Design and Validation of a Multi-service 5G Network with QoE-aware Orchestration

Marco Gramaglia
Universidad Carlos III Madrid
Spain
mgramagl@it.uc3m.es

Ignacio Labrador Pavón
ATOS
Spain
ignacio.labrador@atos.net

Francesco Gringoli
University of Brescia
Italy
francesco.gringoli@unibs.it

Gines Garcia-Aviles
IMDEA Networks Institute &
Universidad Carlos III Madrid
Spain
gines.garcia@imdea.org

Pablo Serrano
Universidad Carlos III Madrid
Spain
pablo@it.uc3m.es

ABSTRACT

While the work on architectural and algorithmic solutions for 5G has reached a good maturity level, the experimental work lags behind, in particular on the development of open source solutions. In this paper, we describe our implementation experiences when deploying a small-scale multi-service network prototype, used to demonstrate some selected advanced features of 5G Networking. We describe our implementation experiences supporting two heterogeneous services over two independent slices, namely, video streaming and augmented reality, showcasing key features such as multi-slice orchestration, RAN slicing and support for local breakout. While the applications running the services rely on proprietary code, the core of our implementation is completely open-source.

CCS CONCEPTS

• **Networks → Network management**; **Mobile networks**;

KEYWORDS

Network slicing; Network Management; Local breakout

ACM Reference Format:
Marco Gramaglia, Ignacio Labrador Pavón, Francesco Gringoli, Gines Garcia-Aviles, and Pablo Serrano. 2018. Design and Validation of a Multi-service 5G Network with QoE-aware Orchestration . In *12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH '18), November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3267204.3267216

1 INTRODUCTION

While specification and documentation activities regarding the 5th generation mobile systems (5G) are progressing at a rapid speed, practical experiments are progressing at a slower pace. Apart from the obvious reasons due to the relatively high development costs, another (and partly related) issue that precluded the prototype of mobile system is the unavailability of resources: mobile equipment is typically expensive, and the frequency bands are licensed. Because of this, operators were the only performing experimentation with cellular systems, while the academia was restricted to technologies based on the ISM band (see e.g. [20] for a recent survey on experimentation with 802.11). One technology that helped this ISM-based experimentation is Software Defined Radio (SDR), which enables researchers to implement all layers of a wireless protocol stack.

The use of SDR has enabled the implementation of novel access schemes and communication paradigms (such as e.g. cognitive radio, full-duplex), but prototypes have been limited to the lower layers of the protocol stack (i.e., the MAC layer), the main reason being the lack of interest on developing complete systems over these prototype boards, given the high development costs. These costs are further exacerbated for the case of mobile systems, as these are characterized by relatively complex protocol stacks (in contrast to e.g. TCP/IP). The situation, though, has recently changed with the emergence of software platforms based on SDR that enable instantiating a complete mobile network, with Eurecom's OpenAirInterface (OAI) [15] and srsLTE [10] among the most popular initiatives. In fact, we have recently carried out an extensive performance comparison of these platforms [11], confirming their adequate performance for prototyping activities and their compatibility with commercial equipment.

The availability of these open projects has leveled up the playground, with now more players (academics, SMEs) being able to develop novel enhancements for cellular systems. This helps accelerating the development of solutions for 5G networking, which will boost the running of field trials under realistic conditions and bring better understanding about the performance of future mobile systems. For instance, thanks to the above mentioned OAI and srsLTE initiatives, we recently released POSENS, the first practical and completely open-source solution for end-to-end network slicing [9].

Despite the above, the gap between theory and practice for 5G networking is still large. In particular, while network slicing has been acknowledged as a key technology to efficiently support services with very diverse requirements [18], there are little experimental "hands on" reports on the use of this technology in practice. In this paper, we contribute to filling this gap by reporting on the development and validation of a multi-slice 5G network prototype, each slice serving a different application, and showcasing several features such as the reallocation of virtual network features or the use of local breakout (LB) to minimize delays [13]. We describe the hardware used, the software installed, and how the different building blocks are connected, providing in this way researchers and practitioners with a "how to" guide while reporting on our experiences and best practices for prototyping. We believe our results provide valuable information to boost the development of further 5G prototype initiatives.

1.1 Related work

We next provide a short summary of the most relevant initiatives to prototype mobile networks. We divide our revision of initiatives in the networking part considering the Radio Access Network (RAN), the Core Network (CN) and the Management and Orchestration (MANO) part.

Concerning *the RAN part*, the most relevant solutions are OAI [15], and the more recent srsLTE [10] from Software Radio Systems. OAI provides an implementation of a subset of LTE R10 elements, while the srsLTE open-source project provides a platform for LTE R8 experimentation, designed for maximum modularity. While in general the performance of OAI is better (both in terms of throughput and resource footprint), srsLTE's source code results easier to customize.

Regarding *the CN part*, the most relevant solutions are the ones associated with the above initiatives: srsLTE has recently released srsEPC, a light-weight CN implementation including the mobility management entity (MME), home subscriber server (HSS) as well as packet and serving gateways (P-GW and S-GW, respectively), while OAI provides the same elements for a basic EPC solution.

The *MANO* part has received a lot of attention from both the open-source community and the enterprises, with a variety of tools such as Open Baton [17], ONAP [7], OPNFV [8] or OSM [4]. They all rely on a Virtual Infrastructure Manager (VIM), an element that can be provided with solutions such as OpenStack [16].

While these solutions provide a subset of elements to implement a 5G mobile network, only recently [1, 6, 9] that researchers were provided with complete tools to implement end-to-end network slicing. Still, hand-on experience is largely missing in the literature, so in this paper we provide details on our experience in implementing specific elements envisioned by 5G Networking on a real life testbed.

1.2 Paper structure

The document is structured as follows: in Section 2 we describe the implemented network services and their baseline deployment in Section 3. Then we focus on some specific advanced items in Section 4 before checking them in Section 5. Finally, Section 6 concludes the paper.

2 SERVICES CONSIDERED

The deployed network provides two services over two network slices, with a focus on the QoS/QoE-aware control, and NFs virtualization and orchestration aspects. The motivation is to feature two different network slices on the cloud infrastructure: one with a reduced latency service and another one with a mobile broadband service. We show how an ETSI NFV MANO platform can be used to deploy, manage and orchestrate different services on different network slices, this including the dynamic re-orchestration of a particular NS forwarding graph, and the placement of certain VNFs in the appropriate host. In both cases, QoS/QoE aspects trigger the re-orchestration function. The software produced for this paper is based on [9] and it is mostly available on GitHub [1].

The two network slices are: *(i)* a Reduced Latency slice (i.e., URLLC), used to read real-time physical measurements triggered by Quick Response (QR) labels, and *(ii)* an enhanced Mobile Broadband (eMBB) slice, which serves contextual captions to streaming media, according to the user profile and surrounding context. Both slices are deployed on the same eNB and share the same spectrum. For the CN part, each tenant (i.e., a service) runs its own instance of the protocol stack (i.e., mobility management, gateways and the upper layers), performing the QoE/QoS policies needed by each scenario. [2] In the following, we describe each slice, listing

[1]https://github.com/wnlUc3m

[2]Although we deploy a multi-tenancy case where each service belongs to a different tenant, other multi-tenancy scenarios can be supported with our solution.

the most important orchestration-related features that have to be implemented (which will be detailed in Section 4).

2.1 eMBB slice: Video Streaming

The first service is hosted by an eMBB slice that has been designed to provide a service consisting of enriching a video streaming signal with context-based add-ons (e.g., subtitles or other graphical elements) depending on the user preferences (i.e., color, language), other conditions (e.g., hearing impairments) and surroundings (e.g., ambient noise). These profiles or environmental conditions can be understood here as QoS/QoE influence factors: the final user generates a trigger to explicitly request the service according to its preferences, or also, certain QoS metrics could automatically activate the service without an explicit user request.

The additional video features are activated by means of MANO procedures that dynamically add the necessary VNFs to the forwarding graph concurrently as the video is streamed. Besides the possible real-life applications of this service, our goal is to demonstrate three different orchestration-related functionality:
(i) On-boarding of the network service itself, i.e., the deployment of the necessary VNFs driven by service descriptors where the operational layout and requirements are defined; *(ii) Dynamic update of the NS Forwarding Graph (FG)* depending on QoS/QoE measurements (QoS/QoE-awareness); *(iii) Placement of VNFs to specific compute nodes*, to simulate the placement of NFs in the edge or core cloud depending on the service requirements.

2.2 URLLC slice: Augmented Reality

The Reduced Latency use case consists of an Android application that performs Augmented Reality (AR) using QR codes. In a real environment, those codes may be distributed in an industrial area close to the equipment where measurements of interest are obtained (e.g., pipes flow or pressure, electric measurements, tank levels, etc.). On each QR code decoding, the mobile terminal performs a request to get real time information that is shown on the mobile terminal.

To get low delays between the User Equipment (UE) and the information server, the latter shall be located close to the user, e.g. in the same element hosting the eNB, or in a edge cloud. This application also serves to demonstrate another fundamental orchestration-related aspect: the use of *local breakout*. That is, by selectively offloading flows from the eNB and routing them directly towards an edge cloud, the low latency traffic communicate with an edge deployment of the information server, incurring into smaller delays.

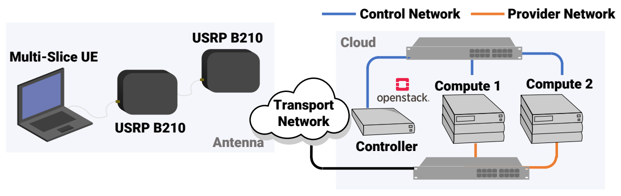

Figure 1: The deployed network setup.

3 BASE IMPLEMENTATION

We first describe what we refer to as the “basic” deployment, consisting of the hardware and software components that provide what we consider as the fundamental features of a 5G mobile network.

3.1 Hardware description

The orchestration platform runs on commodity hardware. More specifically: an Intel 12 Core i7-3930K PC @ 3.2 GHz with 32 GB RAM and 1 TB of storage and two twin AMD FX 8320 eight-core processor PCs @ 3.5 GHz with 32 GB RAM and 1 TB of local storage.

Besides the CPU, memory and storage characteristics, the main criterion to select this hardware is to have two identical nodes, to enable the VNF live-migration functionality. Figure 1 shows the network topology. The twin PCs work as compute nodes in the architecture, i.e., they run some of the VNFs (as we describe next, not all network functions will run as VNFs). The other PC acts as both network controller and storage node. One general-purpose switch is used to interconnect the nodes through a private isolated network, while another switch is used to connect the nodes with the Internet through an OpenVPN Server. All nodes run the Linux CentOS operating system (release 7.2).

The radio part of our deployment is based on two USRP B210 Cards by Ettus Research[3], one on the UE side and the other for the eNB. As the USRP B210 features two antenna ports (one RX and one TX), we cross-connected them using two SMA to SMA cables, adding a 30 dB of attenuation. The left part of Figure 1 represents the connected radio setup. The network functions for the radio part run as physical network functions (PNFs): the eNB runs on a bare metal server and the UE runs in a laptop (the specific software components are described in Section 3.3).

3.2 Management and Orchestration

The two services described in Section 2.1 and 2.2 run on the hardware platform described above, which provides the needed infrastructure (physical and virtual) to the different

[3]https://www.ettus.com/product/details/UB210-KIT

Table 1: Configuration of the Radio Front-end

Parameter	Value
Frame Type	FDD
Band	7
Downlink Frequency	2.68 Ghz
Uplink Frequency	2.56 Ghz
Number of RB in Downlink	75
Tx Gain	90 dB
Rx Gain	125 dB

VNFs and PNFs that compose the two slices. In order to provide the (ETSI NFV compliant) MANO implementation, we focused on a programmatic solution, discarding the usage of out-of-the-box orchestration platforms (Section 1.1) because of their complexity and the lack of certain required features.

More specifically, we implemented the VNF management and orchestration functions by directly building on the VIM APIs. Given the high needed extension required by our network slicing scenario, we selected an ad-hoc approach that allowed us to directly focus on the innovative functionality without the overhead of a more generic solution. Hence, the final approach for implementing the cloud architecture has been a hybrid approach with OpenStack (Pike release) to implement the VIM, and a programmatic solution to implement the remaining blocks. Thus, the VNF Manager and the NFVO components have been specifically developed for this work using the Java programming language together with the OpenStack4J library. Although not a fully fledged ETSI NFV orchestration platform, our implementation includes the needed functionality for advanced 5G Orchestration.

3.3 Softwarized Mobile Network

The mobile network architecture employed within deployment relies on two well-known software implementations of the LTE Stack: OAI [15] and srsLTE [10]. The former is an implementation of an LTE mobile network (including RAN and Core), while the latter is an open source implementation of a UE and the eNB. We describe here the setup that involves the UE implemented using the srsUE software.

As discussed in Section 3.2, the Radio Access Network functions take advantage of the USRP SDR card that use it (on both UE and eNB sides) to modulate and demodulate the in-phase and quadrature (IQ) samples produced and received by the software on one of the available licensed bands (in this case, we use Band 7 at 2.6 GHz). The full configuration parameters for the radio front-end are listed in Table 1.

On the network infrastructure side, the SDR card is used by SRS eNB to provide connectivity to the UE. The SRS suite implements all the functionality of the RAN, while different

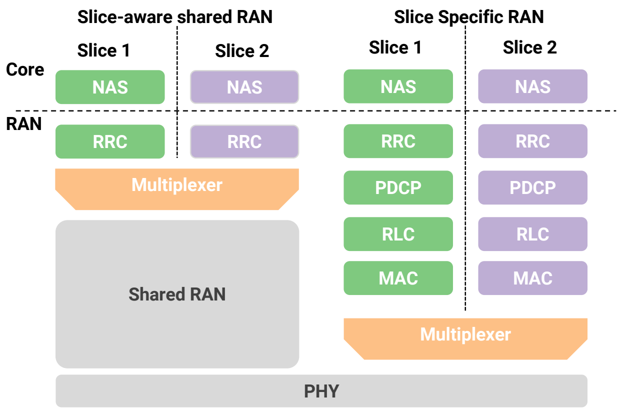

Figure 2: RAN sharing options.

instances of the core network are provided by the OAI core. More specifically, the available software components are:
The RAN: This is the main part of the SRS suite. It implements the Physical (PHY), Media Access Control (MAC), Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP), Radio Resources Control (RRC) and Radio Resource Management (RRM) layers of the LTE stack.
The CN: composed by the Home Subscriber Server (HSS) and the Mobility Management Entity (MME), on the control plane and the Gateway (GW) function, joined into a SP-GW module. All the core modules are from OAI.

Each of the CN functions runs as a VNF. While these are not modified, the RAN network function has been thoroughly configured and substantially modified to provide: *(i)* the appropriate bandwidth and latency characteristics to fulfill the requirements of the mobile broadband low latency services; *(ii)* some means to multiplex and de-multiplex traffic associated to the two different slices. This entails software modifications to the srsLTE mobile network protocol stack.

The latter item is a fundamental enabler as the two implemented services share the same RAN, while each tenant can deploy its own core NF for each service. To enable such functionality in the core network we changed the RAN; more specifically, we implemented the "Slice-aware shared RAN" concept as it is defined in [2] and represented in Figure 2.

The SRS suite used for this implementation defines different classes for the different tasks that have to be fulfilled in the RAN: *(i)* a common library (formerly known as SRSLTE) that performs the encoding/decoding operations, up to the MAC; *(ii)* the RLC, RRC and the PDCP layers; and *(iii)* specific modules for the UE (the UE module for the data plane of the UE) and the eNB (S1AP, that manages the connectivity for the core control plane, and GW for the GW connectivity).

Therefore, to provide the slice-aware shared RAN, we modified the higher layers of the network protocol stack (i.e., PDCP and RRC) on both, eNB and UE. Namely, we provided per slice differentiation for the common c-plane and u-plane procedures, as follows: for the c-plane, we had to allow two registrations against two different Non-access Stratum (NAS) instances, so we duplicated (i.e. an instance for each slice) the UE module and the RRC layer, which triggered two NAS connectivity requests. On the other hand, the u-plane is multiplexed and de-multiplexed at the PDCP module, according to the traffic final destination address. That is, by triggering two NAS registration procedures, the UE obtains two valid IP addresses from each slice GW and creates two virtual network TUN interfaces (one per each slice). Therefore, the UE can connect to two different slices at the same time. More details about the slice-aware shared RAN implementation can be found in [9].

4 SERVICE-SPECIFIC FEATURES

In this section we described the additional pieces of software that we have implemented to provide the service-specific functionality described in Section 2.

4.1 An Amendable Service Function Chain

The main feature of the eMBB network slice is to provide a service that may add context-based add-ons on a live video streaming signal, depending on the user profile, preferences, and certain environmental conditions. These parameters are defined here as QoS/QoE influence factors, and serve as VNFs re-orchestration triggers.

The add-ons can be subtitles, generated depending on the subscriber preferences (language, colour, size, position, etc.) or other support videos, with the sign language translation (e.g., for people with a hearing impairment). These add-ons could be explicitly requested by the user at any time during the video playback or triggered by environmental conditions (e.g., excessive ambient noise), so a good synchronization between add-ons and the main video signal is a must.

The MANO architecture and the orchestrated network functions to provide this service are depicted in Fig. 3. In the figure, the red arrows labelled from 1 to 5 show the initial interactions, where the user request the playback of a certain video from a client (yellow box on the right). This is just a regular video service providing the requested video to the user, where the Streaming Server block (bottom left) is acting just as a proxy for the Main Videos Server (where the video files are actually stored). The video stream pass through the Video Mixer block (orange box), which although it does not mix the video with any other content, it helps to prevent disruptions when add-ons are inserted.

When add-ons are requested by the user (green arrows, labelled from A to H), the Streaming Server communicates with the MANO block, which instantiates the Add-ons Server (green box). This server is split into two internal blocks: the

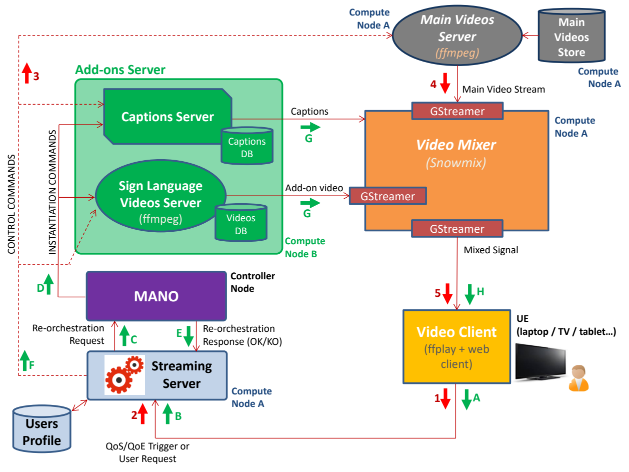

Figure 3: The Orchestration architecture

Sign Language Videos Server (which is used to provide the sign language videos) and the Captions Server (for providing subtitles). Once the appropriate Add-on Server is instantiated, the MANO informs the Streaming Server block, which sends control commands towards the Add-ons Server (dashed line). The selected add-on (video and/or subtitle) is injected into the Video Mixer block which delivers the mixed video signal to the final user.

The Video Mixer block is implemented using Snowmix, an open-Source and very-flexible command line tool for dynamically mixing live audio and video feeds which supports overlaying video, images, texts and graphic elements as well as mixing audio [21]. All these components have been embedded in a special-purpose VNF, deployed on one of the compute nodes. The Add-ons Server is split in two VNFs, corresponding to the two blocks in the figure, i.e., the Captions Server and the Sign Language Videos Server. Each VNF contains a database with the necessary caption and video files, and a service running over TomCat [3]. The service exposes a REST API used by the Streaming Server block to execute the necessary control commands once the VNF instances are up and running. The Caption Server communicates with the Video Mixer block using a Snowmix-specific protocol which makes it possible to specify where and how the add-ons are placed within the video. The Sign Language Videos Server uses ffmpeg [5] to stream the add-on videos into the mixer.

The Main Videos Server is also an independent VNF, running over the same compute node as previously mentioned. It contains a database with all the possible videos the final user could access. The Streaming Server is an independent VNF which performs three primary functions: *(i)* It works as a server for the end user. Specifically, it embeds an HTTP

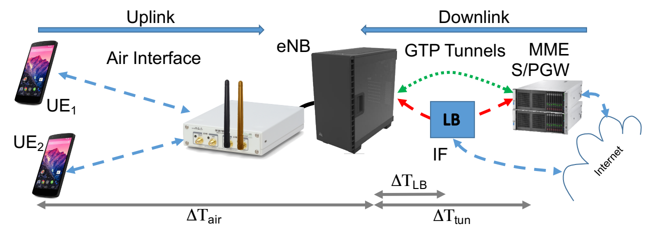

Figure 4: General scenario showing a legacy GTP tunnel (top, green dotteed line) and a GTP with LB (bottom red dashed line).The LB introduces additional interface IF for routing traffic locally.

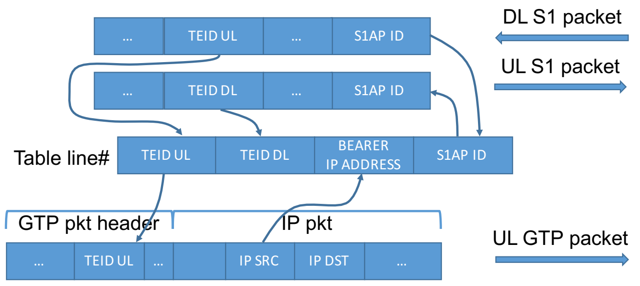

Figure 5: TEID table maintained by the LB threads: it associates TEID numbers to the IP address of each UE. It is used for crafting GTP tunnel return packets.

server to what the final user can access (using a general purpose web browser) to request the video playbacks and the available add-ons. *(ii)* It decodes the user's HTTP requests, implementing the logic to trigger the instantiation of the necessary add-on server (through a request to the MANO) and sending the necessary control commands to these instances, and also, to the Main Videos Server. *(iii)* It maintains a good synchronization between the main video stream and the add-ons. Add-ons can be requested at any time by the final user, so they must appear properly synchronized with the main video and with a minor delay.

The MANO block (purple box in the figure) represents our implementation of the MANO framework. It is basically a process developed using the Java programming language, acting as a server attending the commands from the Streaming Server. Although not explicitly represented in the figure, it also communicates with the underlying ETSI NFVI MANO components (i.e., NFVO, VNF Manager and VIM) to orchestrate and manage the virtual resources. This process is executed on the Controller Node (represented in Figure 1). Finally, regarding the End-User equipment (yellow box in the figure), it consist of two different elements, namely, the video player and a web browser with a GUI. The former is based on the ffplay video player [5], while the later is developed for the end-user to control the service.

4.2 Local breakout

Even though the Local Breakout (LB) feature could be implemented within the eNB software, deploying it as an independent VNF results a more flexible solution, for at least two reasons: *(i)* it can be maintained and upgraded separately, and *(ii)* it would work (in principle) with any eNB. We next present the main characteristics of our LB implementation.

4.2.1 GTP-tunnel and LB. Differently than in a Wireless Local Area Network, traffic generated by mobile users is not routed at the device that terminates the air interface: it is, in fact, delivered inside a GPRS Tunneling Protocol (GTP) tunnel to the remote Gateway controlled by the user's service provider (S/P-GW for LTE, UPF for 5G) where it starts its journey to the destination (top tunnel in green/dotted line in Figure 4). Return traffic is first received by the gateway that tunnels it to the specific eNB at which the destination UE is connected. This method facilitates accounting for roaming users but introduces inefficiencies at the routing level.

To avoid these issues, a LB device can be set up to intercept GTP packets earlier (bottom tunnel in red/dashed line). We report in the following the LB architecture focusing on the mechanisms for transparently intercepting and conveniently routing selected traffic locally.

4.2.2 LB implementation. As shown in Figure 4, the LB node has to *(i)* inspect tunneled packets to extract those matching specific rules and forward them through the new interface IF; and *(ii)* push packets received from IF into the tunnel. As uplink users' packets are embedded in UDP datagrams, the LB can easily access both the fixed length GTP header and the original IP packet [12]. The GTP header is 8 byte long and contains the *Tunnel endpoint identifier* (TEID), a 32-bit value that identifies the bearer to which the inner IP packet is addressed and that is generated by the eNB/MME when the UE node connects to the network (or when it requests a new service). We implemented the tasks for intercepting and pushing packets into the tunnel and for determining the TEID values as three main threads that refer to a common TEID table for storing/fetching TEID values:

S1 sniffer thread. When a UE requests a new service, the eNB and the MME exchange a couple of S1 packets that carry new TEID values: one in downlink with the TEID decided by the MME; followed by one in uplink carrying the TEID chosen by the eNB. The knowledge of the latter is fundamental to the LB for pushing downlink packets received from the IF interface into the tunnel. We show in Figure 5 how the S1 thread correlates the two S1 packets by using the common S1AP-ID field to create a new row in the TEID table with the corresponding TEID values. Because of the

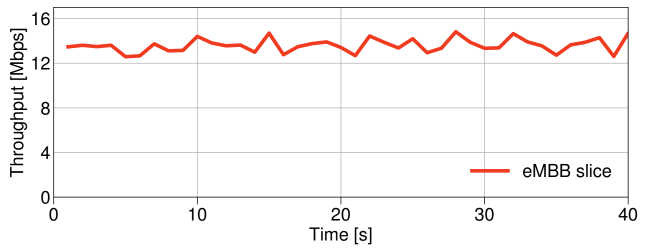

Figure 6: Throughput obtained by the eMBB slice.

complexity in dissecting S1 traffic, the S1 thread forks a `tshark` process and uses a pipe for receiving the TEID data from it. It is worth noting that at this stage the IP address of the UE is not yet known: it will be discovered by the Uplink thread as we explain next.

Uplink thread. To inspect all GTP packets going to the S/PGW, the Uplink thread installs a rule in the netfilter framework of the LB kernel that matches UDP packets addressed to the remote gateway. Setting NFQUEUE [4] as target allows the Uplink thread to receive all packets and decide which must be stopped, stripped by their GTP header and injected through interface IF.

When a UE bearer transmits an uplink data packet for the first time, the thread adds the source IP address found in the packet inside the TEID table: to this end, it looks up the corresponding row by searching the TEID UL value extracted from the GTP packet, as shown at the bottom of Figure 5.

Downlink thread. This thread receives packets from the IF interface and pushes them into the GTP tunnel. It first uses the destination IP address of the packet to look up in the TEID table the value of the TEID DL field: it then crafts a new GTP packet by copying the TEID DL value in the header and concatenating the IP packet in the GTP payload. If no TEID DL value is found, the thread simply drops the packet.

5 FEATURE VALIDATION

In this section we provide the evaluation of the implemented services in practice.

5.1 Service composition

This test simply evaluates whether the proposed mobile broadband service provides the expected results. We used videos from the Spanish Congress of Deputies, which offered open source video feeds of their sessions (including separate video streams for the main video signal and the sign language translations).

The needed throughput is largely supported by the radio system. Values for the available throughput are depicted in Figure 6. The results, averaged over ten experiments, show that the achievable throughput is much higher than the one needed by the streamed video (which ranges between 500 kbits and 1.4 Mbps).

[4] https://www.netfilter.org/projects/libnetfilter_queue

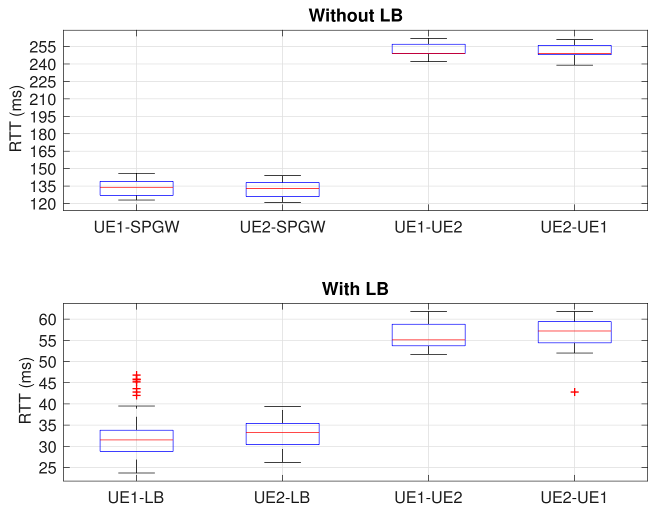

Figure 7: Latency evaluated for the URLLC scenario.

5.2 VNF re-location

One of the objectives of this multi-service deployment is to showcase advanced orchestration functionality, like a service-aware adaptive allocation of functions to different network nodes using VNF mobility concepts. Regarding this, we envisioned the possibility of relocating the VNFs between different compute nodes, which are acting as edge and central cloud nodes in a real 5G network. For our particular case, we provide this feature by using the so-called "live-migration" functionality provided by OpenStack. Namely, we performed two different experiments:

The first one, based on the so-called "block live-migration", just needs a network connection from the source node to the destination. Basically, the complete virtual machine is transmitted without service interruption. In our specific case, VNF instances can be quite big (tens of GBs once instantiated), so it would be necessary to have a very high bandwidth and dedicated network connection to achieve fast migration times. With a common network connection like the ones we have in our testbed the process takes times in the range of seconds to minutes. Moreover, while very high bandwidths may be achievable within the same datacenter, having them available in different geographical locations may be questionable. Furthermore, this approach also reduces the VM processing power during the migration time, so we considered this is not an acceptable approach for a future 5G network.

The second approach is called "shared storage live migration". As the name states, it is based on using a shared storage which is accessible from both, source and destination host.

In this case, performance has been quite good (in the range of few milliseconds), without any service interruption.

Although the performance is quite good using the latter approach, this shared storage node is a drawback in itself. This procedure has well-known inconveniences, among others: *(i)* adding shared storage nodes implies to design the network topology according to this; *(ii)* the shared storage node becomes single point of failure; *(iii)* the network itself becomes a single point of failure *(iv)* the network security shall be improved as the shared storage should be placed in a separate secured network, and *(v)* even with the good performance results experienced in our demo case, the network latency could impact performance, especially for certain very high requiring low latency scenarios in 5G. However, while this aspect goes beyond the purposes of this paper, we think these problems may be solved in different ways. They range from reducing the size of the VMs to be migrated to the usage of containers [19] or unikernels [14]. Another possibility is the improvement of the shared storage option to tackle the known weaknesses, e.g., adding redundancy to the single points of failure and improving the security.

5.3 Low latency through LB

We tested our LB implementation by running the corresponding VNF in the slice providing the low latency service. We attached to our setup the multi-slice UE (UE1) presented in Section 3.3 and an additional single slice UE (UE2) that connected to the low latency slice only.

In one scenario, we co-located the AR server with the gateway, and compared it with another scenario where the AR server was placed together with the LB VNF. An additional delay of 100 ms was added between the eNB and the gateway, to emulate the distances between edge and central cloud. The box and whiskers plot in Figure 7 shows the delay for a UE to server and UE to UE communications. The latter is not related to the provided URLLC service, but it is useful to assess the performance of e.g., machine to machine communications. As expected, the latency with the LB VNF activated is much less than the normal case, with a median value of around 30ms and 135ms for the UE to Server case with or without LB respectively. The gap is even higher for the UE to UE case, with median latency figures of 55ms and 255ms respectively.

6 CONCLUSIONS

In this paper, we described our hands-on experience gained from the implementation of some distinctive functionality of 5G Networking, *(i)* multi-slice orchestration, *(ii)* RAN slicing and *(iii)* local breakout. All our software is based on open source components, and most of it is also released as open source on public repositories.

ACKNOWLEDGMENTS

This work was supported by the H2020 5G-MoNArch project (grant agreement no. 761445), by the Spanish Ministry of Economy and Competitiveness through the 5G-City project (TEC2016-76795-C6-3-R) and by the Madrid Regional Government through the TIGRE5-CM Program under Grant S2013/ICE-2919.

REFERENCES

[1] Mosaic 5G. 2018. Mosaic 5G Ecosystem. http://mosaic-5g.io/.

[2] 5G NORMA. 2017. *D4.2 RAN architecture components - final report.*

[3] Apache. 2018. Tomcat. https://tomcat.apache.org.

[4] ETSI. 2018. OSM. https://osm.etsi.org/.

[5] FFMpeg. 2018. https://ffmpeg.org.

[6] X. Foukas, M. K. Marina, and K. Kontovasilis. 2017. Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking.* ACM MobiCom.

[7] The Linux Foundatio. 2018. ONAP. https://www.onap.org/.

[8] The Linux Foundation. 2018. OPNFV. https://www.opnfv.org/.

[9] G. Garcia-Aviles, M. Gramaglia, P. Serrano, and A. Banchs. 2018. POSENS: a practical open-source solution for end-to-end network slicing. *IEEE Wireless Communications Magazine* (2018).

[10] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith. 2016. srsLTE: an open-source platform for LTE evolution and experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization.* ACM WinTech.

[11] F. Gringoli, P. Patras, C. Donato, P. Serrano, and Y. Grunenberger. 2018. Performance Assessment of Open Software Platforms for 5G Prototyping. *IEEE Wireless Communications Magazine* (2018).

[12] Openair Interface. 2017. Openair LTE Core Network User Plane, User Plane. In *3rd OAI Workshop.*

[13] S. Lee and J. Kim. 2016. Local breakout of mobile access network traffic by mobile edge computing. In *Proceedings of the International Conference on Information and Communication Technology Convergence.*

[14] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, Steven Hand, and J. Crowcroft. 2013. Unikernels: library operating systems for the cloud. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems (ASPLOS)*, ACM.

[15] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet. 2014. OpenAirInterface: A Flexible Platform for 5G Research. *ACM SIGCOMM Computer Communication Review* 44, 5 (Oct. 2014), 33–38.

[16] OpenStack Project. 2018. Open Stack. https://www.openstack.org/.

[17] Open Baton project. 2018. Open Baton. https://openbaton.github.io.

[18] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker. 2017. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Communications Magazine* 55, 5 (May 2017), 72–79.

[19] K-T. Seo, H-S. Hwang, I-Y. Moon, O-Y. Kwon, and B-J. Kim. 2014. Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud *(Advanced Science and Technology Letters).* Science & Engineering Research Support soCiety, 105–111.

[20] P. Serrano, P. Salvador, V. Mancuso, and Y. Grunenberger. 2015. Experimenting With Commodity 802.11 Hardware: Overview and Future Directions. *IEEE Commun. Surveys Tuts.* 17, 2 (2015), 671–699.

[21] Snowmix. 2018. https://snowmix.sourceforge.io/.