Resilient Baseband Processing in Virtualized RANs with Slingshot

Nikita Lazarev[‡], Tao Ji[§], Anuj Kalia[†], Daehyeok Kim[†§], Ilias Marinos[†], Francis Y. Yan[†]
Christina Delimitrou[‡], Zhiru Zhang[¶], Aditya Akella[§ *]
[†]Microsoft, [‡]MIT, [§]UT Austin, [¶]Cornell University

ABSTRACT

In cellular networks, there is a growing adoption of virtualized radio access networks (vRANs), where operators are replacing the traditional specialized hardware for RAN processing with software running on commodity servers. Today's vRAN deployments lack resilience, since there is no support for vRAN failover or upgrades without long service interruptions. Enabling these features for vRANs is challenging because of their strict real-time latency requirements and black-box nature. Slingshot is a new system that transparently provides resilience for the vRAN's most performance-critical layer: the physical layer (PHY). We design new techniques for realtime workload migration with fast RAN protocol middleboxes, and realtime RAN failure detection. A key insight in our design is to view the transient disruptions from resilience events to RAN computation state and I/O similarly to regular wireless signal impairments, and leverage the inherent resilience of cellular networks to these events. Experiments with a state-of-the-art 5G vRAN testbed show that Slingshot handles PHY failover with no disruption to video conferencing, and under 110 ms disruption to a TCP connection, and it also enables zero-downtime upgrades.

CCS CONCEPTS

• **Networks** → **Wireless access points, base stations and infrastructure**; • **Computer systems organization** → **Availability**; **Reliability**.

KEYWORDS

Virtualized Radio Access Networks, Cellular Networks, Resilience, Fault tolerance

ACM Reference Format:
Nikita Lazarev, Tao Ji, Anuj Kalia, Daehyeok Kim, Ilias Marinos, Francis Y. Yan, Christina Delimitrou, Zhiru Zhang, Aditya Akella . 2023. Resilient Baseband Processing in Virtualized RANs with Slingshot. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3603269.3604841

[*]The first two authors contributed equally to this work during their internships at Microsoft.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604841

1 INTRODUCTION

Radio access networks (RANs) are a part of the cellular network infrastructure (e.g., LTE and 5G) that converts wireless signals between the user devices (called UEs, for "user equipment") and radio cell towers into data packets and vice versa. Today, the cellular industry is seeking to replace specialized RAN hardware with software systems running on commodity servers deployed in edge datacenters located close to the radio cell towers. This approach, called virtualized RAN (vRAN), has the benefits of reducing vendor lock-in, rapid feature development and upgrades, easier maintenance, and possibly lower costs [14, 49]. Some cellular network service providers, such as Verizon and Rakuten Mobile, have already deployed vRANs [5, 25], and others such as Vodafone [26] are currently in the process of adopting them. Figure 1 shows a typical vRAN deployment that statically provisions vRAN servers to handle specific radios.

Today's vRAN lacks resilience, with no support for fast failover or zero-downtime upgrades. These features are required to provide high availability for the cellular network which is a critical infrastructure for emergency services, public safety, and other mission-critical applications. Any given vRAN server is likely to crash every few months [28, 33], resulting in severe user downtime lasting over five seconds even if the radio is immediately reconnected to a backup vRAN using our fronthaul migration techniques (§8.1). Planned RAN upgrades happen as often as every day, and today they require manually pre-planned maintenance windows in which parts of the network are taken offline [61, 73]. A resilient vRAN system should migrate processing during failovers and upgrades to another server without significant downtime or network disruption.

Two characteristics of the vRAN software make it challenging to provide resilience: its realtime latency requirements, and its black-box nature. First, the vRAN must complete tasks in strict transmission time intervals (TTIs), measuring 500 µs in 5G's common deployment configurations. In comparison, existing resilience approaches based on virtual machine or container migration/replication techniques impose blackouts lasting several 100 ms [37, 59, 65, 67, 69, 71, 78]. Second, production-grade vRANs use extremely complex and often proprietary software written by domain experts. This makes it infeasible to modify the software to implement custom logic required by existing state replication–based network function resilience techniques [50, 62, 66]. This challenge is exacerbated by the vRAN consisting of modules developed independently by different vendors, such as the layer-1 (Physical Layer, or PHY) and layer-2 (or L2, including Media Access Control and Radio Link Control).

In this paper, we present Slingshot, a new system that takes the first step towards building the required systems support for vRAN resilience. Since the vRAN stack is modular, a practical way towards a resilient vRAN is to make each module independently resilient, exploiting the specific properties of the module. This paper focuses

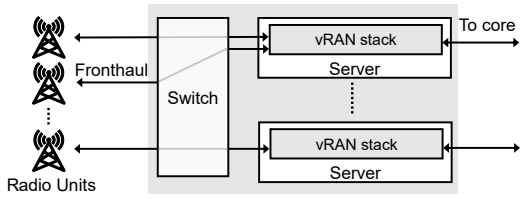


Figure 1: High-level view of a typical vRAN deployment today. Here, each server is statically provisioned to serve a certain subset of connected RUs.

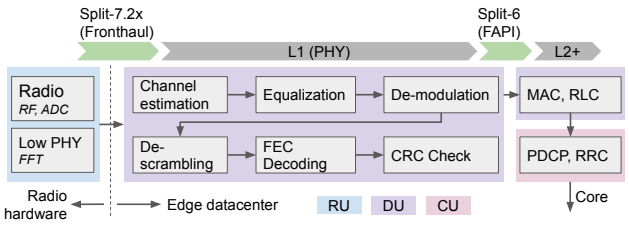


Figure 2: A simplified structural view of a standard vRAN stack with split option-7.2× and 6.

on the vRAN's PHY layer, which has the highest CPU cost and software complexity, and the tightest realtime latency deadlines among all vRAN layers. Slingshot requires no changes to existing vRAN software or hardware components, so it is incrementally deployable. Since we target the vRAN's lowest layer, the techniques we present can be fundamental building blocks for future resilience work on other vRAN layers.

Slingshot's design is based on our observation that the short-term vRAN computation or I/O impairments that can occur during resilience events—such as losing soft PHY state computed in a previous TTI, or dropping some fronthaul packets—are similar to routine wireless signal quality degradation; Slingshot leverages the cellular network's inherent resilience to bad signal quality to preserve connectivity despite short-term impairments. This allows us to design a clean and lightweight migration technique called *PHY migration* that moves the PHY processing to another server by properly managing the two types of traffic processed by the PHY: the PHY–RU fronthaul and the L2–PHY traffic. This technique enables zero-downtime planned upgrades, and, in combination with our novel RAN failure detector, also provides fast failover without significant user disruption.

To provide PHY migration in a transparent and inter-operable way, we design new middleboxes that act as shim layers between different vRAN layers. First, we design a programmable switch–based middlebox between the RU and PHY to manage the high-bandwidth fronthaul traffic without adding the latency and CPU overhead of a software-based alternative. Slingshot introduces two novel realtime in-switch components: one to migrate fronthaul traffic to/from the primary or secondary PHY, and another to detect PHY failures within a TTI. Our failure detection technique is based on the insight that all realtime vRAN layers send packet streams that can be used as natural heartbeats. Second, we design a software middlebox called Orion between the L2 and PHY layers. Orion maintains low-overhead hot standby secondary PHYs and initiates PHY migration by properly managing L2–PHY protocol messages.

We implement the fronthaul middlebox in P4-16 [15] and Python, and Orion in C++, and show that they work with an unmodified commercial vRAN stack consisting of Intel FlexRAN (PHY) [10] and CapGemini 5G (L2+) [6].

We evaluate Slingshot on a state-of-the-art vRAN testbed with end-to-end applications and microbenchmarks. We run a video conferencing application with Slingshot and show zero downtime during PHY failover, compared to 6.2 seconds without Slingshot. We also show that Slingshot moves PHY processing between servers orders of magnitude faster than pre-copy VM migration, and drops no TTIs during planned migrations. Using iperf [3], we demonstrate the end-to-end performance recovery after PHY failure with Slingshot is near-immediate for UDP, and takes 110 ms in the worst case for TCP connections. Finally, we show that Slingshot enables zero-downtime PHY upgrades.

2 MOTIVATION AND BACKGROUND

2.1 Motivation for RAN resilience

The cellular network is a critical infrastructure that must support high availability. Telecom deployments typically require five nines of uptime [56], which permits at most six minutes of downtime per year, and means that vRAN failover and upgrades must be handled with little user downtime. We find that in today's vRANs, these events can have a drastic impact on users' connectivity: without Slingshot, a 5G UE disconnects for 6.2 seconds on average when the vRAN fails, even if the RU is immediately reconnected to a standby vRAN using our in-switch middlebox (§8.1).

Resilience to faults. A resilient PHY must tolerate hardware or software faults. The limited public analyses of server hardware failures suggest mean-time-between-failure values from 10 days [28] to 60 days [33, Table 3]; repairs take several hours [28], violating the five-nines target. vRANs are particularly vulnerable to software crashes: similar to Turlapati and Bhat [70], our experience in operating a vRAN testbed (§8) finds crashes due to imperfect support for real-time applications in Linux (e.g., kernel thread starvation, and co-existence with non-realtime processes).

Resilience to planned upgrades. A key promise of virtualized RANs is easier roll-out new RAN features and updates, OS/security patches, and hardware upgrades. AT&T reports upgrading subsets of their RANs as often as every day, with pre-planned downtime windows for maintenance [61, 73]. Experience from large distributed system deployments shows that mechanisms for zero-downtime upgrades and maintenance are of paramount importance [32, 39]. Upgrades in today's vRANs involve significant user downtime, lasting several seconds.

2.2 A primer on vRAN deployments

Figure 2 shows a simplified view of the main components of the vRAN stack. Most of today's vRAN deployments, such as Rakuten Mobile [23], Vodafone [24] and Deutsche Telekom [9] are based on CPUs running Intel's FlexRAN PHY software. We target these architectures, although our design also applies to other hardware architectures (e.g., DSPs, FPGAs, or GPUs).

Hardware. A vRAN deployment consists of several radio units (RUs) connected to a nearby edge datacenter via fiber-optic fronthaul links. A switch in the edge datacenter connects an RU to a vRAN server; today, this mapping can only be changed rarely (e.g., when new RUs or servers are added). Dedicated commodity servers in the edge datacenter run the RAN's different layers as bare-metal or containerized Linux applications. Of these layers, the layer-1 (PHY) and layer-2 (Media Access Control and Radio Link Control) have strict real-time latency requirements. Higher layers of the vRAN stack as well as the core network do not have real-time latency requirements, and may therefore run in a larger datacenter farther away from the RUs and DUs.

Software. The vRAN consists of large and complex (e.g., several hundred thousand source lines of code) highly-optimized, multi-threaded software applications written by specialized vendors. The PHY performs compute-intensive signal processing tasks, including channel estimation, modulation/demodulation, and forward error correction. We use Intel's FlexRAN PHY [10], which is a production-grade 5G PHY implementation widely used in vRAN deployments. Our proposed techniques are also applicable to other software PHY implementations, such as those from OpenAirInterface [57] and srsRAN [17]. The L2 is primarily responsible for scheduling the frequency and time resources among users (UEs). This layer also connects to higher non-real-time vRAN layers, which in turn connect to the cellular core network. Several vendors provide L2 implementations, including CapGemini [6] and RadiSys [16]. Each process (e.g., PHY or L2) supports handling multiple RUs.

Functional splits and interfaces. To modularize complex RAN systems, the standards define "splits" specifying different ways of partitioning RAN functionality across software and hardware boundaries. A key tenet of vRANs is the use of open specifications that have broad adoption, allowing these components to interoperate. For the fronthaul interface between the PHY and RU, today's vRANs use the "O-RAN split option-7.2×" standard from the O-RAN consortium [12]. The fronthaul carries Ethernet packets containing IQ samples, which the PHY handles with low-latency userspace packet I/O. The PHY/L2 interface uses the FAPI ("Functional API") specification from the Small Cell Forum [19]. In tightly-coupled systems that co-locate the PHY and L2 on the same server, FAPI messages are carried over shared memory. In decoupled systems, which we believe are crucial for vRAN resilience, the PHY and L2 exchange FAPI messages over an Ethernet network with the "network FAPI" (nFAPI) protocol [20], which implements O-RAN's split option-6 [13].

2.3 Availability target

We target a cellular network downtime of under 10 ms during resilience events, which can be considered negligible for cellular deployments. For example, during frequent mobility events called "handovers" where a moving UE transfers from one cell to another, UEs typically experience larger downtime. Some measurements show that handovers happen as frequently as once every 70 seconds while walking, resulting in suspension of service for 24.7 ms on average [52]; today's 5G networks exhibit somewhat higher handover delays [52].

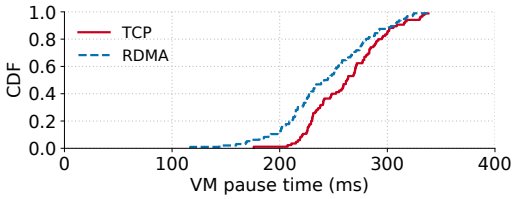


Figure 3: Distribution of VM pause time while migrating FlexRAN in a VM via TCP or RDMA. FlexRAN crashes in all runs.

By targeting a smaller downtime than handovers, we ensure that UEs do not experience unusual disruption during PHY resilience events. Note that resilience events are far rarer than handovers, and degraded performance (e.g., packet losses) for similar ≈10 ms timescales during them is also acceptable, similar to handovers [43, Fig 4].

2.4 PHY downtime with VM migration

While one can consider using general-purpose workload migration techniques such as virtual machine (VM) or container migration for PHY resilience, the PHY's real-time latency requirements preclude using these approaches. These approaches impose blackout periods of up to hundreds of milliseconds [37, 77], which causes UEs to de-synchronize and completely disconnect from the network [29].

Figure 3 shows our measurements of VM pause time while migrating a VM running FlexRAN; we describe the 5G cell's configuration in Section 8. These experiments use a simplified version of FlexRAN that does not use any PCIe devices (e.g., for fronthaul packet I/O or for FEC acceleration), which under-represents the actual migration time. We use QEMU/KVM [4, 11] as the hypervisor, which optimizes VM pause time during migration by using pre-copy techniques that iteratively transfer dirty memory pages. We perform 80 live migration experiments, using RDMA over 100 GbE to speed up migration [1]. Even with these optimizations, the median VM pause time is 244 ms, which is large enough for UEs to fully disconnect from the network after their Radio Link Failure timer (50 ms in our setup) expires [29]. This downtime happens because FlexRAN's signal processing continuously generates dirty memory pages, causing the hypervisor to pause the VM to ensure their consistency after the migration. We also observe that FlexRAN crashes during all migration runs, which is expected since the designers of vRAN's real-time layers optimize the software assuming a low jitter environment. For example, vRAN operators require the server platform to provide sub-10 µs thread interruption times under all circumstances [18], which is several orders of magnitude lower than the pause time with today's VM migration techniques.

3 OVERVIEW OF SLINGSHOT

Our goal is to design a resilience solution for the PHY layer of vRANs that provides the following three properties:

- **Minimal disruption to cellular connectivity:** During failover or upgrade events, user downtime must be less than 10 ms to keep the downtime comparable to routine handovers.

- **Transparency to existing vRAN components:** Since the vRAN stack is highly complex and different layers are usually provided by different vendors, Slingshot should not depend on one particular implementation, which reduces the design's applicability.
- **Low resource and performance overheads:** Given the limited compute resources available in edge datacenters and the need to keep costs low, the design must incur little compute overhead. Also, it must not violate the PHY's realtime latency requirements.

Slingshot meets these goals without any modifications to existing vRAN software. We build Slingshot on our observation that short-term vRAN computation or I/O impairments during resilience events, such as losing soft PHY state from a previous TTI or dropping some fronthaul packets, are similar to routine wireless signal quality degradation (§4). By leveraging the cellular network's inherent resilience to bad signal quality, Slingshot can maintain connectivity despite short-term impairments.

Based on this observation, we design a lightweight stateless migration mechanism for the PHY layer, called *PHY migration*, that moves the PHY processing to a hot-standby secondary PHY process by properly managing the two types of traffic processed by the PHY when a resilience event happens: the PHY–RU fronthaul traffic and the L2–PHY traffic. To achieve PHY migration transparently to the existing vRAN stack while handling the two classes of traffic in protocol-compliant ways, Slingshot uses two types of middleboxes that act as shim layers: one between the RU and PHY (§5), and one between the L2 and PHY (§6).

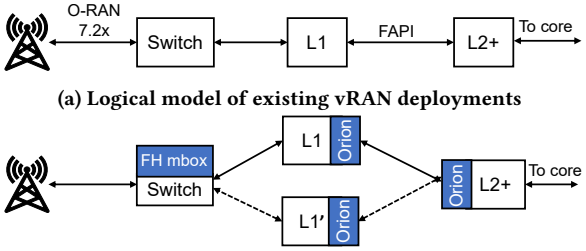
Scope. This paper focuses on 5G's Enhanced Mobile Broadband (eMBB) service that operates in the sub-10 GHz frequency range with 30 KHz subcarrier spacing, which is the primary use case for 5G vRANs today. However, the ideas presented here apply generally to cellular PHY layers, such as mmWave, which operate at higher frequencies and use larger subcarrier spacing.

3.1 Challenges

Realizing PHY migration with the new middleboxes presents the following three challenges.

C-1. Minimizing overhead of middleboxes and hot-standby secondary PHY processes. Since Slingshot middleboxes and the hot-standby secondary PHY process are additional components in the vRAN deployment, they may add compute overhead and latency. We find that handling the high volume of fronthaul traffic with a conventional software-based middlebox reduces the edge datacenter's coverage radius by over 10%, while requiring additional CPU cores and NIC bandwidth. Also, naïvely maintaining the compute-intensive secondary PHY by duplicating the primary PHY's processing results in a 100% compute overhead for resilience.

C-2. Transparent traffic management and failure detection. Managing the two classes of PHY traffic could be straightforward if modifying the RU's firmware or the L2 software to add support for a secondary PHY was possible, but it would violate our transparency requirement. Similarly, using existing low-latency failure detection mechanisms (e.g., [27, 38, 46]) requires modifying the PHY to send periodic heartbeat messages to a failure detector.



(a) Logical model of existing vRAN deployments

(b) Logical vRAN deployment model with Slingshot

Figure 4: Logically, Slingshot provides a resilient L1 to higher vRAN layers and the RU. To do this, Slingshot employs an in-switch fronthaul middlebox, and a software-based FAPI middlebox called Orion.

C-3. Correct processing of RAN protocols. vRANs are built using diverse components from different vendors, which are often closed-source black boxes. Coupled with the notorious complexity of RAN protocols—described by Chen et al. [35] as "characterized by convoluted descriptions in thousands of documents, on millions of pages"—this presents a significant challenge in designing and implementing resilience in an interoperable way.

3.2 Key ideas

We tackle these challenges with three key ideas:

I-1. In-switch fronthaul middlebox (§5). To manage the fronthaul traffic while minimizing overheads, we design the fronthaul middlebox using a programmable switch. We observe that the edge datacenter's switch located between the RU and vRAN servers can naturally handle fronthaul packets without additional compute resources or latency. Today's commodity programmable switches can handle several Tbps of traffic with negligible added latency (e.g., 3.2 Tbps for Arista's 32-port 100 GbE switches [2]), sufficient for processing the fronthaul traffic for hundreds of RUs.

I-2. In-switch fast failure detection (§5.2). To detect PHY failures rapidly, we design a fast failure detection mechanism running on the in-switch fronthaul middlebox. Our key insight is that a healthy PHY is a strict real-time application that sends downlink fronthaul packets to the RU in every slot (synonymous with TTI) duration. We use these packets as a natural liveness indicator to detect PHY failures by monitoring their inter-packet gap, without requiring additional components or modifications to the vRAN software.

I-3. Software-based L2-PHY FAPI middlebox (§6). We observe that interposing on the common FAPI interface—shared between various PHY and L2 implementations—provides a narrow waist that can be used to transparently provide PHY resilience. We realize this in a software middlebox called Orion that properly handles FAPI messages to (1) keep the secondary PHY alive with low overhead by sending null FAPI APIs and (2) hide the existence of the secondary PHY or PHY migration from both the L2 and PHY.

3.3 Slingshot architecture

Figure 4 illustrates a simplified logical view of the "resilient PHY" abstraction provided by Slingshot. The actual placement of the

primary and secondary PHYs (L1 and L1′, respectively), and L2 processes on servers can be configured in different ways. Note that this paper primarily focuses on the most challenging fully-decoupled use case, where all three processes run on different servers. The other extreme case is where all three processes run on the same server, where Slingshot can be used to transparently upgrade the PHY.

PHY migration with Slingshot. During normal operation, the in-switch fronthaul middlebox (FH-mbox) forwards fronthaul packets to/from the primary. Similarly, the L2-PHY software middlebox (Orion) at the L2 sends unmodified FAPI messages to the primary PHY, and sends null FAPI messages with no signal processing work to the secondary. This keeps the secondary PHY alive and hot, while avoiding the high CPU overhead of keeping a duplicate PHY that receives real work. When the primary PHY fails, the FH-mbox detects the failure in realtime by noticing the gap in downlink fronthaul packets from the primary PHY, and notifies the L2-side Orion. The L2-side Orion reconfigures the FH-mbox in realtime to steer the fronthaul traffic to the secondary PHY. It also steers the FAPI traffic to the secondary PHY, which completes PHY migration. Importantly, during PHY migration, Slingshot does not carry over any state from the primary PHY to the secondary.

4 PHY PROCESSING IMPAIRMENTS ≈ WIRELESS SIGNAL IMPAIRMENTS

In Slingshot, we use our observation that the effect of short-term PHY processing impairments, i.e., the loss of PHY computational state and fronthaul packets, resembles the effects of bad signal quality. Slingshot uses cellular networks' inbuilt ability to handle bad signal quality to mitigate the effects of PHY migration. UEs naturally experience performance variations due to the unreliable and shared nature of wireless networks. For example, even stationary 5G UEs with a clear line of sight to the cell tower can experience up to 4× variation in throughput [55]. Importantly, the rate of resilience-related PHY impairments is minuscule when compared to wireless signal impairments, and can therefore be ignored: Assuming a migration per week (e.g., for upgrades), only around 10^{-9} TTIs are effected, whereas even "ultra-reliable" RANs allow signal decoding failures in 0.1% TTIs [58].

This observation allows Slingshot to migrate PHY processing between two PHY processes without transferring any state while meeting our 10 ms availability target (§2.3). This contrasts with the strong consistency-based approaches taken for higher layers of the cellular stack such as the core network, that preserve all state and I/O during migration [48, 56]. Such approaches are not feasible for the PHY because of the much larger amount of state and I/O, realtime latency deadlines, and the lack of source code.

Intuitively, we view the PHY as a task executor responsible for only performing signal processing tasks against radio data. Across TTIs, the PHY maintains only a short-term soft state that spans only a few TTIs; a long-term hard state for the RU, PHY, and UEs is maintained in the higher vRAN layers. As we discuss below, the PHY-level uplink/downlink transmission failures caused the discarding of this state last for only a few milliseconds.

While our discussion focuses on PHY state for brevity, the same arguments apply for the ≈three TTIs of fronthaul packets lost by Slingshot during PHY failures (§8.2). Missing fronthaul packets cause the PHY to process garbage-valued I/Q samples, which is indistinguishable from the PHY performing signal processing on a noisy wireless channel.

4.1 Migrating at TTI boundaries

The PHY performs work at the granularity of TTIs (500 µs in our setup): The L2 issues requests to the PHY in every TTI describing the TTI's signal processing tasks, with information such as the set of UEs active in that TTI, and per-UE modulation and coding schemes. The PHY returns per-TTI responses with the data decoded on the uplink, and transmits encoded IQ samples to the radio on the downlink.

We design Slingshot to migrate PHY processing only at TTI boundaries. Concretely, the primary PHY processes TTIs $0 - i$, and the secondary PHY processes subsequent TTIs. This frees Slingshot from the need to transfer any intra-TTI (e.g., intermediate computation results like the demodulated data before decoding) state across the migration.

We next discuss in detail the state that the PHY retains across TTIs, and why Slingshot can safely discard them. We focus our discussion on uplink processing; similar arguments apply to the downlink (§8.4).

4.2 Inter-TTI state in uplink processing

Average signal-to-noise ratio (SNR). The PHY maintains a moving average for the SNR for every connected UE, which it uses to detect when a UE disconnects from the cell. In Slingshot, we ignore the state of the moving average filter during migration, causing the destination PHY to use a stale or default SNR value before the filter reconverges (for ≈25 ms). We argue that this is acceptable because the changed SNR could have also happened due to impairments in the UE's wireless channel, which the RAN is designed to handle.

Retransmission buffers. Modern RANs use "soft-combining" retransmission schemes [36]. In all such schemes, such as 5G's Hybrid Automatic Repeat reQuest (HARQ) scheme, the PHY retains recent bad UE transmissions that the PHY fails to decode. When the UE subsequently retransmits, the PHY later combines the retransmission with prior transmissions from the UE to improve the likelihood of successful decoding. 5G's HARQ procedure lasts several TTI as it includes an original transmission and up to three retransmissions.

Similarly to the SNR filter, Slingshot ignores HARQ buffers during migration, causing the destination PHY to use a stale or default HARQ buffer. This causes the PHY's CRC-protected forward error correction decoding to fail, resulting in retransmissions at the RAN's higher layers (e.g., RLC and the L3). Importantly, this is no different from a normal PHY operation. Commercial networks aim for PHY decoding failure rates far higher than the $\approx 10^{-9}$ fraction of TTIs affected by once-a-week PHY migrations, even after all four HARQ retransmissions (called the "residual block error rate"): 0.5–2% in mobile broadband [72, 76], and 0.1% in ultra-reliable use cases [58]. We present experiments for PHY migration during HARQ retransmissions in §8.4.

In summary, cellular networks' built-in ability to handle bad signal quality enables migrating PHY processing between two PHY processes without transferring any state. Building on this, we design

our lightweight PHY migration mechanism by introducing two types of middleboxes that sit between the RU and PHY, as well as between the L2 and PHY, to properly migrate traffic between the two PHYs. In the following sections, we present our design of RU-to-PHY fronthaul middlebox (§5) and L2-to-PHY middlebox (§6).

5 IN-SWITCH FRONTHAUL MIDDLEBOX

To support migration for the high-bandwidth and low-latency fronthaul traffic with low overhead, we chose to design a fronthaul middlebox on a programmable switch. We observe that the edge datacenter's switch can naturally inspect all fronthaul packets since it is located at the datacenter's vantage point. Today's commodity programmable switches can handle several Tbps of traffic with negligible added latency (e.g., 3.2 Tbps for Arista's 32-port 100 GbE switches [2]), which is sufficient for hundreds of RUs.

Slingshot's fronthaul middlebox must support (1) steering uplink packets from the RU to the current primary PHY, and (2) blocking downlink control-plane packets from a hot-standby secondary PHY from reaching the RU. In addition, the middlebox should allow changing the active PHY at exactly a TTI boundary, i.e., the RU communicates with the primary PHY for all TTIs $\leq i$, and the secondary PHY for all TTIs $> i$. This is needed to ensure that the RU and PHYs receive only protocol-compliant fronthaul packet sequences, which is needed for interoperability. Without TTI boundary alignment, the RU can receive packets for the same TTI from two PHYs, which can cause the RU to malfunction. To support these features, we develop new ideas including (1) virtual PHY addresses, (2) using fronthaul packet header fields to detect TTI boundaries, and (3) an indirect data structure for storing the RU-to-PHY mapping.

Drawbacks of software-based approaches. While a software-based middlebox can provide these features, it introduces three overheads that our switch-based approach avoids. First, by increasing fronthaul latency, it reduces the radius of the geographical area that a vRAN datacenter can serve. The fronthaul link in 5G vRAN deployments has a strict sub-100 µs maximum one-way delay requirement; our DPDK-based software version of Slingshot's fronthaul middlebox increases the 99.999th percentile one-way fronthaul latency by around 10 µs, reducing the maximum radius by 10%. Second, it increases the operational cost by doubling the required per-server NIC bandwidth by adding an extra hop to each fronthaul packet [40]. Third, it requires additional CPU cores (around 10% of total PHY cores with FlexRAN as the PHY) to be dedicated to the software middlebox.

Note that as we describe in §6, we can afford to use a software middlebox for L2–PHY traffic since its volume is far less than fronthaul traffic. The L2 and PHY exchange user bits, whereas our fronthaul middlebox handles raw floating-point IQ samples. For example, in our testbed, the downlink fronthaul traffic from the RU uses 4.5 Gbps, whereas the downlink L2-PHY traffic uses only around 100 Mbps.

5.1 RU-to-PHY mapping in the data plane

In a conventional RAN deployment, the switch forwards fronthaul packets from the RU to the PHY using a static RU-to-PHY mapping. However, for PHY migration, this static mapping does not work since the RU must communicate with the current primary PHY. To allow dynamically changing the RU-to-PHY mapping, Slingshot lets each RU send fronthaul packets using a *virtual* PHY address, which our in-switch middlebox translates into a physical PHY address.

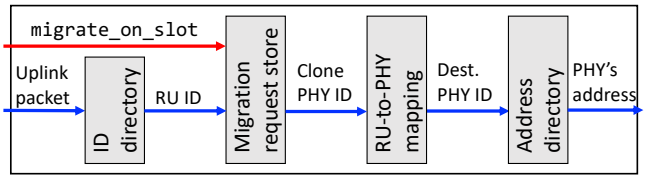


Figure 5: A simplified view of uplink fronthaul traffic processing logic in the fronthaul middlebox data plane.

How can we change this mapping (1) in realtime and (2) at exactly TTI boundaries? The switch control plane can update the mapping, but it has high latency (e.g., 29 ms 99.9th percentile for a rule update in our testbed), and it cannot guarantee that the update will take effect at a TTI boundary. The switch data plane has nanosecond-scale latency, but it is not time-synchronized with the RU and PHY, which are synchronized using the Precision Time Protocol (PTP).

Using packet header fields for timing. To address the time synchronization issue, we observe that fronthaul packets have header fields—the PHY-level frame, subframe, and slot number—that identify the TTI at which the packet was generated. Our in-switch middlebox parses these headers and uses them as triggers to update the RU-to-PHY mapping.

Indirect data structure for RU-to-PHY map. The next challenge is to design a data structure that can be updated in the switch data plane upon receiving a fronthaul packet matching the migration TTI. An intuitive data structure for this purpose is a hash table. However, implementing a general-purpose data plane-updatable 48-bit MAC-to-MAC address mapping (or 32-bits for IP-based fronthauls [31]) is complex, since it requires a full-fledged hash table that handles hash collisions, which is not supported by today's programmable switches. Instead, we introduce an indirection layer to create a collision-free keyspace: We observe that typical vRAN datacenters handle at most a few hundred RUs and PHYs, so we can rely on logical RU and PHY IDs assigned by vRAN operators at installation time. For a datacenter with 256 RUs and PHY processes, eight-bit IDs suffice.

Taking uplink as the example, the switch forwards fronthaul packets as follows: When the switch receives a packet from an RU, it first retrieves the RU's ID by looking up the ID directory (Key: RU's MAC address, Value: RU ID). Next, it looks up the RU-to-PHY mapping with the retrieved RU ID to get a PHY ID, and forwards the packet to the PHY's MAC address retrieved from the address directory (Key: PHY ID, Value: PHY's MAC address). For downlink packets, the switch additionally drops packets that are not addressed from the currently active PHY for the destination RU to avoid unexpected behaviors.

Controlling fronthaul migration. Our software FAPI middlebox (§6) controls PHY migration by sending a `migrate_on_slot` command packet to the switch. This packet contains a future slot number to migrate the PHY processing at, the RU ID, and the PHY server ID to remap this RU to. On receiving this command, the switch stores the command in its migration request store (Figure 5).

On receiving a fronthaul packet, the switch retrieves the packet's RU ID, and compares the packet's slot number with any stored migration requests for this RU. If they match, the switch data reads the PHY's server ID from the migration request and re-maps the RU to this server ID in the RU-to-PHY mapping. This establishes the new mapping and starts routing fronthaul packets to/from the new PHY process.

5.2 In-switch RAN failure detection

Until now, we have assumed that resilience events are instantly detected and trigger PHY migration. While this is straightforward for planned migrations, it is challening to detect PHY failures in a way that satisfies our requirements. Doing so requires a failure detector that (1) works transparently without RAN modifications, (2) detects failures rapidly to minimize dropped TTIs, and (3) has low CPU overhead.

Recent sub-ms failure detection approaches fail to meet these requirements. These approaches, such as those used by FARM [38], Mu [27], and X-Lane [46], use periodic heartbeat messages between a failure detector and the target service. Using such approaches would require modifying the PHY to add heartbeats, as well as dedicating overhead CPU cores for realtime lease message processing. We design a novel technique to transparently detect PHY failures (assuming a fail-stop model), running on our in-switch fronthaul middlebox. Additionally, the technique works generally for detecting the failure of any networked realtime vRAN layer.

5.2.1 Using realtime packets streams as heartbeats. Our key observation is that a healthy realtime vRAN layer sends packets in every TTI to the layer above or below it. These can be used as a natural heartbeat to detect failure for this layer. A healthy PHY—which we target in Slingshot—sends a downlink control plane fronthaul packet to the RU in every TTI, which the in-switch fronthaul middlebox can observe. All realtime vRAN layers emit such TTI-spaced packet streams: the RU emits fronthaul packets, the MAC emits FAPI packets, and the RLC emits RLC Protocol Data Units (PDUs). In our testbed, we measure the maximum inter-packet gap between downlink fronthaul packets to be 393 µs (§8.6).

5.2.2 In-switch inter-packet gap monitoring. We design our PHY failure detection engine as part of our switch data plane, which monitors the inter-packet gap between the PHY's downlink packets. Since today's programmable switches lack timers, we emulate timer ticks by using the programmable switch's packet generator to generate n packets in every timeout period T. To emulate timeout events, we maintain per-PHY counters. Each downlink packet from a PHY sets its counter to 0, and each timer packet reads and increments the counter by 1. When a PHY fails, its counter reaches n after n timer ticks. The next timer packet detects this PHY's failure by observing the saturated counter.

The parameter n governs the worst-case precision at which the switch matches the timeout value T. We empirically set T to 450 µs (Section 8.6) and n to 50, which gives us 9 µs precision and negligible switch overhead (50K packets per second).

Once a timer packet detects the PHY's failure, the switch reformats it into a failure notification packet and forwards it to our software FAPI middlebox (§6) for this PHY. On receiving the failure notification, the FAPI middlebox initiates the migration process and sends a `migrate_on_slot` command to the switch to trigger fronthaul migration.

6 ORION: L2-TO-PHY MIDDLEBOX

While it may seem that PHY migration can be achieved by managing only the fronthaul traffic between the RUs and primary/secondary PHYs using our fronthaul middlebox, this by itself is incomplete for two main reasons. First, the L2 and higher layers are not designed to allow secondary PHYs, and therefore require changes to the L2 software to realize the above approach. Second, even if these layers could deal with secondary PHYs, naïvely maintaining a secondary PHY causes high CPU overhead.

To address these challenges, we design Orion, a new middlebox process that acts as a shim layer between the L2 and PHY and enables efficient and transparent PHY migration. For efficiency, it maintains a hot standby secondary PHY with low CPU overhead during normal operation, and connects it to the L2 on PHY migration. For transparency, it interposes on FAPI protocol (§2.2) messages between the L2 and PHY. Since FAPI constitutes a "narrow waist" interface between different implementations of both the L2 and PHY, our design can support a variety of L2 and PHY implementations (e.g., GPUs [21] and DSPs [22]).

We term the Orion process pairing with a PHY or L2 as a "PHY-side" or "L2-side" Orion, respectively. Orion handles multiple RUs that map to the L2 or PHY processes that it peers with. It supports scenarios where the RU's primary PHY processing runs on the same server as the L2 to minimize overhead, though we focus only on the case where L2 and PHY are fully decoupled in this paper.

In the following, we describe how Orion decouples L2 and PHY over Ethernet (§6.1), how it maintains a hot standby secondary PHY with low CPU overhead (§6.2), and how it manages PHY migration (§6.3).

6.1 Stateless inter-Orion transport

Although there are established protocols such as nFAPI (i.e., networked FAPI) that decouple the PHY and L2 over a wired network, we found them mismatched for Orion's use case. Originally designed for small cells, nFAPI targets L2-PHY communication over a unreliable and slow city-area network. Therefore, it supports features such as reliable transmission using a complex stateful communication protocol (SCTP) and synchronization adjustments, which we do not need.

In contrast with nFAPI's target use case, the PHY and L2 layers in our target vRAN deployments run in the same edge datacenter with a reliable, low-latency network (e.g., 100 GbE). This allows Orion processes to use a lean network protocol with no inter-slot state to communicate with each other, making it possible to migrate PHY processing at TTI boundaries to a different server without migrating Orion's state. Packet losses in our target edge datacenters are extremely rare since vRAN datacenter operators statically provision the required Ethernet bandwidth, and there are no congestion-inducing incast-like situations. When rare packet losses occur, Orion discards the FAPI messages for the slot and injects an "null" FAPI message to its L2/PHY peer (§6.2).

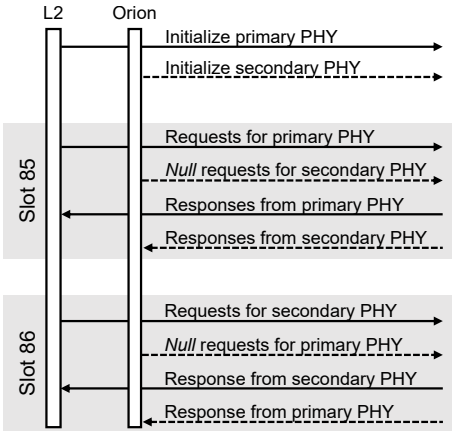


Figure 6: Simplified example of Orion's middlebox actions, showing migration of an RU from primary to secondary PHY at the TTI boundary between slots 85 and 86. The dashed lines show FAPI messages generated or filtered by Orion.

Orion transparently decouples the SHM-coupled L2 and PHY over the datacenter network as follows. When an L2 (or PHY) process attempts to connect to the PHY (or L2) over SHM, it connects to Orion instead. When the L2 sends a FAPI message to the PHY, the "L2-side Orion" receives the message over SHM, and forwards it to the Orion process at the server running the PHY. The "PHY-side Orion" receives the FAPI message over the datacenter network and forwards it to its peer PHY process over SHM. The PHY-to-L2 path works similarly in the reverse direction. Orion's design is agnostic to the physical FAPI channel (e.g., SHM, nFAPI, or even PCIe-based channels).

6.2 Null FAPIs for efficient secondary PHY

What are the challenges in maintaining a hot standby secondary PHY with low overhead, i.e., without scheduling real signal processing work to it? A naïve approach is to simply duplicate all FAPI messages sent by the L2 to the primary PHY. This however results in a large CPU overhead for the secondary PHY, since it now duplicates the primary's signal processing work. An intutive approach of simply not sending FAPI messages to the secondary PHY does not work, because the FAPI specification requires that a PHY must receive valid FAPI work requests in every slot. For example, our PHY (FlexRAN) crashes if it does not receive these requests, which is valid PHY behavior.

We solve this challenge by introducing the concept of "null" FAPI messages. Per the FAPI specification, a PHY must receive uplink and downlink configuration FAPI requests (`UL_CONFIG` and `DL_CONFIG`) in every slot, specifying the uplink and downlink signal processing work for that slot, respectively. These requests include information such as the set of active UEs, the frequency resources and modulation scheme used by each UE, etc.

Our insight is that "null" versions of `UL_CONFIG` and `DL_CONFIG` requests are valid inputs to the PHY; Slingshot uses such requests to keep the secondary PHY alive. A null request has no UE entries, indicating that the PHY needs to do no uplink or downlink signal processing for this slot. The PHY generates no significant computational work for null requests; we quantify this in §8.7.

Figure 6 shows how Orion uses null requests for an example slot 85. When the L2 sends requests to the primary PHY, Orion intercepts them and sends unmodified and null requests to the primary and the secondary PHYs, respectively. Both PHYs subsequently send responses to the L2. The L2-side Orion forwards only the primary PHY's responses to the L2, dropping the secondary's responses.

6.3 Managing PHY migration

Orion provides the initialization, management and control needed for PHY migration:

Initializing the secondary PHY. Orion needs a way to spawn a secondary PHY without understanding the complex details of PHY initialization. When the L2 onboards a new RU, it tries to initialize PHY processing for this RU in an existing PHY process by sending it a FAPI initialization request. The L2-side Orion intercepts this initialization request and stores a duplicate copy of the request. It chooses two servers for the primary and secondary PHY, based on cluster configuration information from Orion's management thread. It then sends one initialization message to the PHY-side Orion processes running on the two chosen remote servers. At this time, the two servers may already be handling primary or secondary PHY processing for other RUs. Each PHY process receives initialization messages from its Orion peer (the "PHY-side Orion"), and initializes PHY processing for the new RU. The stored initialization messages can be used to initialize additional secondary PHYs after the primary fails.

Migration to the secondary PHY. The L2-side Orion initiates migration for the RUs that map to its peer L2; this can be controlled by an external controller or by a manual operator. It does so by (1) switching the PHY with which it exchanges original and null FAPI requests and responses (slot 86 in Figure 6), and (2) triggering the fronthaul migration by sending a `migrate_on_slot` command to the fronthaul middlebox (§5.1). For simplicity, we designate Orion as an exclusive initiator of PHY migration, i.e., the fronthaul middlebox simply executes migration at the slot requested by Orion.

7 IMPLEMENTATION

In-switch fronthaul middlebox. The data plane components written in P4-16 consist of the RU ID and PHY address directory implemented using match-action tables; and the migration request store and the RU-to-PHY mapping implemented using P4 registers. We implement the switch control plane in Python using the Barefoot Runtime APIs, which initializes the above data structures.

In-switch PHY failure detector. The PHY failure detector is implemented as part of our fronthaul middlebox. To emulate timer ticks, we configure Tofino's built-in packet generator to periodically inject packets into the switch data plane using the Barefoot Runtime APIs. We implement the per-PHY timeout counters using P4 registers.

Orion middlebox. We implement Orion and all corresponding FAPI transformations in 8850 lines of C++. For low latency, we use a UDP-based userspace transport implemented using DPDK [8] with

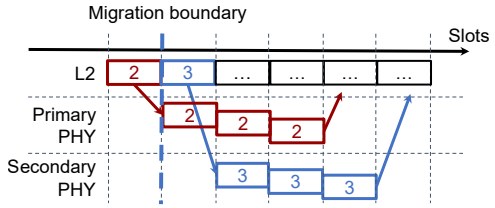


Figure 7: Visualization of pipelined slot processing in Orion.

Radio unit	Foxconn 4x4 RU; 100 MHz at 3.5 GHz
UEs	Raspberry Pi; OnePlus N10 and Samsung A52s
PTP grandmaster	Qulsar QG2 multi-sync gateway
Servers (3×)	HPE Telco DL110; Xeon 6338N CPU
Accelerator	Intel ACC100 for LDPC coding
Ethernet NIC	Intel E810 100 GbE NIC
Ethernet switch	Tofino-based Arista 7170 P4 switch
Operating system	Realtime Linux kernel 5.15
PHY software	Intel FlexRAN v20.11, v21.03, v21.11
L2+ software	CapGemini 5G Solution, Intel testmac
5G core	Metaswitch's Fusion Core

Table 1: Testbed hardware and software configuration.

busy-polling. Note that since the vRAN L1-L2 interface operates at a much lower data rate than the capabilities of the modern datacenter networks, we have not yet fully optimized Orion's performance.

Pipelined slot processing in Orion. Section 6 presents a simplified view where all the processing for a slot happens within the slot duration. Real PHY implementations, such as FlexRAN and srsRAN [17] are more complex: they use a pipeline of tasks to process a slot. As an example, Figure 7 shows FlexRAN's three-slot uplink processing pipeline. Assuming that the PHY migration happens at the slot #2–#3 boundary, the primary PHY will keep producing uplink data for slot #2 even after migration. Orion continues to accept this data to minimize the number of dropped TTIs. This helps Slingshot provide lower UE downtime for the more frequent planned migrations compared to failovers (§8.2).

8 EVALUATION

We evaluate Slingshot on a state-of-the-art 5G vRAN testbed with hardware and software that closely resembles the majority of real vRAN deployments [9, 23, 24] (Table 1). We use a four-antenna RU with 100 MHz bandwidth and 30 KHz subcarrier spacing (i.e., 500 µs TTIs). The cell operates in time-division duplexing (TDD) mode with a "DDDSU" slot format, i.e., three DL slots followed by one UL slot, with a shared/guard slot in between. We use three different types of commercial UEs.

Our testbed has three servers in a rack, connected via a 100 GbE programmable switch. We use two servers to run the primary PHY and the hot secondary PHY, and a different server to run the vRAN's L2+ layers. In real deployments, Slingshot will co-locate primary and secondary PHYs for different RUs within PHY processes, i.e., our design does not require dedicated servers to run just secondary PHYs. We use unmodified commercial RAN and 5G core software, showing our design's transparency and interoperability.

We first present our end-to-end evaluations, and then present microbenchmarks for our two middleboxes.

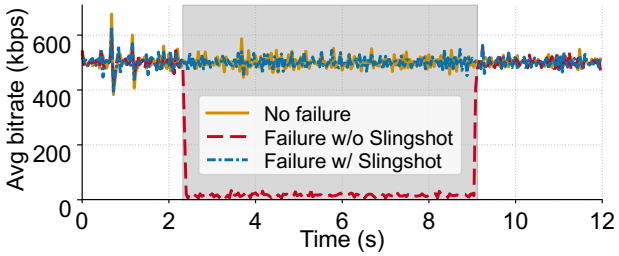


Figure 8: Downlink bitrate in videoconferencing with primary PHY failure within the third second.

8.1 Reliable videoconferencing

Experiment setup. We first show how Slingshot PHY's failure recovery retains the user's quality of experience (QoE) for live video conferencing, which is a latency-critical application where the QoE is critical. Our video sender streams a compressed talking-head video to the UE at a target bitrate of 500 Kbps. We measure the average video bitrate on the receiver since it correlates with QoE and time to recovery [45, 53, 74]. We compare changes in the video bitrate in three scenarios: (1) no failure, (2) PHY failure without Slingshot (baseline), and (3) PHY failure with Slingshot.

Baseline. Our failover baseline that demonstrates the lack of resilience in today's vRANs works as follows: We run an entire vRAN stack (i.e., the PHY layer through the Packet Data Convergence Protocol layer) as a hot backup vRAN on a separate server, with the same configuration as the primary vRAN. Since today's vRANs lack fast failure detection and fronthaul management, when the primary vRAN's PHY fails, we use our fronthaul middlebox to detect it and re-route the fronthaul to the secondary vRAN's PHY.

Results. Figure 8 shows the video bitrate on the downlink. The uplink results are similar and thus omitted. Without Slingshot, the UE fully disconnects from the network and takes 6.2 seconds to reattach; the video bitrate is zero during this time, incurring significant QoE loss. The 6.2 seconds are spent in re-establishing a broken connection with the core network, and match prior measurements (e.g., 5 seconds in a field report by Qualcomm [60]). By contrast, Slingshot keeps the bitrate steady by transparently resolving the PHY failure. We note that Slingshot supports higher video bitrates: later in Figure 12, we show that Slingshot can handle 3.4 Gbps of user traffic.

8.2 End-to-end benchmarks

We next present end-to-end benchmarks to show that Slingshot meets our availability target of sub-10 ms connectivity disruption during PHY failures (§2.3). Our experiments focus on PHY failover where the secondary PHY acts as a backup, since these are more challenging than planned migrations.

Comparison with VM migration. Compared to VM migration-based PHY resilience approaches that can drop several hundred milliseconds of TTIs (Figure 3), Slingshot reduces the number of dropped TTIs by two orders of magnitude to at most three TTIs: If a PHY fails towards the end of over-the-air slot N, our switch middlebox will trigger a timeout after 450 µs (§5.2), i.e., towards the

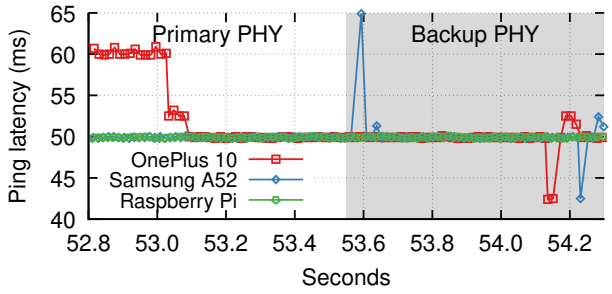


Figure 9: Ping latency with primary PHY failure t =53.546 ms. The Y-axis starts at 40 ms. The transient disruption from failover resembles natural wireless fluctuations.

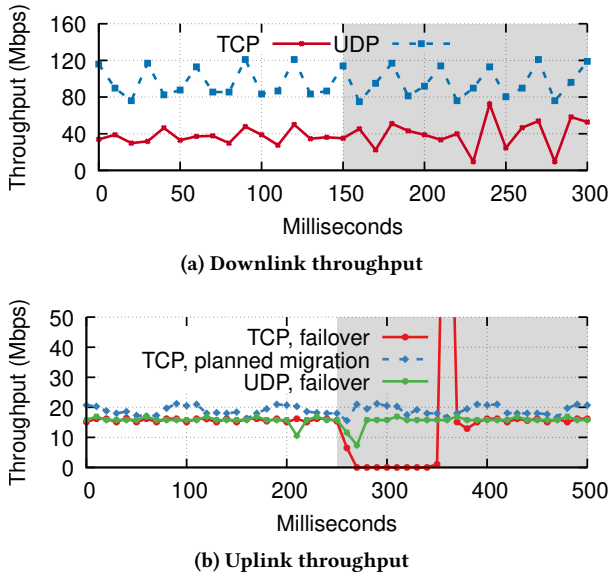
end of slot N+1. It then takes tens of microseconds for the L2-side Orion to receive and react to the switch's failure notification packet, possibly impairing the processing of slot N+2.

Latency impact of PHY failure. We use three UEs simultaneously and measure the ping latency from the UEs to the application server at 10 ms intervals. We trigger failover by manually terminating the primary PHY with a SIGKILL signal. We record the PHY failure time as the time when the L2-side Orion receives a notification about the PHY's failure from the switch middlebox.

Figure 9 shows the latency for each UE. To illustrate how the UE performance degradation during PHY migration resembles natural wireless signal impairments, we plot the ping latency for a ≈2 s period centered at the failure time, with 10 ms between pings. (We show results for shorter timescales in other experiments.) While the latency of two UEs (the OnePlus N10 and the Raspberry Pi) remains unaffected during migration, the latency of the Samsung A52s suffers a 15 ms spike. However, this resembles routine performance fluctuations, seen in the plot's far left and right.

Downlink throughput during failover. To measure the effect of Slingshot's PHY failover on downlink throughput, we use iperf to send downlink traffic from an application server to the UE. We use a single UE in this experiment to measure the throughput in an isolated setting. Figure 10a shows the throughput measured at the UE, reported every 10 ms, zoomed and centered at the failure time. We perform two separate measurements with TCP and UDP traffic. We find that Slingshot preserves the downlink connection without noticeable degradation in UE throughput.

Uplink throughput during failover. Figure 10b shows results from a similar experiment as above, but for uplink throughput measured at the application server. We observe that after the failure, UDP throughput drops from around 15.8 Mbps to 7.4 Mbps, but recovers to 15.8 Mbps within 20 ms. Importantly, the UE retains connectivity for all 10 ms intervals, meeting our availability target. TCP's in-order delivery requirement makes its throughput more sensitive to the packets lost during PHY failure: throughput drops to zero for 80 ms and recovers fully 110 ms after PHY failure. However, the application server keeps receiving TCP packets for much of the 80 ms of period: its throughput jumps to 157 Mbps (cut off from Figure 10) when it finally receives the lost packets retransmitted by the UE's TCP stack. Note that we observe the drop only during



(a) Downlink throughput

(b) Uplink throughput

Figure 10: TCP and UDP throughput changes during resilience events, at t =150 ms and 250 ms, respectively.

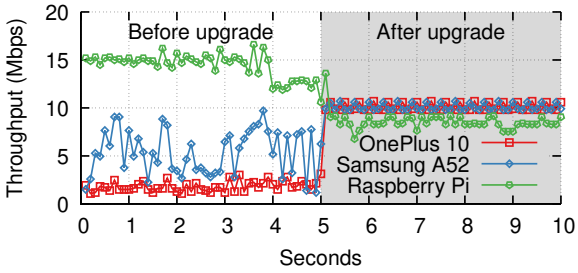


Figure 11: Uplink UDP bandwidth achieved by three UEs before (white) and after (gray) a PHY upgrade.

failover; in case of planned migrations (dashed line in Figure 10b), there is no drop.

8.3 Live PHY upgrades

To demonstrate Slingshot's usefulness for live upgrades, we emulate a scenario where the upgraded PHY has better Forward Error Correction (FEC), which improves its signal decoding success rate. Such performance upgrades are common in practice and today they require pre-planned maintenance windows, which Slingshot can eliminate. We emulate the upgrade by configuring the secondary (new) PHY to use more FEC iterations for decoding the signal. Figure 11 shows that before the upgrade, the two phones get low throughput, with the Raspberry Pi getting an unfairly high share. After the upgrade, the phones' throughput improves and the UEs share the available bandwidth more evenly. The upgrade completes without network downtime.

Metric	1/s	10/s	20/s	50/s
#10 ms blackout intervals	0	0	0	11
Min tput (Mbps) per 10 ms	4.2	3.2	2.1	0
Max tput (Mbps) per 10 ms	18	18	18	18
Max pkt loss rate per 10 ms	50%	62%	67%	100%
Interrupted HARQ seqs over 60 s	0	67	118	315
Avg. UDP pkt loss rate over 60 s	0.1%	0.46%	1.6%	3.9%

Table 2: Metrics for an uplink UDP flow during a stress test for discarding PHY state, with PHY migration rates between 1 migration/s ("1/s") and 50 migrations/s. Slingshot maintains sub-10 ms network downtime with even 20 migrations/second, which includes 118 broken HARQ sequences.

8.4 Stress test for discarding PHY state

We run the following stress test to validate our hypothesis that discarding inter-TTI PHY state such as HARQ buffers (§4) during resilience events does not result in network downtime longer than our 10 ms target. We perform multiple PHY migrations between the two PHY servers at extreme rates of tens of migrations/second, for a measurement period of 60 seconds. Table 2 shows several metrics for an uplink UDP flow from the UE to the application server during the experiment. Even with a migration every 50 ms (i.e., 20 migrations/s), Slingshot keeps the network downtime below 10 ms by providing at least 2 Mbps uplink throughput for any 10 ms duration. In the 20 migrations/second case, we observe 118 HARQ sequences that coincide with the slot that Slingshot chooses for PHY migration, yet we find no prolonged network downtime. This experiment convincingly shows how discarding inter-slot PHY state is safe, and that Slingshot co-exists with multi-TTI operations such as HARQ.

For downlink transmissions, the vRAN's PHY does not maintain HARQ buffers, but PHY migration may cause HARQ acknowledgments sent by the UE's MAC to be dropped. The impact of such impairments is small: our experiment with a 100 Mbps downlink UDP transfer with even 20 migrations/second showed a worst-case reduction of under 20% in downlink throughput measured by the UE at 10 ms intervals.

8.5 Overhead of secondary PHYs

What are the overheads of maintaining a hot inactive secondary PHY (§6.2)? To estimate the CPU cost for a secondary PHY, we measure the marginal cost of adding the PHY to a server that is already running one primary PHY. We find that our use of null FAPI requests makes the PHY compute overhead of secondary PHYs negligible: FlexRAN reports no significant increase in its CPU or FEC accelerator usage. There is no L2 overhead, since Slingshot does not expose the secondary PHY to the L2. The datacenter network overhead is also negligible, e.g., Orion's null FAPI messages use under 1 MBps on our 100 GbE inter-server links.

8.6 Switch microbenchmarks

We measure the amount of switch ASIC resources used by Slingshot's dataplane for a large edge datacenter configuration that serves 256 RUs with 256 vRAN servers (most vRAN deployments are smaller, e.g., the O-RAN's cloud architecture paper targets 64 RUs [7]). The fraction of each switch resource used is small, including crossbar (5.2%), ALU (10.4%), gateway (14.1%), SRAM (5.3%), and hash bits (9.5%). Supporting more RUs/PHYs increases only SRAM usage.

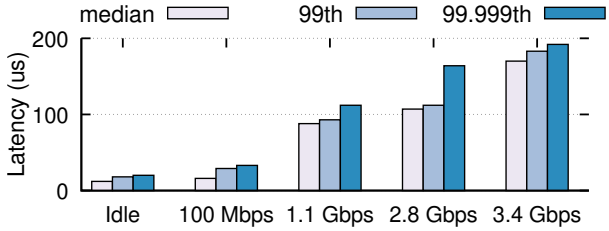


Figure 12: One-way latency added by Orion for different downlink user throughputs.

Inter-packet gap. To choose a timeout value for Slingshot's switch-based failure detector, we measure the maximum inter-packet gap between a healthy PHY's downlink packets (§5.2). We collect switch's ingress timestamps for downlink packets with a P4 program that prepends a nanosecond-precision timestamp to each downlink packet, and then mirrors the packet to a server for analysis. The analyzer computes the inter-packet gap statistics using the collected timestamps. We measure the maximum inter-packet gap across all idle and busy cases to be 393 µs, so we choose a conservative switch timeout of 450 µs for the failure detector.

8.7 Orion microbenchmarks

Orion's FAPI transformations and SHM-to-UDP translation does not noticeably increase latency for UEs. This is because the latency of Orion's intra-datacenter UDP transport (microseconds) is far smaller than cellular network latency (milliseconds). We measure the ping latency from our application server to the UE every 10 ms. This experiment runs the PHY on a single server (i.e., no migration); the L2 runs on a different server for the L2-PHY decoupled configuration. The median latency both with and without Orion is 22.8 ms, with a standard deviation of 0.8 ms in both cases.

Figure 12 shows the one-way L2-to-PHY latency added by Orion for different downlink throughputs, which typically exceed uplink throughput by around 10x. The first two clusters show real testbed measurements. To generate a higher load not currently possible in our testbed, we use FlexRAN's test-mode MAC to send traffic at up to 3.4 Gbps. We find that the latency added by Orion remains under 200 µs. FlexRAN budgets one TTI (slot $N-3$) for FAPI message transfers for downlink slot N, and the latency added by Orion is well under the 500 µs TTI budget.

9 RELATED WORK

Systems support for vRANs. There is a growing body of work that aims to improve vRAN robustness. Concordia [41] presents a deadline scheduling framework that improves the vRAN's ability to co-exist with other workloads. The Nuberu project is the closest in spirit to our work [42]: they propose a vRAN design that works well in non-ideal settings on servers with high CPU interference. These single-server resilience techniques can be combined with Slingshot's cross-server distributed techniques for better resilience.

FSA [34] focuses on network slicing to better isolate different virtual networks with different traffic patterns. It uses a programmable switch to identify and route slices of fronthaul traffic at line rate. FSA is complementary to our work, and Slingshot's switch-based fronthaul middlebox can be extended to support network slicing based on FSA.

RU frontend–based techniques. Projects such as Picasso [44] and Mendes et al. [54] provide a way to run multiple vRAN stacks atop the same physical RU. These approaches can theoretically serve similar goals as Slingshot. For example, the vRAN may broadcast two cells—a primary and a backup—from the same RU, with the UEs attaching preferentially to the primary cell. Upon failure of the primary's vRAN, the UEs migrate to the backup cell. These approaches are orthogonal to Slingshot, and further work is needed to realize these ideas in real vRANs. For example, they require special logic in the RUs, which is not possible with today's commercial radios that we target.

A cellular deployment can also mask the impact of PHY failures from UEs by deploying cells with overlapping coverage, allowing the UE to connect to a different RU in the vRAN backend. Such approaches have cost and applicability constraints, and are orthogonal to our work.

Theoretical work on using dynamic RU-to-PHY mapping. There is a large body of theoretical work on using dynamic RU-to-PHY mappings to improve vRAN energy efficiency or QoS. Sigwele et al. [68] present an approach to bin-pack RUs to the fewest servers during low load to improve LTE's energy efficiency. Other approaches optimize the mappings to improve metrics such as dropped calls [30, 51, 63]; Rodoshi et al. [64] present a comprehensive review. Due to the lack of systems support for dynamically remapping RUs to PHYs, these projects evaluate their benefits only in simulation; PHY migration addresses this gap.

10 FUTURE WORK

Higher vRAN layers. Our north star goal is to design resilience approaches for all types of vRAN components; PHY migration is a first step towards this goal. The different vRAN layers have varied amounts of hard state, compute resource usage, and real-time latency requirements. The PHY layer is at one extreme, with no hard state, very high compute usage, and strict latency requirements. The other extreme is vRAN layers above the L2, which lack real-time latency requirements, and may be handled by VM migration or state snapshotting approaches used successfully for the core network [48, 56]. The L2 layers offer an interesting challenge, since they have both hard state and real-time latency requirements. We believe that we can build L2 migration by combining Slingshot's approach of discarding some state during migration, with recent high-performance state preservation techniques like Zeus [47].

Massive MIMO. Our work focuses on small antenna configurations that are currently the target for vRAN operators. vRAN operators are beginning to adopt massive MIMO configurations, though software and RU support are currently nascent. Massive MIMO PHYs use inter-slot state lasting tens to hundreds of slots for their downlink precoding (i.e., beamforming) and uplink equalization (e.g., zero-forcing) matrices [75]. We note that this is still soft state that can be discarded without affecting correct PHY operation, albeit with a possibly larger impact on UE performance than our experiments observe.

11 CONCLUSION

Resilience support for failovers and live upgrades is a key missing part in today's vRAN deployments. Slingshot takes the first step towards resilient vRANs by building the required systems support for migrating the vRAN's most compute-intensive and latency-sensitive PHY layer. The key insight that makes this work possible is that imperfect migration matches the inherent imperfection of cellular networks. We build upon this insight with a shim-layer approach for a transparent and lightweight PHY migration mechanism with novel in-switch fronthaul and software FAPI middleboxes, as well as an in-switch RAN failure detector. Our evaluations show how Slingshot provides a resilient PHY in a state-of-the-art vRAN testbed with unmodified commercial vRAN software and hardware, migrating PHY processing between servers with only milliseconds of UE service disruption. We believe that our insights and middlebox designs will be fundamental building blocks for future resilient vRANs.

Ethics: This work does not raise any ethical issues.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and constructive feedback. We thank Xenofon Foukas, Bozidar Radunovic, Matthew Balkwill, Yongguang Zhang, and Victor Bahl for their help and feedback on this project. This work was supported in part by NSF awards CNS-2203167 and CNS-2203152, NSF CAREER award CCF-1846046, an Intel Rising Star Award, and an Intel Focused Grant.

REFERENCES

[1] 2016. *QEMU RDMA Live Migration.* https://wiki.qemu.org/Features/RDMALiveMigration
[2] 2020. Arista: 7170 Series Technical Specifications and Features. https://www.arista.com/en/products/7170-series/specifications.
[3] 2020. iperf(1) - Linux man page. https://linux.die.net/man/1/iperf.
[4] 2020. QEMU - A generic and open source machine emulator and virtualizer. https://www.qemu.org/.
[5] 2021. Altiostar and Rakuten Mobile Demonstrate Success Across Performance and Scalability for Open RAN Network. https://www.altiostar.com/altiostar-and-rakuten-mobile-demonstrate-success-across-performance-and-scalability-for-open-ran-network/.
[6] 2021. CapGemini 5G gNodeB. https://capgemini-engineering.com/nl/en/services/next-core/wireless-frameworks/.
[7] 2021. Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN. https://www.o-ran.org/specifications.
[8] 2021. Data Plane Development Kit (DPDK). http://dpdk.org/.
[9] 2021. Deutsche Telekom lights open RAN test site. https://www.mobileworldlive.com/featured-content/top-three/dt-openran-testbed/.
[10] 2021. FlexRAN™ Reference Architecture for Wireless Access. https://www.intel.com/content/www/us/en/developer/topic-technology/edge-5g/tools/flexran.html.
[11] 2021. Kernel Virtual Machine. https://www.linux-kvm.org/page/Main_Page.
[12] 2021. O-RAN Alliance: Operator Defined Open and Intelligent Radio Access Networks. https://www.o-ran.org/.
[13] 2021. O-RAN Hardware Reference Design Specification for Indoor Pico Cell with Fronthaul Split Option 6. https://www.o-ran.org/specifications.
[14] 2021. O-RAN: Towards an Open and Smart RAN. https://www.o-ran.org/s/O-RAN-WP-FInal-181017.pdf.
[15] 2021. $P4_{16}$ Language Specification. https://p4.org/p4-spec/docs/P4-16-v1.2.0.html.
[16] 2021. Radisys 5G NR Software Suite. https://www.radisys.com/connect/connectran/5g.

[17] 2021. SRS: Software Radio Systems. https://www.srs.io/.

[18] 2021. vSphere Performance Equivalent to Bare Metal for RAN Workloads. https://telco.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/microsites/telco/vmware-telco-ran-performance-wp.pdf.

[19] 2022. 5G FAPI: PHY API specification. https://www.smallcellforum.org/reports/5g-fapi-phy-api-specification.

[20] 2022. 5G nFAPI specifications. https://www.smallcellforum.org/reports/5g-nfapi-specifications.

[21] 2022. NVIDIA Aerial SDK: Build and Deploy GPU-Accelerated 5G Virtual Radio Access Networks (vRAN). https://developer.nvidia.com/aerial-sdk.

[22] 2022. Qualcomm Introduces New 5G Distributed Unit Accelerator Card to Drive Global 5G Virtualized RAN Growth. https://www.qualcomm.com/news/releases/2021/06/qualcomm-introduces-new-5g-distributed-unit-accelerator-card-drive-global.

[23] 2022. Rakuten Symphony Symware™ Phase Two Begins with Plans to Commercially Deploy 30,000 Units in Japan. https://symphony.rakuten.com/newsroom/rakuten-symphony-symware-phase-two-begins.

[24] 2022. The Journey to a Cloud-native, Fully Software-defined vRAN Architecture. https://www.vodafone.com/sites/default/files/2022-12/journey-to-cloud-native-fully-software-defined-vran-architecture.pdf.

[25] 2022. Verizon deploys more than 8,000 VRAN cell sites, rapidly marches towards goal of 20,000. https://www.verizon.com/about/news/verizon-deploys-more-8000-vran-cell-sites.

[26] 2022. Vodafone turns on first U.K. 5G open RAN site. https://www.fiercewireless.com/tech/vodafone-turns-first-uk-5g-open-ran-site.

[27] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra J. Marathe, Athanasios Xygkis, and Igor Zablotchi. 2020. Microsecond Consensus for Microsecond Applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 599–616. https://www.usenix.org/conference/osdi20/presentation/aguilera

[28] Kazi Main Uddin Ahmed, Manuel Alvarez, and Math H. J. Bollen. 2020. Characterizing Failure and Repair Time of Servers in a Hyper-scale Data Center. In *IEEE PES Innovative Smart Grid Technologies Europe, ISGT Europe 2020, Delft, The Netherlands, October 26-28, 2020*. IEEE, 660–664. https://doi.org/10.1109/ISGT-Europe47291.2020.9248891

[29] Jesutofunmi Ademiposi Ajayi. 2019. *Live eNodeB Container Migration in LTE Mobile Networks*. Master's thesis. University of Bern.

[30] Sally R. Aldaeabool and Maysam F. Abbod. 2017. Reducing power consumption by dynamic BBUs-RRHs allocation in C-RAN. In *2017 25th Telecommunication Forum (TELFOR)*. 1–4. https://doi.org/10.1109/TELFOR.2017.8249289

[31] ORAN Alliance. 2022. Control, user and synchronization plane specification. *O-RAN Fronthaul Working Group, ORAN-WG4.CUS.0-v10.00* (2022).

[32] M. Baker-Harvey. 2015. *Google Compute Engine uses Live Migration technology to service infrastructure without application downtime.* https://cloudplatform.googleblog.com/2015/03/Google-Compute-Engine-uses-Live-Migration-technology-to-service-infrastructure-without-application-downtime.html

[33] Robert Birke, Ioana Giurgiu, Lydia Y. Chen, Dorothea Wiesmann, and Ton Engbersen. 2014. Failure Analysis of Virtual and Physical Machines: Patterns, Causes and Characteristics. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 1–12. https://doi.org/10.1109/DSN.2014.18

[34] Nishant Budhdev, Raj Joshi, Pravein Govindan Kannan, Mun Choon Chan, and Tulika Mitra. 2021. *FSA: Fronthaul Slicing Architecture for 5G Using Dataplane Programmable Switches*. Association for Computing Machinery, New York, NY, USA, 723–735. https://doi.org/10.1145/3447993.3483247

[35] Yi Chen, Di Tang, Yepeng Yao, Mingming Zha, XiaoFeng Wang, Xiaozhong Liu, Haixu Tang, and Dongfang Zhao. 2022. Seeing the Forest for the Trees: Understanding Security Hazards in the 3GPP Ecosystem through Intelligent Analysis on Change Requests. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 17–34. https://www.usenix.org/conference/usenixsecurity22/presentation/chen-yi

[36] Josep Colom Ikuno, Martin Wrulich, and Markus Rupp. 2009. Performance and modeling of LTE H-ARQ. *International ITG Workshop on Smart Antennas (WSA 2009)* (01 2009).

[37] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 373–387. https://www.usenix.org/conference/nsdi18/presentation/dalton

[38] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. 2015. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 54–70. https://doi.org/10.1145/2815400.2815425

[39] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A Fast and Reliable Software Network Load Balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 523–535. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eisenbud

[40] Kevin Fall, Gianluca Iannaccone, Maziar Manesh, Sylvia Ratnasamy, Katerina Argyraki, Mihai Dobrescu, and Norbert Egi. 2011. RouteBricks: Enabling General Purpose Network Infrastructure. *SIGOPS Oper. Syst. Rev.* 45, 1 (2011). https://doi.org/10.1145/1945023.1945037

[41] Xenofon Foukas and Bozidar Radunovic. 2021. Concordia: teaching the 5G vRAN to share compute. In *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, Fernando A. Kuipers and Matthew C. Caesar (Eds.). ACM, 580–596. https://doi.org/10.1145/3452296.3472894

[42] Gines Garcia-Aviles, Andres Garcia-Saavedra, Marco Gramaglia, Xavier Costa-Perez, Pablo Serrano, and Albert Banchs. 2021. Nuberu: Reliable RAN Virtualization in Shared Platforms. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking* (New Orleans, Louisiana) *(MobiCom '21)*. Association for Computing Machinery, New York, NY, USA, 749–761. https://doi.org/10.1145/3447993.3483266

[43] Ahmad Hassan, Arvind Narayanan, Anlan Zhang, Wei Ye, Ruiyang Zhu, Shuowei Jin, Jason Carpenter, Z. Morley Mao, Feng Qian, and Zhi-Li Zhang. 2022. Vivisecting Mobility Management in 5G Cellular Networks. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 86–100. https://doi.org/10.1145/3544216.3544217

[44] Steven S. Hong, Jeffrey Mehlman, and Sachin Katti. 2012. Picasso: Flexible RF and Spectrum Slicing. *SIGCOMM Comput. Commun. Rev.* 42, 4 (aug 2012), 37–48. https://doi.org/10.1145/2377677.2377683

[45] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[46] Patrick Jahnke, Vincent Riesop, Pierre-Louis Roman, Pavel Chuprikov, and Patrick Eugster. 2021. Live in the Express Lane. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 581–595.

[47] Antonios Katsarakis, Yijun Ma, Zhaowei Tan, Andrew Bainbridge, Matthew Balkwill, Aleksandar Dragojevic, Boris Grot, Bozidar Radunovic, and Yongguang Zhang. 2021. Zeus: Locality-Aware Distributed Transactions. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 145–161. https://doi.org/10.1145/3447786.3456234

[48] Antonios Katsarakis, Zhaowei Tan, Matthew Balkwill, Bozidar Radunovic, Andrew Bainbridge, Aleksandar Dragojevic, Boris Grot, and Yongguang Zhang. [n.d.]. *rVNF: Reliable, scalable and performant cellular VNFs in the cloud.* Technical Report.

[49] Sean Kenney. [n.d.]. Breaking down the pros of Open RAN. https://www.rcrwireless.com/20200925/5g/breaking-down-the-pros-of-open-ran.

[50] Junaid Khalid and Aditya Akella. 2019. Correctness and Performance for Stateful Chained Network Functions. In *USENIX NSDI* (2019).

[51] M. Khan, R.S. Alhumaima, and H.S. Al-Raweshidy. 2015. Quality of Service aware dynamic BBU-RRH mapping in Cloud Radio Access Network. In *2015 International Conference on Emerging Technologies (ICET)*. 1–5. https://doi.org/10.1109/ICET.2015.7389166

[52] 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* (Snowbird, Utah, USA) *(MobiCom '17)*. Association for Computing Machinery, New York, NY, USA, 56–69. https://doi.org/10.1145/3117811.3117838

[53] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. 2021. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*. 229–244.

[54] José Mendes, Xianjun Jiao, Andres Garcia-Saavedra, Felipe Huici, and Ingrid Moerman. 2017. Cellular Access Multi-Tenancy through Small Cell Virtualization and Common RF Front-End Sharing. 35–42. https://doi.org/10.1145/3131473.3131474

[55] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. 2020. A First Look at Commercial 5G Performance on Smartphones. In *Proceedings of The Web Conference 2020*. 894–905.

[56] 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* (New Delhi, India) *(MobiCom '18)*. Association for Computing Machinery, New York, NY, USA, 163–178. https://doi.org/10.1145/3241539.3241564

[57] Navid Nikaein, Mahesh K. Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. 2014. OpenAirInterface: A Flexible Platform for 5G Research. *SIGCOMM Comput. Commun. Rev.* 44, 5 (oct 2014), 33–38. https://doi.org/10.1145/2677046.2677053

[58] Guillermo Pocovi, Hamidreza Shariatmadari, Gilberto Berardinelli, Klaus Pedersen, Jens Steiner, and Zexian Li. 2018. Achieving ultra-reliable low-latency communications: Challenges and envisioned system enhancements. *IEEE Network* 32, 2 (2018), 8–15.

[59] Chandra Prakash, Debadatta Mishra, Purushottam Kulkarni, and Umesh Bellur. 2022. Portkey: Hypervisor-Assisted Container Migration in Nested Cloud Environments. In *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Virtual, Switzerland) *(VEE 2022)*. Association for Computing Machinery, New York, NY, USA, 3–17. https://doi.org/10.1145/3516807.3516817

[60] Qualcomm. 2014. 3GPP RAN2 R2-140089, Mobility Performance in Real Networks. (2014).

[61] Mubashir Adnan Qureshi, Ajay Mahimkar, Lili Qiu, Zihui Ge, Max Zhang, and Ioannis Broustis. 2017. Coordinating rolling software upgrades for cellular networks. In *25th IEEE International Conference on Network Protocols, ICNP 2017, Toronto, ON, Canada, October 10-13, 2017*. IEEE Computer Society. https://doi.org/10.1109/ICNP.2017.8117537

[62] Shriram Rajagopalan, Dan Williams, and Hani Jamjoom. 2013. Pico replication: A high availability framework for middleboxes. In *ACM SoCC* (2013).

[63] Ermínio Augusto Ramos da Paixão, Rafael Fogarolli Vieira, Welton Vasconcelos Araújo, and Diego Lisboa Cardoso. 2018. Optimized load balancing by dynamic BBU-RRH mapping in C-RAN architecture. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. 100–104. https://doi.org/10.1109/FMEC.2018.8364051

[64] Rehenuma Tasnim Rodoshi, Taewoon Kim, and Wooyeol Choi. 2020. Resource Management in Cloud Radio Access Network: Conventional and New Approaches. *Sensors (Basel, Switzerland)* 20 (2020).

[65] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. 2018. VM Live Migration At Scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Williamsburg, VA, USA) *(VEE '18)*. Association for Computing Machinery, New York, NY, USA, 45–56. https://doi.org/10.1145/3186411.3186415

[66] 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

[67] Aidan Shribman and Benoit Hudzia. 2012. Pre-copy and post-copy VM live migration for memory intensive applications. In *European Conference on Parallel Processing*. Springer, 539–547.

[68] Tshiamo Sigwele, Atm S Alam, Prashant Pillai, and Yim F Hu. 2017. Energy-efficient cloud radio access networks by cloud based workload consolidation for 5G. *Journal of Network and Computer Applications* 78 (2017), 1–8.

[69] Radostin Stoyanov and Martin J. Kollingbaum. 2018. Efficient Live Migration of Linux Containers. In *ISC Workshops*.

[70] Sharan Turlapati and Srivatsa Bhat. 2021. Linux kernel support for kernel thread starvation avoidance. *Real-Time Micro-conference, Linux Plumbers Conference 2021* (2021). https://linuxplumbersconf.org/event/11/contributions/1061/

[71] Cheng Wang, Xusheng Chen, Weiwei Jia, Boxuan Li, Haoran Qiu, Shixiong Zhao, and Heming Cui. 2018. PLOVER: Fast, Multi-core Scalable Virtual Machine Fault-tolerance. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 483–489.

[72] Sen Xu, Meng Hou, Yu Fu, Honglian Bian, and Cheng Gao. 2018. Improved Fast Centralized Retransmission Scheme for High-Layer Functional Split in 5G Network. *Journal of Physics: Conference Series* 960 (2018).

[73] Xing Xu, Ioannis Broustis, Zihui Ge, Ramesh Govindan, Ajay Mahimkar, N. K. Shankaranarayanan, and Jia Wang. 2015. Magus: minimizing cellular service disruption during network upgrades. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT 2015, Heidelberg, Germany, December 1-4, 2015*. ACM. https://doi.org/10.1145/2716281.2836106

[74] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning *in situ*: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 495–511. https://www.usenix.org/conference/nsdi20/presentation/yan

[75] Qing Yang, Xiaoxiao Li, Hongyi Yao, Ji Fang, Kun Tan, Wenjun Hu, Jiansong Zhang, and Yongguang Zhang. 2013. BigStation: enabling scalable real-time signal processingin large MU-MIMO systems. *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (2013).

[76] Hang Yin, Nanxi Li, Jing Guo, Jianchi Zhu, and Xiaoming She. 2022. NR Coverage Enhancements for PUSCH. *IEEE Communications Magazine* (2022).

[77] Diyu Zhou and Yuval Tamir. 2021. HyCoR: Fault-Tolerant Replicated Containers Based on Checkpoint and Replay. *CoRR* abs/2101.09584 (2021). arXiv:2101.09584 https://arxiv.org/abs/2101.09584

[78] Diyu Zhou and Yuval Tamir. 2022. RRC: Responsive Replicated Containers. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 85–100.