2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD) | 978-1-6654-1779-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/CAMAD52502.2021.9617804

Experimental validation of dynamic scaling of computational resources in cloud radio access networks

O. Santos
Instituto Politécnico de Castelo Branco
Castelo Branco, Portugal
oas@ipcb.pt

P. Marques
Instituto Politécnico de Castelo Branco
Castelo Branco, Portugal
paulomarques@ipcb.pt

H. Marques
Instituto Politécnico de Castelo Branco
Castelo Branco, Portugal
hugo@ipcb.pt

D. Collins
CONNECT centre, Trinity College
Dublin, Ireland
collindi@tcd.ie

***Abstract*—This paper describes an experiment deployed as a use case on top of the IRIS testbed that implements two concurrent network slices on the same cloud infrastructure: one emulating a mobile virtual network operator public safety service with high throughput and low latency requirements and the other emulating an over-the-top service provider (delay tolerant – best effort slice). A scaling algorithm with the ability to dynamically allocate computational resources among the two slices is described, implemented, and evaluated in terms of performance gains in the priority slice when the shared computational resources are scarce. The results indicate that under depleted computational resources, this algorithm can significantly improve the TCP and UDP throughput performance of priority slices.**

Keywords—cloud RAN; srsLTE; dynamic scaling; slicing

I. INTRODUCTION

Virtualization of network functions is a growing trend in industry and academia, due to its potential to foster significant reductions in operating expenses [1] [2]. Therefore, there is an increasing softwarization of communication networks, where network functions are translated from monolithic pieces of hardware equipment to software components that run over a shared pool of computational, storage, and communication resources, which can be dynamically provisioned as needed [3].

Network softwarization and network slicing are two important 5G technology enablers [4] [5]. Implementing mobile networks over commercial datacentres has proven considerable benefits, however, deploying cloud-based mobile networks and serving multiple network slices with different requirements over the same virtualised physical infrastructure are challenging tasks. The unpredictable temporal and spatial variation of traffic demand makes the situation worse. In this situation, slice-aware elastic resource management approaches are required to guarantee the best possible quality of offered services to each slice [6].

Two critical resources in 5G systems are radio resources and computational resources (e.g., CPUs, RAM, and storage). While management of the former is well known and studied [7] [8] [9], the elastic management of the latter is a relatively new topic in communication systems and the key goal of this experiment.

Slice-aware elastic resource management algorithms consider the QoS requirements, Service Level Agreements (SLA), and demand of network slices operating on the same physical infrastructure to optimally allocate/deallocate, possibly in almost real-time, resources to/from each slice. Therefore, an elastic management of resources, either computational or radio resources, is required to avoid, or minimize the impact of resource shortages while increasing network's CAPEX and OPEX.

For example, as the demand in one network slice increases, more computational resources may be allocated to that network slice and when the demand decreases, the extra computational capacity should revert to the pool of available resources. In fact, one of the most immediate and appealing advantages of a cloudified network is the possibility of reducing costs, by adapting and re-distributing shared resources following (and even anticipating) temporal and spatial traffic variations.

A key feature for the implementation of slice-aware elastic resource management is the ability to monitor the QoS to assess if the SLA are being met across different network slices. This granular KPI visibility in virtual network infrastructures allows the orchestrator to make just-in-time capacity allocation, ensuring that priority slices have the compute and networking resources they need to meet the performance targets required by the service.

A. Objectives of the experiment

The main objective of this experiment is the implementation and validation of an elastic resource management algorithm (ELASTIC) able to manage multiple Network Slice Instances (NSI) over the same physical resources, while optimizing the allocation of computational resources to each slice based on its dynamic requirements.

The experiment deploys two services over two network slices, with a focus on the QoS-aware control and CPU usage. The goal is to have two competing network slices on the cloud infrastructure: one emulating a mobile virtual network operator (MVNO) Public Safety service with high throughput and low latency requirements and the other emulating an over-the-top (OTT) service provider (delay tolerant – best effort slice). A resource management algorithm is implemented and evaluated in terms of performance gains when operating under scarce computational resources.

The main challenges of this experiment can be divided into two distinct dimensions: understanding how the srslte software uses computational resources under different eNodeB configurations and traffic profiles, and how to manage computational resources so that the higher priority slice can cope with stringent SLA requirements without disrupting the low priority slice.

II. EXPERIMENT DESIGN AND IMPLEMENTATION

The experiment has been deployed at IRIS, the reconfigurable radio testbed at Trinity College, Dublin, which provides virtualized radio hardware, software virtualisation, Cloud-RAN, Network Functions Virtualisation (NFV), and Software Defined Networking (SDN) technologies to support experimental research. The experiment setup uses four computing nodes, as can be seen in the diagram shown in Fig. 1. Each computing node has the following specifications and objectives:

Machine A: this is an Ubuntu 18 physical machine, with 4 cores running at maximum speed of 3.5 GHz. It is connected to a B210 Universal Software Radio Peripheral (USRP) through USB3. Its role is to implement the EPC and eNodeB components of the LTE network.

Machine B: this is also an Ubuntu 18 physical machine, with 4 cores running at a maximum speed of 3.5 GHZ. It is connected to a B210 USRP through USB3 and it implements the UE component of the LTE network.

Both USRPs are configured in single antenna mode, using the LTE EARFCN frequencies: DL=2685.0 MHZ, UL=2565.0 MHz.

Machine C: this is a virtual machine with 2 cores, running an IRIS Ubuntu 16 plain image. It is used to exchange traffic patterns with the UE through the LTE network, using the iperf tool. This role could have been implemented in the physical machine A, but it would consume resources and possibly affect the CPU usage results.

Machine D: this is a virtual machine with 2 cores, running an IRIS Ubuntu 16 plain image. This machine implements the ELASTIC algorithm: it receives traffic and CPU usage data from the two probes and determines the actions to perform to comply with QoS requirements.

The experiment has been deployed and run remotely using JFED [10], which provided setup features and SSH access to each virtual machine.

III. BEHAVIOUR OF SRSLTE UNDER STRESS

The srsLTE suite [11] is a free and open-source LTE software developed by Software Radio Systems Limited. It comprises the EPC, eNodeB and UE components built upon the srsLTE library, a high-performance LTE library for software defined radio applications.

The eNodeB software is highly configurable, by using specific options in command line or in the configuration file. Two key configurable parameters are the number of physical resource blocks (PRB) and the Modulation and Coding Scheme (MCS) index to use in the downlink and uplink channels. The eNodeB component supports 15, 25, 50, 75 or 100 PRB.

Computational efficiency is the most challenging aspect of a Software Defined Radio (SDR) application, especially on the LTE receiver, which is much more complex than the transmitter [11]. In order to cope with the stringent time constraints of the LTE PHY layer, the srsLTE Linux implementation of the eNodeB component makes use of parallel processing and real time SCHED_FIFO scheduling policy. In this specific experiment, the main eNodeB process running on machine A deploys 20 child threads scattered among the 4 available CPU cores.

This parallelization is crucial to allow horizontal scalability in Cloud RAN environments, using platforms that allow dynamic scaling, such as kubernetes autoscaling.

A. Testing methodology

The first measurements of this experiment aimed to gather data about the CPU usage in the eNodeB machine and downlink and uplink maximum throughputs for each mode.

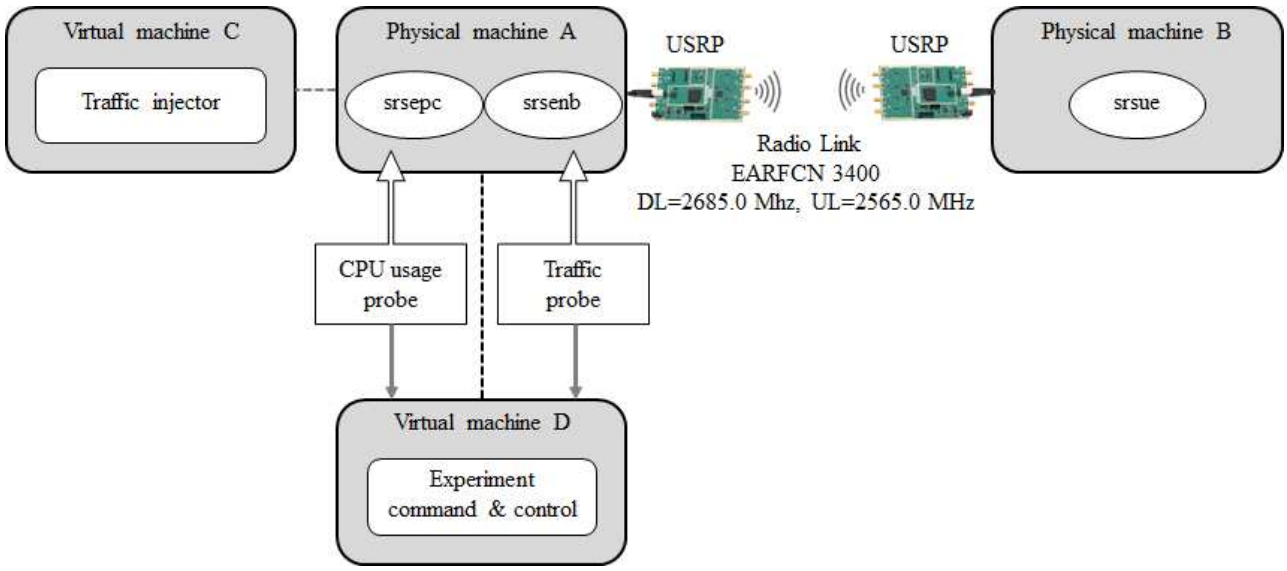

Fig. 1. Experiment setup in IRIS

A mode is a combination of a specific PRB number and MCS index, for example PRB75, MCS 22.

To test the maximum throughput for each mode and its corresponding CPU load, the link was submitted to specific UDP traffic profiles, which were generated employing the iperf tool. UDP was chosen instead of TCP, because preliminary tests with TCP showed significant temporal variability and average throughput below the expected values. This variability may be explained by the congestion control mechanisms of TCP, that considerably reduce the throughput when lost packets are detected; something that happens frequently in wireless networks. Therefore, most of the tests performed in this experiment use UDP, which is more suitable to saturate the LTE network and check its limits.

Fig. 2 illustrates the UDP traffic profiles that were used in the initial tests. Each test took 150 seconds to complete and comprises four different 30 seconds periods, identified in the figure as A, B, C and D.

The first period (A) is used to measure the CPU occupation when the LTE network is idle, without any traffic. The second period (B) measures the CPU occupation and the achieved throughput when the downlink (from the eNodeB to the UE) is saturated with UDP packets. The third period (C) is similar to the previous one, but now in the uplink direction (from the UE to eNodeB). Finally, in the last period (D) the CPU occupation and throughput are measured when both the downlink and uplink channels are flooded with UDP packets. This is the most challenging test, especially in modes with high PRB and MCS.

Fig. 2 also shows the results of one of these tests, applied to the PRB75, MCS22 mode. The CPU usage percentage reported in this paper is always relative to one core, thus, percentages above 100% represent a load of more than one core.

This test was repeated for 20 different eNodeB modes, and the results of 12 of those tests are shown in Table 1. Each column represents the following:

- Mode: the eNodeB configuration regarding PRB and maximum MCS. Only 4 MCS values were chosen; they represent a range of different modulation coding schemes. The last one (auto) means that the MCS index is automatically defined as a function of the signal quality reported by the UE. The maximum MCS index values observed in the tests with auto mode were around 25.
- Downlink max throughput: this is the average throughput observed in the interval B of Fig. 2.
- Uplink max throughput: this is the average throughput observed in the interval C of Fig. 2.
- Downlink factor (alpha): the calculated cost of CPU% for each downlink Mbit/s.
- Uplink factor (beta): the calculated cost of CPU% for each uplink Mbit/s.

The evolution of CPU load throughout the different phases of each test was also registered; this information was then used to calculate the alpha and beta factors. They were calculated from the maximum throughput values and the respective increase in CPU usage.

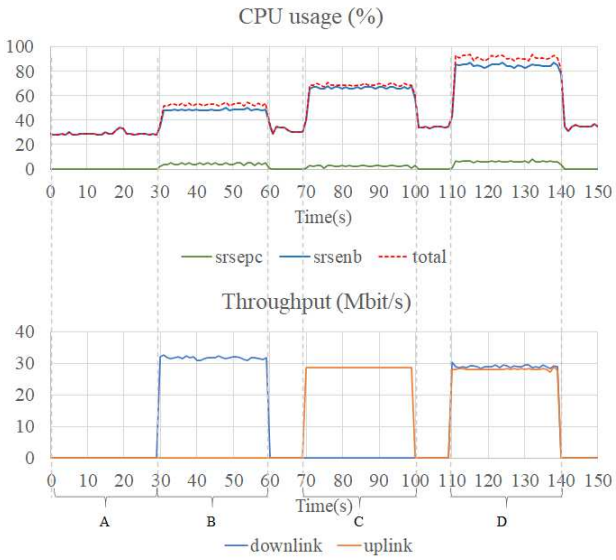

Fig. 2. Test results for mode PRB75 MCS22.

These factors are useful to estimate the CPU load of a specific mode, given the expected downlink and uplink data rates. However, they can be applied to discretionary throughputs only if the CPU usage increases linearly with the data rate. To validate this hypothesis, a test with a UDP ramp profile (linearly increasing throughput) was also designed and implemented. The results obtained in the test provided evidence that indeed the CPU usage varies linearly with the throughput.

Therefore, the CPU usage (C) of a specific mode can be roughly estimated with the formula:

$$C = I + \alpha D + \beta U$$

Where I is the idle CPU usage for that mode, α is the downlink factor, D is the downlink throughput, β is the uplink factor and U is the uplink throughput.

TABLE I. A SUMMARY OF THE INITIAL TEST RESULTS

Mode		**Results**			
PRB	MCS	***Downlink (Mbit/s)***	***Uplink (Mbit/s)***	***alpha factor***	***beta factor***
50	10	7.7	7.7	1.53	2.35
	17	14.8	14.3	1.11	1.47
	22	21.2	20.0	0.86	1.50
	auto	23.6	22.3	0.73	1.72
75	10	11.5	11.1	0.95	1.65
	17	22.4	20.1	0.88	1.76
	22	31.8	28.6	0.75	1.39
	auto	34.6	33.2	0.67	1.63
100	10	15.4	16.5	1.38	2.67
	17	29.8	30.9	0.94	2.07
	22	41.2	44.2	0.94	1.74
	auto	42.9	47.7	0.82	2.13

IV. ELASTIC MANAGEMENT OF COMPUTATIONAL RESOURCES

The ELASTIC algorithm is divided into two different components: the ELASTIC master, which applies to the high priority slice and the ELASTIC slave, which affects the OTT low priority slice as a result of changes in the high priority slice.

The ELASTIC master is continuously probing QoS indicators on the high priority slice. If the current PRB and MCS configuration is not suitable to meet new QoS requirements, action is taken: the slice is configured with new PRB and MCS values and, if this new configuration needs it, more resources are allocated from the OTT slice. Fig. 3 illustrates this algorithm.

The decision about which PRB/MCS mode should be used, given specific downlink/uplink required throughputs, is based on the data of Table 1. The maximum CPU usage of each PRB/MCS mode is also taken from the same table.

The ELASTIC slave basically receives the indication from the master of the amount of resources that it needs to free and changes the PRB and MCS values, so that the slice can run smoothly with the diminished available resources. Obviously, its performance will be decreased as well. Fig. 4 shows how it works.

The OTT slice adaptation to the new level of available resources (changing the PRB and MCS values) is necessary, because without that adjustment the eNodeB software would completely deplete all the available resources. Under these circumstances, the software performs erratically: packet forwarding stops for some periods, radio link failures come up and even software crashes have been observed. Thus, it is crucial to configure the slice with PRB and MCS values that do not deplete the available resources.

The decision about which PRB/MCS mode to use under specific CPU availability is based on a look up table, statically generated from the values obtained in the tests described in section III, thus, it is tailored for this specific

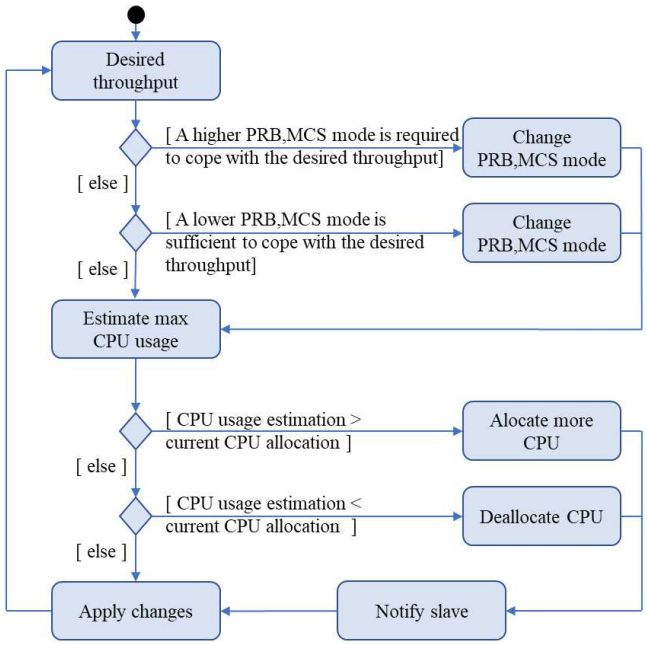

Fig. 3. The ELASTIC master algorithm.

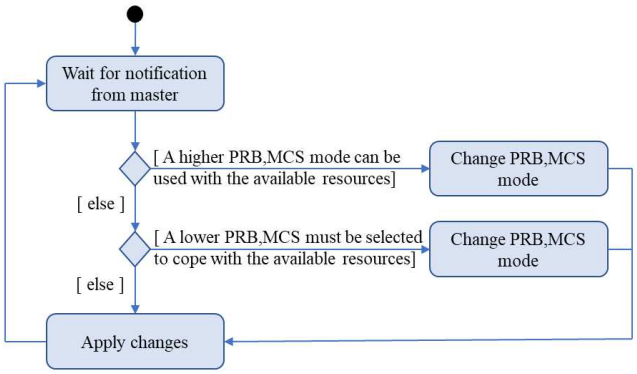

Fig. 4. The ELASTIC slave algorithm.

scenario. Ideally, this table should be dynamically created, using for example machine learning methods, so that it can adapt to different scenarios and even to changes over time in the same scenario. However, in different scenarios, the ELASTIC algorithm remains the same, only the lookup table would have to be changed.

The MCS values for downlink and uplink are sometimes different: they were selected considering that an OTT slice will have to deal mostly with downlink traffic, therefore the priority has been given to the downlink flow.

V. RESULTS

The ELASTIC algorithm was tested with two different traffic profiles: a full speed download/upload TCP profile, and a complex UDP profile with data bursts of different speeds and mixed downlink/uplink flows.

The testing scenario considers that both the slices share the same computational environment, and that initially the CPU resources are equally divided by both slices: 100% for each slice from a total CPU power of 200% (equivalent to a two-core machine). The default eNodeB mode for both slices is PRB75 MCS auto.

Considering that the ELASTIC algorithm works according with the master/slave paradigm, it was not necessary to run both slices at the same time. Instead, results of CPU allocation from the master were recorded and replayed later by the slave component of ELASTIC.

A. ELASTIC behaviour with TCP traffic

The first TCP test measured the difference in throughput in the high priority slice when downloading at full speed, first without ELASTIC and then with ELASTIC activated. Traffic was generated with iperf in TCP mode, for 100 seconds. A second test was performed in the uplink direction.

Fig. 5 shows the results of these tests. As can be seen in the charts, the throughput is substantially higher when ELASTIC is active. This is explained by the automatic switching to the PRB100 mode instead of keeping the default PRB75 mode – keep in mind that this is only possible by dynamically borrowing computational resources from the OTT slice. These tests showed average gains in throughput of 48% and 56%, respectively in the downlink and uplink flows.

The last TCP test involved measuring performance gains with TCP full speed traffic in the downlink and uplink flows simultaneously. In this case, the performance improvement

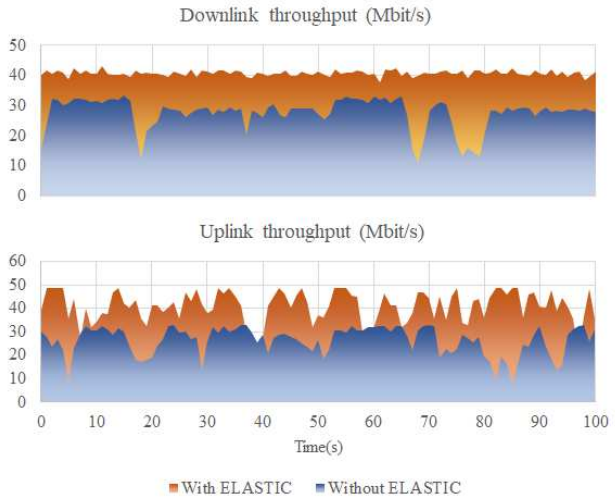

Fig. 5. TCP throughput comparison.

with ELASTIC was 50% in both flows.

B. ELASTIC behaviour with UDP traffic

An UDP test was also devised to evaluate the behaviour of ELASTIC under rapid changing traffic bursts of different throughputs, including simultaneous operation in downlink and uplink and sudden bursts in one direction when it is already operating with high throughput on the other direction.

The results from this test can be seen in Fig. 6. They show that even under these demanding circumstances the ELASTIC mode was able to cope with the requested throughput, while the normal mode struggled to handle high throughputs simultaneously in the downlink and uplink flows. This is particularly evident in the zones A, B and C of the downlink chart and the zone D of the uplink chart.

C. ELASTIC resource allocation

In the previous test (UDP burst traffic profile with simultaneous downlink and uplink) the results obtained with the ELASTIC algorithm are only possible because in this case the eNodeB process has been temporarily switched to higher PRB and MCS values in the critical periods, which requires more than the initial 100% available CPU capacity to work correctly.

Fig. 7 shows the CPU usage measured during the test,

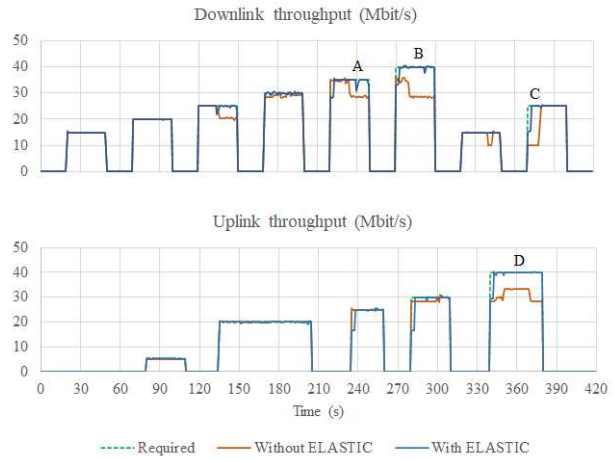

Fig. 6. UDP throughput comparison.

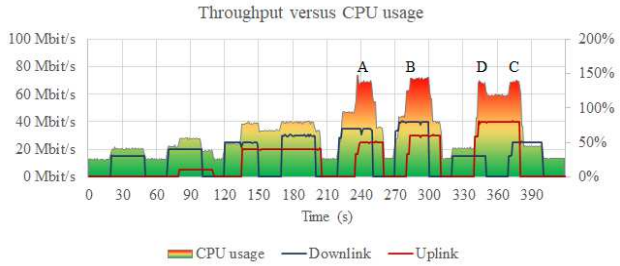

Fig. 7. CPU allocated to the priority slice.

which corroborates that in the most challenging periods (high throughput, especially in the uplink direction) the ELASTIC algorithm switched the eNodeB mode to higher PRB and MCS, thus explaining the CPU usage above 100%.

D. Consequences on the OTT slice

Each time the ELASTIC algorithm needs to switch to a PRB/MCS mode that requires more CPU resources than the initial allocated value for the priority slice, it must allocate that extra CPU resources from the OTT slice. Whenever this happens, the OTT slice must adapt to the new diminished available CPU resources, changing its PRB/MCS mode to lower values. Failure to perform this adaptation results in erratic behaviour, ranging from radio link failures to software crashes, so it is absolutely necessary to switch to PRB/MCS values that can work correctly with the diminished available resources.

In order to evaluate the behaviour of the ELASTIC algorithm concerning the OTT slice adaptation under the conditions of the UDP burst downlink/uplink, the CPU allocation of Fig. 7 was recorded and replayed later in the OTT slice, when it was downloading and uploading TCP traffic at full speed. Fig. 8 illustrates the results which were obtained in this test. The first chart shows the OTT slice's CPU usage during the test. The light green area represents the available CPU resources – the blank "holes" correspond to the resources taken by ELASTIC and given to the priority slice. The blue line is the measured CPU usage. Notice how the ELASTIC algorithm adapted the CPU usage to the available resources so well, just by switching to the most adequate PRB/MCS mode.

The second chart shows what happens to a TCP full speed download during the test. The reduction of throughput during the periods of CPU depletion is noticeable.

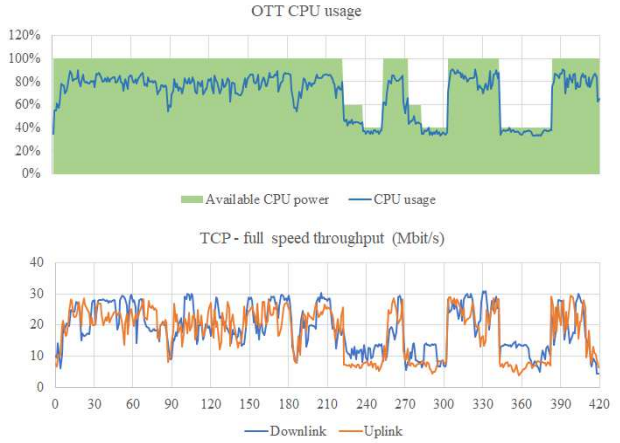

Fig. 8. Adaptation of the OTT slice to depleted resources.

Furthermore, it is also evident the priority given to the downlink flow by the ELASTIC algorithm, which is achieved by being more aggressive in the reduction of the MCS value in the uplink rather than in the downlink. This is a design feature, because uplink traffic requires more CPU power than downlink traffic, so this is an efficient way of reducing the slice's CPU usage. Moreover, in OTT slices it is expected that traffic will flow mostly in the downlink direction, thus this strategy minimizes the impact on QoE as well.

VI. CONCLUSIONS

The main finding of this experiment is that dynamic scaling of shared computational resources in cloud RAN environments is indeed an effective tool to ensure that high priority LTE slices can cope with critical periods of high throughput.

To allocate adequate computational resources for specific traffic profiles, the orchestration algorithm must be able to estimate the optimal PRB/MCS mode to use and its respective computational load. Thus, a learning stage is essential to build lookup tables that relate throughput, PRB/MCS modes and CPU load. The algorithm can then use this lookup table to decide which mode to use and the computational resources to allocate.

The performance tests done in this experiment revealed that the ELASTIC algorithm was able to achieve throughput performance gains of about 50% on TCP traffic and 30% on UDP traffic (during high throughput peaks).

ACKNOWLEDGMENT

This work was supported by the H2020 program under grant agreement No. 732174 (ORCA project) and the Fundo Europeu de Desenvolvimento Regional (FEDER) through the POCI-01-0247-FEDER-046555 (AI4GREEN).

REFERENCES

[1] Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., De Turck, F., & Boutaba, R. (2015). Network function virtualization: State-of-the-art and research challenges. IEEE Communications surveys & tutorials, 18(1), 236-262.

[2] S. Abdelwahab, B. Hamdaoui, M. Guizani and T. Znati, "Network function virtualization in 5G," in IEEE Communications Magazine, vol. 54, no. 4, pp. 84-91, April 2016, doi: 10.1109/MCOM.2016.7452271.

[3] Serrano, P., (2018) "The path towards a cloud-aware mobile network protocol stack", Transactions on Emerging Telecommunications Technologies, John Wiley & Sons, Ltd.

[4] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini and H. Flinck, "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions," in IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pp. 2429-2453, thirdquarter 2018, doi: 10.1109/COMST.2018.2815638.

[5] S. Zhang, "An Overview of Network Slicing for 5G," in IEEE Wireless Communications, vol. 26, no. 3, pp. 111-117, June 2019, doi: 10.1109/MWC.2019.1800234.

[6] Y. Zhang, F. Barusso, D. Collins, M. Ruffini and L. A. DaSilva, "Dynamic Allocation of Processing Resources in Cloud-RAN for a Virtualised 5G Mobile Network," 2018 26th European Signal Processing Conference (EUSIPCO), Rome, 2018, pp. 782-786, doi: 10.23919/EUSIPCO.2018.8552959.

[7] X. Foukas, G. Patounas, A. Elmokashfi and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," in IEEE Communications Magazine, vol. 55, no. 5, pp. 94-100, May 2017, doi: 10.1109/MCOM.2017.1600951.

[8] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," in IEEE Communications Magazine, vol. 55, no. 5, pp. 80-87, May 2017, doi: 10.1109/MCOM.2017.1600935.

[9] P. Rost et al., "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," in IEEE Communications Magazine, vol. 55, no. 5, pp. 72-79, May 2017, doi: 10.1109/MCOM.2017.1600920.

[10] "jFed – Java based framework to support SFA testbed federation client tools". FED4FIRE+. https://www.fed4fire.eu/tools/jfed/ (accessed Dec. 10, 2020).

[11] Gomez-Miguelez, I., Garcia-Saavedra, A., Sutton, P. D., Serrano, P., Cano, C., & Leith, D. J. (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 (pp. 25-32).