Towards a Cloud-Native Radio Access Network

Navid Nikaein, Eryk Schiller, Romain Favraud, Raymond Knopp, Islam Alyafawi and Torsten Braun

Abstract Commoditization and virtualization of wireless networks are changing the economics of mobile networks to help network providers, e.g. Mobile Network Operator (MNO), Mobile Virtual Network Operator (MVNO), move from proprietary and bespoke hardware and software platforms towards an open, cost-effective, and flexible cellular ecosystem. In addition, rich and innovative local services can be efficiently materialized through cloudification by leveraging the existing infrastructure. In this work, we present a Radio Access Network as a Service (RANaaS), in which a Cloudified Centralized Radio Access Network (C-RAN) is delivered as a service. RANaaS describes the service life-cycle of an on-demand, elastic, and pay as you go RAN instantiated on top of the cloud infrastructure. Due to short deadlines in many examples of RAN, the fluctuations of processing time, introduced by the virtualization framework, have a deep impact on the C-RAN performance. While in typical cloud environments, the deadlines of processing time cannot be guaranteed, the cloudification of C-RAN, in which signal processing runs on general purpose processors inside Virtual Machines (VMs), is a challenging subject. We describe an example of real-time cloudifed LTE network deployment using the OpenAirInterface (OAI) LTE implementation and OpenStack running on commodity hardware. We also show the flexibility and performance of the platform developed. Finally, we draw general conclusions on the RANaaS provisioning problem in future 5G networks.

Eryk Schiller, Islam Alyafawi, Torsten Braun
University of Bern, Switzerland, e-mail: {name}@inf.unibe.ch

Navid Nikaein, Romain Favraud, Raymond Knopp
EURECOM, France, e-mail: {name.surname}@eurecom.fr

Romain Favraud is also with DCNS Group, France.

1 Introduction

Every day, we encounter an increasing demand for wireless data use due to a growing number of broadband-capable devices, such as 3G and 4G mobile telephones. To satisfy a higher demand for data rates, service providers and mobile operators expect upgrades and expansion of the existing network, but the required Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) are superior to the revenue growth [9]. The high upgrade and maintenance costs are mainly caused by the current architecture of mobile broadband networks, in which the Radio Access Network (RAN) is built upon the integrated Base Transceiver Station (BTS) architecture. Since mobile broadband providers operate on a large scale, the installation and maintenance of a large number of expensive BTSs over vast geographical areas increase the cost dramatically. Moreover, the new trend of smaller cells will more severely affect both the cost and maintenance problem in the future.

A cost-effective RAN solution, which meets the ever-increasing amounts of mobile data traffic, has to fulfill a set of requirements. First, the new RAN has to quickly and automatically scale with the variable amount of mobile traffic. Second, it has to consume less power providing higher capacity and network coverage at the same time. Finally, it should allow mobile operators to frequently upgrade and operate the service over multiple/heterogeneous air-interfaces.

Only about 15–20% of BTSs operating in the current RAN architecture are loaded more than 50% (with respect to the maximum capacity), which makes the current RAN architecture energy inefficient [8]. An emerging solution to reduce upgrading costs and power consumption is the Centralized-RAN (C-RAN) [13, 17] with resource sharing and exploitation of load patterns at a given geographical area. The C-RAN solution is more adaptable to variations in user data traffic and unpredictable mobility patterns than the current RAN. Moreover, it allows coordinated and joint signal processing to increase the spectral efficiency. Finally, the C-RAN represents a good match between the spatial-temporal traffic variations and available computational resources and hence power consumption.

Since C-RAN signal processing is centralized, it allows us to apply more sophisticated joint spatio-temporal processing of radio signals, which can increase the overall spectral efficiency. The joint signal processing approach is considered by European projects such as Sail [24], iJoin [14], and Mobile Cloud Networking (MCN) [15]. This work has been carried out as part of the MCN vision towards shifting radio communication networks to the cloud computing paradigm. Cloud computing technologies based on virtualization allow us to lower the operational costs even more by running the RAN through a) adoption of general purpose IT platforms instead of expensive specific hardware, b) load balancing, and c) fast deployment and resource provisioning. Running the RAN in the cloud environment is not new. The benefit of such an approach has demonstrated 71% of power savings

when compared to the existing system [7]. However, this approach comes at the cost of higher software complexity.

Recent works [27] have shown the feasibility of LTE RAN functions of software implementation over General Purpose Processors (GPPs), rather than the traditional implementation over Application-Specific Integrated Circuits (ASICs), Digital Signal Processors (DSPs), or Field-Programmable Gate Arrays (FPGAs). Different software implementations of the LTE base station, which is referred to as evolved Node B (eNB), already exist: a) Amarisoft LTE solution, which is a pure-software featuring a fully-functional LTE eNB [2], b) Intel solutions featuring energy efficiency and high computing performance using a hybrid GPP-accelerator architecture and load-balance algorithms among a flexible IT platform [25] and c) OpenAirInterface (OAI) developed by EURECOM, which is an open-source Software Defined Radio (SDR) implementation of both the LTE RAN and the Evolved Packet Core (EPC) [12].

This chapter describes recent progress in the C-RAN cloudification (running software-based RAN in the cloud environment) based on the open source implementations and has the following organization. In Sect. 2, we introduce the concept, architecture, and benefits of centralized RAN in the LTE Network setup. Section 3 presents the critical issues of cloudified RAN focusing on fronthaul latencies, processing delays, and appropriate timing. Our performance evaluation of GPP-based RAN is provided in Sect. 4 and Base-Band Unit (BBU) processing time is modeled in Sect. 5. Possible architectures of cloudified RAN are described in Sect. 6. The description of the cloud datacenter supporting C-RAN resides in Sect. 7. Section 8 illustrates an example RANaaS with its life-cycle management. Finally, we conclude in Sect. 9.

2 Centralized RAN in the LTE Network

C-RAN based networks are characterized by the decomposition of a BTS into two entities namely Base-Band Unit (BBU) and Remote Radio Head (RRH). In C-RAN, the RRH stays at the location of the BTS, while the BBU gets relocated into a central processing pool, which hosts a significant number of distinct BBUs [27]. In order to allow for signal processing at a remote BBU, a point-to-point high capacity interface of short delay is required to transport I/Q samples (i.e., digitized analog radio signals) between RRH and BBU. There are a few examples of link standards meeting the required connectivity expectations such as Open Radio Interface (ORI), Open Base Station Architecture Initiative (OBSAI), or Common Public Radio Interface (CPRI). Even though many recent works have shown the feasibility of C-RAN implementation and the C-RAN importance for the MNOs, there are still three open questions that has to be thoroughly investigated upon a C-RAN system design.

1. **Dimensioning of the fronthaul capacity:** a BBU pool has to support a high fronthaul capacity to transport I/Q samples for a typical set of 10–1000 base-stations working in the BBU–RRH configuration. Due to a low processing budget of RAN, the upper bound for the maximum one-way delay has to be estimated. Moreover, a very low jitter has to be maintained for the clock synchronization among BBUs and RRHs.
2. **Processing budget at the BBU:** in the LTE FDD setup, the Hybrid automatic repeat request (HARQ) mechanism with an 8 ms acknowledgment response time provides an upper bound for the total delay of both fronthaul latency and BBU processing time.
3. **The real-time requirements of the Operating-System and Virtualization-System:** to successfully provide frame/subframe timings, the execution environment of the BBU has to support strict deadlines of the code execution. Moreover, load variations in the cell (e.g., day/night load shifts) impose the requirement on the on-demand resource provisioning and load balancing of the BBU pool.

There are also many other challenges in this field [6], such as front-haul multiplexing, optimal clustering of BBUs and RRHs, BBU interconnection, cooperative radio resource management, energy optimization, and channel estimation techniques. The following subsections focus on the critical issues, and present C-RAN feasible architectures.

3 Critical Issues of C-RAN

In the following subsections, we evaluate the most important critical issues of the C-RAN. We concentrate on the fronthaul capacity problem, BBU signal processing, and real-time cloud infrastructure for signal processing [18].

3.1 Fronthaul Capacity

We start with the description of fronthaul requirements. A very fast link of low delay is necessary as the BBU processes the computationally most heavy physical (PHY) layer of the LTE standards. Many factors contribute to the data rate of the fronthaul, which depends on the cell and fronthaul configurations. Eq. 1 calculates the required data rate based on such configurations:

$$C_{fronthaul} = \underbrace{2 \times N \times M \times F \times W \times C}_{\text{cell configuration}} \quad \times \quad \underbrace{O \times K}_{\text{fronthaul configuration}} \quad , \tag{1}$$

where N is the number of receiving/transmitting (Tx/Rx) antenna ports, M is the number of sectors, F represents the sampling rate, W is the bit width of an I/Q symbol, C number of carrier components, O is the ratio of transport protocol and coding overheads, and K is the compression factor. The following table shows the required fronthaul capacity for a simple set of configurations. An overall overhead is assumed to be 1.33, which is the result of the the protocol overhead ratio of 16/15 and the line coding of 10/8 (CPRI case). One can observe that the fronthaul capacity heavily depends on the cell configuration and rapidly grows with the increased sampling rate, number of antennas/sectors and carrier components.

Table 1 Fronthaul capacity for different configurations

BW (MHz)	N	M	F	W (bits)	O	C	K	$C_{fronthaul}$ (Mb/s)
1.4	1×1	1	1.92	16	1.33	1	1	81
5	1×1	1	7.68	16	1.33	1	1	326
5	2×2	1	7.68	16	1.33	1	1	653
10	4×4	1	15.36	16	1.33	1	1/2	1300
20	1×1	1	30.72	16	1.33	1	1	1300
20	2×2	3	30.72	16	1.33	1	1	7850
20	4×4	3	30.72	16	1.33	1	1	15600

Fig. 1 compares the fronthaul capacity between the RRH and the BBU pool for 20 MHz BW, SISO (max. 75 Mb/s on the radio interface). In the case without compression, the fronthaul has to provide at least 1.3 Gb/s; when the 1/3 compression ratio is used, the required fronthaul capacity drops to 0.45 Gb/s.

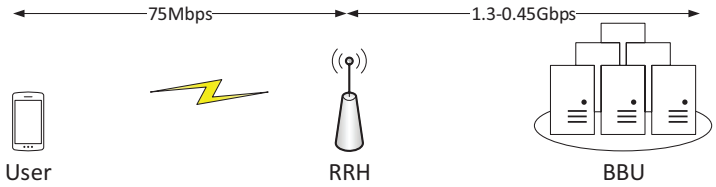


Fig. 1 Fronthaul capacity between the RRH and the BBU pool for 20 MHz BW, Single Input Single Output (SISO). Minimum required fronthaul capacity without compression is estimated at 1.3 Gb/s, the deployment of 1/3 compression ratio decreases the required capacity to 0.45 Gb/s

Further data rate reduction can be obtained by an RRH offloading the BBU functions. As shown in Fig. 2, the functional split can be provided by decoupling the L3/L2 from the L1 (labelled 4), or part of the user processing from the L1 (labelled 3), or all user-specific from the cell processing (labelled 2), or antenna-

specific from non-antenna processing (labelled 1), which is different for the Rx and Tx chain.

Trade-offs have to be performed among the available fronthaul capacity, complexity, and the resulted spectral efficiency. Regardless various possibilities in the BBU functional split, the fronthaul should still maintain the latency requirement to meet the HARQ deadlines. According to *Chanclou et al.* [5], the RTT between RRH and BBU equipped with a CPRI link cannot exceed 700 µs for LTE and 400 µs for LTE-Advanced. Jitter required by advanced CoMP schemes in the MIMO case is below 65 ns as specified in 3GPP 36.104. Next Generation Mobile Networks (NGMN) adopts the maximum fronthaul round-trip-time latency of 500 µs [16][1]. The propagation delay, corresponding to timing advance, between RRH and UE, affects only the UE processing time. The timing advance value can be up to 0.67 ms (equivalent to maximum cell radius of 100 km). Consequently, this leaves the BBU PHY layer only with around 2.3–2.6 ms for signal processing at a centralized processing pool. The next subsection elaborates on the BBU processing budget in the LTE FDD access method.

3.2 Processing budget in LTE FDD

This subsection describes the processing budget problem of the Frequency-Division Long-Term Evolution (LTE-FDD). We concentrate on the Physical Layer (PHY) and Medium Access Control (MAC). PHY is responsible for symbol level processing, while MAC provides user scheduling and HARQ. The LTE FDD PHY is implemented in the asymmetric way using Orthogonal Frequency-Division Multiple Access (OFDMA) and Single-Carrier Frequency-Division Multiple Access (SC-FDMA) on the downlink and uplink respectively. To control the goodput of the air interface, the PHY uses various Modulation and Coding Schemes (MCSs). 3GPP defines 28 MCSs with indexes between 0 and 27. A distinct MCS is characterized by a specific modulation (i.e., QPSK, 16-QAM, 64-QAM having a varying number of data bits per modulation symbol carried) and the so called code rate, which measures the information redundancy in a symbol for error correction purposes [3, 9]. The smallest chunk of data transmitted by an eNB through the LTE FDD PHY is referred to as Physical Resource Block (PRB) and spans 12 sub-carriers (180 kHz) and 7 modulation symbols (0.5 ms), which gives 84 modulation symbols in total. In LTE FDD, we are provided with channels of 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, and 20 MHz bandwidth, which can simultaneously carry 6, 15, 25, 50, 75, and 100 PRBs respectively. Therefore, the workload of signal processing in software is heavily influenced by the MCS index, number of allocated PRBs, and the channel width. Moreover, Hybrid Automatic Repeat Request protocol (HARQ) on

[1] Different protocols have been standardized for the fronthaul, namely CPRI representing 4/5 of the market, OBSAI representing 1/5 of the market, and more recently the Open Radio Interface (ORI) initiated by NGMN and now by the European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG).

the MAC layer introduces short deadline for signal processing on the PHY. Due to HARQ, every transmitted chunk of information has to be acknowledged back at the transmitter to allow for retransmissions. In LTE-FDD, the retransmission time is equal to T_{HARQ} of 8 ms. Let us briefly explain the retransmission mechanism. Every LTE FDD subframe (subframe is later referred to as SF) lasts for 1 ms and contains information chunks carried within PRBs. The HARQ protocol states that the Acknowledgment (ACK) or Negative Acknowledgment (NACK) for a data chunk received at subframe N has to be issued upon a subframe N+4 and decoded at the transmitter before subframe N+8, which either sends new data or again negatively acknowledged chunks. In the following subsection, we briefly summarize the BBU functions.

3.3 BBU Functions

Fig. 2 illustrates the main RAN functions in both TX and RX spanning all the layers, which has to be evaluated to characterize the BBU processing time and assess the feasibility of a full GPP RAN. Since the main processing bottleneck resides in the physical layer, the scope of the analysis in this chapter is limited to the BBU functions. From the figure, it can be observed that the overall processing is the sum of cell- and user-specific processing. The former only depends on the channel bandwidth and thus imposes a constant base processing load on the system, whereas the latter depends on the MCS and resource blocks allocated to users as well as the Signal-to-Noise Ratio (SNR) and channel conditions. The figure also shows the interfaces where the functional split could happen to offload the processing either to an accelerator or to an RRH.

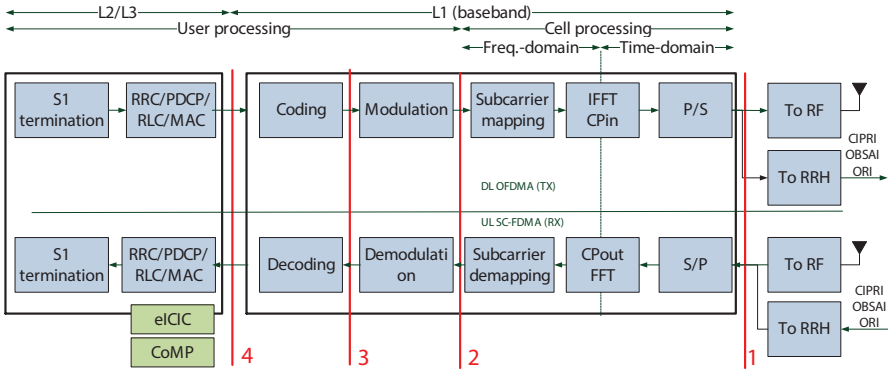
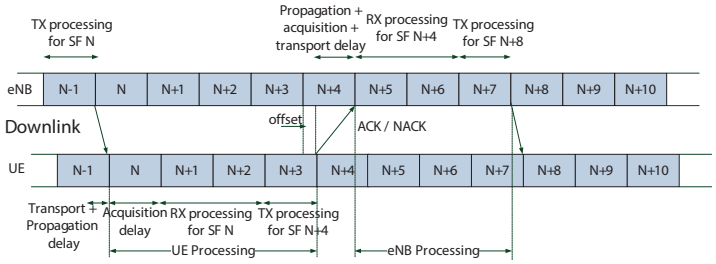
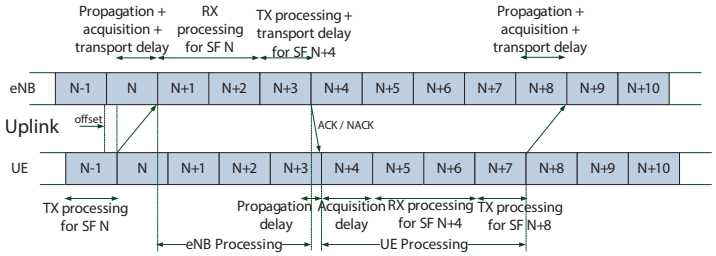


Fig. 2 Functional block diagram of downlink and uplink for LTE eNB

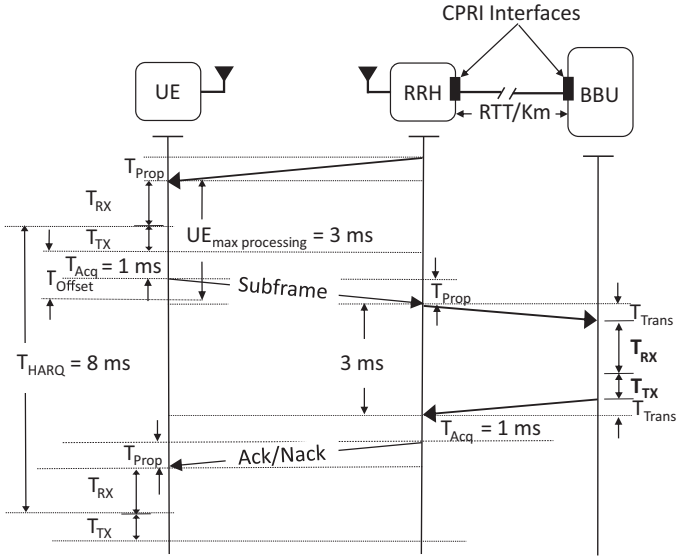
To meet the timing and protocol requirements, the BBU must finish processing before the deadline previously discussed at the beginning of Sect. 3.2. Each MAC



(a) *Downlink HARQ timing.*



(b) *Uplink HARQ timing.*



(c) *HARQ process timing requirement.*

Fig. 3 FDD LTE timing

PDU sent at subframe N is acquired in subframe N+1, and must be processed in both RX and TX chains before subframe N+3 allowing ACK/NACK to be transmitted in subframe N+4. On the receiver side, the transmitted ACK or NACK will be acquired in subframe N+5, and must be processed before subframe N+7, allowing

the transmitter to retransmit or clear the MAC PDU sent in subframe N. Fig. 3(a) and 3(b) show an example of timing deadlines required to process each subframe on the downlink and uplink respectively. Fig. 3(c) graphically represents the communication between the UE, RRH, and BBU. It can be observed that the total processing time is 3 ms. The available processing time for a BBU to perform the reception and transmission is upper-bounded by HARQ round trip time (T_{HARQ}), propagation time ($T_{Prop.}$), acquisition time ($T_{Acq.}$), and fronthaul transport time ($T_{Trans.}$) as follows:

$$T_{rx} + T_{tx} \leq T_{HARQ}/2 - \left(T_{Prop.} + T_{Acq.} + T_{Trans.} + T_{Offset}\right), \quad (2)$$

where $T_{HARQ} = 8\,\text{ms}$, $T_{Prop.}$ is compensated by the timing advance of the UE: $T_{Prop.} = 0$, $T_{Acq.}$ is equal to the duration of the subframe: $T_{Acq.} = 1\,\text{ms}$, and there is no BBU offset on the downlink: $T_{Offset} = 0$. Depending on the implementation, the maximum tolerated transport latency depends on the BBU processing time and HARQ period. The LTE FDD access method puts a particular focus on perfect timing of (sub-) frame processing. To accomplish this goal, the processing system has to fulfill real-time requirements. The next subsection focuses on the real-time cloud system design capable of C-RAN provisioning.

3.4 Real-time Operating System and Virtualization Environment

A typical general purpose operating systems (GPOS) is not designed to support real-time applications with hard deadline. Hard real-time applications have strict timing requirements to meet deadlines. Otherwise unexpected behaviors can occur compromising performance. For instance, Linux is not a hard real-time operating system as the kernel can suspend any task when a desired runtime has expired. As a result, the task can remain suspended for an arbitrarily long period of time. The kernel uses a scheduling policy that decides on the allocation of processing time to tasks. A scheduler that always guarantees the worst case performance (or better if possible) and also provides a deterministic behavior (with short interrupt-response delay of 100 µs) for the real-time applications is required. Recently, a new scheduler, named SCHED_DEADLINE, is introduced in the Linux mainstream kernel that allows each application to set a triple of $(runtime[ns], deadline[ns], period[ns])$, where $runtime \leq deadline \leq period$.[2] The scheduler is able to preempts the kernel code to meet the deadline and allocates the required runtime (i.e., CPU time) to each task period.

A good deadline scheduler can simplify C-RAN deployment, because Software-based Radio providing RAN in software is a real-time application that requires hard deadlines to maintain frame and subframe timing. In the C-RAN setting, the software radio application runs on a virtualized environment, where the hardware is either fully, partially, or not virtualized. Two main approaches exist to virtualization:

[2] `http://www.kernel.org/doc/Documentation/scheduler/sched-deadline.txt`

virtual machines (e.g., Linux KVM[3] and Xen[4]) or containers (e.g., LinuX Container LXC[5] and Docker[6]) as shown in Fig. 4. In a virtual machine (VM), a complete operating system (guest OS) is used with the associated overhead due to emulating virtual hardware, whereas containers use and share the OS and device drivers of the host. While VMs rely on the hypervisor to requests for CPU, memory, hard disk, network and other hardware resources, containers exploit the OS-level capabilities. Similar to VMs, containers preserve the advantage of virtualization in terms of flexibility (containerize a system or an application), resource provisioning, decoupling, management and scaling. Thus, containers are lightweight as they do not emulate a hardware layer (share the same kernel and thus application is native with respect to the host) and therefore have a smaller footprint than VMs, start up much faster, and offer near bare metal runtime performance. This comes at the expense of less isolation and greater dependency on the host kernel.

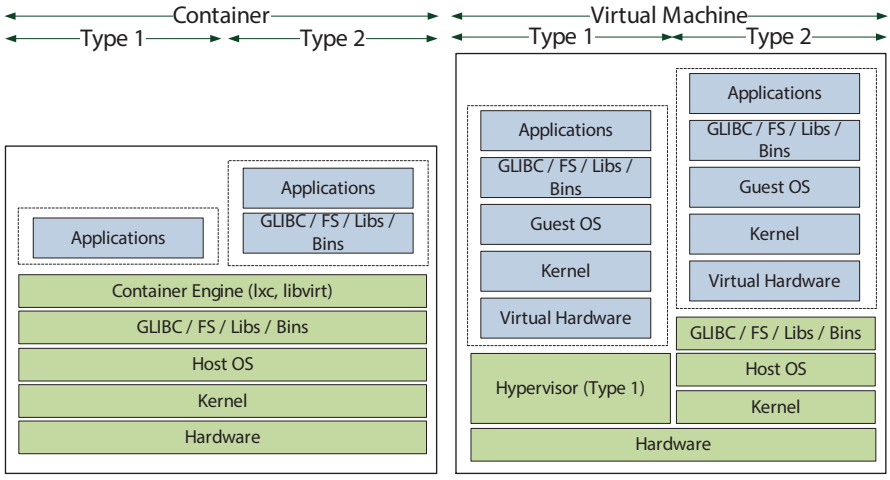


Fig. 4 Comparison of a virtual machine and container virtualized environment

Two other important aspects when targeting RAN virtualization are:

- **I/O Virtualization:** I/O access is a key for a fast access to the fronthaul interface and to the hardware accelerators that might be shared among BBUs. In hypervisor approach to virtualization (i.e., VM), IO virtualization is done through the hardware emulation layer under the control of hypervisor, where as in a container this is materialized through device mapping. Thus, direct access to hardware is easier in containers than in VMs as they operate at the host OS level. In a VM, additional techniques might be needed (e.g., para-virtualization or CPU-assisted

[3] `http://www.linux-kvm.org`
[4] `http://www.xenserver.org`
[5] `http://linuxcontainers.org`
[6] `http://www.docker.com`

virtualization) to provide a direct or fast access to the hardware. When it comes to sharing I/O resources among multiple physical/virtual servers, and in particular that of radio front-end hardware, new techniques such as single/multi root I/O virtualization (SR/MR-IOV) are required.

- **Service composition of the software radio application:** A radio application can be defined as a composition of three types of service [15], atomic service that executes a single business or technical function and is not subject to further decomposition, composed service that aggregates and combines atomic services together with orchestration logic, and support service that provides specific (often common) functions available to all types of service. An atomic service in RAN can be defined on per carrier, per layer, per function basis. For instance, a radio application could be defined as a composition of layer 1 and layer 2/3 services supported by a monitoring as a service.

4 OpenAirInterface based Evaluation of the Cloud Execution Environment

Sect. 1 gives a brief insight into various software-based implementations of BBU. This section, provides an overview of the OpenAirInterface (OAI), which is a key software component in our studies. The main advantage of OAI is that it an open-source project that implements the LTE 3GPP Release-10 standard. It includes a fully functional wireless stack with PHY, MAC, Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP) and Radio Resource Control (RRC) layers as well as Non-Access-Stratum (NAS) drivers for IPv4/IPv6 interconnection with other network services [20]. Regarding the LTE FDD, OAI provides both the uplink and downlink processing chains with SC-FDMA and OFDMA respectively (c.f., Sect. 3.2). For efficient numerical computing on the PHY, OAI uses specially optimized SIMD Intel instruction sets (i.e., MMX/SSE3/SSE4). Fig. 5 presents the OAI multi-threaded signal processing at the subframe level. As an example, the mobile air-interface of a client terminal started transmitting subframe N-1 at time (a). The decoder thread of the OAI lte-softmodem starts processing the subframe N-1 at (1) after the subframe is entirely received at time instance (b). Due to the fact that the

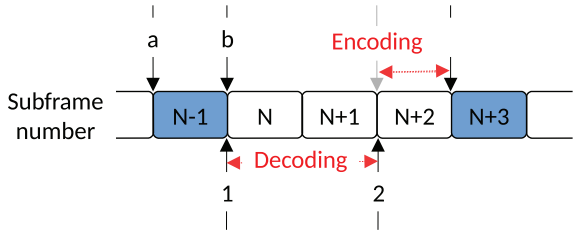


Fig. 5 Processing orders in OAI

encoding thread starting at (2) has to get input from the decoding thread to comply with HARQ retransmission scheme, the decoding thread gets at most 2 ms to finish signal processing. Again, HARQ requires data to be acknowledged at subframe N+3, therefore the encoding thread has to finish before (c) and receives at most 1 ms for processing. This description, however, does not include RRH-BBU propagation delays, which shorten the computing budget (both decoding and encoding) by a few hundred microseconds. Summing up, the OAI decoder gets twice as much time as the encoder; roughly 2 ms are allocated for decoding and 1 ms for encoding[7].

In the following subsections, we evaluate the OAI execution performance on different platforms.

4.1 Experiment Setup

Four set of different experiments are performed. The first experiment (c.f., Subsect. 4.2) analyses the impact of different x86 CPU architecture on BBU processing time, namely Intel Xeon E5-2690 v2 3 GHz (same architecture as IvyBridge), Intel SandyBridge i7-3930K at 3.20 GHz, and Intel Haswell i7-4770 3.40 GHz. The second experiment (c.f., Subsect. 4.3) shows how the BBU processing time scales with the CPU frequency. The third experiment (c.f., Subsect. 4.4) benchmarks the BBU processing time in different virtualization environments including LXC, Docker, and KVM against a physical machine (GPP). The last experiment (c.f., Subsect. 4.5) measures the I/O performance of virtual Ethernet interface through the guest-to-host round-trip time (RTT).

All the experiments are performed using the OAI *DLSCH and ULSCH simulators* designed to perform all the baseband functions of an eNB for downlink and uplink as in a real system. All the machines (hosts or guests) operate Ubuntu 14.04 with the low latency (LL) Linux kernel version 3.17, x86-64 architecture and GCC 4.7.3. To have a fair comparison, only one core is used across all the experiments with the CPU frequency scaling deactivated except for the second experiment.

The benchmarking results are obtained as a function of allocated PRBs, modulation and coding scheme (MCS), and the minimum SNR for the allocated MCS for 75% reliability across 4 rounds of HARQ. Note that the processing time of the turbo decoder depends on the number of iterations, which is channel-dependant. The choice of minimum SNR for an MCS represents the realistic behavior, and may increase number of turbo iterations and consequently causing high processing variation. Additionally, the experiments are performed at full data rate (from 0.6 Mb/s for MCS 0 to 64 Mb/s for MCS 28 in both directions) using a single user with no mobility, Single-Input and Single-Output (SISO) mode with Additive White Gaussian Noise (AWGN) channel, and 8-bit log-likelihood ratios turbo decoder. Note that if multiple users are scheduled within the same subframe on the downlink or

[7] This rule was established empirically, because in full load conditions (i.e., all PRBs allocated in the subframe; the same MCS for all PRBs) the OAI LTE FDD TX requires 2 times less processing time than the OAI LTE FDD RX.

uplink, the total processing depends on the allocated PRB and MCS, which is lower than a single user case with all PRBs and highest MCS. Thus, the single user case represents the worst case scenario.

The processing time of each signal processing module is calculated using timestamps at the beginning and at the end of each BBU function. OAI uses the `rdtsc` instruction implemented on all x86 and x64 processors to get very precise timestamps, which counts the number of CPU tics since the reset. Therefore the processing time is measured as a number of CPU tics between the beginning and end of a particular processing function divided by the CPU frequency[8].

To allow for a rigorous analysis, the total and per function BBU processing time are measured. For statistical analysis, a large number of processing_time samples (10000) are collected for each BBU function to calculate the average, median, first quantile, third quantile, minimum and maximum processing time for all the subframes on the uplink and downlink.

4.2 CPU Architecture Analysis

Fig. 6 depicts the BBU processing budget in both directions for the considered Intel x86 CPU architecture. It can be observed that the processing load increases with the increase of PRB and MCS for all CPU architectures, and that it is mainly dominated by the uplink. Furthermore, the ratio and variation of downlink processing load to that of uplink also increases with the increase of PRB and MCS. Higher performance (lower processing time) is achieved by the Haswell architecture followed by SandyBridge and Xeon. This is primarily due to the respective clock frequency (c.f., Sect. 4.3, but also due to a better vector processing and faster single threaded performance of the Haswell architecture[9]. For the Haswell architecture, the performance can be further increased by approximately a factor of two if AVX2 (256-bit Single

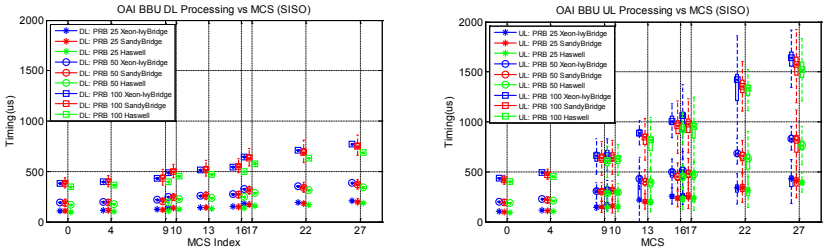


Fig. 6 BBU processing budget on the downlink (left) and uplink(right) for different CPU architecture

[8] `https://svn.eurecom.fr/openair4G/trunk/openair1/PHY/TOOLS/time_meas.h`

[9] `http://en.wikipedia.org/wiki/Haswell_(microarchitecture)`

instruction multiple data (SIMD) compared to 128-bit SIMD) instructions are used to optimize the turbo decoding and FFT processing.

4.3 CPU Frequency Analysis

Fig. 7 illustrates the total BBU processing time as a function of different CPU frequencies (1.5, 1.9, 2.3, 2.7, 3.0, and 3.4 GHz) on the Haswell architecture. The most time consuming scenario is considered with 100 PRBs and MCS 27 on both downlink and uplink. In order to perform experiments with different CPU frequencies, Linux ACPI interface and *cpufreq* tool are used to limit the CPU clock. It can be observed that the BBU processing time scales down with the increasing CPU frequency. The figure also reflects that the minimum required frequency for 1 CPU core to meet the HARQ deadline is 2.7 GHz.

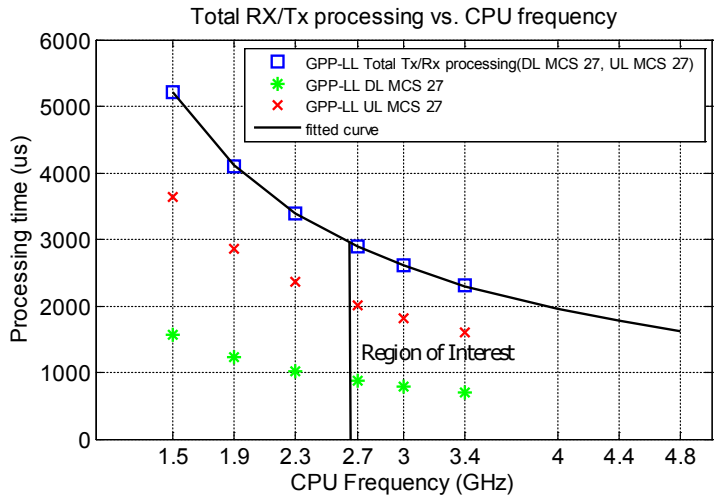


Fig. 7 Total processing time as a function of the CPU frequency

Based on the above figure, the total processing time per subframe, T_{subframe}, can be modeled as a function of the CPU frequency [1]:

$$T_{\text{subframe}}(x)\,[\mu\text{s}] = \alpha/x\,,$$

where $\alpha = 7810 \pm 15$ for the MCS of 27 in both directions, and x is CPU frequency measured in GHz.

4.4 Virtualization Technique Analysis

Fig. 8 compares the BBU processing budget of a GPP platform with different virtualized environments, namely LXC, Docker, and KVM, on the SandyBridge architecture (3.2 GHz). While on average the processing time are very close for all the considered virtualization environments, it can be observed that GPP and LXC have slightly lower processing time variations than that of DOCKER and KVM, especially when PRB and MCS increase.

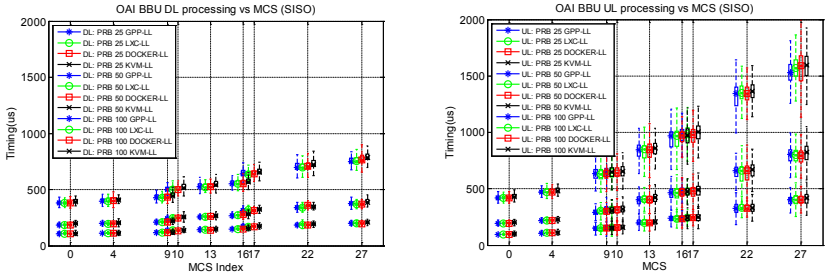


Fig. 8 BBU processing budget on the downlink (left) and uplink (right) for different virtualized environments

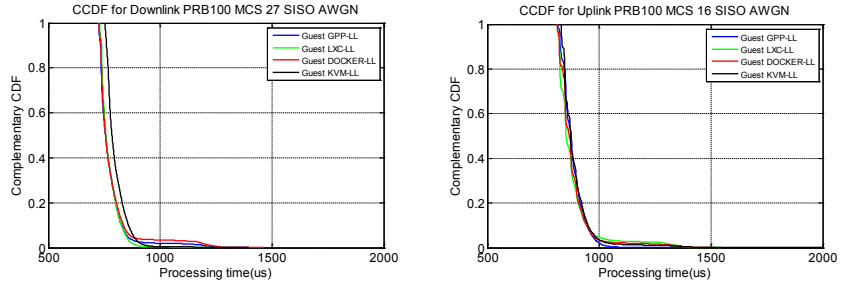


Fig. 9 BBU processing time distribution for downlink MCS 27 and uplink MCS 16 with 100 PRB

Fig. 9 depicts the Complementary Cumulative Distribution Function (CCDF) of the overall processing time for downlink MCS 27 and uplink MCS 16 with 100 PRB[10]. It can be seen that the execution time is stable for all the platforms on the uplink and downlink. The processing time for the KVM (hypervisor-based) has a longer tail and mostly skewed to longer runs due to higher variations in the non-native execution environments (caused by the host and guest OS scheduler). Higher processing variability is observed on a public cloud with unpredictable behaviors, suggesting that cares have to be taken when targeting a shared cloud infrastructure [1].

[10] The CCDF plot for a given processing time value displays the fraction of subframes with execution times exceeding this value.

4.5 I/O Performance Analysis

Generally, the fronthaul one-way-delay depends on the physical medium, technology, and the deployment scenario. However in the cloud environment, the guest-to-host interface delay (usually Ethernet) has to be also considered to minimize the access to the RRH interface. To assess such a delay, bidirectional traffics are generated for different set of packet sizes (64, 768, 2048, 4096, 8092) and inter-departure time (1, 0.8, 0.4, 0.2) between the host and LXC, Docker, and KVM guests. It can be seen from Fig. 10 that LXC and Docker are extremely efficient with 4-5 times lower round trip times. KVM has a high variations, and requires optimization to lower the interrupt response delay as well as host OS scheduling delay. The results validate the benefit of containerization for high performance networking.

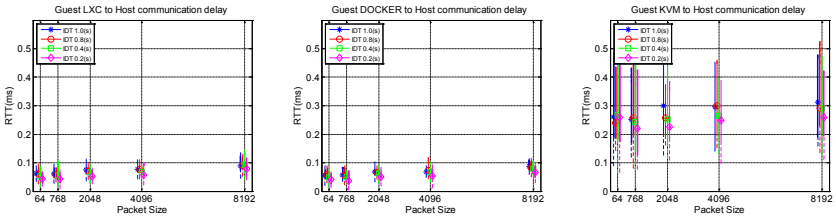


Fig. 10 Round trip time between the host and LXC, Docker, and KVM guests

4.6 New trends in C-RAN signal processing

This chapter is an attempt to analyze three critical issues in processing radio access network functions in the cloud through modeling and measurements. The results reveal new directions to enable a cloud-native radio access network that are outlined below.

New functional split between BBU and RRH: To reduce the fronthaul data rate requirements, optimal functional split is required between BBU and RRH. This depends on the deployment on the cell load, spatial multiplexing (number of UEs / RE / RRH, e.g. MU detection and CoMP), and scenario and can be dynamically assigned between RRH and BBU. In addition some non-time critical function may be performed at a remote cloud. Three principles must be considered while retaining the benefit of coordinated signal processing and transmission, namely (1) minimize the FH data rate, (2) minimize the split on the time-critical path, (3) no split of the deterministic functions. The proposed split is shown in Fig. 11. In TX chain, full PHY layer can be moved from BBU to RRH (c.f. label 4 in Fig. 2) in order to minimize the fronthaul capacity requirements as the operation of PHY layer remain deterministic as long as the L2/MAC layer provides transport blocks for all channels with

the required pre-coding information. When it comes to RX chain, moving cell processing to RRH seems promising as it halves the fronthaul capacity requirements. Additional fronthaul capacity reduction can be obtained if part of user processing can be dynamically assigned to RRH (i.e. log-likelihood ratio) depending on the number of UEs scheduled per resource elements and per RRH. The control plane protocols may be moved to a remote cloud as they are not time-critical functions.

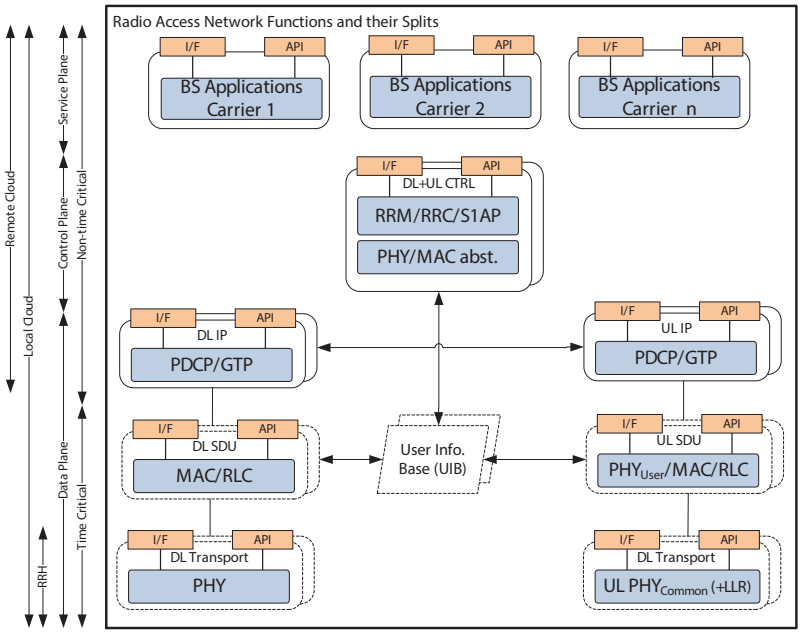


Fig. 11 Functional split between BBU and RRH

Number of CPU cores per BBU: In LTE-FDD, the total RX (Uplink) + TX (Downlink) processing should take less than 3 ms to comply with HARQ RTT, leaving 2 ms for RX and 1 ms for TX. Because TX requires the output of RX to proceed, the number of concurrent threads/cores per eNB subframe is limited to 3 even if each subframe is processed in parallel. By analyzing processing time for a 1 ms LTE subframe, 2 cores at 3 GHz are needed to handle the total BBU processing of an eNB. One processor core for the receiver, assuming 16-QAM on the uplink, and approximately 1 core for the transmitter processing with 64-QAM on the downlink, are required to meet the HARQ deadlines for a fully loaded system. Processing load is mainly dominated by uplink and increases with growing PRBs and MCSs [1, 4]. Furthermore, the ratio and variation of downlink processing load to that of uplink also grows with the increase of PRB and MCS. With the AVX2/AVX3 optimizations, the computational efficiency is expected to double and thus a full software solution would fit with an average of one x86 core per eNB. Additional processing

gain is achievable if certain time consuming functions are offloaded to a dedicated hardware accelerator.

Virtualization environment for BBU: When comparing results for different virtualization environments, the average processing times are very close making both container and hypervisor approach to RAN virtualization a feasible approach. However, the bare metal and LXC virtualization execution environments have slightly lower variations than that of DOCKER and KVM, especially with the increase of PRB and MCS increase. In addition, the I/O performance of container approach to virtualization proved to be very efficient. This suggests that fast packet processing (e.g. through DPDK) is required in hypervisor approach to minimize the packet switching time, especially for the fronthaul transport network. Due to the fact that containers are built upon modern kernel features such as `cgroups`, `namespace`, `chroot`, they share the host kernel and can benefit from the host scheduler, which is a key to meet real-time deadlines. This makes containers a cost-effective solution without compromising the performance.

5 Modeling BBU Processing Time

We confirm with the results from Sect. 4 that the total processing time increases with PRB and MCS, and that uplink processing time dominates the downlink. A remaining analysis to study the contribution of each BBU function to the overall processing time is to be done together with an accurate model, which includes the PRB and MCS as input parameters. In this study, three main BBU signal processing modules are considered as main contributors to the total processing including (de-)coding, (de-)modulation, and iFFT/FFT. For each module, the evaluate processing time is measured for different PRB, MCS, and virtualization environment on the Intel SandyBridge architecture with CPU frequency of 3.2 GHz (c.f., Fig. 12).

The plots in Fig. 12 reveals that processing time for iFFT and FFT increase only with the PRB while (de-)coding and (de-)modulation are are increasing as a function of both PRB and MCS. Moreover, the underlying processing platform adds a processing offset to each function. It can be seen from different plots in Fig. 12 that coding and decoding functions represent most of processing time on the uplink and downlink chains, and that decoding is the dominant factor. The QPSK, 16-QAM, and 64-QAM modulation schemes correspond to MCS 9, 16, and 27. The OAI implementation speeds up the processing by including highly optimized SIMD integer DSP instructions for encoding and decoding functions, such as 64-bit MMX, 128-bit SSE2/3/4. However, when operating the OAI in a hypervisor-based virtualization, some extra delay could be added if these instructions are not supported by the hardware emulation layer (c.f., Fig. 4).

From Fig. 12, we observed that the downlink and uplink processing curves have two components: dynamic processing load added to a base processing load. The dynamic processing load includes user parameters, such as (de-)coding and (de-)modulation, which is in linear relation to the allocated MCS and PRBs. Note

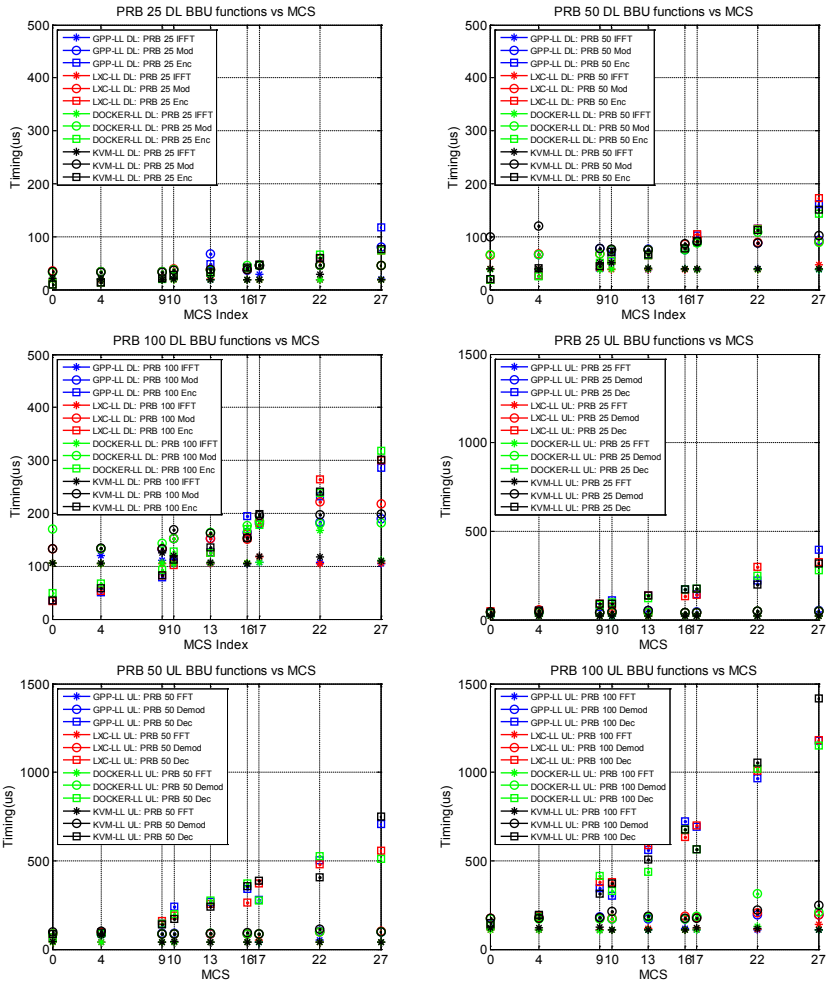
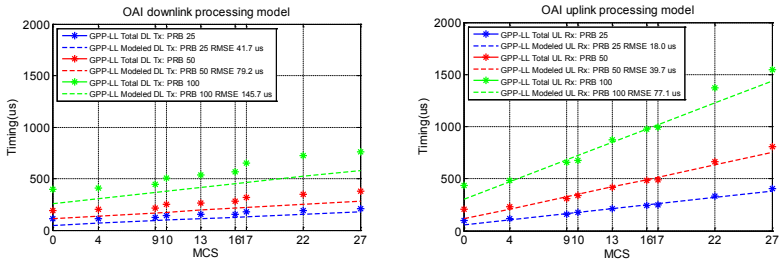


Fig. 12 Contribution of (i-)FFT, (de-)modulation, and (de-)coding to the total BBU processing for different PRB, MCS, and platforms

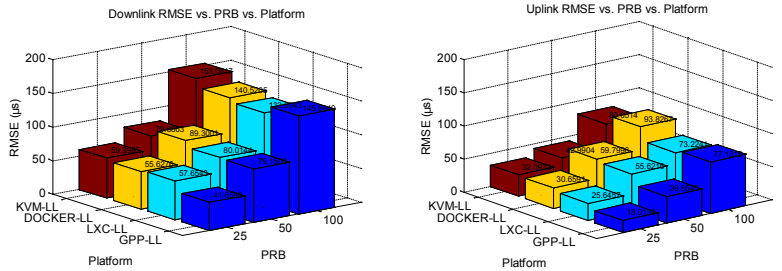
the (de-)coding functions depend also on the channel quality and SNR. The remaining user parameters, namely DCO coding, PDCCH coding, and scrambling, are modelled as the root mean square error (RMSE) for each platform. The base processing load includes iFFT/FFT cell-processing parameter for each PRB and the platform-specific parameter relative to the reference GPP platform.

The fitted curve of the total processing time for GPP is illustrated in Fig. 13(a) and the RMSE for all platforms in Fig. 13(b).

Given the results in Fig. 13, we propose a model that compute the total BBU downlink and uplink processing time for different MCS, PRB, and underlying platform, as indicated by the following formula.



(a) *Fitted curves for GPP-LL platform*



(b) *RMSE for all platforms*

Fig. 13 Modeling BBU processing time

$$T_{\text{subframe}}(x,y,w)\,[\mu\text{s}] = \underbrace{c[x]+p[w]}_{\textit{base processing}} + \underbrace{u_r[x]}_{\textit{RMSE}} + \underbrace{u_s(x,y)}_{\textit{dynamic processing}} \quad ,$$

PRB, MCS, and underlying platform are represented by the triple (x,y,w). The $p[w]$ and $c[x]$ are the base offsets for the platform and cell processing, $u_r[x]$ is the reminder of user processing, and $u_s(x,y)$ is the specific user processing that depends on the allocated PRB and MCS. We fit $u_s(x,y)$ part linearly to $a(x)y+b(x)$, where y is the input MCS, a and b are the coefficients. Table 3 and 2 provide the uplink and downlink modelling parameters of equation 3 for a SandyBridge Intel-based architecture with the CPU frequency set to 3.2 GHz. For different BBU configuration (e.g., Carrier aggregation or Multiple-Input and Multiple-Output (MIMO)), CPU architecture and frequency (c.f., Fig. 6 and 7), a and b has to be adjusted. In our setup, the achieved accuracy using our model is illustrated given an example. Given that PRB equals to 100, Downlink MCS to 27, Uplink MCS to 16, and performing within LXC platform, the estimated total processing time is 723.5 µs (111.4 + 7.4 + 12×27 + 147 + 133.7) against 755.9 µs on the downlink, and 1062.4 µs (108.8 + 13.2 + 41.9×16 + 196.8 + 73.2) against 984.9 µs on the uplink.

Table 2 Downlink processing model parameters in us

x	c	p				$u_s(x,y)$		u_c			
		GPP	LCX	DOCKER	KVM	a	b	GPP	LCX	DOCKER	KVM
25	23.81	0	5.2	2.6	3.5	4.9	24.4	41.6	57.6	55.6	59.4
50	41.98	0	5.7	9.7	13	6.3	70	79.2	80	89.3	79.7
100	111.4	0	7.4	13	21.6	12	147	145.7	133.7	140.5	153

Table 3 Uplink processing model parameters in us

x	c	p				$u_s(x,y)$		u_c			
		GPP	LCX	DOCKER	KVM	a	b	GPP	LCX	DOCKER	KVM
25	20.3	0	5.4	4.8	8.8	11.9	39.6	18	25.6	30.6	32
50	40.1	0	6	9.2	15.8	23.5	75.7	39.6	55.6	59.8	42.9
100	108.8	0	13.2	31.6	26.6	41.9	196.8	77.1	73.2	93.8	80

6 Potential Architectures of C-RAN

While from the operators' perspective such an architecture has to meet the scalability, reliability/resiliency, cost-effectiveness requirements, from the software-defined RAN, two key requirements have to be satisfied: (1) realtime deadline to maintain both protocol, frame and subframe timing, and (2) efficient and elastic computational and I/O resources (e.g. CPU, memory, networking) to perform intensive digital signal processing required, especially for different transmission schemes (beamforming, MIMO, CoMP, and Massive MIMO).

Broadly, three main choices are possible to design a RAN, each of which provide a different cost, power, performance, and flexibility trade-offs.

- **Full GPP:** where all the processing (L1/L2/L3) functions are software-defined. According to China Mobile, the power consumption of the OAI full GPP LTE modem is around 70 W per carrier [7].
- **Accelerated:** where certain computationally-intensive functions, such as turbo decoding and encryption/decryption, are offloaded to a dedicated hardware such as an FPGA, GPU, and/or DSP. The remaining functions are software-defined and performed on the host/guest OS. In this case, the power consumption can be reduced to around 13–18 W per carrier.
- **System-on-Chip:** where the entire Layer 1 is performed in a dedicated hardware (e.g. a SoC), and the layer 2 functions are run on the host/guest OS. This can reduce the power consumption to around 8 W per carrier.

As shown in Fig. 14, the hardware platform can either be a full GPP or a hybrid. In the later case, all or part of the L1 functions might be split and placed either locally at the BBU cloud infrastructure or remotely at the RRH unit. In either cases, some of the L1 functions might be offloaded to dedicated accelerator. It can be seen that a pool of base station (BS) can be virtualized inside the same (or different) cloud

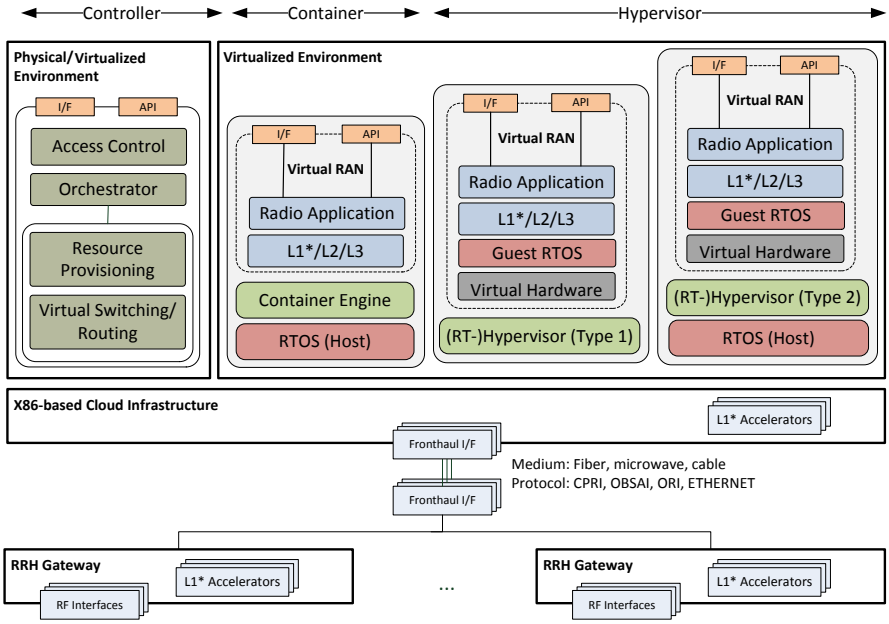


Fig. 14 Potential C-RAN architectures

infrastructure and mapped to RF interface within the RRH gateway. A virtualized RAN (vRAN) can communicate with core networks (CN) through a dedicated interface (e.g. S1 in LTE), and with each other directly through another interface (e.g. X2 in LTE). In addition, vRAN can rely on the same cloud infrastructure to provide localized edge service such as content caching and positioning, and network APIs to interact with the access and core networks [23]. Different service compositions and chaining can be considered, ranging from all-in-one software radio application virtualization to per carrier, per layer or per function virtualization [10]. The virtualization technology can either be based on container or a hypervisor, under the control of a cloud OS, managing the life-cycle of a composite service (orchestrator logic) as well as provisioning the required resources dynamically.

Nevertheless, a full GPP approach to RAN brings the required flexibility in splitting, chaining, and placement of RAN functions while meeting the realtime deadlines along with the following principles [11, 19, 26]:

- **NFV and Micro service Architecture:** breaks down the network into a set of horizontal functions that can be bundled together, assigned with target performance parameters, mapped onto the infrastructure resources (physical or virtual), and finally delivered as a service. This implies that micro virtualized network functions (VNF) are stateless (services should be designed to maximize statelessness even if that means deferring state management elsewhere) and composable (services may compose others, allowing logic to be represented at different levels of granularity; this allows for re-usability and the creation of service abstraction

layers and/or platforms). In addition, they can be autonomous (the logic governed by a service resides within an explicit boundary), loosely coupled (dependencies between the underlying logic of a service and its consumers are limited to conformance of the service contract), reusable (whether immediate reuse opportunities exist, services are designed to support potential reuse), and discoverable (services should allow their descriptions to be discovered and understood by (possibly) humans, service requesters, and service discovery that may be able to make use of their logic).[11]

- **Scalability:** monitors the RAN events (e.g. workload variations, optimization, relocation, or upgrade) and automatically provision resources without any degradations in the required/agreed network performance (scale out/in).
- **Reliability:** shares the RAN contexts across multiple replicated RAN services to keep the required redundancy, and distributes the loads among them.
- **Placement:** optimizes the cost and/or performance by locating the RAN services at the specific area subjected to performance, cost, and availability of the RF front-end and cloud resources.
- **Multi-tenancy:** shares the available spectrum, radio, and/or infrastructure resources across multiple tenants (MNOs, MVNOs) of the same cloud provider,
- **Real-time Service:** allows to open the RAN edge service environment to authorized third-parties to rapidly deploy innovative application and service endpoints. It provides a direct access to real-time radio information for low-latency and high-bandwidth service deployed at the network edge [23]. The Real-time Service shall be automatically configurable to rapidly adjust to varying requirements and utilization of the cloud environment (c.f., Sect. 7).

Table 4 compares the requirements of general-purpose cloud application against the cloud RAN.

Table 4 General purpose cloud applications vs. C-RAN

Application	GP-Cloud Computing	Cloud-RAN
Data rate	Mb/s, bursty	Gb/s, stream
Latency / Jitter	Tens of ms	$<$1 ms, jitter in ns
Lifetime of data	Long	Extremely short
Number of clients	Millions	Thousands – Millions (IoT)
Scalability	High (Micro-service and stateless)	Low
Reliability	Redundancy / load balancing	Redundancy, offloading, load balancing
Placement	Depends on the cost and performance	Specific areas with Radio Frontend. Depends on the cost and performance.
Timescale (operation, recovery)	Non-realtime	Realtime

[11] Micro-service architecture is in opposition to the so-called "monolithic" architecture where all functionality is offered by a single logical executable, see http://martinfowler.com/articles/microservices.html. It has to be noted that the micro-service architecture supports the ETSI NFV architecture [10], where each VNF can be seen as a service.

7 Cloud architecture for the LTE RAN

In cloudified C-RAN, the BBU becomes software-based, hence the concept of C-RAN cloudification, in which the BBU life-cycle is managed through a cloud operating system and run over the cloud infrastructure, is sound and may become an important business connection between mobile telephony operators and cloud providers. Generally, a cloud provider delivers their (publicly available) service in the form of three different flavors, namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [22], however, in the scope of this work, we put a particular focus on the IaaS-based systems. In the IaaS mode, a cloud operator delivers a resource as a so called Virtual Machines (VM), which comes with processing power, RAM, and storage (optionally other services too) accessible through the Internet. A user operating a VM system has the experience of remote access to an ordinary computer, which is accomplished through a virtualization procedure. Virtualization, which is enabled through a special software layer called a hypervisor, allows us to simultaneously run many VMs (instances) on a single physical cloud server.

When a Cloud-RAN is deployed on a public cloud, then multiple instances compete for the same infrastructure (e.g., computing power, storage, RAM). Hence, in an ordinary setup, we cannot be provided with deadlines for required real-time computing. It is therefore necessary to work out new organizational models of publicly available data centers as currently cloud providers do not offer real-time support in their virtual environment. Here, we briefly present our efforts to allow for real-time support in IaaS clouds. We start with an OpenStack installation of a well established cloud orchestration system. OpenStack looks after computing power, storage, and networking resources of the cloud infrastructure (server pools) and orchestrates the execution of VMs including (re-) configuration upon initialization or a user request.

The response time of the full-virtualization KVM-based OpenStack system did not fully satisfy our requirements due to unpredictable processing delays. We therefore decided to modify the host system (cloud compute servers) by installing a low latency kernel and replace the default virtualization technique with the Linux

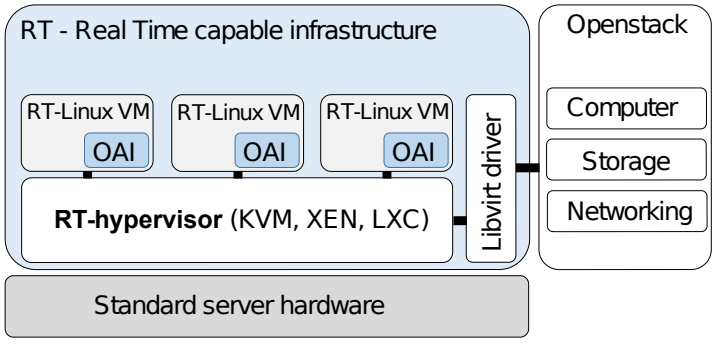


Fig. 15 OpenStack management architecture

Containers (LXC) plugin of OpenStack (c.f., Fig. 15). LXC is Operating System Level virtualization providing high performance as all CPU instructions are natively executed. Moreover, it allows us for the real-time process prioritization on the guest operating system (VM). In our case, the lte-softmodem OAI application is prioritized real-time within the LXC container using the SCHED_DEADLINE or SCHED_FIFO schedulers provided by the low latency Linux kernel. Good performance of RAN satisfied through LXCs could have a big impact on the security of cloud infrastructure as LXCs do not provide a good separation of VMs from physical servers and should be avoided in the case of less time-critical applications such as EPC, HSS, etc. Therefore a heterogeneous cloud infrastructure maintaining both real-time (e.g., LXC-based) and general-purpose (e.g., KVM-based) computing regions[12] can properly serve purposed of the MNO. The region's workload is not know in advance, therefore the cloud provider has to be provided with flexibility to on-demand re-program the infrastructure when required, e.g., to activate a larger number of real-time compute nodes for RAN if the current workload exceeds the capacity of the real-time infrastructure, but the overall cloud-global capacity can still withhold the workload when reconfigured (i.e., adapting the size of real-time and non-real-time regions). To this end, we can employ JUJU[13] and Metal As a Service (MAAS)[14] to program physical cloud compute nodes and provide the concept of *programmable cloud* that dynamically adjusts the cloud region size.

8 C-RAN Prototype

In this section, we demonstrate a RANaaS proof-of-concept (PoC) (c.f., the architecture presented in Fig. 16). Our cloud infrastructure consists of the OpenStack orchestrating software with appropriately designed compute servers. Normally, OpenStack manages large pools of resources, but in our example, it controls a local nano data-center developed to execute RANaaS. Our compute node is deployed on a commodity computer running Ubuntu 14.04 with the low latency Linux kernel version 3.17, while the OpenStack installation uses the LXC plugin on compute nodes to support LXC virtualization. For cloud orchestration OpenStack developed a Heat module that provides a human- and machine-accessible service for the management of the entire life-cycle of a virtual infrastructure and applications. This orchestration engine relies on text-based templates, called Heat Orchestration Templates (HoTs), to manage multiple composite cloud applications and organize them as a stack of virtualized entities (e.g. network, LXCs) called the Heat stack.

Following the LTE protocol stack[15], our demonstration has to instantiate an E-UTRAN part, evolved packet core (EPC), and home subscriber server (HSS). The

[12] A cloud region is an organizational unit of the cloud containing a pool of cloud workers with specific properties such as the same configuration or geographical location.

[13] `http://www.ubuntu.com/cloud/tools/juju`

[14] `http://www.ubuntu.com/cloud/tools/maas`

[15] Here, the work stack does not refer to Heat and should be understood as a protocol stack.

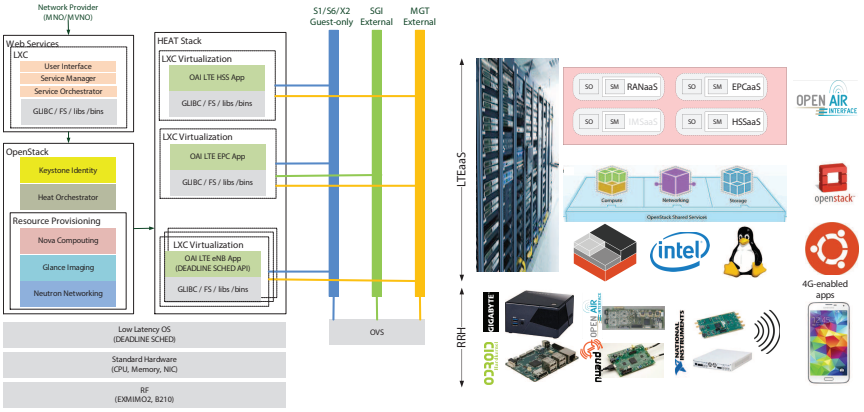


Fig. 16 RANaaS prototype (left) and hardware setup (right)

EPC consists of a mobility management entity (MME) as well as a Serving and Packet data network Gateway (S/P-GW). Mobile Operators (e.g., MNO, MVNO) use the User Interface (UI) to manage the life-cycle of RANaaS. The Service Manager (SM) component receives user queries from the UI and manages the cloud execution through the Service Orchestrator (SO) component, which leverages the use of the Heat API for cloud orchestration.

In the demonstrated scenario, a HoT file describes the whole virtual infrastructure including the LTE network elements as well as the required network setup tailored to a specific business case. Using the HoT template, Heat manages the service instantiation of every required LTE network function implemented in OAI spread among multiple VMs. As we previously explained, RANaaS has strict latency and timing requirements to achieve a required LTE frame/subframe timing. To this end, we use the SCHED_DEALINE Linux scheduler to allocates the requested runtime (i.e., CPU time) upon every sub-frame to meet the deadline.

Listing 1 presents an example HoT file, which instantiates the RAN as a Service (RANaaS) stack. The template is provided to Heat, which automatically spawns a VM using an arbitrary image previously uploaded to OpenStack (enb-1 provides the installation of the OAI lte-softmodem), attaches the network (e.g., PUBLIC_NET defined in OpenStack), and pre-configures the VM through a bash script provided as user-data. Other VMs illustrated in Fig. 16 could be instantiated in a similar way. Heat allows us to use previously defined resource attributes. For instance, if an eNB requires the address of an HSS, one can reference to it through the *get_attr* Heat function, i.e., get_attr: [EPC, first_address], where the EPC is a previously defined resource and first_address is the attribute of the resource (an IP address of the first interface). Consequently, the whole LTE as a Service (LTEaaS) containing an HSS, EPC, and eNBs can be instantiated from a single HoT file with one request to Heat.

Listing 1 The LTEaaS HoT file

```
heat_template_version: 2013-05-23
description: LTEaaS
```

```
parameters:
  key_name:
    type: string
    description: >
      Name of a KeyPair to enable
      SSH access to the instance
    default : cloudkey
resources:
  HSS: ...
  MME: ...
  S+P-GW: ...
  eNB:
   type: OS::Nova::Server
   properties:
    image: enb-1
    flavor: eNB.large
    key_name: cloudkey
    networks: [{ network: PUBLIC_NET }]
    user_data:
     str_replace:
       template: |
         #!/bin/bash
         MY_IP=`ip addr show dev eth0 | \
         awk -F' [ /]*' '/inet /{print $3}'`
         sed -i 's#MY_IP_ADDRESS_REPLACE#'$MY_IP'#g' \
         enb.band7.tm1.usrpb210.conf
         sed -i 's#MME_IP_ADDRESS_REPLACE#'$MME_IP'#g' \
         enb.band7.tm1.usrpb210.conf
         ./build_oai.bash --eNB -w USRP > /tmp/oai.log
         ./lte-softmodem -O \
         enb.band7.tm1.usrpb210.conf
       params:
         $MME_IP: { get_param: mme_ip }
```

LTEaaS describes the service life-cycle of an on-demand, elastic, pay as you go RAN that is running on top of the cloud infrastructure. We believe that life-cycle management is a key for successful adoption and deployment of C-RAN and related services (e.g. MVNO as a Service). It is a process of network design, deployment, resource provisioning, operation and runtime management, and disposal as shown in Fig. 17. In this figure, SM/SO indicates Service Manager/Service Orchestrator, while Keystone and Heat Orchestrator are OpenStack services; the box OpenStack refers to other OpenStack services such as Compute, Storage, Networking, etc. With the help of the UI, the MNO first designs the HoT and spawns other actions such as Deploy, Provision, Manage, and Disposal, which are then managed by the SM/SO that directly communicates with the Heat Orchestrator.

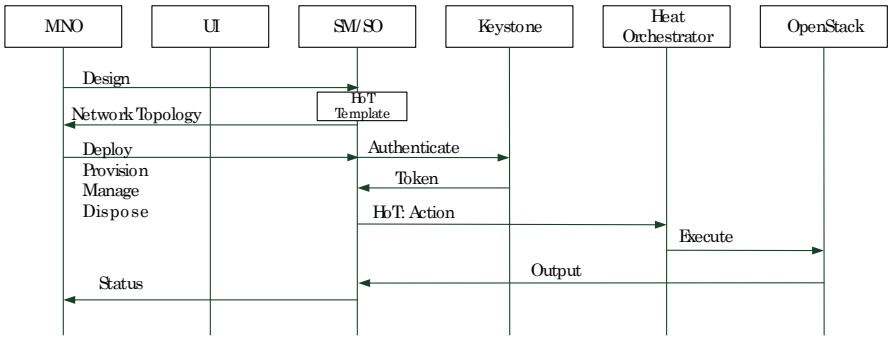


Fig. 17 The RANaaS life-cycle management

8.1 LTEaaS: eNB resource provisioning

This section presents the performance study of the time-critical eNB application running in the LTE as a Service (LTEaaS) architecture. We conducted several experiments particularly relying on the LTE eNB and UE implementation [19, 21] using the OAI platform that implements standard compliant 3GPP protocol stack. We deploy the LTEaaS on the cloud center as shown in Fig. 16 and as described above. The parameters of the real-time OAI eNB are the following: FDD 10 MHz channel bandwidth (50 PRBs) in SISO mode over band 7. MCS are fixed to 26 in downlink and 16 in uplink to produce high processing load. The eNB sends grants to the UE for UL transmission only in downlink SF #0, 1, 2 and 3. Useful UL SFs are then SF # 4, 5, 6, 7. The others UL SFs can possibly be used for HARQ retransmissions.

We compare the feasibility and performance of the proposed LTEaaS architecture using two different linux OS schedulers: namely SCHED_FIFO (not SCHED_OTHER) or SCHED_DEADLINE (low-latency policy) while running the eNB in LXC containers. Linux cgroups and cpu-sets are used to control the CPUs cores accessible to the container. Bandrich C500, a commercial LTE UE dongle is connected to the instantiated eNB using the classical LTE over-the-air attachment procedure. We measure the uplink goodput (data-rate over a period of a second) for each scheduler applying to the eNB and for different numbers of available CPU cores (CPU is i7-3930k 3.2 GHz with hyper-threading and turbo mode disabled). The measurement lasts 120 seconds while iperf is generating UDP traffic between the UE and a local server connected to the EPC.

Fig. 18 and Fig. 19 present the complementary cumulative distribution function of the running time of each RX thread at the eNB, when using SCHED_FIFO or SCHED_DEADLINE with 3 or 2 CPU cores available. Each of these threads corresponds to a specific UL SF. It should be noted that those threads are not the only ones running, as there are also a management thread and a TX thread for each DL SF. In Fig. 18, the value *(1)* of 0.65 ms indicates the BBU and protocol processing time of a fully loaded SF (most of the time for SFs #4, 5, 6 and 7 shown as solid lines in the figure, from time to time corresponding to HARQ retransmission for

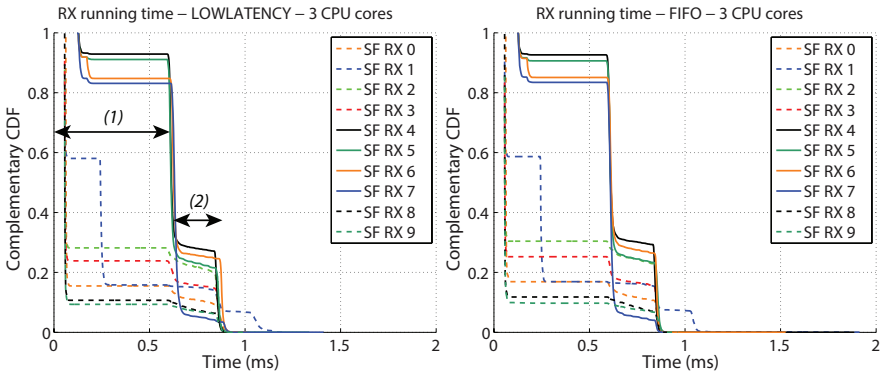


Fig. 18 OAI LTE soft-modem running on 3 CPU cores

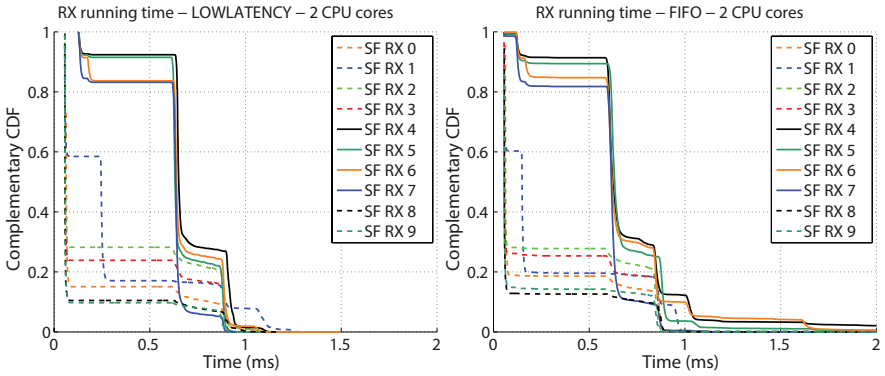


Fig. 19 OAI LTE soft-modem running on 2 CPU cores

the other subframes), while the increase *(2)* of 0.2 ms is related to the RLC packet reassembly event that also triggers the PDCP integrity check.

Both schedulers behave similarly in this scenario when 3 CPU cores are available as shown on Fig. 18. There is no missed deadline in either case, meaning that the processing power is sufficient to directly execute the required threads in their constrained time (2 ms after receiving RF samples for RX, and 1 ms for generating the RF samples for TX).

When only 2 CPU cores are available, the results change for the FIFO scheduler as shown in Fig. 19. Using the low-latency scheduler, the results are similar than with 3 CPU cores and there is no missed subframes. But using the FIFO scheduler, it can be seen that the SF processing time is sometimes larger than 2 ms as indicated by the tails of the curves and during the 120 seconds transfer, there are 708 missed SFs. It represents a loss of 0.6% of the SFs due to late scheduling. Fig. 20 shows that while this loss might seem small, it impacts the average uplink goodput with a more than 6% decrease. The DL channel should present a similar behavior when full loaded.

The results of this experiment are in line with what was presented throughout this chapter and underlines that adequate hardware resources provisioning (programmable cloud concept) and scheduling are mandatory to achieve high performances in cloud architectures.

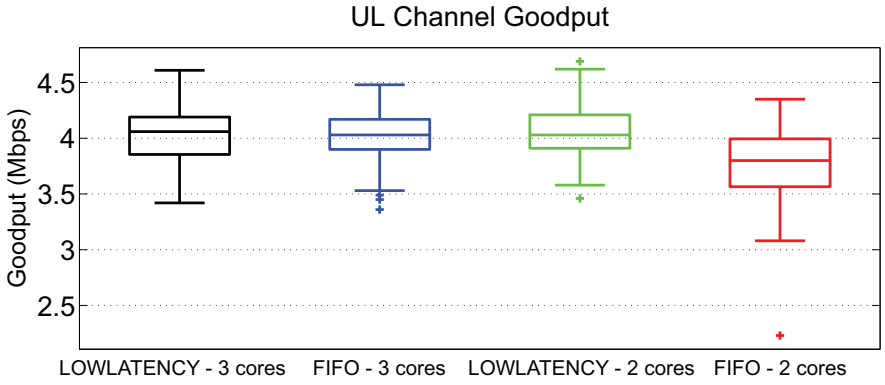


Fig. 20 Impact of the execution environment on the LTE soft-modem uplink performance

9 Conclusions

In this chapter, we have studied and analyzed several important aspects of the radio access network cloudification. First, we have presented C-RAN as a cost effective, scalable, energy efficient, and flexible service for MNOs and MVNOs. Second, current requirements of the LTE standard were translated in terms of various requirements for C-RAN including fronthaul properties, processing software latencies, and real-time capabilities of the operating system. Third, by using OAI, we have evaluated C-RAN in various execution environments such as dedicated Linux, LXC, and KVM. We drew new conclusions on the RRH-based BBU offloading and virtualization environment for C-RAN; we highlighted advantages of containerization over virtualization in C-RAN provisioning. Fourth, we described the properties of RANaaS focusing on the radio-processing organization and micro-service, multi-tenant architecture; we pointed out main differences between RANaaS and general purpose cloud computing. Finally, we described the cloud architecture for LTE RAN and focused on the C-RAN prototype and its life-cycle management.

References

1. Alyafawi, I., Schiller, E., Braun, T., Dimitrova, D., Gomes, A., Nikaein, N.: Critical issues of centralized and cloudified lte-fdd radio access networks. In: Communications (ICC), 2015 IEEE International Conference on, pp. 5523–5528 (2015). DOI 10.1109/ICC.2015.7249202
2. Amari LTE 100, a Software LTE Base Station on a PC. http://www.amarisoft.com/
3. Berardinelli, G., Ruiz de Temino, L., Frattasi, S., Rahman, M., Mogensen, P.: Ofdma vs. sc-fdma: performance comparison in local area imt-a scenarios. Wireless Communications, IEEE **15**(5), 64–72 (2008). DOI 10.1109/MWC.2008.4653134
4. Bhaumik, S., Chandrabose, S.P., Jataprolu, M.K., Kumar, G., Muralidhar, A., Polakos, P., Srinivasan, V., Woo, T.: Cloudiq: A framework for processing base stations in a data center. In: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, Mobicom '12, pp. 125–136. ACM, New York, NY, USA (2012). DOI 10.1145/2348543.2348561. URL http://doi.acm.org/10.1145/2348543.2348561
5. Chanclou, P., Pizzinat, A., Le Clech, F., Reedeker, T.L., Lagadec, Y., Saliou, F., Le Guyader, B., Guillo, L., Deniel, Q., Gosselin, S., Le, S., Diallo, T., Brenot, R., Lelarge, F., Marazzi, L., Parolari, P., Martinelli, M., O'Dull, S., Gebrewold, S., Hillerkuss, D., Leuthold, J., Gavioli, G., Galli, P.: Optical fiber solution for mobile fronthaul to achieve cloud radio access network. In: Future Network and Mobile Summit (FutureNetworkSummit), 2013, pp. 1–11 (2013)
6. Checko, A., Christiansen, H., Yan, Y., Scolari, L., Kardaras, G., Berger, M., Dittmann, L.: Cloud ran for mobile networks #x2014;a technology overview. Communications Surveys Tutorials, IEEE **17**(1), 405–426 (2015). DOI 10.1109/COMST.2014.2355255
7. China Mobile Research Institute: C-RAN White Paper: The Road Towards Green RAN. http://labs.chinamobile.com/cran (2013)
8. Costa-Perez, X., Swetina, J., Guo, T., Mahindra, R., Rangarajan, S.: Radio access network virtualization for future mobile carrier networks. Communications Magazine, IEEE **51**(7), 27–35 (2013). DOI 10.1109/MCOM.2013.6553675
9. Dahlman, E., Parkvall, S., Skold, J.: 4G LTE/LTE-Advanced for Mobile Broadband, 1st edn. Academic Press (2011)
10. ETSI: Network Functions Virtualisation (NFV), White paper. Tech. rep., ETSI (2014)
11. EU FP7 Mobile Cloud Networking public deliverable D2.5: Final Overall Architecture Definition. Tech. rep., EU (2015)
12. EURECOM: Open Air Interface. http://www.openairinterface.org/
13. Haberland, B., Derakhshan, F., Grob-Lipski, H., Klotsche, R., Rehm, W., Schefczik, P., Soellner, M.: Radio Base Stations in the Cloud. Bell Labs Technical Journal **18**(1), 129–152 (2013)
14. Interworking and JOINt Design of an Open Access and Backhaul Network Architecture for Small Cells based on Cloud Networks (iJOIN): an FP7 STREP project co-funded by the European Commission. http://www.ict-ijoin.eu/
15. Mobile Cloud Networking (MCN): an FP7 IP project co-funded by the European Commission. http://www.mobile-cloud-networking.eu.
16. NGMN: Further Study on Critical C-RAN Technologies (v0.6). Tech. rep., The Next Generation Mobile Networks (NGMN) Alliance (2013)
17. NGMN: Suggestions on Potential Solutions to C-RAN by NGMN Alliance. Tech. rep., The Next Generation Mobile Networks (NGMN) Alliance (2013)
18. Nikaein, N.: Processing radio access network functions in the cloud: Critical issues and modeling. In: Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services, MCS'15, pp. 36–43. ACM, New York, NY, USA (2015)
19. Nikaein, N., Knopp, R., Gauthier, L., Schiller, E., Braun, T., Pichon, D., Bonnet, C., Kaltenberger, F., Nussbaum, D.: Demo – Closer to Cloud-RAN: RAN as a Service. Proceedings of ACM MOBICOM Demonstrations (2015)
20. Nikaein, N., Knopp, R., Kaltenberger, F., Gauthier, L., Bonnet, C., Nussbaum, D., Ghaddab, R.: OpenAirInterface 4G: an open LTE network in a PC. In: Proceedings of the 20th Annual International Conference on Mobile Computing and Networking, MobiCom '14, pp. 305–308. ACM, New York, NY, USA (2014). DOI 10.1145/2639108.2641745

21. Nikaein, N., Schiller, E., Favraud, R., Katsalis, K., Stavropoulos, D., Alyafawi, I., Zhao, Z., Braun, T., Korakis, T.: Network store: Exploring slicing in future 5g networks. In: Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture, MobiArch'15, pp. 8–13. ACM, New York, NY, USA (2015)
22. Oracle: Oracle Cloud, Enterprise-Grade Cloud Solutions: SaaS, PaaS, and IaaS. https://cloud.oracle.com/home
23. Patel, M., Joubert, J., Ramos, J.R., Sprecher, N., Abeta, S., Neal, A.: Mobile-Edge Computing. Tech. rep., ETSI, white paper (2014)
24. Scalable and Adaptive Internet Solutions (SAIL): an FP7 IP project co-funded by the European Commission. http://www.sail-project.eu
25. Schooler, R.: Transforming Networks with NFV & SDN. Intel Architecture Group (2013)
26. Wilder, B.: Cloud Architecture Patterns. O'Reilly (2012)
27. Wubben, D., Rost, P., Bartelt, J., Lalam, M., Savin, V., Gorgoglione, M., Dekorsy, A., Fettweis, G.: Benefits and impact of cloud computing on 5g signal processing: Flexible centralization through cloud-ran. Signal Processing Magazine, IEEE **31**(6), 35–44 (2014). DOI 10.1109/MSP.2014.2334952