Nuberu: Reliable RAN Virtualization in Shared Platforms

Gines Garcia-Aviles
i2CAT Foundation

Andres Garcia-Saavedra
NEC Laboratories Europe

Marco Gramaglia
Universidad Carlos III de Madrid

Xavier Costa-Perez
NEC Laboratories Europe,
i2CAT Foundation & ICREA

Pablo Serrano
Universidad Carlos III de Madrid

Albert Banchs
Universidad Carlos III de Madrid &
IMDEA Networks Institute

ABSTRACT

RAN virtualization will become a key technology for the last mile of next-generation mobile networks driven by initiatives such as the O-RAN alliance. However, due to the computing fluctuations inherent to wireless dynamics and resource contention in shared computing infrastructure, the price to migrate from *dedicated* to *shared* platforms may be too high. Indeed, we show in this paper that the baseline architecture of a base station's distributed unit (DU) collapses upon moments of deficit in computing capacity. Recent solutions to accelerate some signal processing tasks certainly help but do not tackle the core problem: a DU pipeline that requires predictable computing to provide carrier-grade reliability.

We present Nuberu, a novel pipeline architecture for 4G/5G DUs specifically engineered for non-deterministic computing platforms. Our design has one key objective to attain reliability: to guarantee a minimum set of signals that preserve synchronization between the DU and its users during computing capacity shortages and, provided this, maximize network throughput. To this end, we use techniques such as tight deadline control, jitter-absorbing buffers, predictive HARQ, and congestion control. Using an experimental prototype, we show that Nuberu attains >95% of the theoretical spectrum efficiency in hostile environments, where state-of-art approaches lose connectivity, and at least 80% resource savings.

CCS CONCEPTS

• **Networks → Mobile networks**; **Wireless access points, base stations and infrastructure**; **Network reliability**.

KEYWORDS

Mobile networks, RAN virtualization, vRAN, O-RAN, 3GPP, 5G, 6G, Distributed Unit, RAN disaggregation, NFV, Network virtualization

ACM Reference Format:
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 *The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '21), October 25–29, 2021, New Orleans, LA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3447993.3483266

1 INTRODUCTION

The virtualization of radio access networks (RANs), based hitherto on monolithic appliances over ASICs, will become the spearhead of next-generation mobile systems beyond 5G [18, 33]. Initiatives such as the carrier-led O-RAN alliance [12] have spurred the market and the research community to find novel solutions that import the flexibility and cost-efficiency of network function virtualization (NFV) into the very far edge of mobile networks [9, 33].

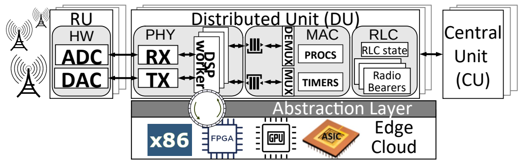

Figure 1: Virtualized RAN architecture [28]

Fig. 1 shows the architecture of a vRAN, with base stations (BSs) split into a central unit (CU), hosting the highest layers of the stack; a distributed unit (DU), hosting the physical layer (PHY); and a radio unit (RU), hosting basic radio functions such as amplification or sampling [1]. As depicted by the figure, vRANs shall rely on cloud platforms comprised of pools of *shared* computing resources (mostly CPUs, but also hardware accelerators brokered by an abstraction layer), to host virtualized functions such as the PHY [28].

However, while CUs are amenable to virtualization in regional clouds, virtualized DUs (vDUs)—namely, the vPHY therein—require fast and predictable computation in edge clouds [7, 28, 33]. Shared computing platforms provide a harsh environment for DUs because they trade off the predictability supplied by dedicated platforms for higher flexibility and cost-efficiency [21, 39]. Indeed, research has shown that resource contention in shared computing infrastructure, even when placing virtual functions on separate cores, may lead to up to 40% of performance degradation compared to dedicated platforms [24, 39]—the so-called *noisy neighbor* problem.

This is certainly an issue for traditional network functions such as virtual switches, firewalls, or even CUs, where metrics such as tail latency are particularly relevant. Consequently, substantial effort has been devoted to the design of generic scheduling frameworks that can balance computing efficiency and latency performance [26, 29]. However, as we show next, the requirements associated with full-fledged DUs are harder: violating deadlines cause users to lose synchronization with the DU, which leads to connectivity collapse.

1.1 The problem

To ease the explanation, we focus on frequency division duplex where uplink (UL) and downlink (DL) transmissions occur concurrently in different frequency bands, and on 5G's baseline numerology ($\mu = 0$ in 3GPP TS 38.211), which yields one transmission time interval (TTI) per subframe (SF), and a SF has a duration of 1 ms.

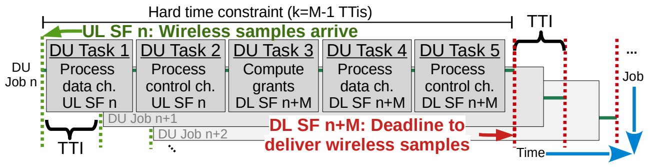
Hard time constraint (k=M-1 TTIs)
TTI
UL SF n: Wireless samples arrive
DU Job n
DU Task 1 Process data ch. UL SF n
DU Task 2 Process control ch. UL SF n
DU Task 3 Compute grants DL SF n+M
DU Task 4 Process data ch. DL SF n+M
DU Task 5 Process control ch. DL SF n+M
DU Job n+1
DU Job n+2
TTI
DL SF n+M: Deadline to deliver wireless samples
Time
Job

Figure 2: Every TTI (=1 ms), a worker must execute a DU job, comprised of a pipeline of interdependent DU tasks to process UL SF n and DL SF $n + M$, within $M - 1$ ms.

Fig. 2 illustrates the basic operation of the baseline 4G/5G DU processor [13, 15, 33]. Every TTI n, a **worker** initiates a **DU job** comprised of a *pipeline* of tasks (hereafter referred to as **DU tasks**): (1) process data and (2) control channels carried by UL SF n, (3) schedule UL/DL radio *grants* to be transported by DL SF $n + M$, and (4) process data and (5) control channels for DL SF $n + M$. A worker executes a DU job in a thread, using computing resources allocated by a task scheduler; and multiple workers perform DU jobs in parallel to handle one DL SF and one UL SF every TTI, as shown in Fig. 2. Importantly, given 3GPP specification (see details in §3), there is a hard constraint on M that *imposes a computing time budget of roughly $M - 1$ ms to process each DU job* (usually, $M = 4$).

Violating this constraint has critical consequences on DUs. To illustrate this, we set up an experiment with two DUs implemented with vanilla srsRAN [13] ($M = 4$), each one associated with one user implemented with srsUE and virtualized over Linux containers sharing 5 Intel Xeon x86 cores @ 1.9GHz. In this experiment, vDU 1 transmits and receives as much data as possible. Conversely, vDU 2 transmits and receives traffic following a random process with different parameters, which generate normally-distributed computing workload with the mean (line) and variance (shaded area) shown at the bottom of Fig. 3: the higher the load variance of vDU 2, the larger the fluctuations of the computing capacity available for vDU 1. Fig. 3 (top) depicts vDU 1's relative network throughput in yellow ("Baseline") as a function of the workload produced by vDU 2. The figure shows that the performance of vDU 1 quickly deteriorates. The reason is that, because both vDUs share the same CPU pool, vDU 1 occasionally suffers from CPU resource deficit when vDU 2 produces a peak in demand. As a result, vDU 1 workers executing DU jobs violate their deadline to send out the corresponding DL SF, as illustrated in Fig. 5, which causes the user to lose synchronization and throughput to drop.

Indeed, completing a DU job *every* TTI is vital to preserve synchronization between the BS and its users and thus attain reliability. However, this is challenged by some compute-intensive operations within DU tasks such as forward error correction (FEC). These

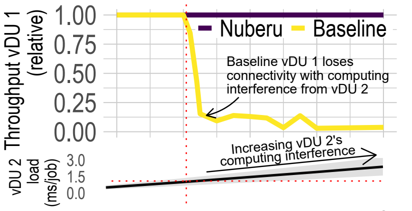

Figure 3: Two vDUs competing for computing resources. The UL/DL data load of vDU 1 is the highest possible while vDU 2's is variable.

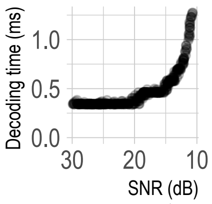

Figure 4: Decoding time of one transport block in a dedicated CPU core.

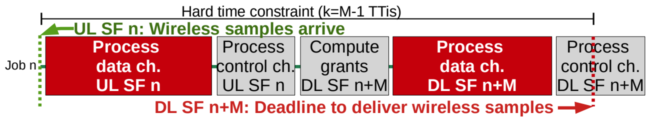
Hard time constraint (k=M-1 TTIs)
UL SF n: Wireless samples arrive
Job n
Process data ch. UL SF n
Process control ch. UL SF n
Compute grants DL SF n+M
Process data ch. DL SF n+M
Process control ch. DL SF n+M
DL SF n+M: Deadline to deliver wireless samples

Figure 5: Computing fluctuations and head-of-line blocking of DU tasks may cause that a worker violates its job's deadline, incurring loss of connectivity as shown in Fig. 3.

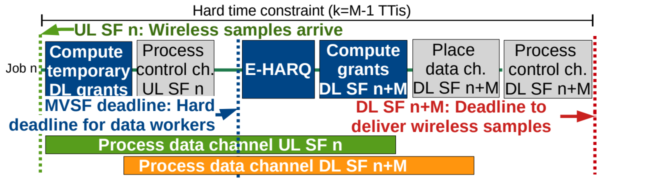
Hard time constraint (k=M-1 TTIs)
UL SF n: Wireless samples arrive
Job n
Compute temporary DL grants
Process control ch. UL SF n
E-HARQ
Compute grants DL SF n+M
Place data ch. DL SF n+M
Process control ch. DL SF n+M
MVSF deadline: Hard deadline for data workers
DL SF n+M: Deadline to deliver wireless samples
Process data channel UL SF n
Process data channel DL SF n+M

Figure 6: To provide reliability, Nuberu decouples UL/DL data tasks from the rest of the pipeline by integrating a temporary grant scheduler, E-HARQ, and congestion control.

operations require substantial processing time even when using solutions such as Intel FlexRAN [19] or Agora [9], which exploit data parallelization, efficient work scheduling, or SIMD programming. More specifically, FlexRAN provides open libraries to perform FEC, rate matching, and cyclic redundancy checks. In turn, Agora builds on FlexRAN to process UL and DL data in data channels (part of DU tasks 1 and 4) with a solution that is optimized for multi-core CPU platforms. To illustrate this, Fig. 4 shows the time required by FlexRAN to successfully decode 4-Kbyte transport blocks modulated with 64QAM, encoded with LDPC and 1/3 code rate, and for different signal-to-noise-ratio (SNR) regimes between 10 dB and 30 dB, in a dedicated CPU core. Note that, albeit these approaches provide high-performing solutions, they require dedicated computing platforms with *deterministic* performance to perform reliably.

Generic computing (or task) schedulers allocate computing resources to DU workers depending on the platform's capacity and the scheduler's policy [26, 29]. However, different DU tasks (pipeline stages) in the pipeline shown in Fig. 2 (in one DU job) cannot run in parallel in different computing cores to expedite the latency of a job because of the inter-dependencies between DU tasks, which cause head-of-line blocking. For instance, UL channels (DU task 1) have to be processed before scheduling UL/DL data (DU task 3), or data grants (DU task 3) must be computed before processing DL data (DU tasks 4) and control channels (DU task 5). More details in §3. The obvious solutions applied today in the market [7, 18, 33], namely, *dedicated* hardware acceleration and over-dimensioning, diminish the very reasons that make virtualization appealing for the RAN in the first place: flexibility and cost-efficiency. On the one hand, research has shown that shared platforms require 5x more resources than dedicated platforms to attain similar performance guarantees in real mobile networks [25]. On the other hand, dedicated accelerators make vDUs more expensive and power-hungry than their pure HW counterparts [30]—let alone the fact that the much-longed hardware/software decoupling is not achieved.

In summary, optimized schedulers and baseband libraries are obviously important but these solutions cannot provide hard latency guarantees for every DU job and, as a consequence, *do not provide carrier-grade reliability in non-deterministic platforms.* Hence, we must find new solutions to enable reliable vRANs without compromising the advantages of virtualization.

1.2 The solution

We propose Nuberu, a novel pipeline architecture for 4G/5G DUs that is suitable for non-deterministic computing platforms. Our design follows one objective: *to guarantee a minimum viable subframe (MVSF) for every TTI during moments of shortage in computing capacity to provide reliability and, provided this, maximize network throughput.* During such shortages, an MVSF encodes those signals and control information required to preserve user synchronization by temporarily holding off data delivery and relying on predictions.

To this end, we set up a deadline within every DU job to begin building an MVSF even if data processing tasks are unfinished. This deadline, depicted in blue in Fig. 6, is set such that there is enough time to process an MVSF before the final job completion deadline (in red in the figure). This is viable because, different from data processing tasks, the tasks involved in building an MVSF require little and roughly deterministic time as we will show in §3. To do this efficiently, we need to decouple data processing tasks such that the information required to build an MVSF is ready on time and network throughput is maximized during computing capacity fluctuations. Consequently, we apply the following techniques.

(1) To process DL data channel tasks:
- We adopt a two-stage DL radio scheduling approach:
  - We issue *temporary* DL grants as early as possible in the DU pipeline, as shown in Fig. 6. Dedicated workers process (encode, modulate, etc.) these grants in separated threads and store the resulting data in a buffer.
  - Upon the MVSF deadline, *final* DL data grants are computed based on those already processed successfully and are available in the buffer. Grants generated in a job n that are not processed on time are hence *delayed* for a later job.
- To mitigate the number of delayed DL data grants, the amount of DL data granted by the temporary scheduler is regulated by a *congestion controller* that adapts the flow of DL data grants to the availability of computing resources.

(2) To process UL data channel tasks:
- Dedicated workers process (demodulate, decode, etc.) UL data carried by each UL SF in separated threads.
- Upon the MVSF deadline, an *early HARQ* (E-HARQ) mechanism infers the *decodability* of UL data based on feedback from the workers, as shown in Fig. 6. This enables us to estimate the radio information that is required to build an MVSF even if UL data processing tasks have not finished on time.
- To maximize the predictive performance of E-HARQ, which depends on the amount of work done before the MVSF deadline, another *congestion controller* adapts the allocation of UL radio resources to the available computing capacity.

As shown by the purple line in Fig. 3, Nuberu can sustain maximum throughput despite severe fluctuations in computing capacity. The details of our design and our experimental evaluation are presented in §4 and §5, respectively. We close our paper with a literature review in §6 and our conclusions in §7.

2 BACKGROUND

In this section, we first introduce the fundamentals of 4G LTE and 5G New Radio (NR) that are relevant to understand this work (§2.1). Then, we present the details of the baseline PHY pipeline architecture introduced earlier (§2.2).

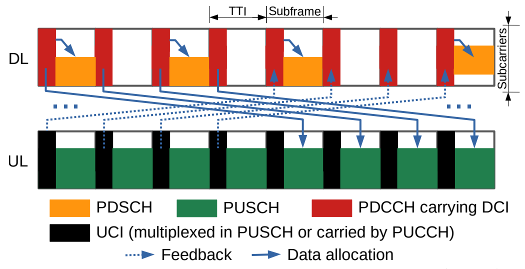

Figure 7: Subframes and PHY radio channels (FDD), $M = 4$.

2.1 A primer in LTE & NR PHY

NR adopts orthogonal frequency division multiplexing access (OFDMA) with cyclic prefix (CP) for both DL and UL transmissions, which enables fine-grained scheduling over a time-spectrum *grid*, and multiple-input multiple-output (MIMO) techniques. While LTE also adopts OFDM in the DL, it relies on single-carrier FDMA (SC-FDMA) for the UL, a linearly precoded flavor of OFDMA that reduces peak-to-average power ratio in mobile terminals.

An example of the radio operation for FDD is depicted in Fig. 7. In the time domain, each frame is comprised of 10 subframes (SF), each with 1-ms duration. Each SF comprises 2 slots, each with 7 (with normal CP) or 6 (with extended CP) OFDM symbols for LTE; and one or more slots, each with 14 (with normal CP) or 12 (with extended CP) OFDM symbols for NR, depending of the (flexible) OFDM numerology employed, which is configurable in 5G. Accordingly, a transmission time interval (TTI), which specifies the time resolution for scheduling, is equal to 1 ms in LTE and configurable to one or multiple slots in NR. Without loss in generality, we will assume a TTI is equal to 1 ms (baseline numerology in NR) to simplify our explanations. In the spectrum domain, SFs comprise a number of subcarriers with inter-subcarrier spacing equal to 15 kHz in LTE and variable, between 15 KHz and 240 kHz, for NR. LTE and NR support different bandwidth configurations up to 20 MHz and 100 MHz, respectively. The time-spectrum grid is divided into *channels* that are summarized in Table 1.

Downlink Channels and Signals	
PDSCH	Physical DL Shared Channel: *Carries user data, higher-layer user information and paging, as indicated in PDCCH.*
PDCCH	Physical DL Control Channel: *Carries resource assignments and UL scheduling grants.*
PBCH	Physical Broadcast Channel: *Carries basic information about the BS, e.g., bandwidth.*
PHICH	(LTE only) Physical HARQ Indicator Channel: *Carries UL Hybrid-ARQ feedback.*
PCFICH	(LTE only) Physical Control Format Indicator Channel: *Indicates the format used for PDCCH and PHICH.*
PSS/SSS	Primary/Secondary Synchronization Signals: *Signals used for synchronization and BS identity.*
Uplink Channels and Signals	
PUSCH	Physical UL Shared Channel: *Carries user data as indicated in PDCCH and, optionally, UL Control Information (UCI).*
PUCCH	Physical UL Control Channel: *Carries UCI including feedback (HARQ, channel quality, etc.) and scheduling requests.*

Table 1: LTE & NR Channels

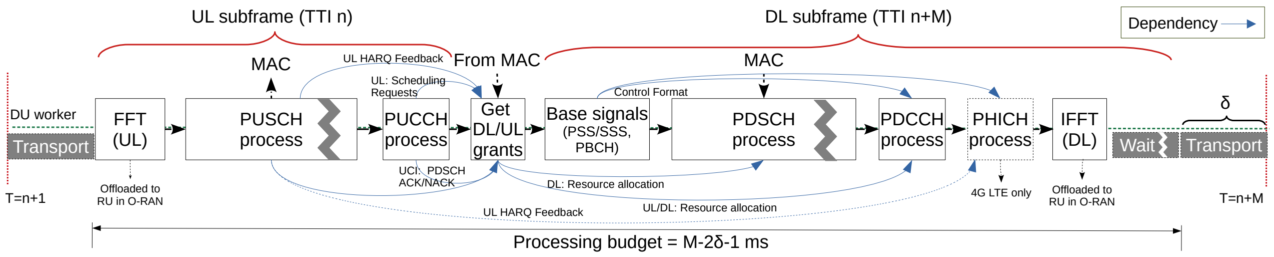

Figure 8: LTE and NR DU pipeline: DU job n

A physical resource block (PRB), comprised of 12 subcarriers and 1 slot, is the smallest radio resource unit that can be allocated; and a transport block (TB) carries data using a variable number of PRBs. The size of a TB depends on the state of higher-layer (MAC, RLC) data buffers and the selected modulation and coding scheme (MCS), which in turn depends on the signal-to-noise ratio (SNR). Every TTI, PDSCH (for DL) and/or PUSCH (for UL) carry one TB (or two, in some MIMO settings) per user. The allocation of radio resources is indicated by DL and UL *grants*, which are encoded into PDCCH's Downlink Control Information (DCI) as shown in Fig. 7.

Hybrid automatic repeat request (HARQ), combining forward error correction (FEC) and ARQ, is used for error control. To this end, explicit feedback is received from the users in UL Control Information (UCI) carried by PUSCH or PUCCH, as shown by the dotted arrows in the figure, and TBs are encoded with low-density parity-check codes (NR) or turbo codes (LTE).

More details can be found in [8, 23] and references therein.

2.2 Baseline DU pipeline

The above operations are inter-dependant, e.g., PDCCH in DL SF n carries the grants needed to process PDSCH in DL SF n and PUSCH in UL SF $n+M$, UCI in UL SF n carries feedback required to compute DL grants in DL SF $n+M$, HARQ feedback from PUSCH processing in UL SF n is required to compute grants in DL SF $n+M$, etc. All of them are required by a DU. To do this, every TTI n an idle worker performs a DU job n comprising the pipeline of Fig. 8 [13, 15, 33]:

(1) **Process uplink subframe n (received during TTI n):**
  1.1. First, wireless samples corresponding to the n^{th} UL SF are transformed into OFDM symbols by performing Fast Fourier Transformation (FFT) and CP removal.[1]
  1.2. Then, the PUSCH and PUCCH are demodulated, decoded and processed, providing UL TBs and UL feedback (DL HARQ, channel quality, scheduling requests).

(2) **Compute UL/DL grants carried by DL SF $n + M$:** Schedule DL and UL radio resources considering UL feedback.

(3) **Process downlink subframe $n + M$:**
  3.1. First, base signals are processed, including PSS/SSS, PBCH, and, in case of LTE, PCFICH.
  3.2. Then, the PDSCH and the PDCCH are processed, encoded, and modulated, according to the UL/DL grants.
  3.3. Finally, the encoded OFDM symbols corresponding the DL SF $n + M$ are converted into wireless samples by adding CP and performing inverse FFT (IFFT). The corresponding wireless samples are then sent to the radio transmission chain, with sub-ms transportation delay δ, at time $n + M$.[1]

Existing DU solutions implement the above baseline pipeline using highly-optimized libraries such as Agora or FlexRAN to expedite some operations involved therein via parallelization, SIMD programming, or dedicated hardware accelerators.

3 DIAGNOSIS

The design presented in §2.2 is not suited for non-deterministic computing platforms such as shared clouds. Namely, existing solutions implementing the above baseline pipeline cannot guarantee the timely execution of individual jobs without the assistance of *dedicated* hardware acceleration or aggressive over-dimensioning [33], which compromise flexibility and cost-efficiency [30].

Timing constraints. 3GPP defines several timing constraints [2]. Relevant to our work are K_3 (latency between ACK/NACK reception in UL UCI and the corresponding DL re-transmission in PDSCH), and K_4 (latency between PUSCH reception and delivery of HARQ feedback). In LTE, $K_4 = 3$ ms, which implies $M = 4$. Though these timings are more flexible in NR, they are set at longer timescales by the CU, and $M = 4$ is the usual choice [20]. As a consequence, there is a hard deadline to process each DU job within $M - 2\delta - 1$ ms, as shown at the bottom of Fig. 8. Violating this deadline prevents timely delivery of DL SFs and, as a result, loss of synchronization and connectivity between the DU and its users, as shown by the baseline performance in the experiment of Fig. 3.

Inter-task dependencies. Regardless of individual processing optimizations [9], different DU tasks within a job have strong dependencies as shown by the blue arrows in Fig. 8:

- DL grants must be computed before PDSCH because they carry information required to encode and modulate DL TBs;
- PUSCH and PUCCH must be processed before computing DL grants because these channels carry users' feedback (DL HARQ, channel quality, etc.) in UCI messages (dotted arrows in Fig. 7), which is required to schedule DL data appropriately;
- UL HARQ feedback can only be computed after processing PUSCH; and this feedback is required to schedule UL grants within the same job to satisfy 3GPP timing constraints;
- UL grants must be computed before processing PDCCH, which carries those grants, and (in case of LTE) before processing PHICH, which carries UL HARQ feedback.
- All the channels and other basic signals (PSS, SSS) must be processed before generating time-domain signals (IFFT).

As a result, known implementations (e.g., Samsung's vDU [33], `srsRAN`[2] and `OpenAirInterface`[3]) perform each DU job in a single-thread pipeline (Fig. 8), or in a multi-thread pipeline where each thread has to wait and be executed in a precise order [15], which

[1] This task may be offloaded to the RU in O-RAN.

[2] http://www.srsran.com/

[3] https://www.openairinterface.org/

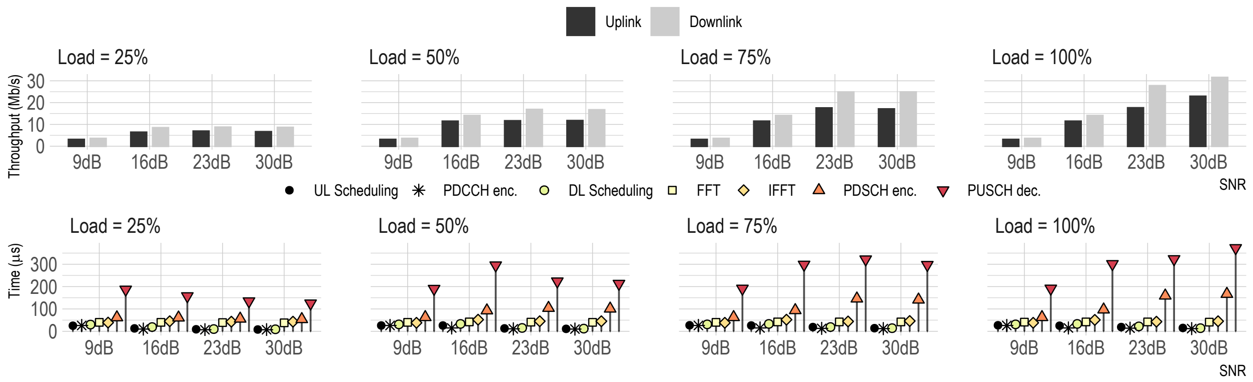

Figure 9: Throughput performance for both uplink and downlink (top). CPU time required by different PHY layer functions (bottom). Different uplink/downlink load (relative to the maximum) and channel conditions (SNR).

boils down to Fig. 8 again. Although solutions like Agora and FlexRAN help to accelerate the processing of individual DU tasks, the aforementioned dependencies prevent running different DU tasks in a job in parallel to expedite the pipeline of Fig. 8.

Non-deterministic tasks. As hinted in our toy experiment shown in Fig. 3, the computing time required by DU tasks highly depends on the instantaneous availability of computing resources. We note moreover that the most compute-intensive tasks also depend on the context, that is, on the data load (rate of TBs to decode/encode) and on the mobility patterns of the users (signal quality) [5], which can induce very quick fluctuations in the demand for computing resources. To illustrate this, we deploy the baseline vDU, implemented in `srsRAN` [13], processing downlink and uplink traffic over one Intel i7 core in a 10-MHz band. Fig. 9 depicts the achieved throughput in both the uplink and downlink (top subplots), and the median time incurred by the CPU to perform DU tasks (bottom subplots). We take these measurements for different load intensities (relative to the capacity in UL and DL, respectively) and average signal-to-noise ratios (SNR) indicating the channel quality for both UL and DL, and adapt the MCS to minimize the workload issued by the decoder (differently to our results in Fig. 4). The results yield two observations. First, processing PDSCH and (especially) PUSCH are the two tasks that consume CPU time the most, which is not surprising as it has been observed before [13]. Second, while the CPU time of the rest of tasks (and others not shown in the figure to reduce clutter) remains practically constant,[4] the time required to process PDSCH and PUSCH highly depends on the context; that is, on the SNR—and so on the mobility patterns of the users, and on the load—and hence on the users behavior. Note that even if shared pools of hardware accelerators are used *à la cloud* to reduce the processing time of some of these tasks, queueing in the abstraction layer brokering access to the accelerators across multiple vDUs incur in similar issues [28].

Conclusion: Because of the above, baseline solutions cannot guarantee the timely completion of DU jobs when facing computing fluctuations, which cause unreliability in scenarios such as that of Fig. 3. We hence claim that a re-design of the DU pipeline is required for non-deterministic computing platforms such as shared clouds.

[4]To be precise, the processing time of these tasks *does* vary with the number of users (e.g., scheduling becomes more complex). However, our results (omitted due to space constraints) indicate this is negligible compared to that of data processing tasks.

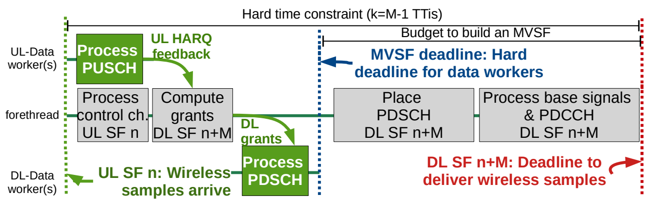
Hard time constraint (k=M-1 TTis)
Budget to build an MVSF
UL-Data worker(s)
Process PUSCH
UL HARQ feedback
MVSF deadline: Hard deadline for data workers
forethread
Process control ch. UL SF n
Compute grants DL SF n+M
DL grants
Place PDSCH DL SF n+M
Process base signals & PDCCH DL SF n+M
DL-Data worker(s)
UL SF n: Wireless samples arrive
Process PDSCH
DL SF n+M: Deadline to deliver wireless samples

Figure 10: MVSF deadline for data processing tasks in job n.

4 NUBERU

To overcome the issues introduced in §3, we present a novel DU architecture, which we name Nuberu. Our solution is specifically engineered for 4G LTE and 5G NR workloads that are virtualized over clouds of shared resources with non-deterministic capacity.

Our design has one goal: to guarantee every TTI a minimum viable subframe (MVSF) that provides the smallest set of signals required to preserve user synchronization while maximizing throughput during shortages of computing resources. To this end, we decouple each job's data tasks into separate threads as shown in Fig. 10:

(1) *DU forethread:* In charge of (*i*) building the MVSF; and (*ii*) coordinating the remaining DU data workers (middle of Fig. 10),
(2) *DL-Data DU workers*: One or more threads in charge of the bulk of PDSCH processing tasks (bottom of Fig. 10),
(3) *UL-Data DU workers*: One or more threads in charge of the bulk of PUSCH processing tasks (top of Fig. 10);

and set up a hard deadline Φ_n for every job n (blue line in Fig. 10) upon which an MVSF is compiled even if data workers have not finished. Note that, as shown before, basic MVSF tasks in charge of the forethread (such as computing grants or building PDCCH) require little or deterministic processing time, which can be estimated *a priori* to calculate $\Phi_n = n+M-\tau$, where τ is the time required to compile an MVSF plus the transportation delay (δ ms). The challenge is to decouple and adapt data processing tasks in a way such that the performance loss caused by computing fluctuations is minimized.

In the following, we first present the overall Nuberu design (§4.1) and then we detail our solutions to address this challenge for DL data processing tasks (§4.2) and UL data processing tasks (§4.3).

4.1 Overall system design

Fig. 11 shows the detailed design of Nuberu. The forethread is responsible for building an MVSF and exploiting the work of data

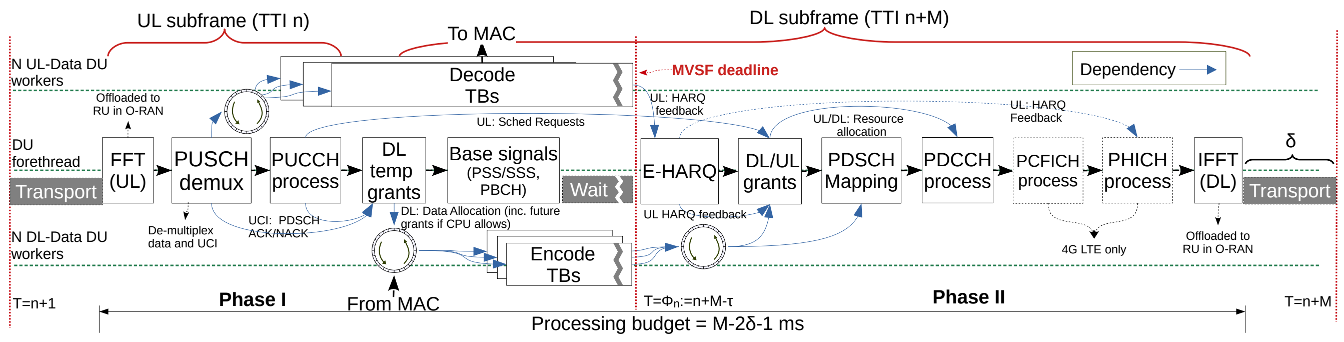

Figure 11: Nuberu—A reliable 4G/5G DU pipeline

workers to maximize performance during shortages of computing resources. Note that, as a consequence of the MVSF deadline, Nuberu divides the time budget of each job into two phases:

Phase I: The forethread carries out the following sequence of basic tasks after FFT[1] to process the received UL SF n and other UL-independent tasks to begin the process of building DL SF $n + M$:

(1) (modified task) If UL SF n carries PUSCH, demultiplex UL TBs from UCI and hand over the coded TBs to UL-Data workers.

(2) Process PUCCH/UCI if UL SF n carries it, and encode base signals (PSS/SSS) and the PBCH for DL SF $n + M$.

(3) (new task) Compute *temporary* DL grants depending on the availability of data, radio resources, *and computing capacity*. This is the first stage of a novel two-stage radio scheduling approach, which uses a congestion controller to issue temporary grants that may be processed before the MVSF deadline by a DL-Data worker. During quick computing fluctuations, some grants may not be processed on time and, to avoid dropping them and waste resources, they are stored in a buffer for handling by the second scheduling stage of a posterior job (see §4.2).

(4) (new task) *Snooze* up until time Φ_n (the MVSF deadline). *This step is key to maximize the time budget used by data workers, yet give the forethread leeway to generate an MVSF that preserves connectivity if data workers violate their deadline.*

Phase II: The forethread awakes at Φ_n and invests the remaining $\tau - \delta$ ms in the following tasks to build DL SF $n + M$ before IFFT[1]:

(1) (new task) *Early-HARQ* (E-HARQ) collects data from UL-Data workers that did not finish on time, and makes a *prediction* about the decodability of the corresponding TBs. This allows us to use *promising* UL TBs that otherwise would have to be dropped because they were not decoded on time. See §4.3.1.

(2) (modified task) Second stage of our radio scheduling approach: given the finished UL-Data workers, the E-HARQ outcome, and the grants actually encoded on time, the final UL/DL grants are computed by the MAC scheduler. Similar to its DL counterpart, UL grants are ruled by a congestion controller, detailed in §4.3.2. In the worst case, when no data worker finished on time (e.g. due to a sudden shortage of computing resources), no DL grants are allocated and the SF becomes an MVSF with the only responsibility of preserving user synchronization.

(3) If DL grants are provided, the encoded TBs are modulated and mapped into the radio resource grid, and all the remaining control channels are processed.

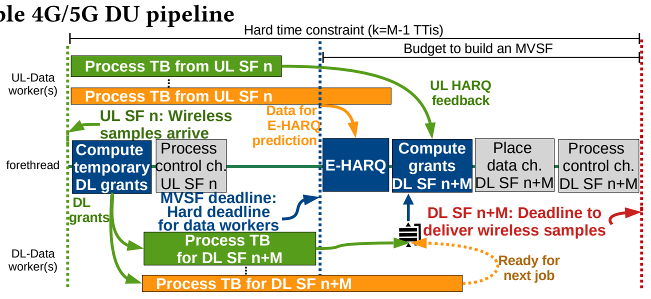
Hard time constraint (k=M-1 TTIs)
Budget to build an MVSF
UL-Data worker(s)
Process TB from UL SF n
Process TB from UL SF n
UL HARQ feedback
UL SF n: Wireless samples arrive
Data for E-HARQ prediction
forethread
Compute temporary DL grants
Process control ch. UL SF n
E-HARQ
Compute grants DL SF n+M
Place data ch. DL SF n+M
Process control ch. DL SF n+M
DL grants
MVSF deadline: Hard deadline for data workers
DL SF n+M: Deadline to deliver wireless samples
DL-Data worker(s)
Process TB for DL SF n+M
Ready for next job
Process TB for DL SF n+M

Figure 12: Decoupling DL data tasks in a job n

4.2 DL data tasks

The first challenge is to schedule only the amount radio resources that can be processed on time, and to buffer those that cannot be timely processed for a later job. We address this with a two-stage radio scheduling approach and a congestion controller.

4.2.1 Two-stage DL radio scheduling. As shown in Fig. 12, we adopt a two-stage approach to compute DL grants. In the *first stage*, Nuberu issues *temporary* DL grants as early as possible during each DU job's Phase I. Dedicated DL-Data workers then process (encode, modulate, etc.) the corresponding DL TBs in separated threads, and, when the task is completed, the worker stores the respective encoded TB in a buffer, as depicted by Fig. 12. In a *second stage*, during Phase II (after the MVSF deadline Φ_n), the forethread computes the *final* DL grants based on those that have already been encoded and, hence, are available in the buffer. In this way, we can begin processing DL TBs as early as possible to maximize the time budget of DL-Data workers without compromising the budget of UL-Data workers, which need to finish before allocating DL resources.

It is expected that the temporary scheduler issues DL grants such that the amount of work they require by the DL-Data worker can be carried out before Φ_n, as illustrated in Fig. 13's case (1). We achieve this by employing a rate controller that is introduced later in §4.2.2. However, during quick computing fluctuations, the DL grants computed by job n's temporary scheduler may be ready only at job $n + K$, for some integer $K > 0$. Therefore, K represents the amount of time that DL data is buffered in the vPHY, which allows us to absorb computing fluctuations appropriately and hence to preserve resiliency during events of computing volatility. This is depicted by cases (2) and (3) in Fig. 13, which are DL TBs that could not be encoded on time and hence can only be placed into a DL SF in a later job. Not only do we attain resiliency with this approach, but we also increase efficiency compared to the baseline, which must drop grants that do not meet their deadlines, wasting resources.

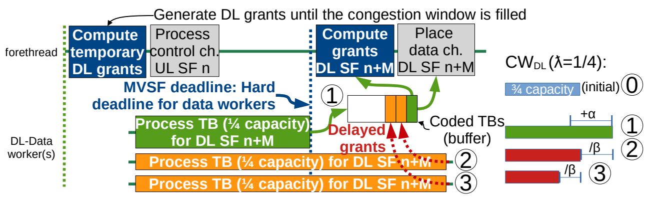
Generate DL grants until the congestion window is filled
forethread
Compute temporary DL grants
Process control ch. UL SF n
Compute grants DL SF n+M
Place data ch. DL SF n+M
MVSF deadline: Hard deadline for data workers
DL-Data worker(s)
Process TB (¼ capacity) for DL SF n+M
Delayed grants
Process TB (¼ capacity) for DL SF n+M
Process TB (¼ capacity) for DL SF n+M
Coded TBs (buffer)
CW_DL (λ=1/4):
¾ capacity
(initial) 0
+α 1
/β 2
/β 3

Figure 13: DL temporary grants and DL congestion control with $\lambda = 1/4$ during job n. Delayed grants (in orange) can only be placed in a later job $n + k$, for some $k > 0$.

4.2.2 DL Congestion Control. Guaranteeing the timely completion of vDU jobs in non-deterministic platforms requires adapting the demand for (shared) computing resources to the instantaneous platform's capacity. This is a fundamentally different paradigm to that of RANs using dedicated hardware, which allocate radio resources based only on wireless conditions and network demand.

The temporary DL grants issued during Phase I generate workload for DL-Data workers. The processing time of the corresponding TBs depends on the amount of data transported therein as well as on the allocated computing resources by some task scheduler. Though allocating computing resources appropriately obviously help, it is not enough to provide the hard guarantees we require. In the following, we propose a mechanism that adapts such computing workload based on observations from the workers' behavior.

Adapting a flow of requests (encoding) to a server capacity (computing platform) falls into the realm of *congestion control*. Hence, we resort to mechanisms amply used in networking protocols such as TCP. To this end, Nuberu's radio schedulers use a *DL congestion window* ($cwnd_{\text{DL}}$) that regulate the flow of DL grants. We adopt an additive-increase / multiplicative-decrease (AIMD) algorithm where $cwnd_{\text{DL}}$ increases by α PRBs every DU job n ($cwnd_{\text{DL}}^{(n+1)} = cwnd_{\text{DL}}^{(n)} + \alpha$) as long as congestion is not detected or the maximum PRB capacity is reached, and multiplicatively decreases by $\beta \leq 1$ ($cwnd_{\text{DL}}^{(n+1)} = cwnd_{\text{DL}}^{(n)} \cdot \beta$) if congestion is detected. Nuberu infers congestion if the buffer of encoded TBs contains $\lambda > 0$ times the vDU's PRB capacity or more.

Fig. 13 shows an example where $cwnd_{\text{DL}}^{(n)} = 3/4$ of the PRB capacity and $\lambda = 1/4$ at job n. Accordingly, in this example the temporary scheduler issues three TBs, each with 1/4 of the PRB capacity to fill $cwnd_{\text{DL}}^{(n)}$. The first TB, case (1), is encoded before the MVSF deadline, added to the buffer, and then immediately placed in DL SF $n+M$, i.e., *it does not need to be stored in the buffer.* Conversely, in cases (2) and (3), the respective workers did not finish their tasks on time. Consequently, the encoded TBs (once the respective workers finish) have to be stored for handling by a later job $n+K$ for some $K > 0$ (and hence will be delivered in DL SF $n+M+K$). Because in this example $\lambda = 1/4$ and every stored TB carries 1/4 of the vDU's PRB capacity, each item triggers a multiplicative reduction in $cwnd_{\text{DL}}$, as shown by the colored bars at the right.

4.3 UL data tasks

As soon as a UL SF arrives, the forethread hands over the (coded) UL TBs carried by PUSCH to idle UL-Data workers to decode them. The challenge, in this case, is to schedule UL radio resources that can be processed by the vDU on time (like the DL case), and compute UL feedback by the MVSF deadline for those that cannot. We address this with another congestion controller and with predictive HARQ.

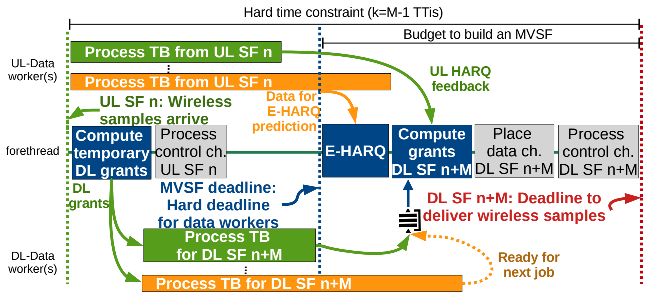
Hard time constraint (k=M-1 TTIs)
Budget to build an MVSF
UL-Data worker(s)
Process TB from UL SF n
Process TB from UL SF n
UL SF n: Wireless samples arrive
Data for E-HARQ prediction
UL HARQ feedback
forethread
Compute temporary DL grants
Process control ch. UL SF n
E-HARQ
Compute grants DL SF n+M
Place data ch. DL SF n+M
Process control ch. DL SF n+M
DL grants
MVSF deadline: Hard deadline for data workers
DL SF n+M: Deadline to deliver wireless samples
DL-Data worker(s)
Process TB for DL SF n+M
Ready for next job
Process TB for DL SF n+M

Figure 14: Decoupling UL data tasks in a job n

4.3.1 Early HARQ. Both types of decoders managed by UL-Data workers in 4G and 5G employ an iterative algorithm to decode data, and a *stopping criteria* based on CRC checks to decide on the decodability of the TB [16, 32]. Then, based on such a decision, feedback is provided to the DU forethread so grants for UL retransmissions (or just new UL data) are allocated.

Like its DL counterpart, it is expected that UL TBs are often decoded before the MVSF deadline if computing resources have been provisioned appropriately and our UL congestion controller (see §4.3.2) adapts fast enough. However, due to the volatility inherent to shared computing environments, UL-Data workers may not finish their task by time Φ_n to provide feedback to build DL SF $n + M$. Simply discarding such TBs is a waste of radio resources. Moreover, the computing capacity has no impact on the decodability of TBs but only on the amount of time required to process them.

To avoid discarding UL TBs that cannot be processed in time, the forethread first performs Early HARQ (E-HARQ) after the MVSF deadline (Φ_n), as shown in Fig. 14, to *predict* the decodability of those TBs based on feedback from the workers. This prediction provides the required information to initiate Phase II, and avoids wasting expensive radio resources when dropping *late* UL TBs, which now do not have a hard deadline to be processed (demodulated, decoded, etc.). E-HARQ has received attention for low-latency communications. The approach is usually to design a stopping criterion for the iterative algorithms used by turbodecoders [32] and LDPC decoders [22], or to predict the decodability of the data to send HARQ feedback early [35]. As mentioned before, Nuberu leverages this technique to provide extra time budget for UL workers.

Let $S_{w,t_n} \in \mathcal{S}$ denote the *state* of an UL-Data worker w at time t in DU job n, where $\mathcal{S}$ is the state space of the decoder. At time Φ_n, the forethread observes the state of each *unfinished* worker, i.e., S_{w,Φ_n} for all $w \in \mathcal{W}_u$ where $\mathcal{W}_u$ is the set of active UL-Data workers, and apply a rule $\Pi(S_{w,\Phi_n}) \in \{\texttt{UNDECODABLE}, \texttt{DECODABLE}, \texttt{UNKNOWN}\}$ to decide upon the decodability, undecodability, or uncertain decodability of the TB. Once E-HARQ infers the decodability of the TB, it signals the UL MAC scheduler so it delivers the appropriate UL HARQ feedback to the users and/or schedules re-transmissions, accordingly, as *if the workers had finished their task.* UNKNOWN TBs are treated by the UL MAC scheduler as UNDECODABLE TBs; however, this information is useful for congestion control, as we will show in §4.3.2. If $\Pi(S_{w,\Phi_n}) = \text{DECODABLE}$, UL-Data worker w is allowed to continue up till a maximum number of decoding iterations wherein, as per 3GPP specification, CRC validation is used as a stopping criterion. Otherwise, the TB is discarded and the UL-Data worker w becomes idle again and ready to receive new work.

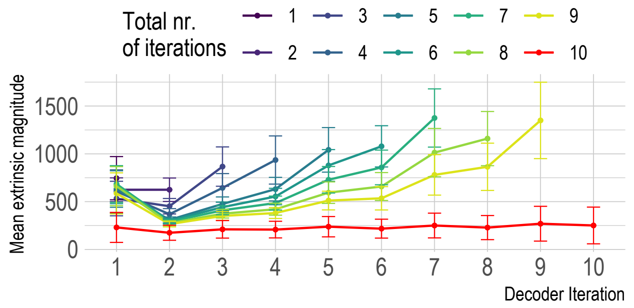

Figure 15: Mean extrinsic magnitude for each iteration of a turbodecoder. Dot/line indicate the average value across multiple PUSCH TBs with different MCS, TBS and SNR. Error bars indicate the standard deviation.

Our approach to design the rule Π and $\mathcal{S}$ draws from different ideas in prior work on turbo and LDPC codes. The key idea behind rests upon the concept of *extrinsic information*, which spawns organically by *belief propagation* algorithms used by both turbo and LDPC codes. We refer the reader to [38] for detailed information about these coding techniques. In a nutshell, belief information is encoded into log-likelihood ratios (LLRs), $L_b := \ln \frac{\text{Prob}(b=+1|\text{input})}{\text{Prob}(b=-1|\text{input})}$, where "input" refers to all the inputs of each decoding node i in a decoder, and b represents the information symbol (bit). The key to iterative decoding is the sequence of *a posteriori LLRs* of the information symbols, $\vec{L}_{x_i}^{(k)}$, which is exchanged every iteration k between the decoding nodes of the decoder so each node takes advantage of the information computed by the others. To improve the bit estimations every iteration, the different nodes need to exchange belief information *that do not originate from themselves*. The original concept of extrinsic information was in fact conceived to identify the information components that depend on redundant information introduced by the incumbent code. Such extrinsic LLRs $\vec{L}_{e_i}^{(k)}$ are used to transform a posteriori LLRs into *a priori LLRs* $\vec{L}_{a_i}^{(k+1)}$ used as an input in the next iteration. For instance, a turbo decoder, consisting of two convolutional decoding nodes, computes two sets of extrinsic LLR vectors every iteration as follows:

$$\vec{L}_{e_i}^{(k)} = \vec{L}_{x_i}^{(k)} - \vec{L}_x - \vec{L}_{a_n}^{(k)}, \quad \forall i \neq n \in \{1, 2\}$$

where $\vec{L}_x$ is the decoder's sequence of input symbols. Extrinsic LLRs can be good estimators of the decodability of the code blocks (CBs) that constitute a TB. We define $S_{w,\Phi_n} := \{K, \vec{\bar{L}}_{e_1}, \ldots, \vec{\bar{L}}_{e_D}\}$, where $D, K \in \mathbb{N}$ are, respectively, the number of decoding nodes and the number of iterations completed by w by time Φ_n. $\vec{\bar{L}}_{e_i} := [\bar{L}_{e_i}^{(1)}, \ldots, \bar{L}_{e_i}^{(K)}]$ is a K-dimensional vector comprised of the mean magnitude of *extrinsic* LLRs at every iteration $k = \{1, \ldots, K\}$, i.e., $\bar{L}_{e_i}^{(k)} = \frac{1}{N} \sum_{b=1}^{N} |L_{e_i,b}^{(k)}|$, where N is the length of the CB being decoded and $L_{e_i,b}$ is the extrinsic LLR of bit b.

Fig. 15 shows the mean $\vec{\bar{L}}_{e_i}$ of both decoding nodes in a turbo decoder for undecodable CBs (in red) and for decodable CBs (in other colors).[5] The color of decodable CBs indicate the maximum number of iterations required till CRC validates their successful decoding. We can observe that the mean extrinsic magnitude of *undecodable* CBs (in red) is small and rather steady across iterations. In contrast, the mean extrinsic magnitude of decodable CBs grow over iterations. That is, there exist patterns of extrinsic information as it propagates across iterations in the decoder that are distinguishable between decodable CBs and undecodable CBs.

[5]Dataset available at https://github.com/agsaaved/nuberu

Algorithm 1 E-HARQ rule $\Pi_{\vec{\gamma}}$

```
1: if $K < 1$ then
2:     if $\mathcal{P}(\text{MCS}, \text{SNR}, N) > \epsilon$ then          ▷ Using [31]
3:         $\Omega \leftarrow$ UNKNOWN
4:     else
5:         $\Omega \leftarrow$ DECODABLE
6: else
7:     if $\left(\bar{L}_{e_D}^{(K)} < \gamma_1\right) \| \left(\bar{L}_{e_D}^{(K)} - \bar{L}_{e_D}^{(2)} < \gamma_2\right)$ then
8:         $\Omega \leftarrow$ UNDECODABLE
9:     else
10:        $\Omega \leftarrow$ DECODABLE
11: Return $\Omega \in$ {UNDECODABLE, DECODABLE, UNKNOWN}
```

In light of the above, we implement a simple rule $\Pi_{\vec{\gamma}}$, parametrized by $\vec{\gamma} = [\gamma_1, \gamma_2]$ and detailed in Algorithm 1, that poses minimal overhead to the DU forethread. Note that, in case we do not have enough extrinsic information to make a prediction because the decoder has not completed any iteration, we resort to simple predictive models that only rely on SNR estimates such as that in [31].

4.3.2 UL Congestion Control. To maximize network performance, we need to generate UL grants that can be decoded before the MVSF deadline or allow sufficient decoding iterations to help E-HARQ make accurate predictions. Otherwise, these UL grants would have to be discarded, which wastes resources. To achieve this, we resort to another *congestion controller* that adapts the emission of UL grants to the availability of computing resources.

Similar to its downlink counterpart, we define a congestion window $cwnd_{\text{UL}}^{(n)}$, which bounds the number of UL PRBs that can be allocated to the DU's users. Again, we adopt a simple AIMD algorithm that increases $cwnd_{\text{UL}}^{(n)}$ additively ($cwnd_{\text{UL}}^{(n+1)} = cwnd_{\text{UL}}^{(n)} + \alpha$) when congestion is *not* inferred, and decreases multiplicatively ($cwnd_{\text{UL}}^{(n+1)} = cwnd_{\text{UL}}^{(n)} \cdot \beta$), with $\beta < 1$, when congestion *is* inferred. The obvious approach to estimate congestion from UL workload is to signal so every time a UL-Data worker does not finish before the MVSF deadline. However, this method does not fully exploit the predictive capability of our E-HARQ mechanism, which can infer the decodability of a TB well before explicit CRC confirmation. Conversely, our approach infers congestion every time E-HARQ cannot provide a prediction with certainty, i.e., outputs UNKNOWN, which occurs every time a UL-Data worker is unable to run sufficient decoding iterations before its deadline (see Alg. 1).

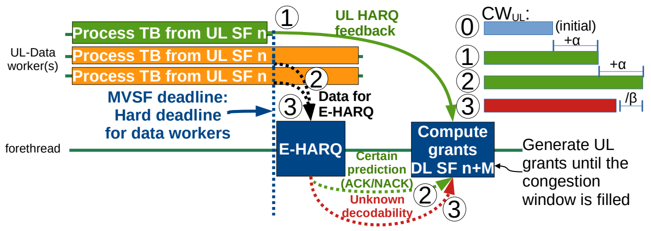

Figure 16: E-HARQ and UL congestion control during job n

Fig. 16 shows an example of a job where PUSCH carries three UL TBs, one of which, marked in green as case (1), is declared decodable (CRC has been verified) or not (the iterative algorithm has reached the maximum number of iterations allowed before the MVSF deadline. This causes the controller to additively increase $cwnd_{\text{UL}}^{(n)}$. Conversely, cases (2) and (3) require of the predictive ability of E-HARQ because they did not finish on time. In case (2), E-HARQ provides a certain estimation (decodable or undecodable), which also increases $cwnd_{\text{UL}}^{(n)}$ because the amount of work done on time is enough to let E-HARQ be accurate. In the last case (3), however, E-HARQ cannot provide a certain estimation because of lack of sufficient extrinsic information and, as a result, congestion is assumed and $cwnd_{\text{UL}}^{(n)}$ is decremented accordingly.

5 VALIDATION AND EVALUATION

To validate and evaluate our pipeline design, we have implemented Nuberu into srsRAN's stack [13] with the parameters shown in Table 2, and use its implementation of the baseline architecture shown in Fig. 8 as our benchmark. This provides a fair comparison because in this way both Nuberu and the baseline approach rely on identical implementations of individual tasks such as "PUSCH process" or "PDSCH process". Consequently, we can show that the gains provided by Nuberu do not come from optimizing signal processing operations within individual tasks but from Nuberu's architectural advantages over unreliable computing platforms.

Table 2: Parametrization of Nuberu

Parameter	Value
Pipeline parallelization depth (M)	4
Nr. DL-Data DU workers	3 per forethread
Nr. UL-Data DU workers	3 per forethread
UL-Data DU worker's buffer size	8 grants
DL-Data DU worker's buffer size	25 grants
Phase II budget (τ)	1.3 ms
UL Congestion Control (α, β)	3, 0.9
DL Congestion Control (α, β, λ)	3, 0.9, 2
E-HARQ (γ_1, γ_2)	600, 200

It is also worth remarking that, different to previous works [9, 19] that do not implement a complete DU pipeline, both Nuberu and its baseline benchmark support end-to-end communication with 3GPP compliant UEs. To this end, we rely on srsRAN's implementation of higher layers of the stack (RLC, PDCP) and its lightweight vEPC to carry out experiments with legacy IP traffic.

Moreover, we use an off-the-shelf Intel(R) Xeon(R) CPU D-1528 server with six x86 cores @ 1.90GHz, and Linux containers to host vDU instances. To avoid uncontrolled effects, we disable hyper-threading and apply CPU shielding to allocate five cores to host vDU instances; the remaining core is reserved for the OS and other processes (e.g., for monitoring). To focus on harsh computing environments, we do not use any hardware acceleration. In summary, this section introduces the following set of results:

- Validate Nuberu's approach and assess its performance in DL scenarios with different computing capacity settings (§5.1);
- Evaluate Nuberu's congestion controller (§5.2);
- Assess Nuberu's E-HARQ under different network conditions and computing capacity settings (§5.3);
- Estimate the network capacity region of a single vDU in different computing capacity settings (§5.4);
- Evaluate Nuberu in scenarios with a variable number of vDUs contending for shared computing resources (§5.5); and
- Estimate the network capacity region in terms of individual aggregate throughput in scenarios with two vDUs (§5.6).

5.1 Downlink

We begin by assessing the performance of Nuberu with downlink traffic. To this end, we set up a scenario comprised of one vDU transmitting as much load as possible with high SNR, and measure throughput (rate of bits successfully decoded by the receiver) and buffering (time elapsed between scheduling a data grant for encoding and actually delivering it over the air). Fig. 17 compares the network throughput achieved by the baseline approach and by Nuberu with different settings of λ, and for different amounts of computing resources allocated to the vDU's encoder, which are measured as the number of bits that are encoded by unit of time, between 8 b/μs and 200 b/μs. As a reference, a *dedicated* Intel Xeon core @ 1.9GHz provides an encoding performance of 200 b/μs.

We first note that the baseline approach is *inelastic*: its throughput collapses suddenly when its DU jobs exceed the available processing time budget (see Fig. 8). In marked contrast, Nuberu preserves high throughput irrespective of the available computing capacity, providing *elasticity* upon computing fluctuations. Indeed, Nuberu's parameter λ serves to choose the desired trade-off between data buffering at the vPHY (shown by the right-most subplot) and throughput. When $\lambda = \infty$, effectively disabling congestion control, Nuberu is greedy and maintains maximum throughput irrespective of the computing capacity but incurs high buffering. Conversely, lower values of λ help reduce buffering at the cost of throughput when there is shortage of computing resources.

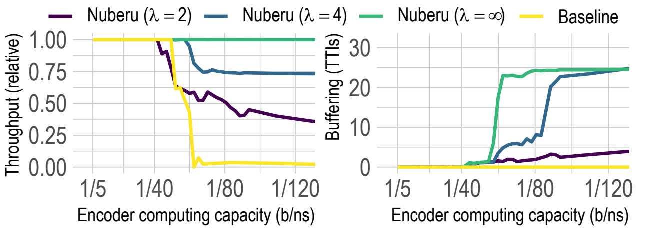

Figure 17: Downlink throughput (left) and vPHY buffering (right) with saturating load. Comparison between the baseline and Nuberu under different computing capacities.

5.2 Congestion Control

We next evaluate the performance of Nuberu's congestion controller. To this aim, we use the same setup as before with an encoder's computing capacity equal to 12.5 b/μs, and with $\alpha = 3$ and $\lambda = 2$. Based on this, we run a set of experiments with different DL load patterns and settings of β, where the load is varied by changing the duty cycle of an on-off generation process with no traffic during the off period and saturating rate during the on period. We plot in Fig. 18 the evolution over time of the throughput achieved for the different loads and values of β considered. Nuberu's congestion control effectively limits the data rate generated by the DU to avoid congestion, as set by parameter λ, when there are shortages of computing resources. To illustrate this, the plot also depicts with

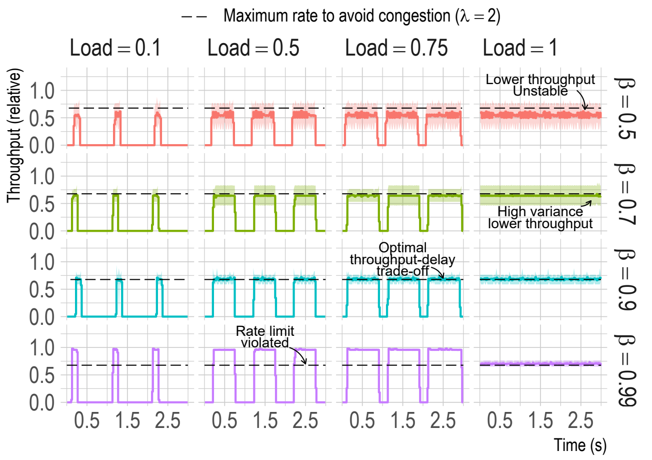

Figure 18: Congestion control performance for different loads and settings of β. $\alpha = 3, \lambda = 2$. Colored lines show mean throughput. Shades cover the area between the 10th and the 90th percentile. Dashed black lines represent the maximum rate limit to avoid congestion as set by $\lambda = 2$.

dashed black lines the theoretical rate limit that achieves this for $\lambda = 2$. On the one hand, $\beta < 0.9$ values are overly aggressive, rendering highly unstable performance and low throughput. On the other hand, $\beta > 0.9$ results in slow convergence to adapt $cwnd_{DL}$ to the optimal value when the load is bursty (< 1), which generates a data rate that violates the buffer constraint set by $\lambda = 2$. We hence use $\{\alpha = 3, \beta = 0.9\}$ in all our experiments as indicated by Table 2.

To provide more insights about the throughput-delay trade-off achieved by the congestion controller, we run more experiments with additional DL load patterns. For each pattern, we measure the mean throughput and one-way end-to-end delay (between end-user applications synchronized with IEEE 1588 protocol [10]) over 1-second windows for Nuberu ($\lambda = 2$, $\alpha = 3$, $\beta = 0.9$) and the baseline, and map each result in the throughput-delay plot shown in Fig. 19. To ease visualization, we only plot a small subset of the results and a curve fitting all of them. As shown by the figure, Nuberu substantially outperforms the baseline despite the use of buffering to accommodate computing jitter. Conversely, the baseline approach frequently loses synchronization, hence providing overly low throughput and, as a result, high end-to-end delay.

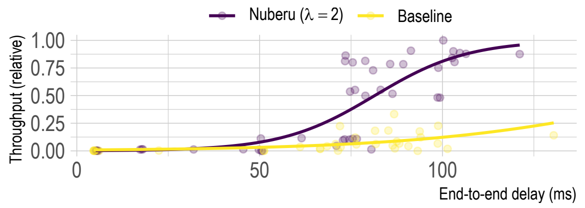

Figure 19: Throughput-delay trade-off for different loads.

5.3 Uplink (Early HARQ)

Different combinations of SNR, MCS, TB size, and computing capacity have different impact on E-HARQ. To assess this, we test different Π_γ (see Algorithm 1) and show in Fig. 20 its accuracy (ratio of predictions made right, at the top), and its ratio of false positives (bottom subplot). The latter is relevant because a false positive,

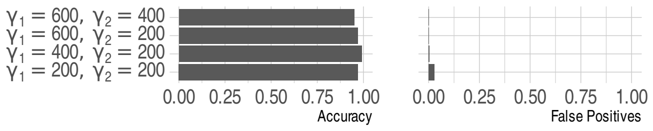

Figure 20: E-HARQ accuracy and false positive rate for different γ settings, and for different combinations of MCS, TB size, SNR, and computing capacity.

i.e., acknowledging a TB that is not decodable, incurs substantially higher cost because the TB has to be recovered by higher layers such as RLC or even TCP (if RLC does not use ARQ). This exercise lets us explore the parameter space of our E-HARQ approach and choose the most appropriate setting. Although $\{\gamma_1 = 600, \gamma_2 = 400\}$ reaches a negligible false-positive ratio ($\sim 0.04\%$) and high accuracy (95%), we have selected $\{\gamma_1 = 400, \gamma_2 = 200\}$ because it provides 99% accuracy while keeping low a false-positive ratio ($\sim 0.1\%$).

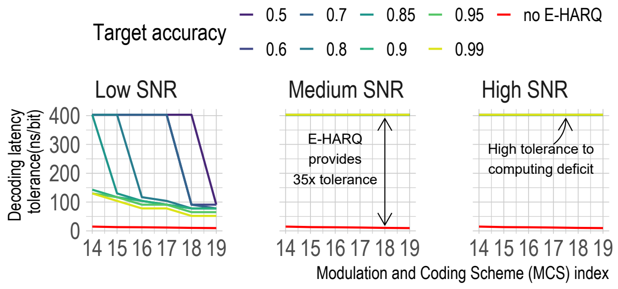

Figure 21: Maximum computing latency supported by Nuberu's E-HARQ given a target accuracy.

Certainly, the actual performance is determined by the (extrinsic) information available before the MVSF deadline in each job. To get additional insight on this, we present in Fig. 21 the minimum computing capacity required by the decoder to attain a given target of accuracy from 50% to 99% for 3 different SNR regimes (low, with SNR below 20 dB, medium, with SNR up to 25 dB, and high), and 6 different MCS indexes (14 to 19) from 3GPP. As a baseline to compare against, we plot with a red line the computing capacity required to run up to 10 decoder iterations in our time budget, which is the usual threshold to decide on the decodability of a TB when E-HARQ is not employed. As a reference, a *dedicated* Intel Xeon core @ 1.9GHz provides a decoding performance of ~ 80 b/μs per iteration. To attain high accuracy during medium and high SNR regimes, we only require sufficient computing resources so our decoder can process ~ 2.5 bits/μs (400 ns per bit). The rest of MCSs alternate between high and mild processing requirements depending on the accuracy target and MCS during low SNR conditions. Remarkably, our E-HARQ mechanism alone lets us reduce by 35x the amount of computing capacity required to carry out the task.

5.4 Single-vDU network capacity region

We now characterize empirically the *network capacity region* of one vDU under high SNR regimes, which generate the highest computational load and are a worst case. To this end, we measure DL and UL throughput for a wide set of UL/DL network load combinations. We also vary the amount of computing resources allocated to the encoder and the decoder by scaling them down to $k \cdot c_0$, where c_0 is the nominal encoding/decoding capacity of a *dedicated* Intel Xeon core @ 1.9GHz, and $k \leq 1$. Our results are shown in Fig. 22 for

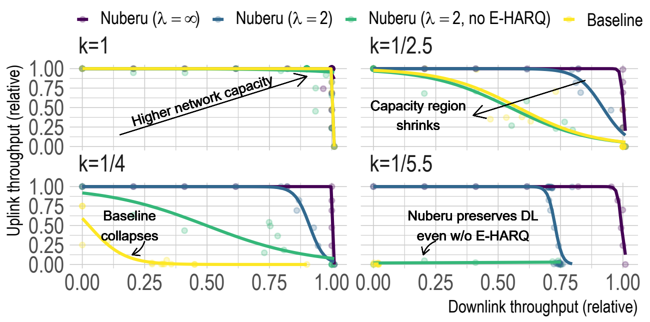

Figure 22: Network capacity region of Nuberu and our baseline. Different computing capacity settings equal to $k \cdot c_0$, where c_0 is the nominal encoding/decoding capacity of a *dedicated* Intel Xeon core @ 1.9GHz.

Nuberu, tuned with and without E-HARQ and $\lambda = \{2, \infty\}$, and for our baseline approach. To ease visualization, we only present the region envelop and a few experimental points around it.

Obviously, when we have dedicated and over-dimensioned computing capacity ($k = 1$), the networking capacity region shows a rectangular shape where uplink and downlink reach maximum spectral efficiency. However, if we reduce the system's computing capacity by $k = 1/2.5$, Nuberu $\lambda = \infty$ (in purple line) preserves the highest theoretical spectrum efficiency. Moreover, Nuberu $\lambda = 2$ (in dark blue line) provides a much higher maximum aggregate throughput than our baseline approach (yellow line). Most of this capacity gain comes from E-HARQ, as it can be seen from the fact that Nuberu falls to the baseline's performance when E-HARQ is disabled (green line). As we reduce the amount of computing resources ($k = 1/5$ and $k = 1/5.5$), the baseline's capacity region keeps shrinking while Nuberu sustains high downlink capacity (even if E-HARQ is disabled) as well as uplink (unless E-HARQ is disabled and the computing conditions are the harshest, i.e., $k = 1/5.5$). Remarkably, with $k = 1/5.5$, which represents an 80% computing resource reduction over the best scenario for the baseline approach, Nuberu reaches ~95% of the maximum theoretical spectrum efficiency.

5.5 Shared computing resources

In the previous sections, we have characterized the performance of a single vDU instance with Nuberu (and the baseline). In the following, we keep the maximum processing speed of our experimental platform (which attains 200 encoded bits/μs and ~ 80 decoded bits/μs per iteration), and deploy multiple vDU instances sharing the common computing resource (five Intel Xeon cores @ 1.90GHz).

Specifically, we let one vDU under test (Nuberu or Baseline) compete for resources with a variable number of competing vDUs that generate DU jobs with a duration that follows a normal distribution with average 1 ms and standard deviation 0.25 ms. Fig. 23 depicts the UL and DL throughput (left and right subplots, respectively) when the vDU under test uses Nuberu with the same settings used before (blueish and green bars), and when it uses the baseline approach (yellow bar), for different scenarios where the number of competing vDUs spans between 0 and 10.

Nuberu's performance remains high in all scenarios. Only when E-HARQ is not employed, UL throughput drops approximately 5% per competing vDU. When using E-HARQ, UL throughput only drops 10% when $\lambda = 2$ and ~2% when $\lambda = \infty$. Conversely, Nuberu

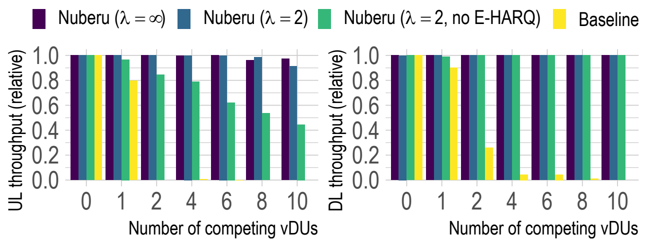

Figure 23: Network performance for a variable number of vDUs contending for computing resources.

attains the maximum spectral capacity in the DL channel regardless of the number of contending vDUs. In marked contrast, the performance of our baseline approach drops significantly with every competing vDU: UL performance collapses when it competes with more than 1 vDU, and DL throughput only reaches 27% with 2 competing vDUs and < 5% with a higher amount of competitors. This result provides strong evidence of the elasticity and high efficiency of Nuberu in shared cloud environments.

5.6 Multiple-vDU capacity region

The results shown above validate Nuberu's superior performance when sharing a platform with different amounts of computing interference. However, they do not evaluate how the available computing capacity is distributed among different instances of vDU, i.e., *fairness*. Consequently, we conclude our evaluation with an empirical characterization of the network capacity region of a system with two vDUs (vDU 1 and vDU 2) sharing the same computing platform. Both vDUs are homogeneous in this case, that is, they both implement Nuberu (with and without E-HARQ, and for $\lambda = \{2, \infty\}$) or the baseline architecture. Different from the experiments of §5.5, which showed competing vDUs with fixed load, we now test a wide set of UL and DL network load combinations; and we reduce the encoder/decoder processing speed of our platform by a quarter ($k = 1/4$ in §5.4) to study harsh computing conditions.

Fig. 24 depicts the throughput performance achieved by vDU 1 *versus* the obtained by vDU 2 for both DL (left plot) and UL (right plot) directions. Like in Fig. 17, we only present the capacity frontier with fitting lines and a few experimental points around it to simplify the plot. "Nuberu ($\lambda = \infty$)" maximizes the throughput of both vDUs in both DL and UL; "Nuberu ($\lambda = 2$)" maximizes UL performance and roughly halves DL throughput to control buffering at the PHY; and "Nuberu ($\lambda = 2$, no E-HARQ)" further halves UL throughput to avoid dropping UL data that cannot be decoded on time. In contrast, while the baseline approach reduces DL throughput in a similar way

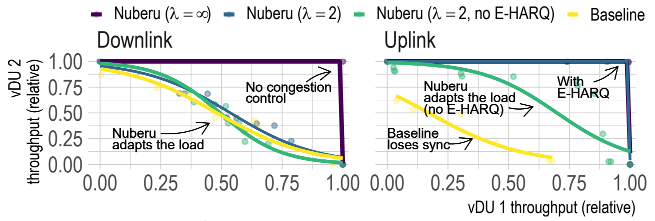

Figure 24: Network capacity region in a 2-vDU scenario. Encoder/decoder processing speed equal to $c_0/4$, where c_0 is the nominal speed of a *dedicated* Intel Xeon core @ 1.9GHz.

as Nuberu when $\lambda = 2$, its UL performance is roughly a third of the maximum spectrum capacity. This is because the users frequently lose synchronization with the DUs and cannot communicate while re-attaching. Importantly, this result proves that the individual performance gains provided by Nuberu do not come at the cost of fairness but because of a more efficient pipeline design when there are shortages of computing resources.

6 RELATED WORK

Network Function Virtualization (NFV) has received a lot of attention over the years [14, 21, 24, 37, 39]. NFP [37] exploits parallelism across sets of network functions to mitigate resource contention. ResQ [39] shows how to increase performance isolation by using appropriate processor cache isolation and I/O buffer sizes. PicNIC [21] opportunistically enforces isolation by applying a set of techniques: CPU-fair weighted fair queueing at receivers, congestion control, and admission control with shaping. Only last year, Microscope [14] and SLOMO [24] introduced performance diagnosis tools for network functions. The literature is rather vast. However, the focus has conventionally been on middleware, such as L2/3 forwarders, load balancers, firewalls, IDSs, VPNs, and others. Moreover, because metrics such as tail latency are particularly relevant in these applications, substantial effort has been devoted to the design of generic scheduling frameworks that can balance computing efficiency and latency performance [26, 29]. However, these approaches, though relevant, they alone do not solve the problem addressed in this paper since (*i*) the dependencies between DU tasks within each DU job impose constraints to parallelize DU tasks in different processors to expedite DU jobs, and (*ii*) while these techniques can reduce job latency, they cannot guarantee hard deadlines, which as we showed in this paper, is a requirement to provide reliable DUs.

Virtualized RANs are gaining substantial interest among mobile stakeholders [7, 18, 33] and academia [11]. Solutions such as Orion [11] or that in [27] enable the virtualization of *radio* resources efficiently but do not address problems related to the virtualization of the computing tasks involved in a DU pipeline. vrAIn [5] and others [4, 40] propose machine-learning-based controllers to optimize the allocation of radio and computing resources at long timescales (seconds). Though these solutions are useful for network slicing, they do not provide reliability to vDUs during (quick) fluctuations of computing capacity. Indeed, (pre-)commercial vDUs employ *dedicated* hardware accelerators assisting the baseline pipeline presented in §2 to provide reliability [7, 18, 33]. There is substantial work in this direction, e.g., [3, 6, 17]. Worth highlighting are Intel FlexRAN [36] and NVIDIA Aerial [34]. These are L1 solutions used in commercial vRANs, which can accelerate operations such as FEC using dedicated FPGAs (the former) and GPUs (the latter). However, they are closed-source and due to the lack of publicly available information, we cannot compare them with Nuberu. FlexRAN does provide openly its libraries to perform FEC [19], one important operation within a DU that we have used in our experiments of Fig. 4. Although these libraries improve the latency of important operations involved in some DU tasks, they are not enough to provide carrier-grade reliability in shared computing platforms because they do not guarantee meeting the hard deadline of DU jobs in these environments. A remarkable piece of work is Agora [9], which builds on FlexRAN's open libraries to provide a high-performing pipeline to process UL and DL data carried by PUSCH and PDSCH that is suitable for multi-core CPU platforms. However, like FlexRAN's open solutions, Agora does not implement a complete DU pipeline, requires dedicated platforms to provide reliable performance, and does not propose solutions to guarantee reliable full-fledged DU operation in shared platforms.

The use of specialized hardware for compute-intensive and repetitive tasks, such as FFT or FEC, seems unavoidable for carrier-grade performance. However, in line with the view of O-RAN, a carrier-led initiative, we believe that successful RAN virtualization has to go through shared pools of computing resources as depicted in Fig. 1 (see, e.g., [28]) to provide affordable and flexible solutions. The cost is however non-deterministic computations that lead to the unreliability problems presented in this paper. Latency-driven task schedulers and optimized baseband libraries like the ones mentioned above are obviously important because they expedite operations such as FEC. However, they are not enough to guarantee meeting the latency constraints of DUs and, consequently, they are not enough to provide carrier-grade reliability in non-deterministic computing platforms such as shared clouds. We have shown in this paper that a re-design of the baseline DU pipeline is required to achieve this goal. To the best of our knowledge, Nuberu is the first approach to design a complete DU pipeline that is specifically suitable for non-deterministic (e.g. shared) computing platforms.

7 CONCLUSIONS

Shared computing platforms are particularly relevant for the success of RAN virtualization as they alleviate the cost incurred by dedicated platforms and provide flexibility. However, research has shown that resource contention in shared computing infrastructure provides non-deterministic performance and tail latency. The Distributed Units (DU) of virtualized RANs are especially sensitive to this problem due to *hard time constraints* to execute the DU pipeline. In this paper, we showed that a re-design of the baseline DU pipeline is required to provide carrier-grade reliability in these platforms. We then proposed *Nuberu*, a novel DU design specifically engineered for 4G and 5G vRANs that are virtualized over clouds of shared resources with constrained and/or fluctuating computing capacity. To the best of our knowledge, Nuberu is the first full-fledged DU design that addresses the aforementioned reliability problem. Nuberu has one key objective: preserve synchronization with the users to provide reliability regardless of computing fluctuations. Provided that, Nuberu employs congestion control, predictive HARQ, and jitter-absorbing buffers to maximize performance under compute resource shortages. Finally, a comprehensive evaluation of Nuberu has been conducted in an experimental proof-of-concept. Our results show that Nuberu dramatically improves the robustness of vRANs in harsh computing environments over state-of-art solutions, e.g., providing as much as $\sim 95\%$ spectrum efficiency when computing resources are reduced by $\sim 80\%$.

ACKNOWLEDGEMENTS

We would like to thank our shepherd and reviewers for their valuable comments and feedback. This work has been supported by the European Commission through Grant No. 101017109 (DAEMON project) and Grant No. 101015956 (Hexa-X project), and the CERCA Programme/Generalitat de Catalunya.

REFERENCES

[1] 3GPP. 2020. 5G;NG-RAN; Architecture description . 3GPP TS 38.401 version 16.2.0 Release 16.

[2] NGMN Alliance. 2019. 5G E2E Technology to Support Verticals URLLC Requirements.

[3] Alex Aronov, Leonid Kazakevich, Jane Mack, Fred Schreider, and Scott Newton. 2019. 5G NR LDPC Decoding Performance Comparison between GPU & FPGA Platforms. In *2019 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. IEEE, 1–6.

[4] Jose A. Ayala-Romero, Andres Garcia-Saavedra, Xavier Costa-Perez, and George Iosifidis. 2021. Bayesian Online Learning for Energy-Aware Resource Orchestration in Virtualized RANs. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488845

[5] Jose A. Ayala-Romero, Andres Garcia-Saavedra, Marco Gramaglia, Xavier Costa-Perez, Albert Banchs, and Juan J. Alcaraz. 2020. vrAIn: Deep Learning based Orchestration for Computing and Radio Resources in vRANs. *IEEE Transactions on Mobile Computing* (2020), 1–1. https://doi.org/10.1109/TMC.2020.3043100

[6] Vinay Chamola, Sambit Patra, Neeraj Kumar, and Mohsen Guizani. 2020. FPGA for 5G: Re-configurable Hardware for Next Generation Communication. *IEEE Wireless Communications* (2020).

[7] Cisco, Rakuten, Altiostar. 2019. Reimagining the End-to-End Mobile Network in the 5G Era. *White Paper* (2019).

[8] Erik Dahlman, Stefan Parkvall, and Johan Skold. 2018. *5G NR: The next generation wireless access technology*. Academic Press.

[9] Jian Ding, Rahman Doost-Mohammady, Anuj Kalia, and Lin Zhong. 2020. Agora: Real-time massive MIMO baseband processing in software. In *Proceedings of ACM CoNEXT '20*. ACM.

[10] John C Eidson. 2006. *Measurement, control, and communication using IEEE 1588*. Springer Science & Business Media.

[11] Xenofon Foukas, Mahesh K. Marina, and Kimon Kontovasilis. 2017. Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. Association for Computing Machinery, New York, NY, USA, 127–140. https://doi.org/10.1145/3117811.3117831

[12] Andres Garcia-Saavedra and Xavier Costa-Perez. 2021. O-RAN: Disrupting the Virtualized RAN Ecosystem. *IEEE Communications Standards Magazine* (2021).

[13] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. 2016. srsLTE: an open-source platform for LTE evolution and experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*. 25–32.

[14] Junzhi Gong, Yuliang Li, Bilal Anwer, Aman Shaikh, and Minlan Yu. 2020. Microscope: Queue-Based Performance Diagnosis for Network Functions. In *Proceedings of ACM SIGCOMM '20*. ACM, 390–403.

[15] Wang Tsu Han and Raymond Knopp. 2018. OpenAirInterface: A pipeline structure for 5G. In *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. IEEE, 1–4.

[16] Lajos Hanzo, Jason P Woodard, and Patrick Robertson. 2007. Turbo decoding and detection for wireless applications. *Proc. IEEE* 95, 6 (2007), 1178–1200.

[17] Yan Huang, Shaoran Li, Y Thomas Hou, and Wenjing Lou. 2018. GPF: A GPU-based Design to Achieve˜ 100 μs Scheduling for 5G NR. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 207–222.

[18] Intel. 2017. vRAN: The Next Step in Network Transformation. *White Paper* (2017).

[19] Intel. 2019. *FlexRAN LTE and 5G NR FEC Software Development Kit Modules*. https://software.intel.com/content/www/us/en/develop/articles/flexran-lte-and-5g-nr-fec-software-development-kit-modules.html

[20] Florian Kaltenberger, Aloizio P Silva, Abhimanyu Gosain, Luhan Wang, and Tien-Thinh Nguyen. 2020. OpenAirInterface: Democratizing innovation in the 5G Era. *Computer Networks* (2020), 107284.

[21] Praveen Kumar, Nandita Dukkipati, Nathan Lewis, Yi Cui, Yaogong Wang, Chonggang Li, Valas Valancius, Jake Adriaens, Steve Gribble, Nate Foster, and Amin Vahdat. 2019. PicNIC: Predictable Virtualized NIC. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, 351–366.

[22] Jiangpeng Li, Guanghui He, Hexi Hou, Zhejun Zhang, and Jun Ma. 2011. Memory efficient layered decoder design with early termination for LDPC codes. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2697–2700.

[23] X. Lin, J. Li, R. Baldemair, J. T. Cheng, S. Parkvall, D. C. Larsson, H. Koorapaty, M. Frenne, S. Falahati, A. Grovlen, and K. Werner. 2019. 5G New Radio: Unveiling the Essentials of the Next Generation Wireless Access Technology. *IEEE Communications Standards Magazine* 3, 3 (2019), 30–37.

[24] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. 2020. Contention-Aware Performance Prediction For Virtualized Network Functions. In *Proceedings of ACM SIGCOMM '20*. ACM, 270–282.

[25] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez. 2019. Resource Sharing Efficiency in Network Slicing. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 909–923. https://doi.org/10.1109/TNSM.2019.2923265

[26] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: A Microkernel Approach to Host Networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 399–413. https://doi.org/10.1145/3341301.3359657

[27] Jose Mendes, XianJun Jiao, Andres Garcia-Saavedra, Felipe Huici, and Ingrid Moerman. 2019. Cellular access multi-tenancy through small-cell virtualization and common RF front-end sharing. *Computer Communications* 133 (2019), 59–66.

[28] O-RAN Alliance. 2020. Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN v02.01 (O-RAN.WG6.CAD-v02.01). Technical Report.

[29] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 361–378. https://www.usenix.org/conference/nsdi19/presentation/ousterhout

[30] Rethink Technology Research. 2020. Special Report: Open Networks. *Technical Report* (2020).

[31] P. Rost and A. Prasad. 2014. Opportunistic Hybrid ARQ—Enabler of Centralized-RAN Over Nonideal Backhaul. *IEEE Wireless Communications Letters* 3, 5 (2014), 481–484. https://doi.org/10.1109/LWC.2014.2327982

[32] P Salija and B Yamuna. 2016. An efficient early iteration termination for turbo decoder. *Journal of Telecommunications and Information Technology* (2016).

[33] Samsung. 2019. Virtualized Radio Access Network: Architecture, Key technologies and Benefits. *Technical Report* (2019).

[34] Soma Velayutham. 2019. *NVIDIA CEO Introduces Aerial — Software to Accelerate 5G on NVIDIA GPUs*. https://blogs.nvidia.com/blog/2019/10/21/aerial-application-framework-5g-networks/

[35] Nils Strodthoff, Barış Göktepe, Thomas Schierl, Cornelius Hellge, and Wojciech Samek. 2019. Enhanced machine learning techniques for early HARQ feedback prediction in 5G. *IEEE Journal on Selected Areas in Communications* 37, 11 (2019), 2573–2587.

[36] Sujata Tibrewala. 2018. *The 5G network transformation*. https://software.intel.com/en-us/articles/the-5g-network-transformation

[37] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. 2017. NFP: Enabling Network Function Parallelism in NFV. In *Proceedings of ACM SIGCOMM '17*. ACM, 43–56.

[38] Yang Sun and Joseph R Cavallaro. 2011. A flexible LDPC/turbo decoder architecture. *Journal of Signal Processing Systems* 64, 1 (2011), 1–16.

[39] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. 2018. ResQ: Enabling SLOs in Network Function Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 283–297. https://www.usenix.org/conference/nsdi18/presentation/tootoonchian

[40] Lanfranco Zanzi, Vincenzo Sciancalepore, Andres Garcia-Saavedra, and Xavier Costa-Pérez. 2018. OVNES: Demonstrating 5G network slicing overbooking on real deployments. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1–2.