobility Management Function
AUSF	Authentication Server Function
N3IWF	Non-3GPP Interworking Function
NSSF	Network Slice Selection Function
PCF	Policy Control Function
SMF	Session Management Function
UDM	Unified Data Management
UDR	Unified Data Repository
UPF	User Plane Function
DL	Downlink
DN	Data Network
DPDK	Data Plane Development Kit
EPC	Evolved Packet Core
FAR	Forwarding Action Rule
GTP	GPRS Tunnelling Protocol
IE	Information Elements
LB	Load Balancer
NF	Network Function
NFV	Network Functions Virtualization
NGAP	New Generation Application Protocol
PDR	Packet Detection Rules
PDR-LL	PDR Linear Search Algorithm
PDR-PS	PDR Partition Sort Algorithm
PDR-TSS	PDR Tuple Space Search Algorithm
PDU	Packet Data Unit
PFCP	Packet Forwarding Control Protocol
QoS	Quality of Service
RTT	Round-Trip Delay
SBI	Service-Based Interface
SDF	Service Data Flow
RAN	Radio Access Network
TEID	Tunnel Endpoint Identifier
UE	User Equipment
UP	Uplink

E ARTIFACT APPENDIX

Abstract

We have released our implementation of L²5GC and other artifacts in a Github repository. It contains the scripts for setting up the environment, our UE RAN simulator to generate various UE-related events, and a number of plotting scripts useful to generate the results presented in §5.

Scope

The scope of the artifact is to make implementation of L²5GC publicly available for the community and industry to experiment with L²5GC. It also includes the steps to setup and reproduce the experiment results presented in §5.

Contents

The artifact consists of the complete source code and all the necessary scripts for setting up L²5GC and free5GC:
- Source code for L²5GC, free5GC with gtp5g kernel driver (for free5GC based UPF), ONVM based UPF (onvm-upf)
- test-script3.0.5: simulator for generating UE-events
- Scripts to build and run L²5GC and free5GC
- Scripts to generate GTP encapsulated data plane pcap traces
- Misc. scripts for environment clean up and plotting results

Hosting

L²5GC is publicly available at https://github.com/nycu-ucr/l25gc (commit hash `74cb035`). The "README.md" covers the details about the artifact and includes steps for setting up the environment and reproducing the results presented in this paper.

Requirements

Hardware Dependencies: The node to run L²5GC needs to have at least 12 CPU cores and two DPDK-compatible NICs. This can achieve the demonstrated performance.
Software Dependencies: This artifact requires Ubuntu 20.04 with Linux kernel version 5.4, OpenNetVM v20.05. We use Moongen as the traffic generator running on another node, using pcap traces available with this artifact.
More details can be found in the artifact documentation.