How much is Fronthaul Latency Budget Impacted by RAN Virtualisation ?

H. Gupta⋆, D. Manicone•, F. Giannone•, K. Kondepu•, A. Franklin⋆, P. Castoldi•, L. Valcarenghi•
•Scuola Superiore Sant'Anna, Pisa, Italy
⋆Indian Institute of Technology Hyderabad, India
Email:cs16mtech01001@iith.ac.in

***Abstract*—In the New Radio Access Network architecture (New RAN), currently envisioned by 3GPP, the evolved NodeB (eNB) functions can be split between a Distributed Unit (DU) and Central Unit (CU). Furthermore, as per the Virtual RAN (VRAN) approach, such functions can be virtualised (e.g., in simple terms, deployed in virtual machines). In such scenario, the fronthaul network connecting DU and CU must fulfill different latency and capacity requirements based on the selected functional split.**

This study experimentally evaluates in a federated testbed how the fronthaul latency budget (i.e., the latency requirement of the fronthaul network connecting DU and CU), specified by Standard Developing Organisations (SDO) (3GPP in this specific case), is impacted by virtualising some of the RAN functions. In particular, Option 7-1 (i.e., intra-PHY split) and different virtualisation methods are considered for the CU. Furthermore, it evaluates how jitter (i.e., delay variation) impacts the DU-CU communication.

The obtained results show that light virtualisation methods (e.g., *Docker*) impact less the fronthaul latency budget than heavy virtualisation methods (e.g., *VirtualBox*). In addition, a maximum jitter of about 40 μs can be tolerated in the fronthaul.

***Index Terms*—5G, functional split, virtualisation, latency, jitter**

I. Introduction

5G networks are expected to be massively deployed and offer an unprecedented capacity to address the demanding requirements of throughput, latency, and scalability of current and future 5G applications, such as eMBB (enhanced Mobile Broadband), mMTC (massive Machine Type Communications) and URLLC (Ultra-Reliable and Low Latency Communications) [1], [2]. A New Radio Access Network (New RAN) architecture supporting the, so called, New Radio (NR) access technology has been proposed to increase performance with limited deployment costs. In the New RAN the evolved NodeB (eNB) functions are split into two new network entities [3]: the Central Unit (CU) deployed in centralised locations and the Distributed Unit (DU) deployed near the antenna. The function distribution is based on the chosen functional split.

The choice of supporting different functional splits is motivated by the limitations shown by the Common Public Radio Interface (CPRI), so far used to connect the Radio Equipment Control (REC) (i.e., CU) and the Radio Equipment (i.e., DU) [4]. CPRI is based on carrying time domain baseband IQ samples between REC and RE. Thus, CPRI needs a high capacity fronthaul, low latency, low delay variation and fine synchronisation. Guaranteeing such requirements in the fronthaul is particularly challenging and expensive [5]–[9].

New upper layer functional splits have been proposed by 3GPP in TR 38.801 [3] and a Next Generation Fronthaul Interface (NGFI) is under definition [10]. As reported in 3GPP TR 38.801 [3], different splits demand different requirements in terms of latency and capacity to the fronthaul network connecting DU and CU. Moreover, recent approaches are proposing the virtualisation of the New RAN functions (e.g., the CU) paving the way to the so-called Virtual RAN (VRAN) [11]. However the impact of such virtualisation on the fronthaul latency budget (i.e., the latency requirement of the fronthaul network connecting DU and CU) is yet to be fully evaluated.

One of the main component of virtualisation is the hypervisor [12]. Hypervisors monitor virtual machines and allocate the physical resources of the host operating system. Different hypervisors characterised by different virtualisation "depth", may have different performance in virtualising the same scenario. Therefore, in VRAN, the latency introduced by the hypervisor operations could be crucial for the fronthaul latency budget.

This paper evaluates experimentally the fronthaul latency and jitter (i.e., delay variation) budgets for different radio channel bandwidths, when different virtualisation methods are utilized. Option 7-1 (i.e., intra-PHY) functional split is considered and EPC and CU only are virtualised by using *VirtualBox*, *Kernel-based Virtual Machine* (*KVM*) and *Docker*. The experimental evaluation is performed in the 5G segment of the Advanced Research on Networking testbed (*ARNO-5G*). *ARNO-5G* allows to emulate the behaviour of a 5G network and run performance tests to evaluate several functional split requirements. ARNO-5G is federated in the Fed4FIRE federation.

The obtained results show that light virtualisation methods (e.g., *Docker*) impact less the fronthaul latency budget than heavy virtualisation methods (e.g., *VirtualBox*). In addition, a maximum jitter of about 40 μs can be tolerated in the fronthaul.

II. The ARNO-5G Testbed

Fig. 1 shows a block diagram of the ARNO-5G testbed. This section describes the physical characteristics of the devices composing the ARNO-5G testbed, the federation of the testbed and the utilized LTE emulation platform.

TABLE I
ARNO-5G TESTBED

Devices Name	Devices Type	Processor Type	OS
PC 1	mini-pc (Up-board First Generation)	Intel Atom x5-Z8350 Quad Core Processor	Ubuntu 14.04 (4.7 kernel)
PC 2	Dell T410 PowerEdge desktop servers	Intel Xeon E5620	Ubuntu 14.04 (3.19 low-latency kernel)
PC 3	Dell T410 PowerEdge desktop servers	Intel Xeon E5620	Ubuntu 14.04 (3.19 low-latency kernel)
PC 4	Mini-ITX	Intel I7 7700 Quad Core (@ 4.0GHz)	Ubuntu 14.04 (3.19 low-latency kernel)
PC 5	mini-pc (Up-board First Generation)	Intel Atom x5-Z8350 Quad Core Processor	Ubuntu 14.04 (4.7 kernel)
PC 6	mini-pc (Up-board First Generation)	Intel Atom x5-Z8350 Quad Core Processor	Ubuntu 14.04 (4.7 kernel)
PC 7	Desktop Computer	Intel I7 7700 Quad Core (@ 4.0GHz)	Ubuntu 14.04 (3.19 low-latency kernel)

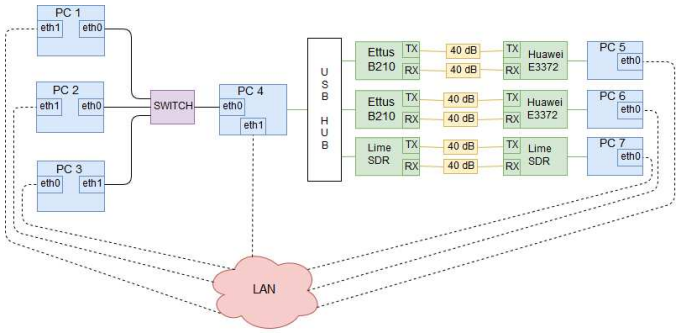

Fig. 1. The ARNO 5G testbed

A. Overview of the ARNO-5G Testbed

Table I summarizes the principal characteristics of the devices composing the ARNO-5G testbed. The kernel hosted in PC 1 is directly pre-compiled by OpenAirInterface (OAI) platform for including the GPRS Tunnelling Protocol (GTP) kernel module. *PC 4* is connected Universal Software Radio Peripherals (USRPs) through an *USB 3.0 HUB* in order to easily attach/detach each USRPs. The Ettus B210 and the LimeSDR are fully integrated, single-board, USRP platforms and they act as radio front-end performing Digital to Analog and Analog to Digital Conversion (DAC/ADC), Digital Up and Down Conversion (DUC/DAC), low pass filtering, and amplification.

The Huawei E3372 LTE dongles are utilized as User Equipments (UEs). They support LTE category 4 and frequency-division duplexing (FDD) communication in the following bands: 900 MHz, 1800 MHz, 2100 MHz and 2600 MHz. They support a maximum rate of 150 Mb/s in downlink and 50 Mb/s uplink with a signal bandwidth of 20 MHz. The dongles are connected to the Ettus B210 USRPs through SMA cables with 40 dB of attenuation. Note that one of the LimeSDR is also used to act as OAI UE.

All the ARNO-5G PCs, have a management plane interface and also a data plane interface. The management and data planes belong to two different networks. The former one (i.e., 10.30.x.x) belongs to the lab LAN to assure the continuous reachability of the machines. The latter one is handled by a Cisco Catalyst 2960G switch (indicated as SWITCH in Fig. 1) and it is used exclusively as ARNO-5G testbed data plane for the communication between the LTE network entities. All the PCs are connected to the management plane through 100 Mb/s Ethernet links while they are all connected to the SWITCH by a 1 Gigabit Ethernet link. The SWITCH is configured to have different subnets for the backhaul link and for the fronthaul link. Therefore the EPC and the CU interfaces belonging to the backhaul link are configured in the 10.10.20.x subnet while the fronthaul link and the interfaces of the CU and DU are configured in the 10.10.30.x subnet.

The utilised mobile network software is the OpenAirInterface (OAI) by Eurecom [13]. The core network utilizes *openair-cn*, an implementation of the Evolved Packet Core (EPC) network [14]. It implements the EPC 3GPP specifications, that means it contains the implementation of the following network elements: the Serving Gateway (S-GW), the PDN Gateway (PDN GW), the Mobile Management Entity (MME) and the Home Subscriber Server (HSS). For the RAN, *openairinterface5g* provides a standard-compliant implementation of Release 10 LTE and later for the evolved NodeB (eNB) and User Equipment (UE) [15]. It also provides an implementation of some eNB functional splits and therefore a possible deployment of Cloud/Virtualised RAN (C/V-RAN). The functional splits implemented by the OAI platform are the IF5 and IF4.5 also known as Option 8 and Option 7-1 in the 3GPP terminology [3].

Option 8 (i.e., PHY-RF split) separates the RF and the PHY layer. All the RF functions reside in the DU and the layers from the PHY layer to Radio Resource Control layer (RRC) reside in the CU.

In Option 7-1 (i.e., intra-PHY) split in the uplink direction, Fast Fourier Transform (FFT), Cyclic Prefix (CP) removal and possibly Physical Random Access Channel (PRACH) filtering functions reside in the DU and the rest of PHY functions reside in the CU. In the downlink direction, Inverse Fast Fourier Transform (IFFT) and CP addition functions reside in the DU, the rest of PHY functions reside in the CU. In other word, Option 7-1 functional split is made before/after the resource mapping/demapping respectively.

B. Federation of the ARNO-5G Testbed

ARNO testbed is federated in the Fed4FIRE federation and it accepts users from only one trusted central authority (iMinds) identity provider. Here, experimenters can configure experiments interconnecting resources from multiple testbeds at the same time, reserve and access them via iMinds tools such as jFed [16].

Once successfully logged in, experimenters can set up their experiments by choosing which types of resources and from which testbeds, configure those resources (operative system, software to be installed, network configurations, measurement options, etc.), launch the experiments and access the resources.

Through jFed tool, an experimenter can select the ARNO testbed, namely "Sant'Anna Pisa testbed" and provide their slice name. This process creates a Docker container in ARNO testbed. Finally experimenters can enter each OAI component of ARNO testbed through ssh based on the specific container. More details about how to reserve the components of ARNO testbed can be found in [17].

III. Performance Evaluation Parameters and Evaluation Scenarios

This paper evaluates experimentally the latency and jitter budgets that Option 7-1 functional split can support in the fronthaul. The *fronthaul latency budget* is defined as the one-way latency requirement of the sole fronthaul network interconnecting the DU and the CU. The *fronthaul jitter budget* is defined as the maximum supported jitter (i.e., latency variation) by the fronthaul. For Option 7-1 split the one-way fronthaul latency constraint specified by 3GPP is 250 μs [3], mainly due to the 4ms limit of the Hybrid ARQ (HARQ) [18] protocol, while no constraint is specified for the jitter. The experimental evaluations aforementioned are performed for different virtualisation softwares and for signal bandwidths equal to 5 MHz and 10 MHz, corresponding to 25 and 50 Physical Resource Blocks (PRBs).

The latency and the jitter experienced along the fronthaul link are emulated by means of the linux utility traffic control *tc*. The *tc* utility is based on a token bucket filter and it is capable of increasing the delay and jitter experienced on a link by a packet by storing it in the output interface for a specified amount of time before its transmission on the link. A delay *d0* is applied to the Ethernet interface of the machine in which the DU is deployed and a delay *d1* is applied to the Ethernet interface of the machine in which the CU is deployed. In this way a one-way latency is inserted in the fronthaul link. For evaluating the fronthaul latency budget, *d0* and *d1* are increased with steps of 10 μs until DU, CU, and UE disconnect. For evaluating the fronthaul jitter budget, instead, the jitter follows a normal distribution and it is added to the latency values *d0* and *d1* with steps of 10 μs until DU, CU, and UE disconnect. Two different scenarios are considered: in the first one the latency is set close to the fronthaul latency budget and the jitter is varied to understand if the jitter could cause a reduction of the fronthual latency budget. In the second one the latency is set quite below the fronthaul latency budget and the jitter is varied to understand if the jitter could be an additional constraint for the fronthaul.

The considered experimental evaluation scenario is shown in Fig. 2. Regardless of the considered hypervisor, in such scenario, the EPC and CU are virtualised while the DU is deployed in a physical machine. In particular, the EPC is deployed in *PC 1*. The Mobile Management Entity (MME) is

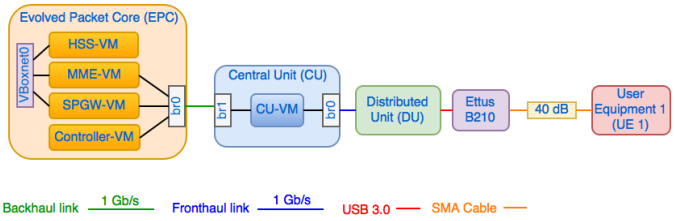

Fig. 2. Scenario considered for the experimental evaluation

deployed in a VM called *MME-VM* and the Home Subscriber Server (HSS) is deployed in a second VM called *HSS-VM*. The Serving Gateway (S-GW) and the PDN Gateway (PDN-GW) are deployed in a third VM called *SPGW-VM*. The *CU* is hosted in *PC 2* and is deployed in another VM called *CU-VM*. The DU runs always in *PC 4* in a physical machine and the UEs, connected to the RAN through SMA cables with 40 dB of attenuation, are deployed by means of a Huawei E3372 dongle connected to *PC 5*.

The other experimental parameters, independent of the utilized virtualisation method, are shown in Table II.

TABLE II
Experimental Parameters

Parameter	Value
Experiment Duration	100000 TTIs
Frame Duration	10 ms
Duplexing Mode	FDD
PHY Layer Abstraction	NO
Number of CUs	1
Number of DUs	1
Number of UEs	1
Carrier Bandwidth	5MHz, 10 MHZ

Different virtualisation methods are considered: *VirtualBox*, *Kernel-based Virtual Machine* (*KVM*) and *Docker*.

Using *VirtualBox*, *MME-VM*, *HSS-VM* and *SPGW-VM* host Ubuntu 16.04 with 4.8 generic kernel featuring a one core virtual CPU and 1 GB of RAM. Instead *CU-VM* hosts Ubuntu 14.04 with 3.19 low-latency kernel featuring a 8 core virtual CPU and 16 GB of RAM.

The utilized virtual Network Interface Controller (NIC) modes are bridged and host-only networking. Here, the bridge networking mode allows a VM to intercept data from/to the physical network effectively by creating a new network interface in software. Therefore we bridge a virtual ethernet interface in bridge networking mode in both *MME-VM* and *SPGW-VM* to the physical ethernet interface in *PC 1*, referred as *br0* in Fig. 2. Whereas in *CU-VM*, two virtual interfaces are bridged in bridge networking mode with corresponding physical ethernet interfaces in *PC 2* (referred as *br0* and *br1* in Fig. 2). This because the first *CU-VM* virtual interface has to connect to the *MME-VM* for the LTE control plane communications (S1-C interface) and with the *SPGW-VM* for the LTE data plane communications (S1-U interface). Instead, the second *CU-VM* virtual interface is used for the fronthaul communication with the *DU*.

The host-only networking mode, is a networking mode that can be thought of as a hybrid bridged networking: the virtual machines can communicate to each other and the host as if they were connected through a physical Ethernet switch but they cannot communicate to the external (world) host since there is not a networking interface. Therefore, such networking mode is used for the internal communications between the entities composing the EPC: a virtual interface, different from the above mentioned, is set on *MME-VM*, *HSS-VM* and *SPGW-VM* allowing the communications between the *MME-VM* and the *HSS-VM* (S6a interface) and between the *MME-VM* and *SPGW-VM* (S11 interface) through the host-only adapter vboxnet0.

The NICs used for the virtualised EPC are:

- Bridge adapter *enp0s3* to physical interface *eno2* with subnet 10.30.x.x (used for the management plane);
- Host-only adapter *enp0s8* to *vboxnet0* interface, with subnet 192.168.x.x (used for internal EPC service configuration and relationship);

The NICs used for the virtualised CU are:

- Bridge adapter *enp0s3* to physical interface *eth5* with subnet 10.30.x.x (used to set S1-C and S1-U interface between CU and MME and between the CU and SPGW respectively);
- Bridge adapter *enp0s9*, to Physical interface *eth1* with subnet 10.10.x.x (used to set the fronthaul between DU and CU).

The second virtualisation software used for the deployment of the scenario shown in Fig. 2 is Kernel-based Virtual Machine (*KVM*). *KVM*, an open source software, is a full virtualisation solution for Linux on x86 hardware containing virtualisation extensions (Intel VT or AMD-V). It consists of a loadable kernel module that provides the core virtualisation infrastructure and a processor specific module. Using *KVM*, it is possible to run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualised hardware: a network card, disk, graphics adapter, etc. For our purpose we characterised the *MME-VM*, *HSS-VM* and *SPGW-CU* with Ubuntu 16.04 with 4.8 generic kernel featuring a one core virtual CPU and 1 GB of RAM. Instead *CU-VM* hosts Ubuntu 14.04 with 3.19 low-latency kernel featuring a 8 core virtual CPU and 16 GB of RAM. In *PC 1* we bridge the management physical interface with a first interface of each VMs in passthrough source mode and the physical data plane interface with a second interface of each VMs in bridge source mode. In the passthrough source mode option, a virtual function of a Single-Root Input/Output Virtualisation (SRIOV) capable Network Interface Controller (NIC) is attached directly to a target VM without losing the migration capability. Therefore all the packets are sent directly to the network devices. In the bridge source mode option, packets whose destination is on the same host physical machine where they are originated from are directly delivered to the target device. Regarding the *CU*, because no internal host communication is needed, we set three different virtual interfaces in passthrough source mode each connected to three different physical interfaces. Thus, the NICs used for virtualised EPC are:

- passthrough adapter *enp0s3* to physical interface *eno2* with subnet 10.30.x.x (used for the management plane);
- bridge adapter *enp0s8* to *vboxnet0* interface with subnet 192.168.x.x (used for internal EPC service configuration and relationship);

The NICs used for the virtualised CU are:

- passthrough adapter *enp0s3* to physical interface *eth5* with subnet 10.30.x.x (used to set S1-C and S1-U interface between CU and MME and between the CU and SPGW respectively).
- passthrough adapter *enp0s9* to physical interface *eth1* with subnet 10.10.x.x (used to set the fronthaul between DU and CU).

Finally, *Docker* has also been used for the deployment of the scenario shown in Fig. 2. *Docker* is an open platform for developers and it is a mechanism that helps in isolating the dependencies per each application by packing them into containers. Containers are more scalable to deploy than virtual machines. Virtual machines have a full OS with its memory management installed with the associated overhead of virtual device drivers. Containers are therefore smaller than Virtual Machines and enable faster start up with better performance, less isolation and greater compatibility which is possible due to sharing of the hosts kernel. *Docker* containers can share a single kernel and share application libraries. Containers present a lower system overhead than Virtual Machines and the performance of the application inside a container is almost the same as compared to the same application running on a Physical Machine but better as compared to Virtual Machine.

There are different types of network modes available to connect *Docker* container with the host machine or to an external host. We connect our container through *host network* which uses the same protocol stack as the host device is using. A considerable advantage of using *Docker* containers is that we can bypass the overhead of bridge adapter used in previous approaches. Because OAI set many system variable values at run time, we run *Docker* container with privileged mode so that the container has got write permissions to set system variables. The following interfaces ((i.e., host's network interfaces) are set up in the utilized *Docker* containers:

- Physical interface *eth5* with subnet 10.30.x.x (to set S1-C and S1-U interface between CU and MME and between the CU and SPGW respectively).
- Physical interface *eth1* with subnet 10.10.x.x (used to set the fronthaul between DU and CU).

IV. Experimental Results

This section presents the experimental results obtained in the considered scenarios for both fronthaul allowable latency and jitter budgets.

Fig. 3 shows the fronthaul allowable latency budget for the considered virtualisation methods with different signal bandwidth values (i.e., 5 MHz and 10 MHz). Using *VirtualBox*,

the fronthaul allowable latency budget is 40μs for 5 MHz signal bandwidth. For 10 MHz signal bandwidth, CU and DU never communicate. This is due to the large number of samples generated at DU which cannot be handled with current configuration of the considered *PC 2* in which *CU-VM* is deployed.

If *KVM* is used, the fronthaul allowable latency budget is 190 μs for 5 MHz bandwidth and 140 μs for 10 MHz bandwidth, respectively. By using *Docker*, the fronthaul allowable latency budget is 35 μs in the case of 5 MHz bandwidth and 165 μs for the MHz bandwidth. Thus, if *VirtualBox* is used, the fronthaul allowable latency budget is very low when compared to when the CU is deployed in other virtualisation methods. By using *KVM* and *Docker* the allowable latency budget is close to the 3GPP constraint specified in TR 38.801. This is mainly due to how the packets are forwarded by the host network interface to the virtualized one and how they are managed. Regardless of the utilized virtualisation methods, the fronthaul latency budget is also a function of the signal bandwidth. Such dependence is due to the heavier processing required by the higher number of PRBs.

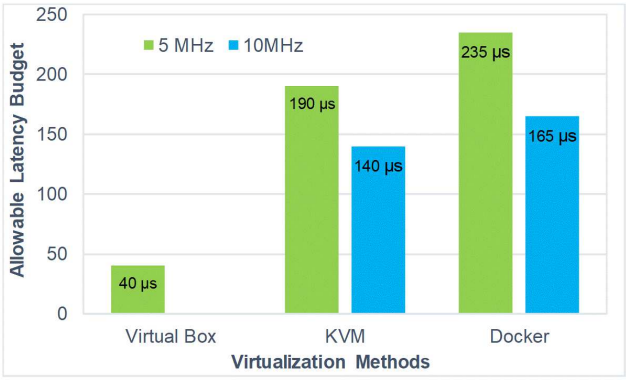

Fig. 3. Fronthaul allowable latency budget

While fronthaul latency requirements for different functional splits are specified by the 3GPPP [3], fronthaul jitter budget has not been fully evaluated yet.

Fig. 4 shows the obtained fronthaul allowable jitter budget results using the three virtualisation methods as above when the jitter is applied to a latency value close to the fronthaul allowable latency budget. With *VirtualBox*, the experiments are carried out by setting a fixed latency on the fronthaul link equal to 20 μs for 5 MHz signal bandwidth. The obtained fronthaul allowable jitter budget is 25 μs and no communication was observed in case of 10 MHz signal bandwidth. In the *KVM* case, the experiments are carried out by setting a fixed latency on the fronthaul link equal to 170 μs for a 5 MHz signal bandwidth and equal to 120 μs for a 10 MHz signal bandwidth. The fronthaul allowable jitter budget is equal to 20 μs for both signal bandwidths as shown in Fig. 4. With the *Docker*, the experiments are carried out by setting a fixed latency on the fronthaul link equal to 220 μs for 5 MHz signal bandwidth and equal to 150 μs in case of 10 MHz bandwidth. The obtained fronthaul allowable jitter budget is 25 μs in the first case and 35 μs in the second one. Thus, for all the considered virtualisation methods the latency budget is negligibly impacted by the jitter.

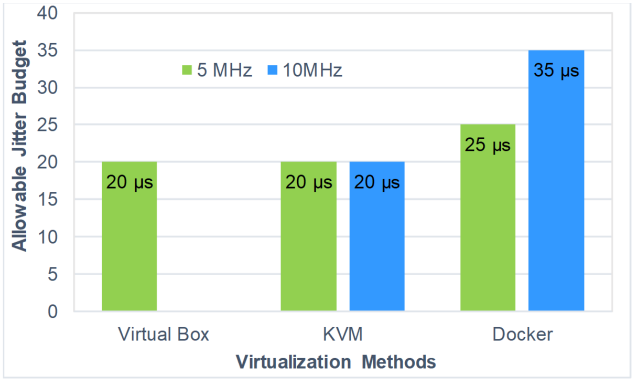

Fig. 4. Fronthaul allowable jitter budget with latency close to the allowable latency budget

To observe the impact of jitter on the fronthaul link, the latency value is set far from the budgets reported in Fig. 3. The obtained results are shown in Fig. 5 for all the considered virtualisation methods. When the CU is virtualised with *VirtualBox*, the experiments are conducted by setting a fixed latency on the fronthaul link equal to 20 μs for 5 MHz signal bandwidth. The obtained fronthaul allowable jitter budget is 25 μs and no communication was observed in case of 10 MHz signal bandwidth. Whereas, in *KVM* case, the latency value is fixed to 100 μs for 5 MHz and equal to 80 μs for 10 MHz signal bandwidth. The obtained fronthaul allowable jitter budget is 25 μs for the 5 MHz signal bandwidth and 30 μs for the 10 MHz case. With the *Docker*, the experiments are carried out by setting a fixed latency on the fronthaul link equal to 150 μs for 5 MHz signal bandwidth and equal to 100 μs in case of 10 MHz bandwidth. The obtained fronthaul allowable jitter budget is 40 μs and 35 μs for 5 MHz and 10 MHz, respectively. Thus the maximum allowable jitter budget is achieved by utilizing the Docker technology and it is about 40 μs.

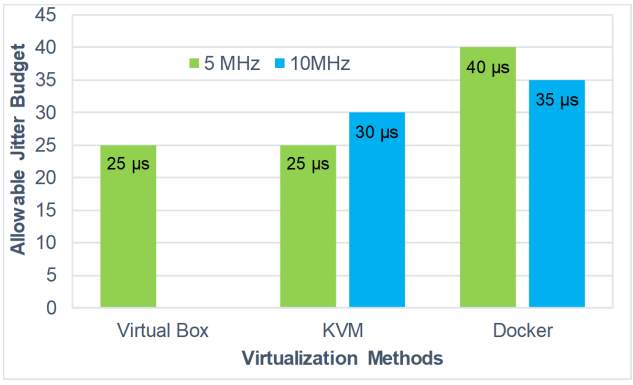

Fig. 5. Fronthaul allowable jitter budget with a fixed latency far from the allowable latency budget

V. Conclusions

This paper presented the experimental evaluation of the impact of virtualizing eNB functions on the fronthaul latency and jitter budget when different virtualisation methods are utilized. The experimental evaluation was performed in a testbed utilizing OpenAirInterface as mobile network software, desktop computers, USRPs, and LTE dongles.

Results showed that lighter virtualisation methods (e.g., *Docker*) are impacting the fronthaul latency budget for Option 7-1 (i.e., intra-PHY) split less than heavier virtualisation methods (e.g., *VirtualBox*). However, in all the cases, the fronthaul latency budget reduction depends on the considered signal bandwidth. The higher the bandwidth the higher the computations required the higher the fronthaul latency budget reduction. Furthermore, the performed experimental evaluation showed that a jitter of at most 40 μs can be tolerated.

Acknowledgment

This work has been partially funded by the H2020-ICT-2014-1 Wishful project (grant no. 645274).

References

[1] Ericsson mobility report, June 2015, last accessed Jan. 22, 2016.
[2] 5G radio access, June 2015, last accessed Jan. 22, 2016. http://www.ericsson.com/res/docs/whitepapers/wp-5g.pdf
[3] 3rd Generation Partnership Project; Technical Specification Group Radio Access Network, Study on new radio access technology; radio access architecture and interfaces, 3GPP TR 38.801 V2.0.0 (2017-03).
[4] Common Public Radio Interface (CPRI) Specification V7.0, Tech. Rep., 2015 (accessed on Jun. 11, 2017). http://www.cpri.info/downloads/
[5] 5G PPP, "View on 5G Architecture," White Paper, July 2016. [Online] Available: https://5g-ppp.eu/white-papers/ (accessed on Jul. 20, 2017)
[6] D. Chitimalla, K. Kondepu, L. Valcarenghi, M. Tornatore, and B. Mukherjee, "5G Fronthaul-Latency and Jitter Studies of CPRI Over Ethernet," J. Opt. Commun. Netw. 9, 172-182, 2017.
[7] L. Valcarenghi, F. Giannone, D. Manicone, and P. Castoldi, "Virtualized eNB latency limits," Proc. of ICTON, 2017.
[8] Small Cell Forum Document 159.07.02, Small cell virtualization functional splits and use cases, Small Cell Forum, January 2016 (accessed on Jul. 20, 2017)
[9] L. Valcarenghi, K. Kondepu, and P. Castoldi, "Time- Versus Size-Based CPRI in Ethernet Encapsulation for Next Generation Reconfigurable Fronthaul," J. Opt. Commun. Netw. 9, D64-D73, 2017.
[10] Next Generation Fronthaul Interface (1914) Working Group [Online] Available: http://sites.ieee.org/sagroups-1914/ (accessed on Jul. 20, 2017).
[11] A. Checko, H. Christiansen, H. Yan, Y. Scolari, G. Kardaras, M. Berger, and L. Dittmann, "Cloud RAN for mobile networks ? A technology overview," IEEE Commun. Surv. Tutorials, vol. 17, pp. 405-426, 2015.
[12] A. Blenk, A. Basta, M. Reisslein and W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," IEEE Commun. Surv. Tutorials, vol. 18, no. 1, pp. 655-685, Firstquarter 2016.
[13] http://www.openairinterface.org
[14] https://gitlab.eurecom.fr/oai/openair-cn
[15] https://gitlab.eurecom.fr/oai/openairinterface5g
[16] jFed http://doc.fed4fire.eu/getanaccount.html
[17] ARNO-5G Testbed arnotestbed.santannapisa.it
[18] CMCC, "Transport requirement for CU&DU functional splits options," R3-161813 (document for discussion), 3GPP TSG RAN WG3 Meeting #93, Goteborg, Sweden, 22nd-26th August 2016