Building virtualized 5G networks using open source software

Bruno Dzogovic
Oslo Metropolitan University
Norway
bruno.dzogovic@hioa.no

Van Thuan Do
Wolffia AS
Norway
Vt.do@wolffia.no

Boning Feng
Oslo Metropolitan University
Norway
boning.feng@hioa.no

Thanh van Do
Telenor and Oslo Metropolitan University
Norway
thanh-van.do@telenor.com

***Abstract*— The upcoming 5G mobile networks will not only bring high data rates but also deliver flexibility and adaptability, which is conveyed by the virtualization of the mobile network. Unfortunately, virtualization of mobile networks is not well understood, and the work described in this paper aims at investigating and elucidating the particular matter. As a starting point, OpenAirInterface, an open source mobile communication software is selected for first virtualization and then cloudification. In addition to the descriptions of the virtualization and cloudification, the paper also provides key findings and lessons learned of the experiments.**

Keywords—Cellular networks, Mobile networks, Virtual network function, Virtualized mobile networks, Software mobile communication, Open source mobile communication

I. Introduction

Since the emergence of GSM [1] in 1991 the popularity of mobile communication keeps increasing. The number of mobile subscriptions is currently over 7.6 billion corresponding to the world population and still increasing. To meet the demands on performance, coverage and quality of service the mobile community puts a lot of effort to constantly improve mobile communication both in terms of connectivity and services by introducing new technologies. 2G/GSM was replaced by 3G/UMTS [2], which in addition to voice and short message provides data connections. With rising demand on higher bit rates 3G was replaced by 4G/LTE [3], which offers sufficiently high data rate for the access to popular Internet services such as social networks, mail, instant messaging, maps, etc. Again, 4G will be soon taken over by 5G [4] which is aiming at responding to widest range of services and applications, ranging from *enhanced mobile broadband (eMBB)* to *massive machine-type communications (mMTC)* and *ultra-reliable and low-latency communications (URLLC)*.

In addition to advanced wireless access technologies, the two fundamental technology enablers of 5G are softwareization and virtualization of network functions, [5] and software-defined [6][7][8], programmable network functions as well as infrastructure resources. However, the virtualization of the mobile network is still in very early phases and the current acquired experience is sparse. Indeed, the feasibility is fully proven. This paper presents the work done at the Secure 5G4IoT Lab at the Oslo & Akershus University College within the scope of the H2020 SCOTT project which is aiming at acquiring knowledge and experiences on the network virtualization, while paving the way for 5G. To achieve its objectives, the project decides to adopt an open source software approach. In fact, to build an earlier 5G mobile network in a lab environment, an open source 4G/LTE is selected first for softwareization and then for virtualization. The paper starts with a brief review of open source mobile communication software and follows with the description of the current open source software 4G/LTE OpenAirInterface integration. One of the major step of the project will be the virtualization of OpenAirInterface. The main target of the paper is clarification of the mobile network cloudification. The findings and lessons learnt will also be conferred in a comprehensive way. The paper concludes with a conclusion which also encompasses suggestions for further works.

II. Brief Review of Open Source Mobile Communication Software

A. 2G/GSM open source software

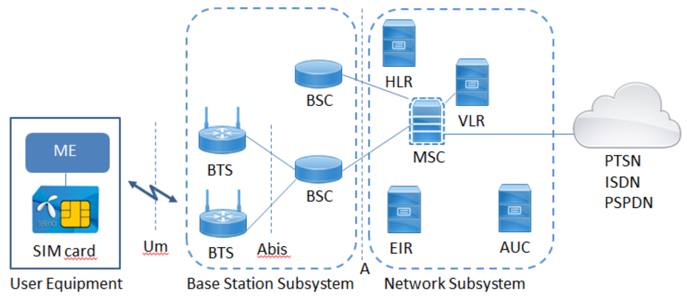

Figure 1 The GSM network architecture

Founded in 1998 the open source initiative quickly became popular and was spreading to a large variety of application areas such as browser, big data, cloud infrastructure, financial, business process management, CAD, billing, accounting, etc. it reached mobile communication by 2007 and in 2010 the development of OpenBTS [10], an open source GSM was completed and a pilot took place in Burning Man, USA. The second GSM open source software is OpenBSC, which had the first public demonstration in 2008. In 2014, OpenAirInterface, an open source LTE software from EURECOM was demonstrated at the Mobile World Congress in Barcelona. Short introductions will be successively provided for each of the open-source software.

The traditional GSM (Global System for Mobile Communications, originally Groupe Spécial Mobile) network is made of components called network elements such as Base Transceiver Station (BTS), Base Station Controller (BSC), Mobile Switching Center (MSC), Home Location Register (HLR), Visitor Location Register (VLR), etc. as shown Figure 1. These network elements are manageable dedicated logical entities including hardware and software components, which are built and delivered by telecommunication manufacturers.

With open source software, these network elements are realized by software components installed on generic personal computers or enterprise-grade servers. Open source communication software enables various research projects as the one described in [11], where a fake base station aka “IMSI catcher” was built.

OpenBTS

In this approach, a smallest GSM network is obtained by the provision of the air interface between the mobile phone and the base station, done by an open-source Base Transceiver Station software running on a commodity processor.

OpenBSC

This approach implements a software base station controller (BSC) [12], running on a commodity Linux PC which can be combined with off-the-shelf BTS hardware to establish a GSM network.

B. *4G/LTE open source software*

OpenAirInterface

It is a software standard-compliant implementation of a subset of Release 10 LTE for UE, eNB, MME, HSS, SGw and PGw [13], on standard Linux-based computing equipment (Intel x86 PC/ARM architectures), which is freely distributed by the OpenAirInterface Software Alliance under the terms stipulated by the OSA license model.

III. Current standard 4G/LTE OpenAirInterface installation

As shown in Figure 2, a current validated OpenAirInterface network installation consists of three nodes as follows:

- A commercial LTE enabled mobile equipment (*COTS UE*)
- An LTE base station, realized by an *OAI soft eNB* running on a low latency Linux commodity computer
- An LTE core network, realized by an *OAI soft EPC* running a Linus commodity community computer.
- Undoubtedly, the dedicated hardware network elements are now replaced by software network functions such as soft eNB and soft EPC.

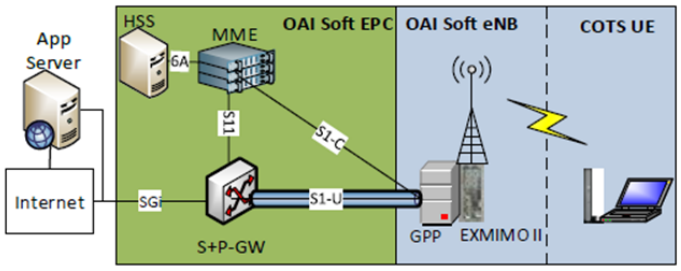

Figure 2 Standard OAI architecture (source: OSA [13])

IV. Virtualization of OpenAirInterface

To establish an early 5G network, we carried out the virtualization of the OAI software stack of network functions, mentioned in the previous sections.

A. *Virtualization*

Virtualization refers to the process of abstracting computing resources such that multiple applications can share a single physical hardware. Quite often, virtualization relates to server virtualization, where a particular physical server is abstracted and decomposed into virtual entities. The virtual constituents are assembled into a hypervisor, which makes up the virtualization software like KVM (Kernel-based Virtual Machine) [14], VirtualBox [15] or VMware [16]. A virtual constituent can be a virtual CPU, virtual RAM or virtual NIC (Network interface controller) etc. Even storage can also be virtualized, and this allows alleviated sharing of resources between users. Furthermore, a network can be virtualized as well by creating virtual links, subnetworks, gateways and layer-2 bridges, etc.

The major obstacle of the virtualization is the introduced overhead, as well as complexity. In fact, the OAI network functions are high-performance applications that require low latency and near real-time response. When running within a Virtual Machine (VM), a network function may perform some I/O calls or other operations involving system calls. The system calls go through additional layers of abstraction, introduced by virtualization, which incurs overhead resulting to slower responses.

B. *Containerization*

To circumvent the issue of overhead we choose to use containers instead of virtual machines.

Containerization [17] also called container-based virtualization and application containerization is an operating system level virtualization method for deploying and running distributed applications without launching an entire VM for each application. Multiple isolated systems, called containers, are instead run on a single control host, accessing a single kernel. As shown in Figure 3, containers include the components necessary to run the software, such as files, environment variables and libraries. Since containers run on the same operating system as the host computer, they are more efficient than VMs that are running different operating systems. In fact, the key benefits of containers are efficiency gains for memory, CPU and storage. Another major advantage is the fact that containers can be created much faster than hypervisor-bases instances.

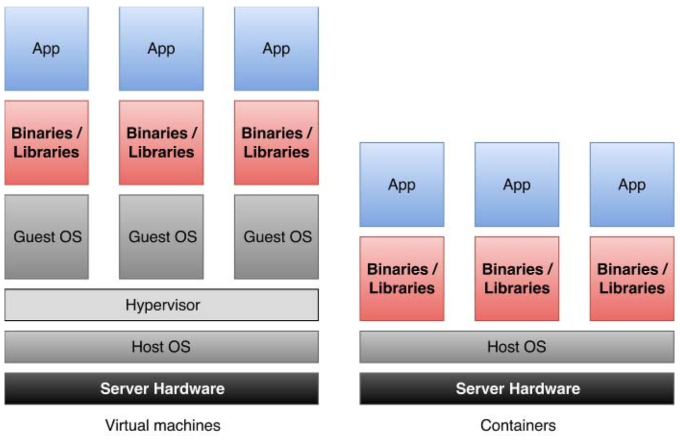

Figure 3 Virtualization versus Containerization

C. *Containerized OAI deployment at the Secure 5G4IoT*

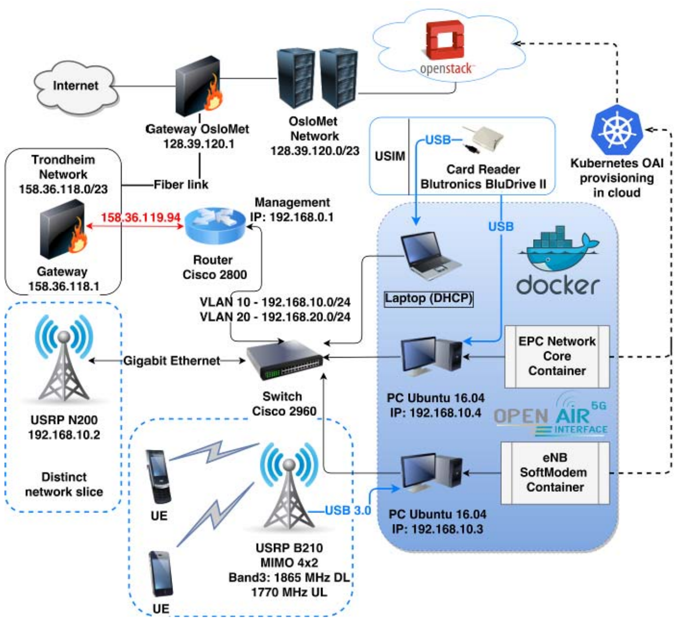
openstack
Internet
Gateway OsloMet 128.39.120.1
OsloMet Network 128.39.120.0/23
USIM
USB
Card Reader Blutronics BluDrive II
Kubernetes OAI provisioning in cloud
Trondheim Network 158.36.118.0/23
Fiber link
158.36.119.94
Management IP: 192.168.0.1
Router Cisco 2800
Gateway 158.36.118.1
USB
docker
Laptop (DHCP)
VLAN 10 - 192.168.10.0/24
VLAN 20 - 192.168.20.0/24
Gigabit Ethernet
EPC Network Core Container
Switch Cisco 2960
USRP N200 192.168.10.2
PC Ubuntu 16.04 IP: 192.168.10.4
OPEN AIR INTERFACE 5G
eNB SoftModem Container
Distinct network slice
UE
USB 3.0
PC Ubuntu 16.04 IP: 192.168.10.3
USRP B210 MIMO 4x2 Band3: 1865 MHz DL 1770 MHz UL
UE

Figure 4 Deployment of OpenAirInterface core network and eNB in Docker containers

The deployment of OpenAirInterface in containers can be performed in several ways. The easiest and simplest approach is to run the OpenAirInterface core network in a single container, i.e. both EPC (Evolved Packet Core) including HSS (Home Subscriber Server) and eNB. The advantage of such deployment is that the core network can be pre-configured and run automatically instantaneously. Therefore, everything can be initiated within matter of seconds. To ensure scalability, the OpenAirInterface core network can entitle a separate container to deploy the HSS in few replicas, supporting database load-balancing and installing the other core network functions in a separate container(s) on the same or even different physical hosts, while eNB is running on a third container remotely (from a distributed cloud edge network).

At the Secure 5G4IoT lab, we adopt a configuration which uses a separate container to deploy the entire EPC, as well as another one for the eNB, as shown in Figure 4.

There are few container implementation possibilities, but in the particular testbed, Docker [17] is used for the containerization of the OpenAirInterface. Docker is a very powerful tool for software containerization, which introduces containers that can wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, and system libraries. Factually, anything that can be installed on a server. This guarantees that the software will always be executed in the exact form, disregarding the environment. Containers running on a single machine share the same OS kernel; they start instantly and use less RAM. Images are constructed from layered filesystems and share common files, making disk usage and image downloads much more efficient.

Comparatively to virtual machines, a container is a virtualization instance in which the kernel of an operating system enables multiple isolated user-space instances. Moreover, containers do not need to run a complete operating system (OS) image for each instance. Instead, containers are able to run separate instances of an application within a single shared OS. This new feature provides the flexibility to build instantaneously and move applications without the need to rewrite or redeploy their code, which makes up for faster integration and access to analytics, big data and services. It also enables advanced techniques of deployment and orchestration in form of clusters and stacked services.

As shown in Figure 4 the testbed consists of the following nodes:

- **EPC:** 192.168.10.3; PC running Ubuntu 16.04
- **eNB:** 192.168.10.4; PC running Kali Linux connected through USB 3.0 interface with a USRP B210 [18] and through Ethernet with a USRP N200 [19] **Error! Reference source not found.**
- Two smartphones Huawei P9 lite, equipped with self-programmed Milenage algorithm SIM cards
- Blutronics BluDrive II SIM card programming device
- Cisco 2800 router and Cisco 2960 switch, separated in two VLANs

The testbed is situated at Oslo Metropolitan University, where a dedicated cloud infrastructure is built in a datacenter. As shown on Figure 4, the host running the EPC reaches the eNB through the routing plane in two possible ways: one is through layer-3 networking and L2TP tunneling encapsulation, consequently. Due to different performance metrics, the separate solutions differ with regard to the scenario that they are appointed to. For example, the usage of network overlay introduces additional latency and therefore, it would only be suitable for scenarios where the communication does not require low latency, i.e. the case of some Internet of Things devices for home appliances.

For a complete Layer-3 networking solution, the SDN *Calico* [20] is used for formulating a Network Function Virtualization. Calico is simplified SDN (Software-defined Networking) solution that can enable various features of networking, including unparalleled scalability, security via policy-based networking and routing in cloud networks, as well as physical server-based networking. Calico provides secure network connectivity for containers and virtual machine workloads. It creates and manages a flat layer 3 network, assigning each workload a fully routable IP address. Workloads can communicate without IP encapsulation or network address translation for bare metal performance, easier troubleshooting, and better interoperability. In environments that require an overlay, Calico uses IP-in-IP tunneling or can work with other overlay networking such as Flannel or OvS. Calico also adds dynamic implementation of network security rules. Using simple policy language, it is possible to achieve fine-grained control over communications between containers, virtual machine workloads, and bare metal host endpoints. This allows extensive security resolution for IoT verticals, since the current are not regulated due to lack of research and solutions [21].

Proven in production at scale, Calico features integration with Kubernetes, OpenShift, Docker, Mesos, DC/OS, and OpenStack. To integrate Calico with OpenStack, *Etcd* is installed as a provider of a distributed key/value database that is accessible from all compute hosts and Neutron networking servers. The next entity being installed is the Felix (Calico agent), which runs on each compute host and reads information from Etcd that stipulates the workloads and their properties. Consequently, the BIRD routing daemon [22] runs on each compute host, propagating local workload routes to other compute hosts and infrastructure routers. To successfully integrate Calico in the Neutron networking of OpenStack, the Calico driver is installed and executed on each machine where the Neutron server runs i.e. the three servers that are constructed to provide high availability. The driver helps Etcd and Felix to comprehend the Neutron networking operations in OpenStack e.g. instance, security, subnet operations etc. The last agent required is the Calico DHCP for dynamic address assignment, which also runs on each compute host, providing DHCP service for the locally-hosted workloads. At this point, Calico enables a policy for connection between the particular containerized OpenAirInterface network core instances and the outside world, including the remote eNB that is set at the network edge.

The L2TP encapsulation for lower-security IoT applications is resolved with *Open Virtual Switch (OvS)* network overlay. The traffic is thus unencrypted, without implementation of IPSec encryption. Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license that can enable interconnection between containers or virtual machines. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag. It has also been integrated into many virtual management systems including OpenStack, openQRM, OpenNebula and oVirt. In addition, it is designed to support distribution across multiple physical servers similar to VMware's vNetwork distributed vSwitch or Cisco's Nexus 1000V [23]. A script is built to automate the initialization process of the tunnel between the two remote containers on each start of the operating system, which enables reachability to remote locations for particular IoT network verticals, as shown in Figure 5.

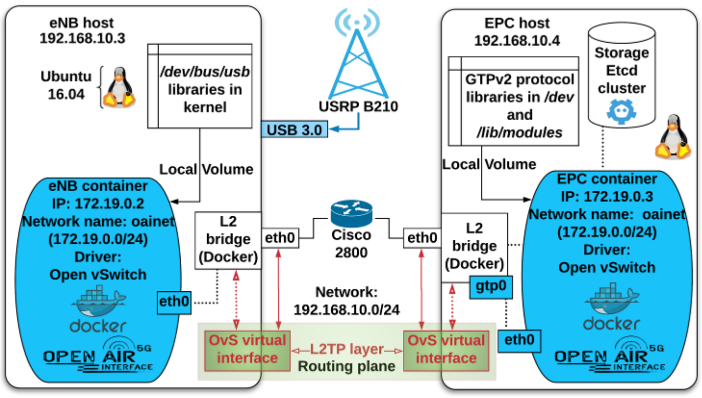

Figure 5 Interconnection between remote containers using network overlay with Open VSwitch

V. CLOUDIFICATION OF OPENAIRINTERFACE

The Oslo Metropolitan University has its own cloud infrastructure based on OpenStack. The cloud consists of 10 compute nodes, 3 controller nodes and 6 Ceph-OSD storage servers. It is also equipped with a complete monitoring solution, based on Prometheus and Grafana. There is also an infrastructure alerting system, comprised of Elasticsearch Kibana, InfluxDB and Grafana.

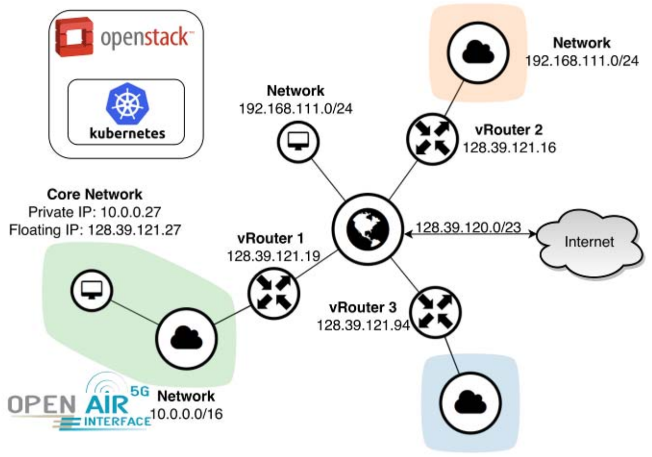

Figure 6 OpenStack deployment of OpenAirInterface core network

As depicted on Figure 6, the OpenStack cloud is built on top of Kubernetes clusters and is managed remotely. *Kubernetes* [24] is a powerful system, developed by Google, for managing containerized applications in a clustered environment. It aims at providing better ways of managing related, distributed components across varied infrastructure. Kubernetes, at its basic level, is an orchestration system for managing containerized applications across a cluster of nodes. Therefore, the OAI pre-built images can be instantiated in a remote cloud automatically, with a simple creation of a YAML template that is very practical for the deployment of remote eNB components.

To integrate the OpenAirInterface network core in the OpenStack Neutron networking component, without network overlay but using a network underlay, OpenStack Heat templates are used. The templates (HoTs) are pre-created to manage multiple composite cloud applications and organize them as a stack of virtualized entities i.e. containers. The Heat orchestration component enables service for managing the entire lifecycle of infrastructure and applications within OpenStack clouds [25]. Creating multiple templates, it can provide different network slices for different tenacities, merging the underlying operation of the OvS network in the OpenStack's Neutron networking component. This way, the procedure allows automation of the deployment within the disposition of the Continuous Integration/Delivery (CI/CD) paradigm.

One addressed issue is the support for SCTP (Stream Control Transmission Protocol) in OpenStack, which is the protocol through which the eNB communicates with the MME in the core network. The virtual machine in which the OpenAirInterface core is running, needs to have the SCTP module enabled, which is integrated into the Linux kernel. As the module is loaded, it is possible then to create SCTP sockets inside containers with standard Linux tools. A Linux Ubuntu image is pre-built with all the modules required for the core network to run. Moreover, the OpenStack cloud needs a special SCTP security group rule that enables the traffic over floating IPs. On the main controller node, a security group rule is thus created with the following command executed at the main controller node: `openstack security group rule create --protocol sctp`

The --protocol flag allows the creation of a security rule based on the protocol type employed, in this case SCTP. Also, this feature allows avoiding the tunneling of traffic with a network overlay, which can add a significant network latency and traffic incongruity. On the other hand, the S1 traffic between the eNB and the MME/SPGW can also be encrypted using IPSec at the tunnel, and the viability of implementing OvS tunnel is still possible. However, this combination may be appropriated for applications that are not restricted in terms of low-latency or real-time operation. To successfully deploy the core network in OpenStack using Heat orchestration templates, the OpenStack version should at least be "Kilo" from 30 April 2015. However, the particular version of OpenStack is "Ocata" from 22 February 2017, which has added support for multiple other OpenStack components [25], as well as full support for SCTP. As the Ubuntu 16.04 image is configured to support Heat templates, exported to QCOW2 format, it is imported to the OpenStack cloud using the OpenStack "Glance" component for image creation. Because the OpenAirInterface core is a demanding deployment, it requires dedication of more available resources to the tenant that runs the process. For that purpose, an instance with extra-large flavor is assigned, namely: 8 vCPUs, 16 GB of RAM and 160GB disk space.

The 5G4IoT network edge is simulated as a physical server on a distinct network, which is connecting from Trondheim to Oslo via optical link and provides direct connection to the University network via different routing domains (as indicated on Figure 7).

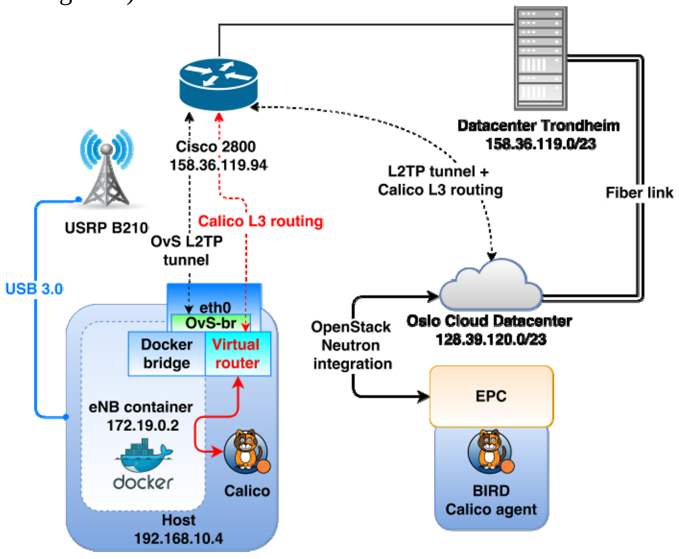

Figure 7 Cloudification of the OpenAirInterface EPC

The EPC is running in the cloud, which currently communicates to the remotely connected eNB through multiple routing domains. The end-to-end latency is measured via the hops required for the physical routing plane to reach the Calico virtual routing plane in the OpenStack Neutron. The OvS and Docker bridges are eliminating the additional latency overhead due to the direct bridging with the physical interface of the host at which they are running.

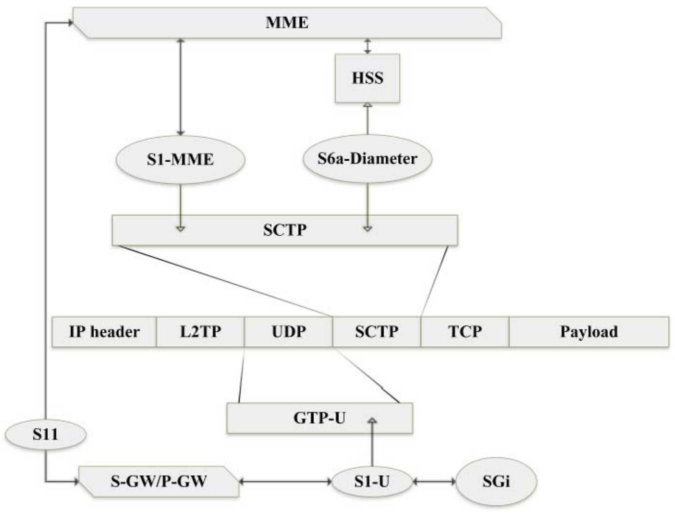

Figure 8 Customizing the TCP header due to encapsulation

Consequently, the encapsulation of multiple protocols yields a bigger packet size and the IP header is thus modified. The TCP transmission window is also slightly increased to accommodate the requirements for bigger IP fragments. As depicted on Figure 8, the L2TP protocol requires increased MTU size to acclimatize larger packet transmission, particularly of at least 1648 octets. Also, the SCTP protocol can be categorized subsequently in this class, which adds supplementary size to the header because of its encapsulation in the UDP protocol. Additionally, the UDP also has the GTP-U protocol encapsulated in its header and they altogether form the complete packet size required to establish a successful connection between the eNB and EPC.

VI. Key findings and Lessons learned

Through the experiments, a few interesting important findings and deductions were acquired as follows:

Meticulous planning is essential

The installation of OpenAirInterface core network in a Docker container is a process that demands detailed groundwork. The aim behind packing a software into a container is to enable immutability and consistency, which is to allow the software to run on any OS platform and/or kernel, as well as virtualized environment. Therefore, establishment of a basic underlying image for the container is crucial. Particularly, OpenAirInterface is heavily tested by the open-source community on Ubuntu 14.04 and 16.04, with kernel versions 3.4, 4.7.x to 4.8.x. Hence, building a Docker image out of those specific Linux versions will yield stability for the core network operation, as well as the eNB container running on a separate machine.

Hardware performance tuning

In addition to the operating system demands, there are various constraints and requirements that must be met before proceeding with the core network deployment, as follows:

- The Linux kernel must be low-latency, because the application deployed works in real-time and any discrepancies of such character are unacceptable.
- The underlying hardware must be perfected to support real-time applications. This fathom securing consistent CPU speed without any fluctuations, which can implicate overclocking. The CPU is configured by default to work in power-saving modes, that when not fully utilized, downclocks the unit in order to save power. These modes are known as C-states and P-states [26], which must be disabled in the BIOS of the underlying motherboard. As for best operation, the contemporary Intel CPU employs 4 cores. The units support hyperthreading, which can disturb the real-time operation of the network core, while balancing the workload from core to core in dependence of the threads used by particular processes in the operating system. However, if the CPU is overclocked and set to work on a single frequency, in this case 3.0 GHz, the fluctuations of the speed will unlikely occur, despite the usage of hyperthreading. Also, hyperthreading is desirable if multiple containers are run and scaled in a cluster for automatic deployment using an orchestrator (i.e. Kubernetes), and if the OpenAirInterface eNB is replicated into multiple instances for different network slices.
- The RAM memory needs a consistent timing control, which has a CAS latency set to as minimal as possible, without disturbing the motherboard's north-bridge-to-RAM frequency ratio. If the read and write speeds of the RAM are slow, then the real-time function of the network core is distraught and will cause failures or unpredictable behavior. Derisory read/write RAM timings also cause bottlenecking of the eNB radio access, especially at the uplink (UL) channel.

Appropriate LTE frequency band must be selected

Due to the hardware limitations, the LTE frequency band selection can play a crucial role in successful deployment of a stable environment. The band 7 (2500-2570 MHz uplink and 2620-2690 MHz downlink, with 120 MHz duplex spacing) offers channel bandwidths of 5, 10, 15 and 20 MHz. However, the increasing of the bandwidth requires much more powerful CPU, and the only bandwidth that can be used with the current hardware configuration is the minimal 5 MHz. Despite that, the particular PC does not provide the required resources to run the eNB base station at 2.6 GHz, which cancels operation after irrelevantly short period. Lowering the frequency, as well as the channel bandwidth, can enable the PC to stabilize the operation of the eNB instance. Therefore, the band 3 is more appropriate, which does not require as much computing resources as the higher bands. With the range from 1710-1785 MHz for uplink and 1805-1880 MHz uplink, the band 3 offers more channel bandwidths, namely: 1, 3, 4, 5, 10, 15 and 20 MHz. The both tested bands work with FDD duplex mode, with the difference that a better stability is achieved within the lower frequencies because of the diminished utilization of computing resources.

Increasing the MTU of transmitted packets on all interfaces is obligatory

As previously stated, the default MTU of 1500 octets will yield the UE unable to utilize the HTTP protocol for browsing the Internet. With slight increase of the MTU (around 1648 octets), the desirable results are achieved within satisfactory levels of operation. However, if the MTU is increased more, the risk of fragmentation attack vulnerability increases in straight-proportional relations to the packet size, and thus it should be set to the lowest possible suitable levels. The process of creation of virtual interfaces for the OvS and Calico SDN is automated within a script that also sets the adequate MTU size to 1648 octets.

VII. Conclusion

This paper describes a successful attempt to build an earlier 5G mobile network using OpenAirInterface, a 4G/LTE open source. However, instead of virtualization a containerization of the OpenAirInterface was carried out in a satisfactory way, although a few challenges have been encountered and considerable efforts were requested to surmount them. The cloudification was also performed successfully and automation scripts are generated for further redeployment. The next step will be to explore the flexibility and dynamicity brought by the virtualization and cloudification of the mobile network, as well as migration of the network core and the eNB to the cloud and network edge for improved performance and decreased latency. The concept of network slicing, i.e. to set up multiple logical mobile networks on the same hardware network infrastructure will be experimented and guidelines for network slice configuration will be elaborated. The dynamicity of the network slice establishment and termination will also be tested and verified. The generic orchestration process is a central topic that calls for more research.

REFERENCES

[1] Mouly, M. & Pautet, M-B: *The GSM System for Mobile Communications*, ISBN-13: 978-0945592150 ISBN-10: 0945592159

[2] Hillebrand, F: GSM and UMTS: The Creation of Global Mobile Communication, ISBN: 978-0-470-84322-2, Wiley, Oct 2001

[3] Cox, C.: *An Introduction to LTE: LTE, LTE-Advanced, SAE and 4G Mobile Communications*, ISBN: 978-1-119-94353-2, Wiley, Mar 2012

[4] 5G Infrastructure Public Private Partnership (5G PPP): *View on 5G Architecture* (Version 2.0), 5G PPP Architecture Working Group - 2017-07-18

[5] ETSI: GS NFV 002 *Network Functions Virtualization (NFV)*; Architectural Framework, v.1.1.1, 10-2013

[6] Open Network Foundation (ONF): *SDN architecture*, Issue 1 June, 2014 ONF TR-502

[7] Open Network Foundation (ONF): *SDN architecture, Issue* 1 June, 2014 ONF TR-521

[8] Open Network Foundation (ONF): *Applying SDN Architecture to 5G Slicing*, Issue 1 April 2016 ONF TR-526

[9] H2020 SCOTT project: https://scottproject.eu/

[10] Iedema, M: *Getting Started with OpenBTS*, ISBN 978 1 491 91065 8, published by O'Reilly Media, Inc, Copyright 2015 Range Networks

[11] Do, Thanh van, Nguyen, Hai Thanh Nguyen, Nikolov Momchil and Do, Van Thuan: Detecting IMSI-Catcher Using Soft Computing in Communications in Computer and Information Science, ISSN 1865-0929 ISSN 1865-0937 (electronic)

[12] Osmocom, *OpenBSC.* [Online]
Available at: *https://osmocom.org/projects/openbsc/wiki/OpenBSC*
[Accessed December 2017]

[13] OpenAirInterface Software Alliance, *OpenAirInterface.* [Online]
Available at: *www.openairinterface.org/*
[Accessed December 2017]

[14] Redhat, Inc., *KVM – Kernel Based Virtual Machine*, Copyright. [Online]
Available at: http://www.redhat.com
[Accessed November 2017]

[15] Oracle Corporation, *Oracle VM VirtualBox User Manual, Version 5.2.2.* [Online]
Available at: *http://www.virtualbox.org*
[Accessed December 2017]

[16] VMware: *Enterprise Java Applications on VMware - Best Practices Guide*, 2011

[17] Docker: *Docker for the Virtualization Admin*, 2016; htttp: www.docker.com

[18] Ettus Research, Inc., *USRP B200.* [Online]
Available at: *https://www.ettus.com/product/details/USRP-B200mini-i*
[Accessed November 2017]

[19] Ettus Research, Inc., *USRP N200* [Online]
Available at: *https://www.ettus.com/product/details/UN200-KIT*
[Accessed November 2017]

[20] Tigera Inc., 2017. *Calico.* [Online]
Available at: *https://projectcalico.org/*
[Accessed November 2017]

[21] Siddiqui, M. S., Escalona, E., *Policy Based Virtualised Security Architecture for SDN/NFV enabled 5G Access Networks*, IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2016

[22] Kyuongha, K., Yanggon, K., *The Security Appliance to BIRD software router, ICUIMC '14 Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, Article No. 37 , Siem Reap, Cambodia — January 09 - 11, 2014

[23] Linux Foundation Collaborative Project, 2016. *Open VSwitch* [Online] Available at: *http://www.openvswitch.org*
[Accessed December 2017]

[24] Kubernetes, 2017. *Kubernetes.* [Online]
Available at: *https://kubernetes.io*
[Accessed 20 November 2017]

[25] OpenStack, 2017. *OpenStack.* [Online]
Available at: http://www.openstack.org
[Accessed December 2017]

[26] Sistla, K. V., Rowland, M., Varma, A., Steiner, I. M., Bace, M., Borkowski, D., Garg, V., Akturan, C. & Ananthakrishnan, A. N., *Dynamically Modifying a Power/Performance Tradeoff Based on Processor Utilization*, U.S. Patent US9760409B2, issued 12 September, 2017

ACKNOWLEDGEMENT:

This paper is a result of the SCOTT project (www.scott-project.eu) which has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway.