

CHRONOS: Scalable and Cost-Effective Cloud-based RAN Emulation System

Ujjwal Pawar^{†,*}, Andrew E. Ferguson^{†,*}, Yufeng Xia[†], Xenofon Foukas[‡],
Mahesh K. Marina[†] and Bozidar Radunovic[‡]

[†]The University of Edinburgh [‡]Microsoft Research

Abstract

The trend towards softwarized and AI driven radio access network (RAN) in 5G and beyond has created a need for RAN emulators to test and enhance cellular network performance prior to real-world deployment. However, large-scale RAN emulation remains challenging due to the complexity of the RAN protocol stack and the high cost of hardware-based solutions. Software alternatives are more affordable but current solutions sacrifice scalability or fidelity. This paper presents CHRONOS, a cloud-based RAN emulation system that overcomes these challenges by abstracting the compute-intensive PHY layer, virtualizing emulation time, introducing RAN slot level synchronization, and scalable middlebox based traffic forwarding. CHRONOS operates at the FAPI boundary: it models PHY outcomes that affect higher-layer behavior, while deliberately excluding complex physical layer signal processing. Experiments using OpenAirInterface based prototype on cloud infrastructure demonstrate CHRONOS's ability to emulate a RAN with 10,000 mobile devices and 10,000 base stations on 130 cloud instances (with 6400 CPU cores), two orders of magnitude larger scale than existing solutions, reflecting its potential for enabling scalable RAN emulation in an affordable manner, all while maintaining fidelity.

CCS Concepts

• **Networks** → **Mobile networks**; **Cloud computing**; **Network experimentation**.

Keywords

Mobile networks, RAN, emulation, cloud computing

ACM Reference Format:

Ujjwal Pawar^{†,*}, Andrew E. Ferguson^{†,*}, Yufeng Xia[†], Xenofon Foukas[‡], Mahesh K. Marina[†] and Bozidar Radunovic[‡]. 2026. CHRONOS: Scalable and Cost-Effective Cloud-based RAN Emulation System. In *The 32nd Annual International Conference on Mobile Computing*

and Networking (MobiCom '26), October 26–30, 2026, Austin, TX, USA. ACM, New York, NY, USA, 18 pages. <https://doi.org/10.1145/3795866.3844156>

1 Introduction

Mobile cellular RAN has been going through a dramatic transformation for the past several years [21], embracing softwarized and data-driven operation leveraging AI/ML techniques via disaggregated architecture with open interfaces, controllers (RICs [15]), and apps (xApps, rApps) [32]. This created the need for controlled RAN evaluation and testing environments that are realistic and scalable. Such environments should allow sandbox testing of software upgrades and new features prior to deployment in production networks. They are equally key to realistic analysis of network behavior and evaluation of innovations *at scale* as well as to flexibly obtain realistic data to train and test AI/ML based RAN Apps, given real-world RAN data is either not easily accessible due to commercial sensitivity concerns, or unavailable due to limited operational deployments of Open RAN at present.

To meet these needs there exist three potential methodologies in the form of simulators, emulators and real testbeds. While simulators (e.g., ns-3 [35]) offer a high level of flexibility and control to realize large-scale RAN scenarios, they achieve that by trading off realism/fidelity and do not allow real-world virtualized RAN (vRAN) software to be tested. Real-world cellular testbeds (e.g., Powder [24]), on the other hand, are closest to reality but are expensive to scale and limit controlled, repeatable experimentation. RAN emulators [2, 13, 49] provide the happy middle ground between simulators and real testbeds, but they currently cannot scale beyond 100s of base stations (for a more detailed discussion, see §2).

Motivated by the above, we pursue a new *cloud based approach to RAN emulation* in this work. Conceptually, it falls under the software based RAN emulation category but crucially looks to unshackle it to scale freely while maintaining fidelity. Being cloud-friendly allows leveraging the commoditized compute and network resources of the cloud *on-demand* for RAN emulation, when needed and at the scale that is needed, cost effectively. We aim to achieve this flexibility



This work is licensed under a Creative Commons Attribution 4.0 International License.

MobiCom '26, Austin, TX, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2505-0/2026/10

<https://doi.org/10.1145/3795866.3844156>

* Co-Primary Authors

on any cloud infrastructure (public, or private, of any size) without sacrificing fidelity.

RAN emulation in a cloud environment and at large scale, while a very compelling proposition, presents a new set of challenges (§3). Given the resource demanding nature of the RAN, particularly at the physical (PHY) layer, naively realizing the whole RAN stack would be prohibitively expensive – emulating RAN scenarios with 100s of base stations would need 1000s of CPU cores and TB/s of network capacity. Another challenge arises from unpredictable compute preemption and network latencies, which conflict with the strict sub-millisecond latency requirements of the RAN software [37] and can therefore introduce performance artefacts. Preemption latencies stem from a variety of causes, from the lowest levels (BIOS firmware, platform timing noise, etc.) and up (hypervisor-level preemption on virtualized cloud instances, scheduling effects in multi-tenant environments, etc). Cloud systems are not provisioned or tuned as telco-grade RANs, so hosts lack the low-level control needed to optimize them accordingly. This issue is further amplified when realizing larger-scale RAN emulation scenarios across multiple cloud instances, due to the high networking latencies between instances. A further challenge stems from the heterogeneity of both preemption and network latencies, which makes it difficult to keep emulated RAN components running on different instances tightly synchronized.

To overcome these challenges, we propose a cloud-based RAN emulation system called CHRONOS (§4), building on our earlier work in [45]. CHRONOS is designed as an L2/L3-focused emulator for studying the behavior of real RAN control and data-plane software at scale. CHRONOS brings together several key ideas to enable RAN emulation at scale, cost-effectively, and while maintaining fidelity: i) to address the high compute and network demands of the RAN, it features lightweight emulation of the compute-and-network-intensive PHY layer and channel of the RAN and associated user devices (UEs) via a virtualized and distributed PHY implemented via a Middlebox based on the standard FAPI interface [30]; ii) it introduces a custom Supervisor that virtualizes emulation time, shielding emulated network functions from compute preemptions and inter-cloud-instance network latencies; iii) to support large-scale RAN emulation by scaling across multiple cloud instances while preserving correctness, it operates at the granularity of RAN slots and uses slots as synchronization barriers through a Slot Coordinator.

CHRONOS is designed to run any standards-compliant RAN software out of the box, without modification. The fidelity target is therefore the behavior exposed above the PHY abstraction boundary: MAC scheduling, HARQ interactions, RLC/PDCP behavior, RRC procedures, mobility, RIC control loops, and application-facing data-plane effects.

We develop a prototype implementation of CHRONOS on Ubuntu Linux. The platform part of CHRONOS implementation includes: a custom KVM-based Supervisor, a clock source kernel-module that synthesizes clock ticks for guest VMs, as well as a Slot Coordinator and Slot Monitor; the latter implemented in userspace. The Middlebox supports both 4G and 5G. The device (UE) implementation stems from OpenAirInterface, with minor interface modifications to allow for interfacing with the Middlebox. Finally, the vRAN software in our prototype implementation is based on OpenAirInterface, although any FAPI compliant vRAN stack can be used; CHRONOS naturally inherits the capabilities of the vRAN stack used as the system-under-test. Since CHRONOS connects at the FAPI boundary, it also inherits the CU/DU/RU functional split and deployment topologies supported by the underlying RAN stack – CHRONOS does not introduce an independent split model. We demonstrate that CHRONOS is portable across multiple different cloud providers, including Microsoft Azure, Google GCP and CloudLab.

We extensively evaluate CHRONOS with experiments over the cloud. Our results demonstrate the fidelity of CHRONOS in terms of both control and data plane operations when compared with a standard over-the-air RAN deployment with real UEs, using the same OpenAirInterface stack. Crucially, we show the excellent scaling capability of CHRONOS design, which allows us to achieve scenarios with 10,000 UEs / gNBs using 130 cloud instances with an aggregate of 6400 cores. Furthermore, we present several different use cases that showcase the capabilities offered by CHRONOS for scalable, flexible, controllable and low cost RAN emulation.

In summary, we make the following key contributions:

- We introduce cloud-based approach as a path to scalable, cost-effective and flexible RAN emulation (§3).
- We propose CHRONOS, a tailored system design that addresses the associated challenges with a suite of innovations including a custom Supervisor providing virtual time abstraction, slot-based synchronization and scalable Middlebox (§4). The design explicitly targets L2/L3 RAN emulation by placing the abstraction boundary at FAPI.
- We build a prototype implementation of CHRONOS, with a platform component portable across different cloud providers and the Middlebox component supporting both 4G and 5G RAN emulation (§5). We also document the configuration and bootstrapping workflow used to instantiate experiments across the emulated RAN components. We open-source the CHRONOS implementation along with all the documentation at <https://github.com/netsys-edinburgh/Chronos> to benefit the research community.
- Through extensive evaluations, we demonstrate that CHRONOS not only achieves two orders of magnitude improvement in RAN emulation scale over existing solutions but also does

so while maintaining fidelity (§6), despite the undesirable characteristics of cloud platforms.

- We demonstrate the utility and versatility of CHRONOS with four diverse use cases (§7).

Approach / Systems	Type	High Fidelity	Support Real vRAN Software	Large Scale	Low Cost
Network simulators [35]	SW	×	×	✓	✓
Cellular testbeds [19, 24, 29, 47]	HW	✓	✓	×	×
Hardware RAN emulators [2, 6, 13, 42, 55]	HW	✓	✓	×	×
Software RAN emulators [11, 43, 49]	SW	✓	✓	×	✓
Trace-based emulators [53]	SW	×	×	✓	✓
Cloud based RAN emulation (CHRONOS – this work)	SW	✓	✓	✓	✓

Figure 1: Approaches for RAN testing and evaluation.

2 Background and Motivation

RAN overview – Modern RAN stacks are made up of virtualized RAN (vRAN) software and a physical radio unit (RU). RUs along with the antennas are deployed at cell towers. The vRAN software consists of two components: a distributed unit (vDU), typically deployed at an edge cloud close to the cell towers, and a centralized unit (vCU or L3). The vDU can further be split into the physical layer signal processing (also called L1) and packet scheduling and retransmissions (L2).

Most vDU operations must be finished within a slot time (typically 0.5 ms or less), otherwise the network performance will be affected [37]. Therefore special care is dedicated to making sure that the software runs close to real time. The L1 layer runs complex signal processing algorithms, but it is essentially stateless [41]. The other layers (L2 and L3) have a lot of complicated per-user state, such as packet queues, channel and throughput history. The state drives the algorithms which affect some of the key network operations such as packet scheduling and handovers, some of which are real-time or near-real-time. It is the interplay among these algorithms that defines the performance of a cellular network and that vendors and operators continuously try to optimize. Many such optimization efforts over years were key to continuous improvement of the network performance (e.g., SON [17], ICIC [40], eICIC [25], xApps [36], rApps [48]).

Existing approaches and their limitations – There has been considerable prior effort in the field of cellular RAN testing and evaluation (see Figure 1 for a summary). As an example of a simulator, consider ns-3 [35], a widely used open-source network simulator. It is very flexible to model different scenarios and architectures (including cellular networks) and at large scales. But it cannot integrate with commercial RANs, and requires a different, event-based RAN implementation with its own idiosyncrasies. Testing the behavior of specific RAN stacks under certain scenarios is therefore not possible.

On the complete opposite end of the spectrum, physical testbeds [19, 24, 29, 47] integrate a real-world environment into the testbed itself. While offering high fidelity and an accurate environment for experimentation, they lack flexibility particularly to cost-effectively expand the scale beyond the geographical footprint or topology of the testbed beyond its existing configuration. Controlled hardware (HW) testbeds [55] utilize a large number of interconnected SDRs to significantly increase flexibility while still retaining high fidelity. However, their scale is physically limited by the number of SDRs and their interconnects (e.g., Colosseum contains 256 SDRs).

Several commercial one-box hardware-based RAN emulators [2, 6, 13] exist that provide the highest fidelity in a controlled manner. These solutions have a high cost, especially for large-scale RAN emulation, limiting their use to the labs of major telco vendors and operators. There also exist a subclass of hardware based or hardware dependent emulators that focus only on emulating channels (e.g., [5, 42]).

Software (SW) RAN emulators run on any compute, thus cost less. Some solutions avoid the time-sensitive vDU component altogether (e.g., [14]) vastly limiting their utility, while others encapsulate the entire L1 layer in software [11, 43] that requires enormous computational and network resources. The most promising recent work is EMANE [49] which abstracts away the more demanding computational tasks of the L1 layer, but retains its high-level characteristics. However, its scalability remains limited; it supports only tens of devices.

For application and transport layer focused studies, trace based network emulators like CellReplay [53] abstract the network path including the cellular link by recording propagation delay and available bandwidth traces, and replaying them.

3 Cloud-based Approach and Challenges

Clouds have been designed to provide an almost infinite amount of commoditized compute and network resources on demand. As such, they are ideally suited for large scale emulations. However, they have been designed for general purpose workloads, and cannot be optimized for demanding and latency sensitive workloads like the RAN as this would affect performance of most of the other workloads. If one wants to design a scalable RAN emulator for a cloud, they need to consider several significant interrelated challenges, outlined below.

3.1 C1: RAN Emulation Requirements

The foremost challenge is to confront the high RAN resource requirements while meeting the stringent latency constraints.

RAN resource requirements – The 5G RAN network functions can be very demanding in terms of compute and network resources, requiring several CPU cores and high network bandwidth for their operation. Fully emulating all layers of a large scale network with hundreds of base stations would require

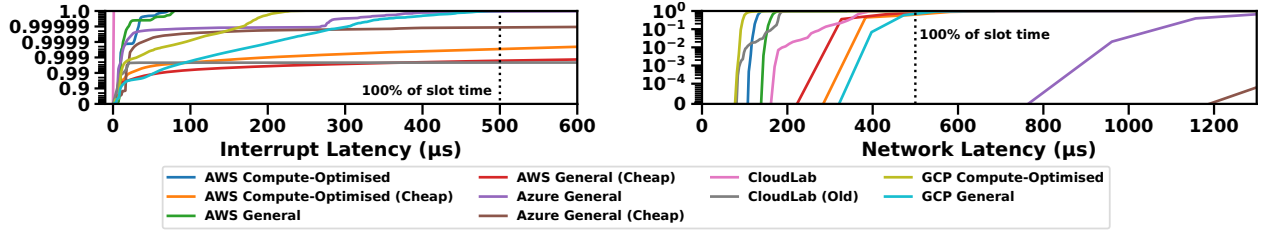


Figure 2: CDFs of compute interruption and network latencies for different cloud providers.

tens of thousands of CPU cores and terabits of network capacity, which would make the RAN emulation unviable, due to the cost or the availability of resources. As an example, we deploy a single 100 MHz 4×4 MIMO 5G cell based on the open source srsRAN software stack [12] and measure its CPU and network utilization when fully saturated with traffic. We observe that a minimum of 8 CPU cores and a 10G NIC interface is required to achieve full downlink throughput (~ 1 Gbps DL). Similar observations can also be made for other RAN stacks [8, 31]. Most of these resources are for the PHY processing, which, in practice, is offloaded to dedicated hardware accelerators (ASICs, FPGAs, GPUs) [3, 10, 51]. However, such offloading is not a typical option in cloud settings. On the other hand, PHY processing on general-purpose CPUs violates the strict latency constraints [51], as discussed next.

RAN latency constraints – The RAN operates in a time slotted fashion, with each slot having a duration in the order of hundreds of microseconds up to a millisecond. During each slot, the MAC scheduler of the RAN needs to perform the radio resource allocation in the control plane for both the uplink and the downlink direction. This involves deciding which users to schedule, their traffic flow priorities, how many resource blocks to allocate, etc. At the same time, the RAN also has to deal with the processing of the actual packets of users in the data plane (e.g., segmentation, adding headers, retransmissions). Both these control and the data plane operations must take place within the duration of the slot [37]. Otherwise, emulation fidelity will suffer due to packet drops, user detaches or worse (e.g., RAN crashes).

Utilizing unmodified commercial RAN software – National mobile networks deploy RAN products from a variety of vendors. We cannot expect that these commercial products can be easily adapted for RAN emulation environments. Instead, any RAN emulator should be able to evaluate unmodified commercial RAN software from multiple providers.

3.2 C2: Compute Interruption & Network Latencies

The multi-tenant nature of cloud platforms introduces uncontrollable compute preemptions. These stem from all layers of the compute stack, from the lowest levels (firmware, BIOS, and system configuration choices that are not optimized for RAN

workloads) to higher-layer cloud abstractions (e.g., hypervisor-level preemption and multi-tenancy scheduling effects). Further, realizing large-scale RAN emulation scenarios in a cloud environment often necessitates multi-instance deployments, which also introduce inter-node networking jitter.

Compute preemption – A typical commercial RAN deployment runs the vDU on carefully optimized bare-metal servers, often pinning DU threads to dedicated CPU cores and minimizing the number and duration of interruptions on those cores in order to tightly control tail execution latency (e.g., tail latency targets below $10\mu s$ [37]). Such control is generally unavailable in cloud environments. The underlying servers are not provisioned or tuned as telco-grade RAN hosts, and tenants do not have the BIOS or firmware-level access required to apply the low-level optimizations commonly used in production RAN deployments. In virtualized instances, tenants run inside virtual machines and are exposed to hypervisor-induced preemption and scheduling effects that are outside their control. Even when cloud providers offer high-performance instances, these compute preemptions remain present and can significantly perturb RAN slot-level execution. As a result, compute preemptions remain a practical source of timing instability.

We measured compute-preemption latencies on a variety of cloud instances (Azure, AWS, GCP and CloudLab) across both cost-optimized and high-performance instances. Using osnoise [9], we observe several platforms with tail latencies exceeding 20% of a 0.5 ms 5G slot. The latency distributions shown in Figure 2 (Left) vary substantially across cloud platforms, reflecting differences in platform technology, server tuning and hypervisor behavior; in general, lower-cost, older, or general-purpose (rather than compute-optimized) cloud instances exhibit worse tails. Even the best-performing cloud instances we tested still show tail latencies of around $70\mu s$, far above the preemption budget typically targeted in telco edge deployments [37].

Network latencies – Cloud instances belonging to the same virtual network are often placed far apart within the cloud infrastructure, frequently across different racks and connected through multiple layers of switching, each adding additional latency. We measured network latency and variability using ping RTT between two instances of the same type, co-located within the same cloud region. The results for different cloud

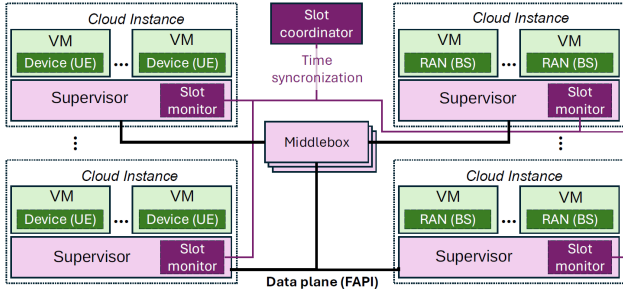


Figure 3: CHRONOS Architecture

platforms, and for different instance types, are shown in Figure 2 (Right). Considering a typical 5G slot duration of 0.5 ms, all instances except for the CloudLab bare-metal instance exhibit median network latencies that consume around 20% of the slot time, while the tail latency approaches 40%. This is significantly higher than the prescribed requirements, and therefore substantially reduces the processing budget left for RAN computation. At cloud scale, this is a fundamental challenge: as deployments grow across multiple instances, the probability that all participating nodes are placed close to one another in the physical network (e.g., under the same ToR switch) becomes increasingly small.

Overall, these observed latencies show that the RAN software may spend a large fraction of each slot time ($> 20\%$) waiting on network latency. Since each missed deadline translates into packet loss, this directly impacts emulation fidelity; in our measurements, running the RAN software *as-is* in the cloud leads to packet losses of up to 20%.

Heterogeneous preemptions and network jitter – The compute preemptions and network jitter cause another significant issue for realizing large-scale RAN emulation in a cloud environment. The compute preemptions are not equal in size, nor do they occur at the same time across multiple cloud instances. Similarly, the network latencies exhibit considerable jitter, for example due to the effects of other traffic on the cloud provider’s network, or even other traffic within the cloud instances used for the RAN emulator.

This can cause the RAN emulation scenarios involving multiple cloud instances not to work at all, let alone correctly. Correct operation requires emulated components to be perfectly synchronized. Such synchronization is lost, if for example, a base station (gNB) believes it is in one slot and a corresponding device (UE) finds itself in a different slot. In presence of varied compute preemptions and network jitter, keeping all emulated RAN components in sync is hard because cloud instances lack the precise microsecond level PTP based time synchronization capability used in production 5G RAN deployments.

4 CHRONOS System Design

4.1 Overview

RAN requirements (C1) inherently conflict with the non-real-time behavior of cloud platforms (C2). To make RAN resource requirements manageable and support unmodified real vRAN software (C1), CHRONOS relies upon PHY abstraction to significantly reduce the computational and network overheads of the PHY layer. We implement an abstracted and distributed PHY in a Middlebox, enabling resource efficient and scalable PHY abstraction. However, despite the improvements in resource efficiency, the non-real-time cloud platform offers no guarantees that RAN processing deadlines will be met. To that end, CHRONOS introduces a Supervisor that virtualizes the system clock. Operating in virtual time allows deadlines to be respected; cloud-platform-induced delays (C2), that would otherwise cause deadlines to be missed, are masked.

Additionally, because compute preemptions and network latencies vary across cloud instances (C2), explicit synchronization is required to preserve fidelity. This gives rise to two key challenges: identifying appropriate synchronization points and enforcing synchronization at scale. We address the first by exploiting RAN workload semantics, which naturally align with RAN slot boundaries. To address the second, we implement a global Slot Coordinator that enforces barrier synchronization across all components at every slot boundary, thereby ensuring system-wide alignment at the beginning and end of each slot. All components of the system are designed with scalability as a first-class principle.

4.2 Architecture

The above outlined design innovations are reflected in the CHRONOS system architecture, as illustrated in Figure 3. The emulator consists of four primary components: the emulated devices (UEs), the Middlebox, the Slot Coordinator, and the vRAN system itself (i.e., the base station, BS). The RAN software and the emulated UEs connect to the Middlebox through a virtual PHY (discussed in §4.3). Since CHRONOS interfaces with the RAN through standard-compliant interfaces, it can operate with any standards-compliant RAN implementation and naturally inherits the features and behavior of the underlying stack. In particular, CHRONOS operates at the FAPI [30] boundary between the PHY and the MAC layers. It does not redefine the CU/DU/RU split; the supported functional splits, cell layouts, and deployment topologies are those implemented by the RAN stack used. The Middlebox component processes the data-plane traffic. It is fully disaggregated and can scale horizontally, allowing the system to scale to 10,000s of UEs / BSs (§6.2).

5G RAN operates in time slots (a widely used slot duration in today’s 5G networks is 0.5 ms). We introduce time synchronization across all components at the slot boundaries. These

synchronization points are not manually selected by the experimenter. They are automatically derived from the configured RAN numerology, which determines the slot duration and therefore the barriers at which CHRONOS coordinates progress. To ensure synchronization, the Slot Coordinator acts as the central point of synchronization using the barrier protocol. It informs all connected components of the start of a slot, and waits for all of them to report that they finished processing the slot before advancing to the next slot. The synchronization protocol is simple yet scales well (§6.2), justifying its centralization.

In order to allow the real-time processing of the RAN and UEs in a cloud environment, we develop a customized Supervisor that virtualizes time and advances it based on the progress of the entire system, rather than based on the real time that passes in the physical world. This gives the RAN and UEs the illusion that no deadline violations ever occur (due to compute preemptions, network latencies, etc.), ensuring correctness without RAN code modifications. The RAN and UE components run in VMs controlled by the Supervisor. The mobile core and end application hosts can be similarly be realized in VMs over the Supervisor. A Slot Monitor component within the Supervisor communicates with Slot Coordinator, ensuring synchronization between components.

While virtualized time is not a new concept in network emulation [33, 38, 46, 56], existing systems do not provide a principled mechanism to set the *dilation factor* (ratio of emulation/virtual time to real time) based on system load. Notably, SliceTime [56] uses virtualized time to synchronize virtual machines with an ns-3-based network simulation through barrier-based execution, thereby removing the simulator’s real-time constraint. However, SliceTime targets general network emulation in which network behavior is still provided by the simulator, while real software executes only at the edge as external prototypes. As a result, the fidelity question in SliceTime is primarily about the interaction of external applications with the simulated environment, rather than about preserving the behavior of a real network stack itself. In contrast, CHRONOS applies virtualized time to mobile network emulation while integrating a real RAN software stack, thereby preserving stack fidelity in a way that simulator-based approaches do not. This distinction is important because SliceTime acknowledges that fidelity of simulation-based emulation is inherently limited by abstraction and by discrepancies between real systems and their software models.

4.3 System Components

RAN and UE: Physical (PHY) layer of RAN is very resource intensive (C1). However, the PHY layer is also almost *stateless* (c.f. [41]). All the state is stored and all the decisions are made

at the higher layers. The decisions are communicated to the PHY layer through an industry-standard FAPI protocol [30].

CHRONOS abstracts the PHY layer (the southbound FAPI) and connects it to an unmodified RAN software from any vendor. It can emulate various physical environments without affecting the fidelity of the results (§6.1). At the same time, this approach significantly reduces the compute requirements. We demonstrate this by comparing CHRONOS to an emulation setup where full PHY is present. We run a single UE, and measure both the CPU utilization and network bandwidth. We observe (Figure 4a) that the CPU and network bandwidth are significantly higher for the setup with the full PHY layer. The trend observed from the full PHY layer makes it prohibitively higher for use at larger scales – running an emulation scenario with hundreds or thousands of UEs would require vast compute resources compared to CHRONOS using PHY bypass. **Middlebox:** The purpose of the Middlebox is to act as a virtualized and distributed PHY layer, allowing efficient and scalable abstraction of PHY-level processing for both the emulated UEs and RAN. It processes FAPI messages. In each slot, a UE sends one FAPI message to the virtualized PHY defining the uplink traffic for the slot. Similarly, a RAN sends a FAPI message defining the downlink traffic. The Middlebox stores the FAPI messages as they arrive until messages from all components are received, and then aggregates the uplink and disaggregates the downlink messages.

A downlink FAPI message to a UE contains a list of information blocks specifying which spectrum chunks it should listen to or send in. It also contains some information blocks that are common for all UEs in a cell, such as spectrum allocation dedicated to a RACH procedure. The Middlebox disaggregates this message by creating multiple messages, one per UE, each containing the information blocks pertaining to that UE and the common ones, and sends it out.

The uplink FAPI messages contain information about the outcome of the transmissions (e.g., CRC result or RACH presence). Upon receiving multiple uplink FAPI messages from UEs in a cell, the Middlebox aggregates them by combining them into a single FAPI message to be delivered to the RAN.

It is important to notice that only messages from the UEs attached to a single RAN cell are aggregated/disaggregated together, and there is no state sharing across RAN BSs/cells. This allows us to fully parallelize operations across RAN cells. However, once parallelized, in each slot some cells will inevitably have more traffic than others, and finish later. Time dilation and barrier synchronization make sure that we adapt the global time to the stragglers. Together, these techniques allow us to achieve horizontal scaling of the data plane.

Handover operations are also inherently scalable (evaluated in §7.1), as each involves only localized message exchanges between the UE and the two RAN cells involved. Furthermore,

the stateless design of the FAPI protocol eliminates the need for explicit context management.

Supervisor: As discussed in §3, the RAN software has real-time requirements that are incompatible with the latency guarantees offered by the cloud. The purpose of the Supervisor is to ensure that RAN deadlines are always met. It transforms the unreliable cloud into a stable platform, atop which RAN and emulated UEs can be run. This must be done with no modification to the RAN.

To achieve this, we provide time virtualization through a Supervisor that splits the workload into bounded tasks, each with a rendezvous point. By preventing time from advancing beyond this rendezvous point, it prevents the overrunning of deadlines (Figure 4b). This is illustrated experimentally (Figure 4c) by varying the computation in a compute-bound processing task (analogous to RAN slot processing). Upon completion, the task reports how much time (internally) has elapsed. We run with and without Supervisor. While both achieve the same amount of work, the elapsed (internal) time is lower under Supervisor because it has stopped time. Without Supervisor, task time does not stop at the end of the slot, causing deadlines to be exceeded; for a RAN workload this causes failures, e.g., packet loss, a crash, or a loss in fidelity.

We implement a custom kernel module that interacts with the Slot Coordinator register exposed by the VM hypervisor, giving us full control over the timing of the operating system running in the VM. This allows us to start and stop time upon demand, and at a very fine granularity.

One of the key benefits of the RAN workload semantics is that it provides natural rendezvous points at the slot boundaries. RAN workload is triggered by FAPI messages within a slot and all the interaction between components is through those messages. Therefore, any time discrepancy within the slot does not affect the RAN behavior.

In addition to the RAN and UE components, time virtualization must also be applied to other system elements such as the cellular core and traffic generators. This is straightforward, as these components lack real-time constraints and operate on time scales significantly larger than the RAN slot. They do not block the barrier but simply wait for it to complete before progressing to the next slot.

Slot Coordinator ensures synchronization across all the individual components of CHRONOS, ensuring that at any given point in (real) time, they are all working within the same slot. It implements a barrier synchronization method, imposing boundaries between slots as rendezvous points. No component can proceed past a rendezvous point until all other components have completed processing up to the rendezvous point. This is demonstrated experimentally (Figure 4d) by running ten identical and looping CPU-bound processes in parallel, equivalent to RAN slot processing. With Slot Coordinator, the processes are tightly synchronized: the synchronization

loss (difference between the fastest and slowest process) is minimal and constant even when increasing the number of loop iterations (i.e., mean RAN slot processing duration); Slot Coordinator prevents synchronization loss by strictly enforcing synchronization at each slot (iteration) boundary. In contrast, when running without Slot Coordinator, the difference increases with the number of iterations. The interruptions are not equally distributed across the processes, nor do they have the same magnitude. At larger scales these differences accumulate across the processes, resulting in loss of synchronization.

Slot Monitor controls the system time in each Supervisor and communicates with the Slot Coordinator. As the barrier synchronization method requires only one message per cloud instance and per slot, we opt for a simple, centralized design. It scales with a number of cloud instances with only minimal impact on the dilation time (Figure 19).

5 Implementation

The Supervisor is implemented based on the KVM Hypervisor, allowing us to dynamically control the time of guest VMs. A key requirement is the ability of the cloud provider to support *virtualization*: the ability to run our own VM hypervisor inside the cloud instance. All major clouds support this feature (AWS, GCP, Azure, IBM, CloudLab, OVH, etc.). The Slot Monitor runs inside the guest VM and interfaces with both the Supervisor and the Slot Coordinator to manage the virtualized system time. The Middlebox is built on the EMANE emulator, substantially rearchitected for scale.

An emulation experiment is specified by a single configuration describing the RAN stack image, CU/DU/RU placement supported by that stack, numerology and bandwidth, number of cells and UEs, UE attachment and mobility inputs, traffic generators, SINR/channel parameters, and the cloud instances assigned to each component. During bootstrapping, Kubernetes first instantiates the Supervisor-enabled VMs, then starts the Slot Coordinator and Middlebox shards, launches the core network and vRAN components, registers the monitors, and finally starts the emulated UEs and traffic generators after all components have joined the slot barrier. The slot duration used by the barrier is derived from the configured numerology, so experiments configure radio parameters rather than manually selecting synchronization points.

For the vRAN, we use OpenAirInterface [8] (OAI) for both 4G and 5G. Accordingly, CHRONOS inherits OAI’s features and capabilities. The 5G core used was Open5GS [7]. Kubernetes manages the deployment of CHRONOS at scale. We note that the number of UEs that can be attached to a single cell is a property of the underlying vRAN implementation rather than of CHRONOS itself: for the 5G OAI stack, the documentation reports support for 16 UEs by default. CHRONOS imposes no

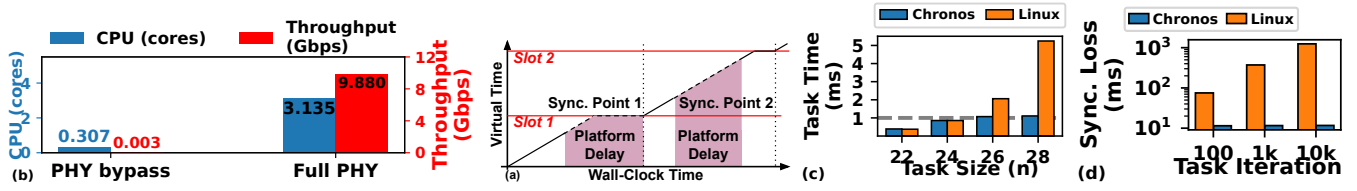


Figure 4: (a) CPU vs Network throughput for PHY bypass vs Full PHY based emulation; (b) CHRONOS time evolution: virtual time advances until it meets the slot time, then is frozen (Supervisor) until the synchronization point (Slot Coordinator) is reached by all instances; (c) Virtualized time for tasks of increasing computation, running with and without Supervisor; (d) Synchronization loss for tasks of increasing iterations, running with and without Slot Coordinator.

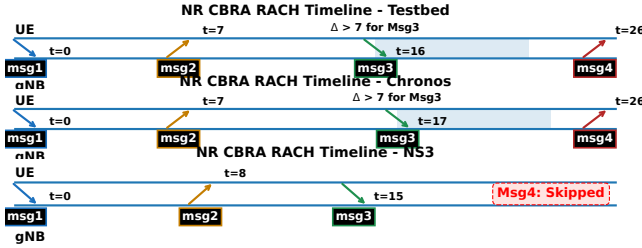


Figure 5: Control plane fidelity: CBRA Procedure

additional limit, and adopting a vRAN stack with a higher per-cell UE capacity would raise this bound accordingly.

CHRONOS emulates channel effects through SINR-based link abstraction rather than a waveform-level PHY [49]. For each transport block, it maps the UE’s SINR and the MAC-selected MCS to a BLER using AWGN reference curves and samples the decoding outcome. This outcome is reported through FAPI as an uplink CRC result or a downlink HARQ ACK/NACK, with a configurable SINR gain modelling HARQ combining across retransmissions. Because the feedback is introduced through the standard FAPI interface, the vRAN’s link-adaptation and HARQ procedures execute unmodified.

CHRONOS is inherently portable across a wide variety of cloud providers. We demonstrate the portability by deploying on multiple clouds, including CloudLab [26], Microsoft Azure, and Google GCP. Further implementation details can be found in Appendix A.

6 Evaluation

CHRONOS is capable of fulfilling a wide variety of RAN emulation experiments, and therefore supporting a wide variety of use cases. For example, CHRONOS can be used to model user mobility and its effect on the network at city scales; it can be used to collect training/test data for AI/ML based RAN applications. We discuss these applications of CHRONOS, and more, in Section 7. Here in this section, we dig deeper into the behavior of CHRONOS, validating its fidelity, and showing its scalability and ability to operate on cloud platforms.

6.1 Fidelity

For fidelity evaluation, we deploy a testbed consisting of two OAI 5G RAN base stations, an Open5GS core, and a Commercial Off-The-Shelf (COTS) UE, with the RAN operating

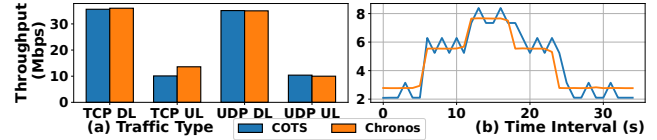


Figure 6: Data plane fidelity results, comparing iperf3 throughput between CHRONOS and a COTS UE: (a) ideal channel condition; (b) real channel trace.

in TDD mode at 20 MHz. We then create an emulation counterpart of this setup using CHRONOS, where the RAN and core versions are identical to the real testbed, but using CHRONOS instead of using the COTS UE.

Control-plane – We consider the contention based random access procedure (CBRA), which is invoked during the attachment of UEs [1]. Figure 5 illustrates the timing of events during the procedure, showing that both CHRONOS and the testbed follow an identical message / event flow, and a very similar timeline. Fidelity of CHRONOS is demonstrated as both the timing of events and the total duration closely match. We also compare this with the NS-3 5G-LENA attach procedure. The ns-3 deviates from 3GPP standards, CBRA sequence omits Msg4 (Contention Resolution): RA success is declared right after Msg2, as noted in the code (“in principle we should wait for contention resolution, but... there is no need”). Real NR requires a full Msg4 round-trip, adding 4–8+ slots of latency. Preamble collisions are also impossible in ns-3, since preambles are assigned sequentially, so every UE succeeds on the first attempt. This leads ns-3 to overestimate RACH success and underestimate attach latency. Implementing Msg4 would require random preamble selection, handling collisions, temporary C-RNTIs, Msg4 identity checks, and RACH retries adding significant protocol complexity.

Data-plane – We first study the achievable UE throughput. For this purpose, we generate TCP and UDP iperf traffic in both the uplink and the downlink direction and for ideal channel conditions (i.e., MCS = 28). As illustrated in Figure 6(a), both the emulated and the COTS UE achieve similar performance. We next evaluate the UE under varying signal quality conditions. For this purpose, we feed the CQI trace collected by [39] to the link adaptation algorithm of OAI in both the real and the emulated cases, and we generate TCP iperf traffic.

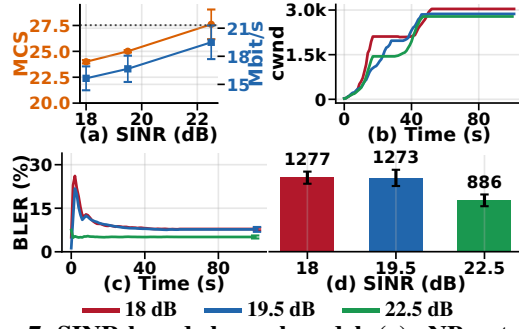


Figure 7: SINR-based channel model: (a) gNB outer-loop settled MCS and DL goodput vs SINR; (b) congestion window; (c) BLER evolution; (d) total HARQ retransmissions.

As illustrated in Figure 6(b), the throughput achieved by the CHRONOS UE closely matches that of the COTS UE, which further demonstrates the emulation fidelity. CHRONOS achieves this fidelity through time dilation; we explore how this feature enables flexible provisioning in Appendix B. Figure 7 validates the SINR-based link abstraction described in Section 5. As SINR increases from 18 to 22.5 dB, the link-adaptation loop converges to progressively higher MCS values (24 $\rightarrow$ 28), increasing DL goodput from 15.65 to 19.49 Mbit/s toward the loss-free ceiling of 21.32 Mbit/s. This confirms that the injected decoding outcomes drive the expected link-adaptation and HARQ behavior. Figure 7(b)-(d) show the corresponding transport-layer response: cwnd converges to a similar $\sim$ 2800-3000-segment plateau under all three conditions, reaching it slightly later at 18 dB ($\sim$ 4.9 s) than at 19.5/22.5 dB ($\sim$ 44 s); settled BLER falls from 7.85% to 7.70% to 5.10% as SINR increases, and total HARQ retransmissions fall correspondingly from 1277 to 1273 to 886. The same channel model is used in the closed-loop xApp experiment in §7.2.

Finally, we evaluate the impact of CHRONOS on data-plane latency. We measure ICMP ping RTT under two MAC scheduler configurations: conf#1 (inactivity=0, minrxtx=6) and conf#2 (inactivity=3, minrxtx=7). Here, ulsch_max_frame_inactivity controls how long an idle UE waits before being auto-scheduled, while minrxtx sets the minimum number of slots between a downlink reception and the UE’s uplink response. Together, these parameters can delay uplink grants and increase RTT. As expected, compared to conf 1, conf 2 (Figure 8(a)) yields higher RTT. Crucially, the COTS UE and CHRONOS show similar behavior, indicating CHRONOS closely matches real UE behavior given same scheduler configuration.

6.2 Scalability

To evaluate the scalability of CHRONOS, we deploy a multi-instance cluster on Cloudlab [26]. Specifically, we deploy a total of 132 instances: 90 56-core instances to support the RAN functions, 40 32-core instances for Middlebox, a single 32-core instance for the Slot Coordinator; one 64-core instance was used as the Kubernetes controller. Orchestration

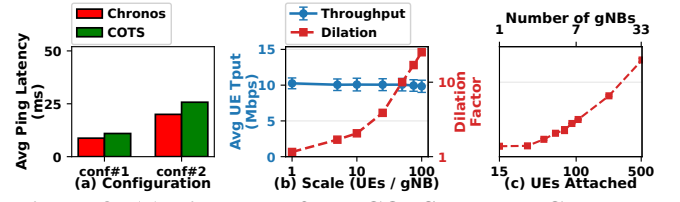


Figure 8: (a) Ping RTT for a COTS UE and CHRONOS under two MAC schedulers; (b) Data-plane throughput and dilation factor with increasing scale; (c) Dilation factor with increasing multi-UE-gNB pairs (15 UEs per gNB).

of the CHRONOS components atop the cluster was handled by Helm charts (Appendix A).

We first demonstrate the scalability of CHRONOS. We setup an experiment with a varying number of gNBs, where each gNB having a single UE connected to it. By increasing the number of gNBs from one to ten thousand, we demonstrate the ability of CHRONOS to realize large city-scale scenarios. The results, shown in Figure 9a, demonstrate that the data plane latency (as measured using irtt [34]) of the connected UEs does not substantially change as the scale (number of gNBs) increases by four orders of magnitude. In contrast, when we run EMANE with the same setup, EMANE fails to maintain fidelity: the data plane latency increases dramatically as the scale increases. We do not scale EMANE beyond 100 gNBs / UEs due to implementation issues preventing it from running at these scales, combined with per-packet data plane latencies that are sufficiently large (multiple seconds) to render the data plane effectively unusable. In other words, we achieve more than two orders of magnitude larger scale than what was previously possible.

Next, we assess the relative scalability of CHRONOS with respect to EMANE in terms of the percentage of UEs that were able to successfully establish a data plane connection with increasing number of UEs and gNBs (one UE per gNB). Figure 9b shows the results. Both systems successfully attached 5 UEs without any data plane failures. However, starting from 10 UEs, EMANE begins experiencing data plane failures, with not all of the UEs being able to establish a data plane connection. In contrast, CHRONOS achieves near 100% data plane connectivity for all scales of UEs tested, showing three orders of magnitude better scalability than EMANE. This behavior of CHRONOS is due to its dynamically-variable dilation factor that enables scale (particularly large scales).

We further demonstrate the scale of CHRONOS by considering throughput-focused workloads. In a smaller cluster on CloudLab (with 4 56-core cloud instances to support the RAN functions, 3 32-core instances for the Middlebox, a single 32-core instance for the Slot Coordinator, a 64-core instance as the Kubernetes controller), we deploy gNBs and UEs (one UE per gNB as before) but with iperf as the UE data plane application, running at 10Mbps for each UE. We

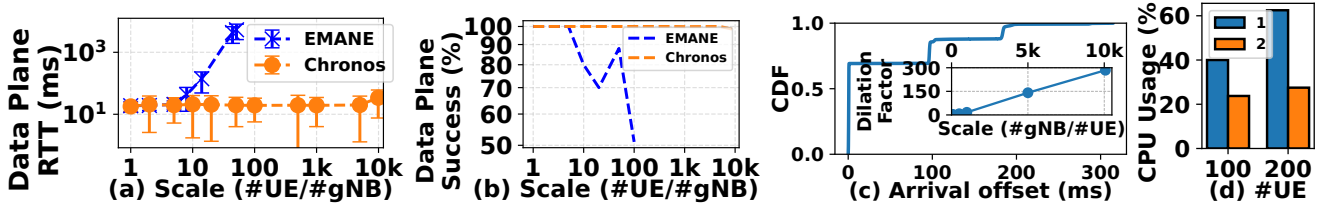


Figure 9: (a) Fidelity of the data plane (measured using the average UE RTT latency) with increasing scale (number of gNBs/UEs) for CHRONOS and EMANE; (b) Number of successfully connected UEs with increasing number of UEs/gNBs for both CHRONOS and EMANE; (c) (outer) Barrier arrival distribution from slot issue, plotted as a CDF; (inner) Variation of the dilation factor with scale (number of UEs or gNBs) when compute resource is fixed; (d) Average per-Middlebox CPU utilization for a monolithic (one Middlebox) and split (two Middlebox) setup.

measure both the average per-UE achieved throughput and the dilation factor. The results, shown in Figure 8(b), demonstrate that CHRONOS is able to scale while maintaining fidelity even under more-demanding data-plane tasks.

The experiments in Figure 8(b) use a 1:1 UE-to-gNB mapping, but CHRONOS is not restricted to this topology. To demonstrate this, we evaluate multi-UE-per-node configurations: 15 UEs connected to a single 5G gNB, and 100 UEs connected to a single 4G eNB. Both cases achieve a clean 100% UE attach. How the dilation factor changes with scale for the 15 UEs per 5G gNB configuration is shown in Figure 8(c), while Appendix C (Figure 18) shows the equivalent experiment for 100 UEs per 4G eNB.

We now measure the dilation factor for the scaling experiments (Figure 9a). The results, shown in Figure 9c (inner), reveal *how* CHRONOS emulates the desired scales despite constrained compute. At higher scales, CHRONOS begins competing for resources with itself, resulting in compute preemptions to the workload. As previously demonstrated, CHRONOS handles preemptions by increasing the dilation factor. The result is a linear relationship between scale and the dilation factor. This makes intuitive sense: time dilation allows compute to be traded for time, so halving the available compute resources doubles the experiment time.

We dig deeper into the effect of this at the largest scale. Figure 9c (outer) shows the barrier arrival offset of the Middlebox from the slot issue. The tail to 300 ms matches the observed dilation factor of 300.

6.3 Public Cloud Operation

We next evaluate the ability of CHRONOS to operate on the public cloud. We first consider how CHRONOS copes with the characteristics of cloud platforms, then evaluate our design choice of disaggregating the Middlebox. Finally, we consider the overhead of nested virtualization.

6.3.1 Cloud Characteristics. As highlighted earlier (Figure 2), we have measured multiple cloud platforms and observed significant compute preemptions and network latencies. Such

interruptions cause significant challenges for running unmodified RAN stacks in cloud environments (C2). We evaluate the effect of these interruptions on unmodified RAN stacks, as well as demonstrating the ability of CHRONOS to ensure correctness despite them. We run both CHRONOS and EMANE (which uses an unmodified OAI RAN stack) on a controlled testbed as well as on a public cloud platform (Microsoft Azure). The results (Figure 10a) reveal significant fidelity degradation of EMANE / OAI on the cloud. The EMANE UE RTT nearly doubles compared to the same setup on the controlled testbed. In contrast, the fidelity of CHRONOS remains constant.

While this implicitly demonstrates that CHRONOS can operate correctly despite cloud compute preemptions and network latencies, we now quantify this explicitly. On the controlled testbed, we again setup a single-UE instance of CHRONOS. We introduce deliberate compute preemptions (via *stress-ng* running in the cloud instance, specifically targetting CPU cores allocated to the VM hosting the CHRONOS RAN stack) and network variability (inducing a deliberate network delay at the Middlebox, affecting the latency of traffic going to and from the UEs and gNBs) and show that despite these, CHRONOS operates with no loss in fidelity. In Figures 10c and 10d, we plot the UE data plane latency and CHRONOS dilation factor against the size of these compute preemptions and deliberate network latencies. It can be noted that despite the increasing interruptions, the UE data plane latency (an indicator of CHRONOS fidelity) remains constant. The dilation factor increases with the increasing interruptions, showing that CHRONOS is compensating for them by taking more time to process each slot.

6.3.2 Scaling Out. A key feature of CHRONOS is the distributed Middlebox (§4.3), enabling it to scale out across multiple cloud instances. Without this, CHRONOS would be scale-limited by largest single cloud instance size, around 384 CPU cores for a typical cloud [16].

We compare the distributed CHRONOS Middlebox with a monolithic equivalent (e.g., the approach taken by EMANE). Specifically, we compare the Middlebox distributed across two cloud instances (both 8-core) with a single Middlebox that serves all gNBs / UEs. In Figure 9d, we look at the average

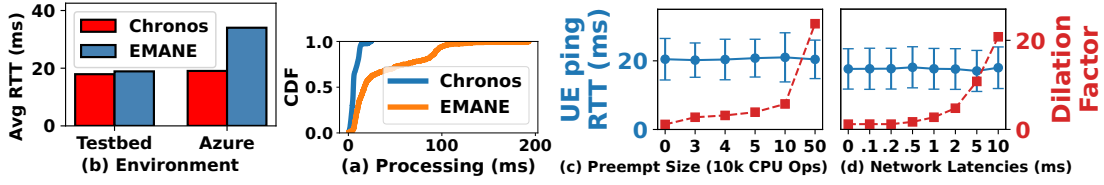


Figure 10: (a) CHRONOS vs EMANE deployed on a controlled testbed and Azure cloud. (b) Middlebox processing time of distributed Middlebox (CHRONOS) vs monolithic (EMANE) approaches, running 250 gNBs/UEs, demonstrating the lower processing tail and scaling advantage of the CHRONOS architecture. (c) and (d): Average UE ping latency and dilation factor vs. (c) compute preemption duration and (d) induced network latency.

per-Middlebox CPU utilization, as we double the scale from 100 to 200 gNBs / UEs (one UE per gNB). The CPU utilization for the monolithic Middlebox is approximately double that of the distributed setup, demonstrating that the distributed Middlebox can scale out and not become a processing bottleneck – indeed, the large-scale experiments (§6.2) utilized a distributed setup with the Middlebox running across 40 cloud instances. In contrast, the monolithic Middlebox limited to a single machine will rapidly saturate under increasing load.

We now compare the processing latency of monolithic Middlebox (EMANE) vs the distributed equivalent used by CHRONOS. We run 250 gNB scenario (one UE per gNB), connected to first the monolithic Middlebox, then the CHRONOS distributed setup with ten Middlebox instances handling the same load. The results (Figure 10b) show the high tail latency that occurs when a single Middlebox becomes bottlenecked. The distributed Middlebox used by CHRONOS avoids this by scaling out rather than bottlenecking.

We also show that the overhead from scaling out across multiple cloud instances is minimal (Appendix D). While saturating the Middlebox will not break CHRONOS due to its dynamic dilation, the Middlebox will become the bottleneck, causing significant increases in the time required to emulate large-scale scenarios. Without a distributed Middlebox, realizing such large-scale scenarios becomes impractical.

6.3.3 Nested Virtualization. CHRONOS realizes time dilation through virtualization, whereby gNBs and emulated UEs run in a virtual machine in which time can be controlled. On cloud platforms themselves deploying virtualized instances, the workload runs under *nested virtualization*, i.e. a VM that is itself running in a cloud VM instance through the cloud hypervisor. It is important to characterize the impact that this architecture has on the performance of CHRONOS – what overhead does running multiple hypervisors have?

We compare CHRONOS running on two clouds: one with virtualized instances over Microsoft Azure, and the other with bare-metal instances (no hypervisor) over CloudLab. We deploy a single UE and run both RTT and throughput tests. The fidelity of the two experiments was identical, and the dilation factor differs only by a small margin – for RTT it was 1.08 (Cloudlab) vs 1.20 (Azure), while for throughput it was

1.08 (CloudLab) vs 1.21 (Azure). From this, we conclude that nested virtualization has a negligible overhead on CHRONOS. Note that this difference in dilation factors is due to a number of factors, of which nested virtualization is but one. Other factors include the cloud characteristics as evaluated previously (§6.3.1). Unfortunately, isolating the effect of nested virtualization was not possible due to Azure not offering bare-metal instances as a generally-available machine type.

7 Use Cases

In this section, we present a representative set of use cases to demonstrate the utility and versatility of CHRONOS.

7.1 City Scale User Mobility & Its Impact

As a first use case, we demonstrate the ability of CHRONOS to support large-scale RAN emulation scenarios. We consider device (UE) mobility at city scale and its impact on signalling traffic, using a dataset from a major cellular operator’s city-wide deployment. The dataset includes about 1000 base stations (cells) and an inter-cell handover probability matrix. In this scenario, we emulate 1000 UEs and examine how mobility affects the distribution of users per cell and the resulting handover signalling traffic. UE mobility classes (stationary, pedestrian, vehicular) follow standard 3GPP assumptions [28]. Each UE’s handover rate is computed as a function of speed and base-station density using a homogeneous Poisson point process model [50].

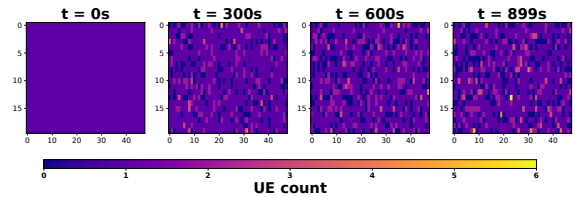


Figure 11: UE density evolution over a 15 min period. Each pixel aggregates 20 cells (pixel color = #UEs).

At the start of the emulation experiment, every cell has one UE each. We then visualize the UE density evolution over time with heatmaps in which each pixel aggregates 20 cells (color represents UEs per pixel). At $t = 0$, the network is initialized with a uniform occupancy of one UE per cell across 1000

cells (arranged as a 20×50 grid), as shown in Figure 11. The heatmaps at $t \in \{300, 600, 899\}$ s show progressive deviation from uniformity. By $t = 300$ s most cells have 0–2 UEs while some with 3–5 UEs; by $t = 600$ s max UEs/cell reaches ≈ 6 ; and by $t = 899$ s stable hotspots of 7–8 UEs/cell emerge in areas where inter-cell handovers are likely.

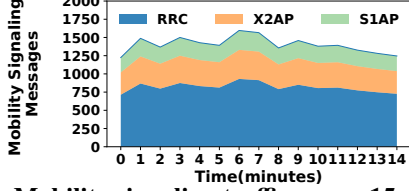


Figure 12: Mobility signaling traffic over a 15-min period.

We quantify mobility-induced control-plane signalling during a peak hour (Figure 12). Message rates for RRC, S1AP and X2AP over time are shown. During simultaneous handovers, RRC messages spike first (e.g., MeasurementReport, RRCReconfiguration), followed by X2AP signalling (HandoverRequest/RequestAck, UEContextRelease) and finally S1AP messages (PathSwitchRequest/Ack). CHRONOS can reproduce spatial UE dispersion and estimate time-varying handover signalling demand, enabling city-scale capacity planning.

7.2 Data for AI/ML based RAN Apps

We demonstrate how CHRONOS supports both realistic RAN KPI data collection and closed-loop RIC experimentation. AI/ML-based RAN applications span scheduling, interference mitigation, resource slicing, application-level QoS and more [18, 20, 27, 44, 54]. However, existing studies often rely on small testbeds, homogeneous traffic, and only a few UEs [18, 20, 27, 44], limiting the diversity of data available for training and evaluation. CHRONOS combines configurable traffic, channel conditions, and cell load with an unmodified vRAN stack and standard RIC interfaces, including compatibility with platforms such as OpenRAN Gym [22].

KPI data collection. We use CHRONOS with the OpenAirInterface5G RAN stack and collect per-UE KPIs through the KPIMON xApp over FlexRIC [52]. Our scenario comprises 10 cells and 44 UEs across 20MHz and 40MHz cells, with low (1 UE), medium (4 UEs), and high (15 UEs) load configurations. UEs are assigned different channel conditions using CHRONOS’s SINR-based channel emulation and generate a mixture of UDP streams, web bursts, low-rate DNS traffic, and voice-like keepalives using D-ITG [23]. All cells use proportional-fair scheduling.

Figure 13 (a) shows the resulting per-UE RB grants over 150ms intervals. Low-load cells exhibit a concentrated distribution because a single UE faces no contention and therefore receives similar RB grants across intervals. Medium-load cells become bimodal as UEs alternate between larger allocations

when fewer flows are active and smaller allocations when several UEs compete for resources. In high-load cells, many UEs with different traffic demands and channel conditions contend simultaneously, causing the proportional-fair scheduler to distribute RBs across a much broader range of per-UE allocations. Thus, changes in contention, traffic, and channel conditions reshape the complete scheduler-level KPI distribution rather than only its mean, providing diverse data for training and evaluating RAN applications.

Closed-loop control. We further deploy an xApp that monitors a UE (arbitrarily chosen from a medium-loaded cell) through E2SM-KPM indications and adjusts the Outer-Loop Link Adaptation (OLLA) target BLER, which controls MCS aggressiveness, through E2SM-RC. The UE operates at 18dB SINR with 3dB HARQ gain and carries a TCP DL traffic. The xApp applies an A–B–A control sequence, changing the target BLER from $0.05 \rightarrow 0.20 \rightarrow 0.05$ with 150sec holds. Each control request is acknowledged by the E2 node and produces the expected reversible MAC response. This is shown in Figure 13 (b-d), where settled MCS changes from $23.9 \rightarrow 25.0 \rightarrow 24.0$, mean cwnd from $2558 \rightarrow 527 \rightarrow 2669$ segments, and goodput from $16.6 \rightarrow 11.5 \rightarrow 16.9$ Mbit/s. Settled BLER tracks the commanded target closely throughout ($5.1\% \rightarrow 23.8\% \rightarrow 4.9\%$), and the looser target drives a corresponding rise in HARQ retransmissions ($1,747 \rightarrow 5,956 \rightarrow 1,922$), which caps cwnd early under CUBIC and thus lowers goodput.

Appendix E reports the E2 message sequence, confirming that CHRONOS preserves E2AP message ordering, control delivery, and scheduler-visible effects in emulated RAN time. Time dilation only stretches wall-clock execution; it does not change the sequence or semantics of E2-based control.

7.3 Controlled Testing of vRAN Software

CHRONOS enables reproducible, slot-accurate testing of RAN software within CI/CD pipelines. Tests can delay, drop, reorder, or modify selected messages at specific slots or HARQ processes: time virtualization ensures the same fault is injected at the same point across runs. Each build can be compared against established control-plane and performance baselines without relying on COTS UEs.

We integrated CHRONOS with the openairinterface5g CI/CD pipeline (commit c4c1dae) and tested failure handling during NR random access. We replaced the UE’s HARQ ACK for downlink Msg4 with a NACK while leaving all other signalling unchanged. The gNB retransmitted Msg4 until the HARQ limit and correctly declared random-access failure, but failed to release the associated UE resources (Fig. 14(a)). Repeated failures created “zombie” UE contexts that remained in the scheduler and could eventually exhaust gNB resources.

We patched the gNB to call `nr_mac_trigger_ul_failure()` when the Msg4 HARQ retries are exhausted. This removes the RNTI and clears the corresponding scheduler and HARQ

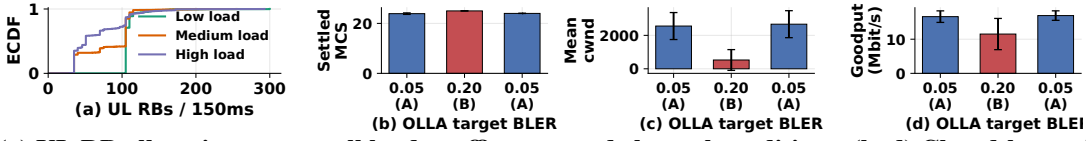


Figure 13: (a) UL RB allocation across cell load, traffic type, and channel conditions. (b–d) Closed-loop xApp control under time dilation: changing the OLLA target BLER (0.05 $\rightarrow$ 0.20 $\rightarrow$ 0.05) produces reversible changes in (b) MCS, (c) mean cwnd, and (d) TCP goodput.

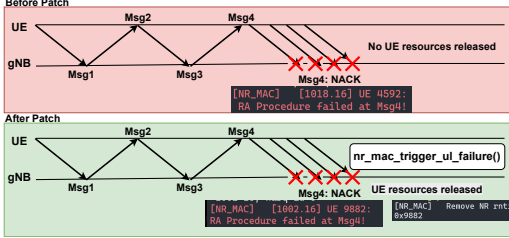


Figure 14: CHRONOS integration in CI/CD pipeline: (a) before patch, (b) after patch.

state. Repeating the same CI test confirmed that the patched build consistently released the UE resources (Fig. 14(b)). This experiment shows how CHRONOS extends RAN CI/CD beyond basic functional testing: reproducible, slot-accurate fault injection can expose state-management and control-plane bugs that conventional tests may miss.

7.4 Low Cost and Reliable PHY Channel Emulation

CHRONOS supports full-stack PHY emulation on commodity CPUs without specialized accelerators. Software-only PHY emulators [11] are easy to deploy but may exceed slot deadlines under load, causing timing drift, increased latency, and reduced throughput. Hardware-accelerated platforms such as FlexRAN [4] provide better timing fidelity but require hardware (e.g., Intel ACC100), increasing cost and limiting access.

CHRONOS preserves slot correctness by advancing emulated RAN time only after the PHY tasks for each slot are complete. An eBPF probe dynamically attached to the RAN’s slot handler detects slot boundaries without modifying the RAN stack. Time advances normally for the nominal slot duration and is then dilated if processing requires longer. Consequently, slots may take longer in wall-clock time, while control- and data-plane operations remain aligned in emulated RAN time.

We compare CHRONOS against the vanilla srsRAN-ZMQ PHY emulator on identical hardware: an Intel Xeon 5512U server with 56 cores and 128GB RAM. Both configurations use a 20MHz channel, Open5GS and one srsUE, with uplink and downlink TCP traffic generated using *iperf3*. Figure 15a shows that the vanilla emulator frequently exceeds the 1ms slot deadline under both idle and traffic workloads; CHRONOS preserves deterministic slot progression in emulated time.

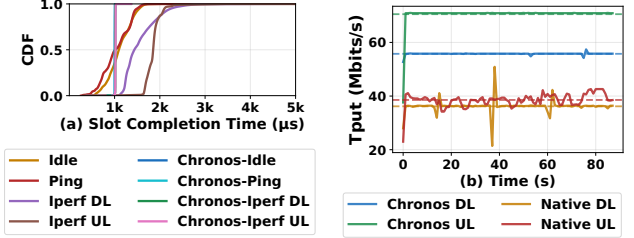


Figure 15: (a) Slot completion time with and without CHRONOS. (b) Throughput impact.

These overruns also affect end-to-end performance. Repeated deadline misses cause processing to spill into subsequent slots, eventually desynchronizing the gNB and UE and reducing throughput. As shown in Figure 15b, the vanilla emulator therefore delivers lower and less stable throughput, while CHRONOS maintains predictable performance. Thus, CHRONOS enables slot-correct CPU-only PHY experimentation, at the cost of slower wall-clock execution when processing exceeds real-time deadlines.

8 Limitations

CHRONOS is an L2/L3-focused RAN emulator that operates at the FAPI boundary between PHY and MAC. This allows CHRONOS to achieve very large scale, but represents PHY behavior only through outcomes exposed to MAC and higher layers via FAPI. So, CHRONOS is appropriate for experiments where PHY impact is represented through FAPI-visible outcomes, but not for evaluating timing-critical L1 behavior.

Note that CHRONOS does not perform channel estimation, symbol-level processing, waveform or I/Q sample processing, detailed beamforming or precoding, RF impairment modelling, or other L1 operations below the FAPI boundary. Hardware testbeds and full-PHY emulators remain the appropriate tools for benchmarking PHY implementations, accelerator efficiency, radio front-end behavior, or compliance with sub-slot L1 deadlines.

9 Conclusions

We have presented CHRONOS, a system that enables high-fidelity, scalable RAN emulation in a cost effective manner through a novel cloud based approach. Our system improves L2/L3-focused RAN emulation scale by two orders of magnitude over existing solutions, while maintaining fidelity; we demonstrate its utility through a variety of use cases.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their helpful comments and suggestions that greatly improved this paper. We also thank CloudLab for providing the computing resources used in this work. The work of M. Marina was supported in part by the UKRI/EPSC grant UKRI860.

References

- [1] 5G/NR - Initial Access/RACH. https://www.sharetechnote.com/html/5G/5G_RACH.html.
- [2] AMARI UE Simbox Series. <https://www.amarisoft.com/test-and-measurement/network-testing/network-products>.
- [3] Dell Open RAN Accelerator Card. Technical report. Dell Solution Brief.
- [4] Intel FlexRAN. <https://www.intel.com/content/www/us/en/developer/topic-technology/edge-5g/tools/flexran.html>.
- [5] Keysight Propsim. <https://www.keysight.com/us/en/products/channel-emulators/propsim-platforms.html>.
- [6] Keysight UESIM. <https://www.keysight.com/us/en/product/P8800S/uesim-ue-emulation-ran-solutions.html>.
- [7] Open5GS. <https://open5gs.org/>.
- [8] OpenAirInterface. <https://openairinterface.org/>.
- [9] OSNOISE. <https://docs.kernel.org/trace/osnoise-tracer.html>.
- [10] Platform and Performance Advantages to Accelerate vRAN Deployments. Technical report. Intel Solution Brief.
- [11] srsRAN gNB with srsUE. <https://docs.srsran.com/projects/project/en/latest/tutorials/source/srsUE/source/index.html#multi-ue-emulation>.
- [12] srsRAN Project. <https://www.srslte.com/>.
- [13] Viavi TM500. <https://www.viavisolutions.com/en-us/products/tm500-network-tester>.
- [14] Viavi TM500 Cloud. <https://www.viavisolutions.com/en-us/products/tm500-cloud>.
- [15] O-RAN Architecture Overview. <https://docs.o-ran-sc.org/en/latest/architecture/architecture.html>, 2025.
- [16] Supported CPU options for Amazon EC2 instance types. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/cpu-options-supported-instances-values.html>, 2026.
- [17] 3GPP. Self-Organizing Networks (SON) for 5G networks. *ETSI TS 128 313 V16.0.0*, 2020.
- [18] N. Apostolakis, M. Gramaglia, L. E. Chatzieftheriou, T. Subramanya, A. Banchs, and H. Sanneck. Athena: Machine learning and reasoning for radio resources scheduling in vRAN systems. *IEEE Journal on Selected Areas in Communications*, 42(2):263–279, 2023.
- [19] P. Bahl, M. Balkwill, X. Foukas, A. Kalia, D. Kim, M. Kotaru, Z. Lai, S. Mehrotra, B. Radunovic, S. Saroiu, et al. Accelerating Open RAN Research Through an Enterprise-scale 5G Testbed. In *Proc. ACM MobiCom '23*, pages 1–3, 2023.
- [20] M. Belgiovine, J. Gu, J. Groen, M. Sirera, U. Demir, and K. Chowdhury. Megatron: Machine learning in 5G with analysis of traffic in open radio access networks. In *2024 International Conference on Computing, Networking and Communications (ICNC)*, pages 1054–1058. IEEE Computer Society, 2024.
- [21] L. Bonati et al. Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead. *Computer Networks*, 182:107516, 2020.
- [22] L. Bonati et al. OpenRAN Gym: AI/ML development, data collection, and testing for O-RAN on PAWR platforms. *Computer Networks*, 220:109502, 2023.
- [23] A. Botta, A. Dainotti, and A. Pescapè. A tool for the generation of realistic network workload for emerging networking scenarios. *Comput. Networks*, 56(15):3531–3547, 2012.
- [24] J. Breen et al. Powder: Platform for Open Wireless Data-driven Experimental Research. *Computer Networks*, 197:108281, 2021.
- [25] S. Deb et al. Algorithms for Enhanced Inter Cell Interference Coordination (eICIC) in LTE HetNets. *arXiv preprint arXiv:1302.3784*, 2013.
- [26] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Riccart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.
- [27] S. D’Oro, L. Bonati, M. Polese, and T. Melodia. OrchestRAN: Orchestrating network intelligence in the open RAN. *IEEE Transactions on Mobile Computing*, 23(7):7952–7968, 2023.
- [28] ETSI. Study on channel model for frequencies from 0.5 to 100 GHz. 2020.
- [29] A. E. Ferguson et al. Campus5G: A Campus Scale Private 5G Open RAN Testbed. *SIGCOMM Comput. Commun. Rev.*, 55(3):19–28, Jan. 2026.
- [30] S. C. Forum. FAPI specifications. <https://www.smallcellforum.org/work-items/fapi>.
- [31] X. Foukas and B. Radunovic. Concordia: Teaching the 5G vRAN to share compute. In *Proc. ACM SIGCOMM '21*, pages 580–596, 2021.
- [32] A. Garcia-Saavedra and X. Costa-Pérez. O-RAN: Disrupting the Virtualized RAN Ecosystem. *IEEE Communications Standards Magazine*, 5(4):96–103, 2021.
- [33] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker. To infinity and beyond: time warped network emulation. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles, SOSP '05*, page 1–2, New York, NY, USA, 2005. Association for Computing Machinery.
- [34] P. Heist. Irtrt (isochronous round-trip tester). Available: <https://github.com/heistp/irtrt>.
- [35] T. Henderson, M. Lacage, G. F. Riley, C. Bèrnièr, and S. Floyd. Network Simulations with the ns-3 Simulator. In *Proc. ACM SIGCOMM '08*, 2008.
- [36] M. Hoffmann et al. Open RAN xApps Design and Evaluation: Lessons Learnt and Identified Challenges. *IEEE Journal on Selected Areas in Communications*, 41(10):2345–2357, 2023.
- [37] Intel. Intel-HPE Verified Reference Configuration for vRAN on the HPE ProLiant DL110 Gen11. Technical report.
- [38] D. Jin, Y. Zheng, H. Zhu, D. M. Nicol, and L. Winterrowd. Virtual Time Integration of Emulation and Parallel Simulation. In *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, pages 201–210, 2012.
- [39] W.-H. Ko, U. Ghosh, U. Dinesha, R. Wu, S. Shakkottai, and D. Bharadia. EdgeRIC: Empowering real-time intelligent optimization and control in NextG cellular networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1315–1330, Santa Clara, CA, Apr. 2024. USENIX Association.
- [40] C. Kosta et al. On Interference Avoidance through Inter-Cell Interference Coordination (ICIC) based on OFDMA mobile systems. *IEEE Communications Surveys and Tutorials*, 15(3):1245–1271, 2013.
- [41] N. Lazarev, T. Ji, A. Kalia, D. Kim, I. Marinos, F. Y. Yan, C. Delimitrou, Z. Zhang, and A. Akella. Resilient Baseband Processing in Virtualized RANs with Slingshot. In *Proceedings of the ACM SIGCOMM 2023 Conference, ACM SIGCOMM '23*, page 654–667, New York, NY, USA, 2023. Association for Computing Machinery.
- [42] X. Lin, L. Kundu, C. Dick, E. Obiodu, T. Mostak, and M. Flaxman. 6G Digital Twin Networks: From Theory to Practice. *IEEE Communications Magazine*, 61(11):72–78, 2023.
- [43] A. Mamaghani, U. Ghosh, S. Shakkottai, D. Bharadia, and I. K. Jain. Tiny-Twin: A CPU-Native Full-stack Digital Twin for NextG Cellular

- Networks, 2026.
- [44] E. Moro, M. Polese, A. Capone, and T. Melodia. An Open RAN framework for the dynamic control of 5G service level agreements. In *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 141–146. IEEE, 2023.
 - [45] U. Pawar, A. E. Ferguson, Y. Takano, J. Larrea, X. Foukas, M. K. Marina, and B. Radunovic. Towards Scalable and Cost-Effective RAN Emulation Leveraging the Public Cloud. In *Proceedings of the 26th International Workshop on Mobile Computing Systems and Applications, HotMobile '25*, page 43–48, New York, NY, USA, 2025. Association for Computing Machinery.
 - [46] D. Pediaditakis, C. Rotsos, and A. W. Moore. Faithful reproduction of network experiments. In *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '14*, page 41–52, New York, NY, USA, 2014. Association for Computing Machinery.
 - [47] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejwski, M. Sherman, Z. Kostic, X. Gu, H. Krishnaswamy, S. Maheshwari, P. Skrimponis, and C. Gutterman. Challenge: Cosmos: A city-scale programmable testbed for experimentation with advanced wireless. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, MobiCom '20*, New York, NY, USA, 2020. Association for Computing Machinery.
 - [48] RCR Wireless News. xApps vs. rApps: Network automation fundamentals. https://www.rcrwireless.com/2021/11/09/open_ran/xapps-vs-rapps-network-automation-fundamentals, 2021.
 - [49] B. Ryu, R. Knopp, M. Elkadi, D. Kim, and A. Le. 5G-EMANE: Scalable Open-Source Real-Time 5G New Radio Network Emulator with EMANE. In *Proc. IEEE MILCOM 2022*, pages 553–558. IEEE, 2022.
 - [50] S. Sadr and R. S. Adve. Handoff rate and coverage analysis in multi-tier heterogeneous networks. *IEEE Transactions on Wireless Communications*, 14(5):2626–2638, 2015.
 - [51] L. L. Schiavo, G. Garcia-Aviles, A. Garcia-Saavedra, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez. CloudRIC: Open Radio Access Network (O-RAN) Virtualization with Shared Heterogeneous Computing. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, ACM MobiCom '24*, page 558–572, New York, NY, USA, 2024. Association for Computing Machinery.
 - [52] R. Schmidt, M. Irazabal, and N. Nikaein. FlexRIC: an SDK for next-generation SD-RANs. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '21*, page 411–425, New York, NY, USA, 2021. Association for Computing Machinery.
 - [53] W. Sentosa, B. Chandrasekaran, P. B. Godfrey, and H. Hassanieh. Cell-Replay: Towards accurate record-and-replay for cellular networks. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 1169–1186, Philadelphia, PA, Apr. 2025. USENIX Association.
 - [54] C. Sun et al. SpotLight: Accurate, Explainable and Efficient Anomaly Detection for Open RAN. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, ACM MobiCom '24*, page 923–937, New York, NY, USA, 2024. Association for Computing Machinery.
 - [55] D. Villa, M. Tehrani-Moayyed, C. P. Robinson, L. Bonati, P. Johari, M. Polese, and T. Melodia. Colosseum as a Digital Twin: Bridging Real-World Experimentation and Wireless Network Emulation. *IEEE Transactions on Mobile Computing*, 23(10):9150–9166, 2024.
 - [56] E. Weingärtner, F. Schmidt, H. Vom Lehn, T. Heer, and K. Wehrle. SliceTime: A Platform for Scalable and Accurate Network Emulation. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

A Further Implementation Details

We developed a prototype implementation of CHRONOS on Ubuntu 20.04 as per the design described in §4, consisting of all the system components shown in Figure 3.

Platform: The Supervisor is implemented based on the KVM Hypervisor (in kernel version 5.15.160) and setup to use our clock source. The clock source is implemented as a kernel module to generate a new “fake” Time Stamp Counter (TSC) value every 50 microseconds using a high-resolution timer. This fake TSC counter is exposed as symbol and is used by KVM kernel module (and therefore by guest VM) instead of using host TSC. This module also provides an interface, via shared memory, to dynamically adjust the flow of time, enabling on-demand time dilation.

The Slot Monitor running in userspace can pause time through this shared memory interface. Upon receiving a slot start message from the Slot Coordinator, the Slot Monitor starts a timer for the slot duration. Once the timer expires, it uses the shared memory interface provided by the local clock source to pause the clock, keeping it paused until the next slot start is received from Slot Coordinator. In the actual implementation, we do not completely pause time upon timer expiry (in contrast to what is shown in Figure 4b), instead slowing it down by several orders of magnitude. This effectively pauses time while allowing for nanosecond-level operating system timers to continue, preventing freezes in the operating system and standard library.

Our largest deployment includes 130 cloud instances. In order to manage that scale, we run our platform on Kubernetes. To make sure the Kubernetes cluster controller is fast and reactive, we deployed it on its own instance without time dilation. Since most workers in the system run on time-dilated instances, we needed to carefully adjust the timers and timeout for different types of instances. Additionally, we modified the discovery and connection protocols used by the components in the CHRONOS architecture to enable all the components to be created and initialized (but not yet running) prior to dilation being introduced. This speeds up the deployment of emulation experiments.

Software stack: The Middlebox is built on the EMANE emulator atop the OpenAirInterface RAN [8]. We substantially rearchitected it for scale, including replacing the Cisco-derived nFAPI library’s `select()` based I/O with `epoll()`. The use of `epoll()` does not introduce tail-induced scaling issues. This is partly by design: the Middlebox is distributed and scales out, preventing one instance from becoming a bottleneck, but also through the implementation: each gNB is handled internally by a separate thread; `epoll()` only handles two sockets in this context. We demonstrate this in Figure 16. Two Middlebox’s, one handling 25 gNBs and another handling 250 gNBs (equivalent to 1000 gNBs and 10,000 gNBs when run in our 40 Middlebox distributed experiment), were instrumented and

the processing time attributable to `epoll()` plotted as CDFs. These are near-identical, demonstrating that `epoll()` processing does not increase with scale, and therefore is not a scaling concern.

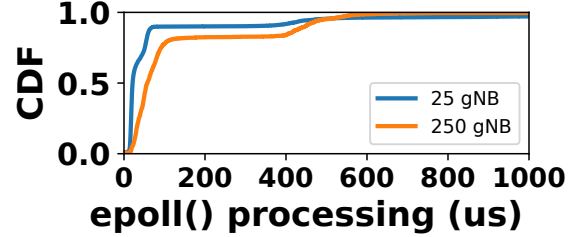


Figure 16: The scaling penalty (or lack of) from the use of `epoll()` in the Middlebox. The Middlebox was run twice, once with 25 gNBs and again with 250 gNBs, and the processing latency attributable to `epoll()` was plotted.

Beyond this, we also added lightweight UE-side features for seamless Kubernetes orchestration. On the RAN side, we adapt OpenAirInterface for both 4G and 5G, increasing scalability by modifying hard-coded limits (e.g., maximum supported UEs). We also improved reliability with several significant bug fixes, including a critical fix enabling multiple handovers. At larger scales, we also ran into scaling limitations of the Open5GS core we used [7]. We overcame these limitations by splitting the core into multiple instances. This ensured that no one core instance handled more than a few hundred gNBs while allowing us to scale. With all these modifications, we were able to run a deployment with 130 cloud instances and 6400 CPU cores, emulating 10,000 gNBs and UEs. This is 300× more than what has previously been emulated at the same fidelity level [49].

Portability: The CHRONOS platform (Supervisor and Slot Monitor) is inherently portable across a wide variety of cloud providers. A key requirement is the ability of the cloud provider to support *virtualization*: the ability to run our own VM hypervisor inside the cloud instance. All major clouds support this feature (AWS, GCP, Azure, IBM, CloudLab, OVH, etc.). We demonstrated the portability by deploying on multiple clouds, including CloudLab [26], Microsoft Azure, and Google GCP. Our implementation of Supervisor is specific to x86-64 processors. This can be easily ported to other architectures supported by KVM hypervisor and provides access to the CPU-specific timestamping register.

B Provisioning Flexibility

We demonstrate that time dilation enables more efficient use of compute resources (and therefore of experimental costs, since cost and compute are closely tied for cloud workloads). We do this by running an experiment with a single UE and gNB. Initially we run the UE with relatively little data plane

activity – only ICMP ping. However, at a predetermined time, we switch the UE into a mode with high data plane activity: UE begins a TCP session, downlinking as much traffic as possible. The results, shown in Figure 17, demonstrate that CHRONOS uses the dilation factor to dynamically adjust the amount of wall-clock time given to the experiment in order to ensure that the tasks are completed within the deadlines. During the course of this experiment, we capture the dilation factor, and plot it over time. When there is little network activity, the amount of processing required is minimal, and the dilation factor is relatively small. However, when there is significant network activity, the amount of processing required is much higher, and CHRONOS adjusts the dilation factor to ensure that no deadlines are exceeded.

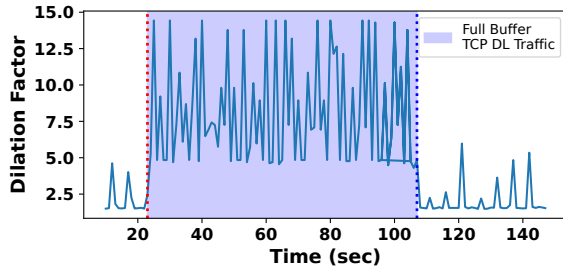


Figure 17: Dilation factor over time, for an experiment with significantly varying computational needs over time. The dilation factor can dynamically adapt to the current computational requirements of the experiment.

Here, the takeaway is that *a CHRONOS experiment can be provisioned for the typical-case behavior, rather than the worst-case*. If dynamic time dilation was not possible, the experimental setup would require sufficient compute resources to be provisioned to handle the worst-case (e.g., a sudden burst of traffic). But in many experimental scenarios, this worst-case only occurs momentarily, and therefore provisioning the entire experiment for this worst-case leads to inefficient use of resources. Therefore the dynamic nature of time dilation enables resources to be provisioned for the typical-case, leading to the most efficient use of compute resources.

C 4G LTE eNB / UE Scalability

We observe the same trend for CHRONOS’s 4G LTE (eNB) implementation [45] as for the 5G NR (gNB) implementation: increasing scale, with fixed compute, increases the dilation factor. As shown in Figure 18, scaling from 1 to 3 eNBs (100 UEs per eNB) increases the dilation factor from 1.38 to 3.86. At the same total UE count the two technologies show different absolute dilation factors e.g., 3.86 at 300 UEs in 4G versus ~10 at 300 UEs in 5G (Figure 8(c)) because the two sweeps use different UE-per-node ratios: 100 UEs/eNB in 4G versus 15 UEs/gNB in 5G, so reaching 300 UEs requires only 3 eNBs in 4G but 20 gNBs in 5G. The larger number of concurrently

emulated RAN nodes contributes to more processing cost, and thus a higher dilation factor at matched UE counts.

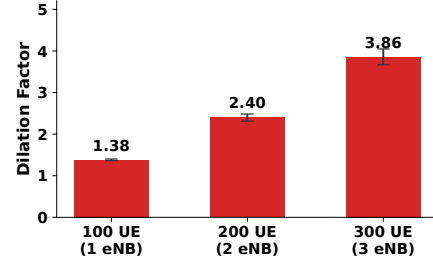


Figure 18: Dilation factor as the number of 4G LTE eNBs / UEs increases (100 UEs per eNB, fixed compute).

D Negligible Penalty for Scaling Out

As noted in §6.3.2, the Middlebox in CHRONOS is able to horizontally scale-out across multiple cloud instances, compared to alternative approaches that can only vertically scale-up on a single instance. This design is an enabler for scale because it matches the topology of cloud architectures. However, a consequence of scaling out is the communication overheads between multiple distributed instances. While the CHRONOS design will not break from these overheads due to its dynamic time dilation feature, it will increase the amount of time that experiments take. We now aim to quantify this overhead from scaling out, and in doing so, show that it is negligible.

We run a single Slot Monitor on each cloud instances, varying the number of instances and recording the dilation factor. The results, in Figure 19, reveal only a small linear increase in the dilation factor as the number of instances increases. This is logical, because as the number of instances in the experiment increases the chance of one particular instances experiencing larger compute preemptions or network latency also increases. All the Slot Monitor’s run in parallel, so it is the maximum compute preemption or network latency that determines how long the Supervisor and Slot Coordinator wait before advancing to the next slot.

Therefore it is logical that the time dilation is linearly increasing with the number of cloud instances. But crucially, the increase is very small to the point that it is negligible, especially when compared to scenarios involving scale and/or throughput (e.g., Figure 9a, Figure 8b)). This demonstrates that the cost to scaling out CHRONOS is negligible, and thus extremely large scales (requiring tens or hundreds of cloud instances) are achievable.

E Closed-loop xApp Message Sequence

This section summarizes the E2AP message sequence for the closed-loop xApp experiment in §7.2. The xApp first establishes its E2 session with FlexRIC, subscribes to KPM indications with a 1000 ms reporting period, observes the

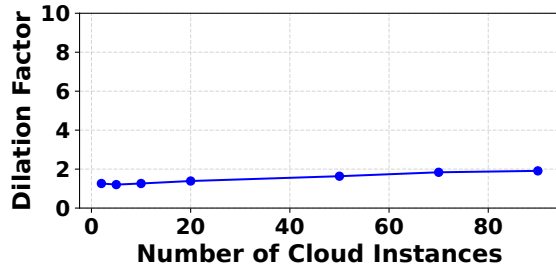


Figure 19: The overhead of scaling out Middlebox across multiple cloud instances, visualized as the increase in dilation factor as the number of cloud instances supporting the experiment increases.

UE during the baseline phase, and then sends two E2SM-RC control requests to retune the downlink OLLA target BLER. The native xApp log confirms that both control requests are

acknowledged by the E2 node within approximately 2 ms, after which KPM indications continue.

Table 1: E2AP timeline for the closed-loop xApp experiment. Times are relative to the xApp start.

Time	Message	Detail
+2.009 s	Subscription	KPM subscribed, 1000 ms period
+5.010 s	Baseline	UE observed; initial target held
+155.010 s	Control request	OLLA DL BLER target set to 0.200
+155.012 s	Control ACK	Request acknowledged by E2 node
+305.012 s	Control request	OLLA DL BLER target set to 0.050
+305.014 s	Control ACK	Request acknowledged by E2 node