VNF modeling towards the Cloud-RAN implementation

Veronica Karina Quintuna Rodriguez, Fabrice Guillemin

VNF modeling towards the Cloud-RAN implementation

Veronica Quintuna Rodriguez and Fabrice Guillemin
Orange Labs, 2 Avenue Pierre Marzin, 22300 Lannion, France
{veronicakarina.quintunarodriguez,fabrice.guillemin}@orange.com

***Abstract*—We address in this paper the problem of modeling Virtual Network Functions (VNFs) in order to devise execution principles. We specifically represent a VNF as a chain of sub-functions or micro-services and investigate its processing on a multi-core platform. Instead of reserving dedicated resources to a VNF, we specifically examine the gain that can be obtained by resource pooling, namely a pool of cores which is shared by active VNFs. The driving use case of this study is the virtualization and the cloudification of Radio Access Network functions, in particular Base Band Unit (BBU) functions. We validate this approach by simulation and show that an adequate threading model can reduce the VNF runtime. Obtained results open the door towards the Cloud-RAN implementation since some BBU tasks might be hosted in geographically distant data centers.**

Keywords: NFV, VNF, Cloud-RAN, BBU, channel coding, multi-core, global scheduling.

I. Introduction

Network Function Virtualization (NFV) is a promising technique [1] which provides network operators with high flexibility and agility to instantiate network functions on demand, e.g., control functions such as IP Multimedia Subsystem (IMS) or Evolved Packet Core (EPC). Instead of implementing such functions on expensive dedicated hardware, Virtualized Network Functions (VNFs) can be run on general purpose servers, ideally by using open-source software. Moreover, the possibility of instantiating on the fly control functions and configuring network elements via Software Defined Network (SDN) protocols (e.g., OpenFlow) allows network operators to create customized networks on demand, and thus, to explore new businesses offering innovative services.

Two current use cases investigated by network operators are virtual EPC (vEPC) and Radio Access Network as a Service (RANaaS). The first case is targeted at companies wishing to deploy private mobile networks by using the spectrum and the access network of a network operator. A vEPC can then be instantiated by a network operator and configured by a company according to own needs (firewall, private numbering, user's profile, etc.). RANaaS is one step further by offering to a client the possibility of deploying his own radio access network, even if the radio spectrum is still under the control of an operator. In this paper, we shall pay special attention to the RAN and explore how a RAN can be handled as a VNF by a network operator.

With regard to the implementation of NFV, one possibility is to instantiate a VNF as a software package on a Virtual Machine (VM) or a container with dedicated resources in terms of computing and storage. This is typically the case of current open-source software packages for EPC and Radio Access Network (RAN) functions, namely Open Air Interface (OAI)-cn and OAI-eNB, respectively [2], [3].

The basic reason for implementing a VNF on a single VM or more generally on the same data center in the case of multiple VMs is that some network functions cannot easily be dissociated (e.g., P-Gateway and MME in the case of EPC). Even when the functional decomposition of a VNF is possible, some performance constraints can limit the separation of some elements (e.g. in the case of RAN, the time budget for the TX base-band process is 1 millisecond).

On the other hand, a dedicated resource allocation to each VNF is extremely expensive since resources keep assigned even when the VNF is not active and any resource sharing model cannot be applied. In addition, beyond unused resources, the energy consumption aspects of NFV are also of prime interest. If NFV enables a split between hardware and software-based network functions, supporting VNFs on not optimized hardware may lead to additional costs in terms of energy. This is a major issue for network operators and more generally for the whole of the IT industry in the aim of reducing the energy footprint in the next future. Hence, any attempt to save energy is utmost important. In our understanding, resource pooling is one possibility to achieve energy efficiency.

In order to reach efficient resource utilization, we rely in this paper on the decomposition of a VNF into a set of sub-functions which can be executed independently of each other as much as possible. This is consistent with the notion of micro-services [4] as well as with the network slicing concept [5]. In this perspective, a VNF is considered as a service chain that can be disseminated throughout the network. However, a major challenge is to place the various components on non co-localized data centers while meeting performance and resilience objectives [6].

The present work investigates the implementation of a VNF in the form of a chain of components to be executed on a multi-core platform. Instead of dedicating resources to each VNF, we assume that VNFs are served by one high capacity server, i.e., the global capacity is not split into virtual servers. This is motivated by the general results given by Kleinrock, chapter 5 [7], where it is shown that it is better to have jobs in

a big queue instead of splitting jobs into sub-classes processed by dedicated servers.

In order to illustrate the implementation of VNFs, we consider the case of Cloud-RAN, which is a proposed architecture for future mobile networks [8], [9], [10]. The RAN functions are today hosted by eNodeBs (eNBs) but in the next future it is very likely that those functions can be grouped higher in the back-haul network, thus, dissociating Radio Remote Head (RRH) and Base Band Unit (BBU). In this case, centralized BBUs are executed on a large pool of computing resources that accomplish the virtualized RAN functions. Improving the response time in the execution of BBU functions is highly important because, beyond the hard real-time constraints of LTE, the runtime of some BBU functions might not have a deterministic behavior.

The goal of this work is to determine the amount of time which is possible to gain when applying parallel processing techniques in the execution of BBU functions. The resulting gain shall allow the separation of the BBU-pool and antennas (RRHs). In addition, the performance of virtual BBUs in terms of latency (i.e., runtime) determines the number of cells (MIMO/SISO) that can be hosted in the BBU-pool, as well as the required back-haul capacity.

In order to reduce the BBU runtime, we present a VNF threading model for the virtualization of BBU down-link functions. The proposed model defines threads at a finer granularity and maximizes the number of processes that can be executed in parallel. As a proof of concept, we demonstrate by simulation the gain in terms of latency which is a step towards the RAN cloudification. The simulation scenario assumes urban macro-cells with heterogeneous traffic conditions for multiple UEs.

The major goal of the present study is to investigate which distance can be introduced between BBU and RRH functions and how many BBUs can be grouped together. The ultimate goal is to determine where to implement a BBU hostel and more generally a Central Office (Central Office (CO)) depending on the target radio coverage. This actually amounts to cloudifying RAN functions. For this purpose, we introduce general modeling principles for VNFs.

This paper is organized as follows: In Section II, we present the VNF model using the case of BBU functions. Implementation guidelines of a virtualized BBU are described in Section III. The scheduling strategy is detailed in Section IV. In Section V, we present the simulation results for BBU DL functions and the achieved gain in terms of latency. Concluding remarks are presented in Section VI.

II. VNF MODELING

A. Model Description

VNF modeling needs to be based on the general philosophy of high performance computing architectures which maintains thousands of threads in parallel [11]. Due to the intrinsic nature of a network function only some tasks can run independently one of each other, thus their parallel execution is not always possible.

In this context, we consider a VNF as a suite of executable processes. Each of them executes a specific sub-function in such a way that the global VNF can be realized. Some processes can be started only when the output of the previous one is available. Two such processes are necessarily executed in series. On the contrary, some tasks can run in parallel even if the subsequent task can be executed only when the output of all parallel processes is available.

This leads to the representation of a VNF as depicted in Figure 1. The tasks t_{n-1} and t_n are executed in series but t_{n+1} is composed of m sub-tasks $t_{n+1,1}, \ldots, t_{n+1,m}$, which can be executed in parallel. Task t_{n+2} can be executed only when all the tasks $t_{n+1,j}$, $j = 1, \ldots, m$ are completed.

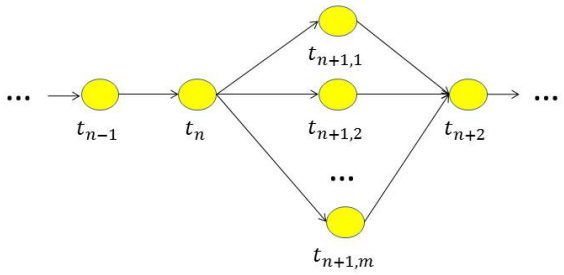


Fig. 1. VNF modeling

With regard to the execution of VNFs on a Graphics Processing Unit (GPU)/Central Processing Unit (CPU)-based server, we can assume that the computing capacity is shared among the various active VNFs. The server can be composed of a single or several cores. In the case of a multi-core platform, we assume that cores are shared among the various VNFs. The scheduler contained in the kernel of the operating system is in charge of allocating the capacities of the cores.

Each task of a VNF can be considered as a job requiring a certain amount of service but upon service completion, a job can simply leave the system or give rise to another job with another service requirement, or to several jobs each with a certain amount of service, which can give rise to another job only when all the parallel jobs are completed.

We can hence easily see that the presence of batches can have a significant impact on those tasks which are executed in series, if the server capacity is shared among all active tasks. A VNF can momentarily receive more processing than other VNFs. Moreover, if the subsequent task of a VNF can start only when all the sub-tasks of a batch are all completed, then badly scheduling tasks can introduce significant latency in the execution of the global VNF. An abundant literature addresses the problem of batch processing and constraint based programming. In the following, we shall focus on non-preemptive Round-Robin scheduling to test the benefit of parallel execution of sub-tasks of a VNF before considering more sophisticated scheduling algorithms.

B. The case of RAN

To illustrate the decomposition of a VNF into sub-functions or components, we consider in this section the case of RAN in the framework of LTE. Note that the current trend in 3GPP is to keep the architecture of RAN for the first phase of 5G similar to that of LTE. As a consequence, it is relevant to consider the current architecture for devising the RAN of 5G. We shall more precisely focus on distributed radio elements (RRH) and centralized BBUs with shared processing, i.e., the BBU functions are hosted in a multi-core data center.

Separating the BBU and RRH functions is motivated by many reasons, beyond the usual ones such as better computing resource utilization, energy savings, operational cost reduction, etc. Grouping several BBUs on the same data center also enables multi-point cooperation, advanced coordinated beamforming, optimal radio resource allocation, and efficient power management.

The BBU processes the data of User Equipments (UEs) as follows. Packets from the IP layer are handled by the PDCP, RLC and MAC layers before to be presented in a Transport Block (TB) format to the physical layer (PHY) as shown in Figure 2.

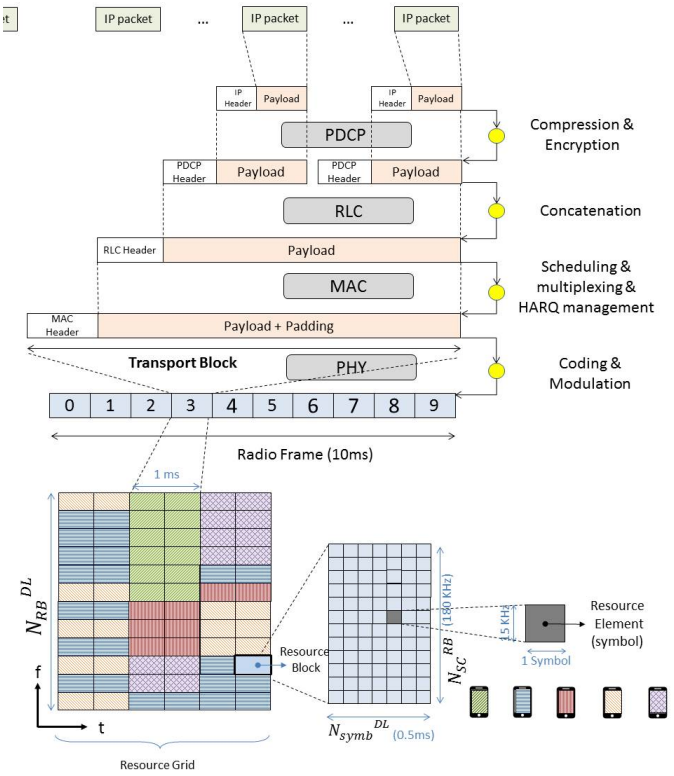


Fig. 2. BBU elements

A TB is a group of Resource Block (RB) with a common Modulation and Coding Scheme (MCS). The modulation Order (denoted by Q_m) represents the number of bits per symbol. The data rate for an UE depends on the number of data bits contained in the TB. Thus, the Transport Block Size (TBS) corresponds to the data carried for an UE in a Transmission Time Interval (TTI).

LTE standardization defines the procedure for choosing the TBS for FDD and TDD frame configurations. In the rest of this paper, we shall consider the FDD scenario and we shall analyze only the functions involved in the transmission process (Tx) for the down-link (DL) direction. Furthermore, we shall consider only the best effort bearer, in spite of the fact that the 3GPP standards specify a multi-bearer architecture. In the horizon of 5G, it is very likely that all services will be supported by the best effort bearer. An adequate traffic management at the IP layer, possibly taking into account cross layer information (including that of the radio layer), will then be necessary to meet the QoS requirements. A quality of service architecture is however out of the scope of this paper.

The runtime of the VNF that implements the BBU functions is especially impacted by the sub-functions listed below and shown in Figure 3:

- Compression and encryption procedures of the PDCP sub-layer;
- Concatenation tasks executed by the RLC sub-layer;
- Radio scheduling and HARQ management handled by the MAC layer, and
- Sub-functions performed by the PHY layer, which are channel coding, modulation and OFDM signal generation.

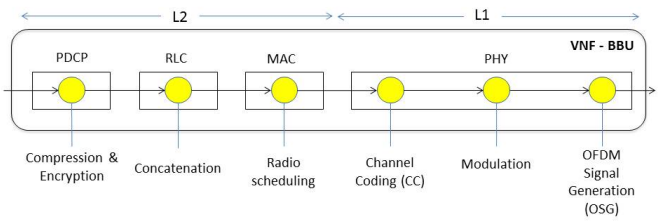


Fig. 3. VNF model of the BBU

The VNF threading model for the BBU is shown in Figure 4, which is consistent with the VNF template depicted in Figure 1. It presents the various sub-functions to be executed for different UEs. PDCP sub-functions are able to be executed in parallel running one thread per UE. RLC sub-functions are sequentially executed after the PDCP sub-function.

The MAC layer defines the TBS and MCS which are function of the traffic load and the radio channel conditions. The radio scheduler selects the UEs to be served in each TTI, i.e., one sub-frame of 1 millisecond. The scheduler allocates resources according to some algorithm, for instance Round Robin or Proportional Fair among the various UEs which have data to receive or transmit. In general, the scheduler takes into account the radio conditions (namely Channel Quality Indicator (CQI)) to allocate radio resources while achieving some fairness criterion. In general, the CQI derives from channel models which are defined by vendors. The most

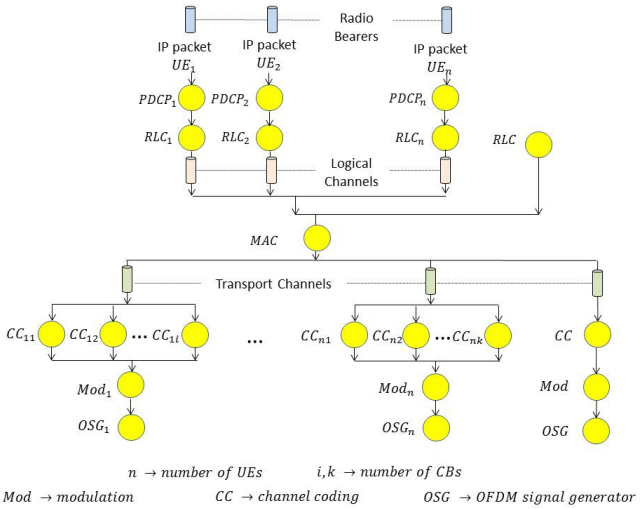


Fig. 4. VNF threading model of the BBU

important factor involved in the CQI measurement is the Signal-to Interference Noise Ratio (SINR).

Current mobile networks use high-throughput FPGA architectures to perform channel coding which can be considered as the bottleneck in the virtualization of RAN functions [12]. As a consequence, we investigate the gain that can be achieved when virtual coding sub-functions are instantiated and executed in parallel as much as possible. We propose a multi-threading scheme which processes transport blocks by segments namely Code Blocks (CBs). LTE defines 6120 bits as the maximum Code Block Size (CBS).

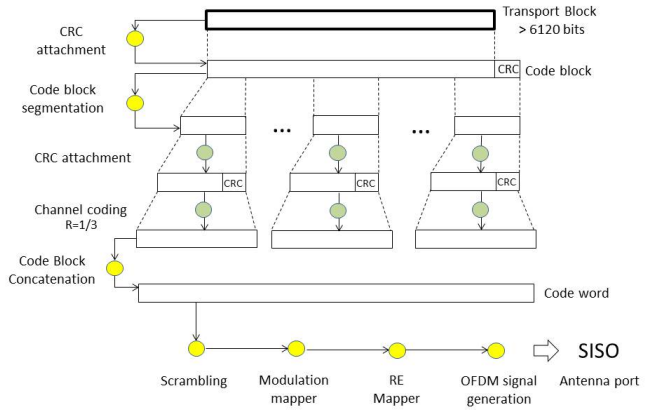


Fig. 5. PHY threading model

The CB is the data structure which triggers the encoding function. The number of parallel channel coding threads per UE depends on the TBS. When the TBS is not big enough, parallelism is not performed since only one CB is produced. Conversely, when the TBS takes the maximum standardized value, the number of parallel threads is given by $\lceil \mathrm{TBS}/\mathrm{CBS} \rceil$. See Figure 5 for an illustration.

Modulation sub-function is sequentially executed after the code block concatenation. Finally, the OFDM signal is generated after the modulation sub-function is completed.

Figure 6 shows an illustration of additional procedures required for MIMO. Note that the transport block size is also different.

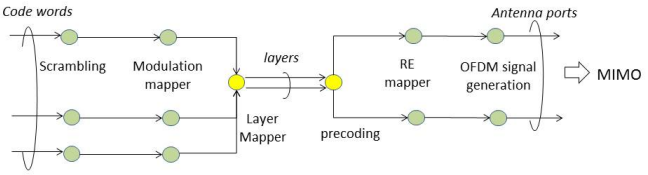


Fig. 6. MIMO threading model

The aim of the threading model presented above is to investigate the performance gain in terms of execution time when running PHY sub-functions with a high level of parallelism. Concretely, we focus on the channel coding sub-function since it is the most expensive one. These claims are validated by simulation in Section V.

Looking for a high-performing virtual RAN, we also investigate the scheduling algorithm which dictates how much CPU time is allocated to each sub-function, thread and/or job. We describe general implementation guidelines in the next section.

III. Implementation of a virtualized BBU

A. Implementation guidelines

The implementation of a VNF on a data center as depicted in Figure 1 calls for some implementation principles. Basically, when a software-based network function runs on a multi-core platform, it generates multiple jobs which are executed according to a certain scheduling algorithm. (See Figure 7 for an illustration.)

The scheduling strategy plays a crucial role in the performance of VNFs, since it allocates the processing capacity and decides which runnable job will be executed, i.e., which processor executes a job and in what order the jobs are processed. Scheduling algorithms are implemented in the kernel of operating systems (OS), thus virtualization solutions where the resource allocation of CPU and memory is handled by an external entity (e.g., the hypervisor in the case of VM) have lower performance than those where the resource provisioning is kept by the OS (e.g.,containers).

The computing platform architecture is also important for the performance of VNFs, especially when considering the memory access time. Generally speaking, the time required to access instructions and data in memory is rarely negligible in general purpose computers.

Commodity parallel computing platforms based on GPUs can significantly increase the performance, since they give direct access to the instruction set and parallel runnable elements. The performance analysis of computer architectures and memory access mechanisms is beyond the scope of this work. It is nevertheless important to note that recent

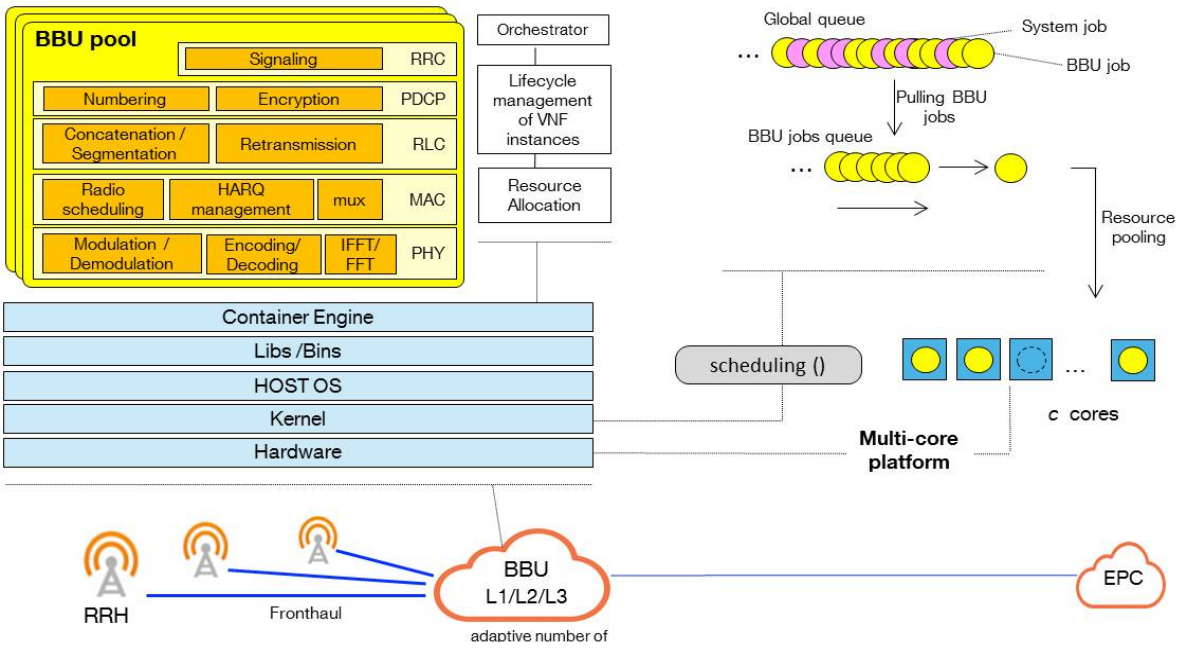


Fig. 7. BBU virtualization environment

studies have compared the LTE BBU execution on GPU and CPU based architectures [13]. Results show that GPU servers substantially increase the performance.

In the following, we focus our study on the fact that there is an important level of parallelism in the base-band processing of LTE. It can be exploited when applying an appropriate threading-based model that works together with the scheduling strategy of a multi-core platform.

B. Implementing BBUs in a virtualized environment

Let us consider a multi-core platform where two or more homogeneous cores are connected to a shared memory. This architecture represents a pool of resources which can host various eNBs. The main challenge is to guarantee the individual performance of each eNB while avoiding waste of resources. Thus, the computing platform processes the BBU functions of various radio network elements which are geographically distant. The front-haul capacity should also maintain the latency requirements in order to meet LTE deadlines.

We propose a container-based virtual environment in which the BBU functions run as applications. This approach improves the performance because there is a single kernel handling the BBU processes or jobs, in this way their execution uses less RAM and makes the runtime more efficient. See Figure 7 for an illustration.

We assume that all cores are controlled by a global scheduler. Partitioned scheduling is also possible in multi-core systems, however dedicating resources for a particular task or sub-function limits the performance of the VNF runtime [7].

The global scheduler makes its decisions based on the scheduling policy. We assume that all BBU-jobs have the same and the highest priority in the system. Thus, the scheduling policy allocates cores among the runnable BBU-threads. The scheduling strategy is detailed and illustrated in the following section.

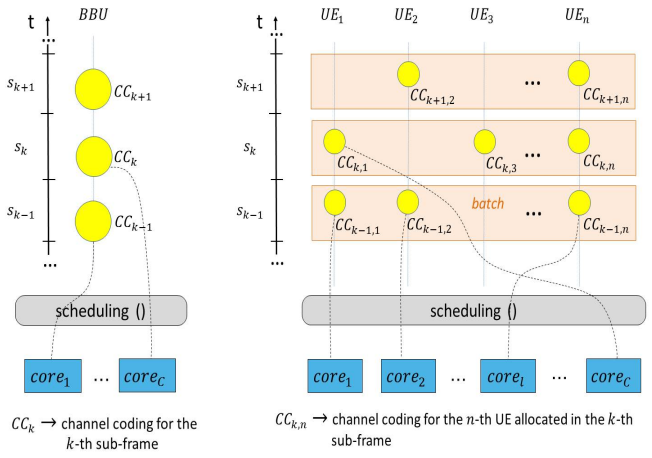
IV. Scheduling Strategy for Parallel Processing

In the framework of parallel computing on multi-core platforms, we define a global scheduler which selects the next job to be processed among those which are ready to be executed. The scheduler allocates a single processing unit (core) to each job according to the Round Robin criterion. We use non-preemptive scheduling, in this way, a running job cannot be interrupted and it is executed until completion.

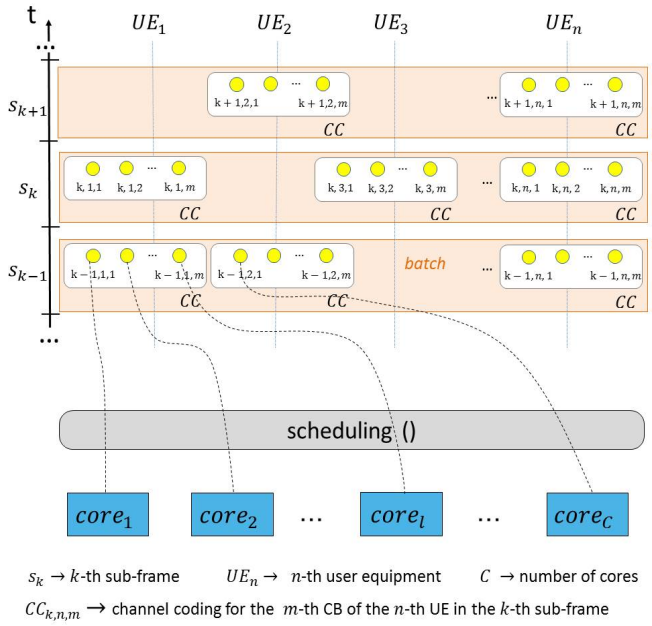
We focus on the channel coding function which is decomposed into several sub-functions that can be processed in parallel. See Figure 8 for an illustration. The execution of sub-functions occurs at the same instant on separate cores. Note that parallel computing is not possible on a single core, since only one sub-task is able to run at any instant. When sub-tasks are carried out simultaneously on one core (or even multiple-cores) by time-sharing computing resources, we then refer to concurrent computing [14]. Parallel and concurrent computing can be used together, nevertheless, we focus our analysis on parallel computing.

As shown in Figure 8, we consider data decomposition in such a way that the channel coding sub-function is performed on different sub-sets of data at the same time, i.e., each parallel task works on a portion of data. Note that functional parallelism is not applicable on the PHY layer of BBUs, since modulation, channel coding and IFFT sub-functions must be executed in series.

In the framework of software-based BBUs, each LTE sub-frame generates a job for channel coding processing, i.e., a



(a) No parallelism (b) UEs in parallel



(c) CBs in parallel

Fig. 8. Global scheduler for parallel computing

new job requires service every millisecond. When parallel computing is not applied, as shown in Figure 8(a), a channel coding job denoted by CC_k for the k-th sub-frame is executed on a single core even when more cores are available. If the channel coding job CC_k arrives when the previous one CC_{k-1} is not finished, the scheduler allocates to it the next free core. In the case that all cores are busy, the job is queued. This scenario is not desirable when executing base-band functions, mainly when closed-loop processes are active, e.g., the HARQ process defined in LTE.

In order to speed up the channel coding sub-function, we decompose each channel coding job CC_k in a suite of sub-jobs. Each parallel sub-job works on a portion of BBU data. The sub-job belonging to the n-th UE allocated in the k-th sub-frame is denoted by $CC_{k,n}$. Channel coding jobs arrive in batches at the computing platform every millisecond. (See Figure 8(b) for an illustration.) The batch size (i.e., the number of active jobs) is variable and corresponds to the number of UEs allocated into a sub-frame. The service time of a job in this batch varies in function of the TBS which is determined by the radio scheduler in the MAC layer. Batches are processed in a First-Come-First-Serve discipline. The global scheduler selects jobs in a circular order and assigns them one core to each.

We now consider a finer granularity in the decomposition of channel coding sub-functions. Thus, each job $CC_{k,n}$ of the n-th UE is split in a set of m parallel sub-jobs, where the m-th sub-job is denoted by $CC_{k,n,m}$. This is illustrated in Figure 8(c). The resulting number of parallel channel coding jobs, i.e., the batch size, varies every millisecond (or more generally, according to the periodicity of radio scheduling). Thus, the batch size is the product of active UEs and the number m of CBs. Note that m can take different values for each UE. As in the preceding threading model, incoming batches join the tail of a single queue and each job gets access to a particular core under Round-Robin criterion.

We evaluate these scenarios by simulation. Performance results and simulation settings are described in the following section.

V. Performance Analysis of a Virtualized BBU

A. Simulation settings

In this section, we evaluate by simulation the execution time of virtualized BBU functions. First, we analyse the performance on a single-core platform. Then, we present the gain in terms of latency on a multi-core platform employing the threading model depicted in Figure 4. The simulation emulates a container-based virtualization environment with a global scheduling as described in Section III. We utilize the software-based RAN solution named OAI-5G, in order to determine the execution time of each runnable job.

We emulate real traffic conditions for an urban macro cell with multiple connected UEs. Concretely, we consider heterogeneous traffic where UEs with small needs in terms of radio resources coexist with UEs requiring high data rates. In addition, the simulation scenario takes place during the peak-hour, i.e., we assume a greedy consumption of radio resources with the aim to evaluate the worst case. The simulation tool implements the Release 12 of the LTE specification for the PHY layer which is defined by the 3GPP TS 36.213, version 12.4.0 [15].

The simulation is performed for an LTE bandwidth of 20MHz, FDD transmission and SISO configuration. We focus our study on the execution of the PHY DL taskset which is composed of channel coding, modulation, and IFFT functions. We are concretely interested in the runtime performed by each TTI (1 millisecond).

The simulation selects the modulation order Q_m and more concretely the MCS index I_{MCS} based on the emulated radio conditions, i.e., the SINR. It is shown in Table I.

TABLE I
MODULATION ORDER AND SINR

CQI	Modulation	Q_m	MCS index I_{MCS}	SINR [dB]
1-6	QPSK	2	0 - 9	< 3
7- 9	16-QAM	4	10 - 16	< 9
10- 15	64-QAM	6	17 - 28	< 20

In the same way, we define the number of Physical Resource Blocks (PRB) for an UE in function of the emulated traffic in the cell and of the data load per user. Note that for an LTE Bandwidth of 20 MHz the available number of PRB is 110 [15].

We can easily identify the I_{TBS} from the I_{MCS} looking up the table 7.1.7.1-1 presented in the LTE Physical layer specification 3GPP TS 36.213 version 12.4.0 Release 12 [15]. Used values are show in Table II.

TABLE II
MODULATION AND TBS INDEX

MCS index I_{MCS}	Q_m	TBS index I_{TBS}
0-9	2	$I_{TBS} = I_{MCS}$
10- 16	4	$I_{TBS} = I_{MCS} - 1$
17- 28	6	$I_{TBS} = I_{MCS} - 2$

Based on the TBS index, I_{TBS}, and the number of Physical RB, N_{PRB}, we determine the TBS from the table 7.1.7.2.1.1-1 of the LTE PHY specification 36.213 version 12.4.0 Release 12 [15]. It is for transport blocks not mapped to two or more layer spatial multiplexing, i.e., SISO configuration. A fragment is shown in Table III

TABLE III
TRANSPORT BLOCK SIZE TABLE

	N_{PRB}					
I_{TBS}	4	5	6	7	8	9
12	904	1128	1352	1608	1800	2024
13	1000	1256	1544	1800	2024	2280
14	1128	1416	1736	1992	2280	2600

B. Performance results: PHY runtime

The simulation shows the runtime of the PHY layer for a single cell and multiple connected UEs. In a first step, we do not consider the parallel execution of BBU functions. Figure 9 presents separately the runtime for the Modulation, Channel coding and IFFT sub-functions.

The execution time of the modulation sub-function depends on the modulation order and on the number of PRBs. The IFFT sub-function depends only on the number of PRBs. Finally the Channel Coding depends on the TBS, i.e., the quality of radio conditions, the amount of data to be transmitted by each UE and the traffic load in the cell. Simulation results which are based on the OAI performance show that the runtime of the whole PHY layer takes around 785 microseconds. It does not leave enough margin for processing L2 and L3 BBU functions. This fact may limit the implementation of centralized BBUs. We present below the gain that can be obtained when using implementation guidelines that were described in Section III.

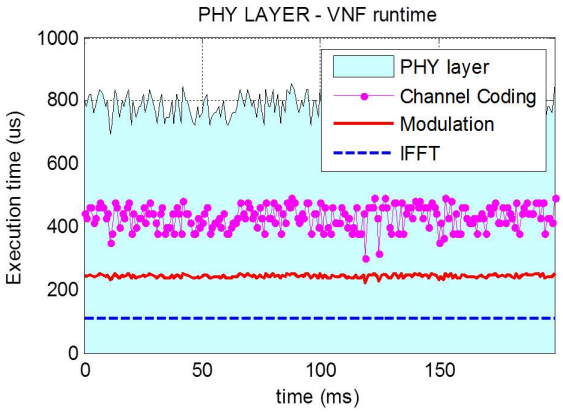
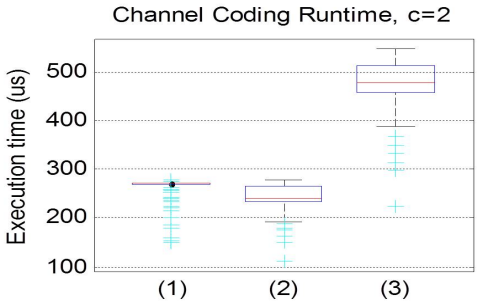


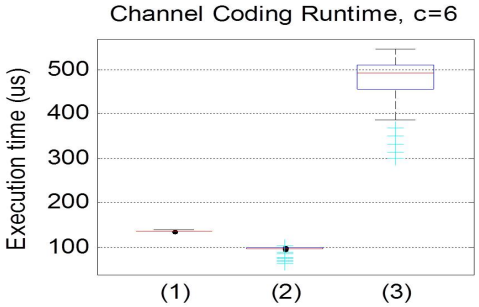
Fig. 9. Execution time of PHY sub-functions

C. Gain in terms of latency

In order to reduce the execution time of the BBU, we focus our analysis on the channel coding sub-function. We simulate a multi-core platform with c cores and a global scheduler managing all of them as detailed in Section IV.



(a) $c = 2$



(b) $c = 6$

Fig. 10. Parallel execution of channel coding.

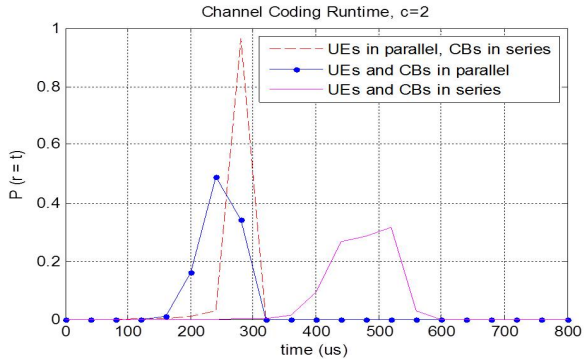
We evaluate the gain in terms of latency when using the threading model presented in Section II. We are able to execute one thread per UE, and even one thread by CB. It is possible to

execute up to eleven threads per UE when it has the maximum number of RBs and the maximum modulation order according to the LTE-Release 12.

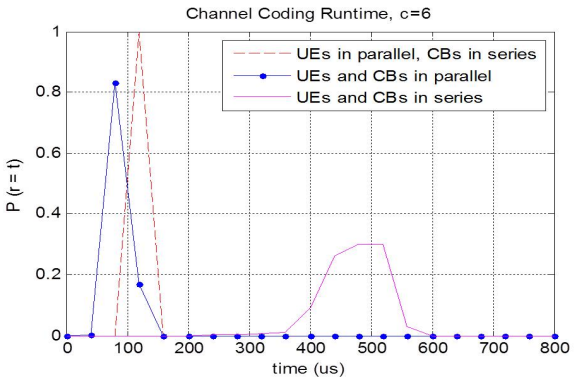
Figures 10(a) and 10(b) show the gain of the parallel execution of channel coding in a multi-core platform with 2 and 6 cores respectively. Results are shown for the following scenarios:

1) UEs in parallel, CBs in series: one thread per UE.
2) UEs and CBs in parallel: one thread per CB.
3) UEs and CBs in series: no parallelism.

It clearly appears that the execution of the channel coding sub-function with one thread per CB can halve the runtime even when using only twice the processing capacity. When the processing capacity is significantly increased, (e.g., for a multi-core platform with 6 cores), the runtime of the channel coding can be reduced almost five times compared with the original model which is not based on parallel threads.



(a) $c = 2$



(b) $c = 6$

Fig. 11. Probability density of channel coding runtime.

Figures 11(a) and 11(b) show the probability density distribution function for each scenario. Results show that for a multi-core platform with $c = 6$, the execution time of the channel coding when running one thread per CB or even one thread per UE becomes less variable (i.e., runtime values are more concentrated around the mean). This fact is relevant for the incoming cloudification of RAN functions.

VI. Conclusion

We have studied in this paper the execution of VNFs on multi-core platforms. Results show that NFV implementation requires an adequate computing platform, an efficient global scheduling algorithm, as well as an optimum threading model in order to minimize the runtime of VNFs.

In the case of software-based RAN functions, we defined threads at a fine granularity and as a result, the most expensive sub-function in terms of latency (namely channel coding) was strongly decreased. This fact is a step towards the Cloud-RAN implementation, since some tasks of the BBU can be moved higher in the network, this offers more possibilities in the functional split.

In a future work, we shall investigate how these sub-functions might be executed on various data-centers separated from each other. Moreover, we shall investigate how to account of the uplink and what strategy is the most suited for sharing computing resources.

References

[1] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[2] Navid Nikaein, Raymond Knopp, Florian Kaltenberger, Lionel Gauthier, Christian Bonnet, Dominique Nussbaum, and Riadh Ghaddab. Openair-interface 4G: an open LTE network in a PC. In *International Conference on Mobile Computing and Networking*, 2014.

[3] Florian Kaltenberger, Raymond Knopp, Navid Nikaein, Dominique Nussbaum, Lionel Gauthier, and Christian Bonnet. OpenAirInterface: Open-source software radio solution for 5G. In *European Conference on Networks and Communications (EUCNC), Paris, France*, 2015.

[4] Johanna Heinonen, Pekka Korja, Tapio Partti, Hannu Flinck, and Petteri Pöyhönen. Mobility management enhancements for 5G low latency services. In *2016 IEEE International Conference on Communications Workshops (ICC)*, pages 68–73. IEEE, 2016.

[5] Xuan Zhou, Rongpeng Li, Tao Chen, and Honggang Zhang. Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine*, 54(7):146–153, 2016.

[6] David Dietrich, Ahmed Abujoda, and Panagiotis Papadimitriou. Network service embedding across multiple providers with Nestor. In *IFIP Networking Conference, 2015*, pages 1–9. IEEE, 2015.

[7] Leonard Kleinrock. *Queueing Systems*, volume II: Computer Applications. Wiley Interscience, 1976.

[8] China Mobile. C-RAN: the road towards green RAN. *White Paper, ver*, 2, 2011.

[9] I Chih-Lin, Corbett Rowell, Shuangfeng Han, Zhikun Xu, Gang Li, and Zhengang Pan. Toward green and soft: a 5G perspective. *IEEE Communications Magazine*, 52(2):66–73, 2014.

[10] I Chih-Lin, Jinri Huang, Ran Duan, Chunfeng Cui, Jesse Xiaogen Jiang, and Lei Li. Recent progress on C-RAN centralization and cloudification. *IEEE Access*, 2:1030–1039, 2014.

[11] Shane Ryoo, Christopher I Rodrigues, Sara S Baghsorkhi, Sam S Stone, David B Kirk, and Wen-mei W Hwu. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73–82, 2008.

[12] Islam Alyafawi, Eryk Schiller, Torsten Braun, Desislava Dimitrova, Andre Gomes, and Navid Nikaein. Critical issues of centralized and cloudified LTE-FDD radio access networks. In *2015 IEEE International Conference on Communications (ICC)*, pages 5523–5528. IEEE, 2015.

[13] Q Zheng, Y Chen, S Abeyratne, S Dreslinski, and T Mudge. Revisiting cloud RAN from a computer architecture point of view, July 2016. [Online; posted July-2016].

[14] R Pike and A Gerrand. Concurrency is not parallelism. *Heroku Waza*, 2012.

[15] LTE, evolved universal terrestrial radio access, physical layer procedures (3GPP TS 36.213 version 12.4.0 release 12). Standard, European Telecommunications Standards Institute, 2015.