Orchestration and Reconfiguration Control Architecture

ELASTIC: Experimental validation of resource management algorithms for elastic network slicing based on end-to-end QoS

Revision: v.1.0

Call identifier	ORCA-OC2-EXP-IND
Date of report	10/28/2019
Organization	Allbesmart LDA
Submission date	10/18/2019
Authors	Paulo Marques, Luis Pereira, Tiago Alves
Coordinator[1]	Paulo Marques
Patron	Diarmuid Collins (TCD)

[1] Coordinator is the main (technical and administrative) contact person for this experiment or extension.

SUMMARY

The main objective of this experiment is the validation of elastic resource management algorithms able to serve multiple Network Slice Instances (NSI) over the same physical resources while optimizing the allocation of computational resources to each slice based on its requirements and demands.

The experiment deploys a use case on top of the IRIS testbed that provides 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 MVNO Public Safety service with high throughput and reduced latency requirements and the other emulating an OTT service provider (delay tolerant – best effort slice). A resource management algorithm is implemented and evaluated in terms of performance gains when operating under computational resource limitations.

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

The experiment setup created in IRIS uses four computing nodes: two machines are connected to B210 USRPs and implement the LTE network, one the EPC and eNodeB components and the other the UE component. The third machine is used to exchange traffic patterns with the UE through the LTE network. Finally, the fourth machine implements the ELASTIC resource management algorithm.

The ELASTIC algorithm proved to be very effective to increase the TCP throughput of the high priority slice if more CPU resources are required to comply with stringent QoS requirements. Testing revealed gains of 48% in downlink, 55,6% in uplink and 49,8% in simultaneous downlink and uplink. ELASTIC was also successful dealing with UDP traffic bursts, even with high throughput demand in both directions at the same time.

The main conclusion of this experiment is that when two competing cloud RAN LTE slices are implemented over the same computational infrastructure, intelligent management of computational resources is an effective instrument to ensure that high priority slices can cope with demanding QoS requirements under shortage of computational resources. The ELASTIC algorithm implemented in this experiment proved to be effective to increase the performance of a high priority slice without disrupting the operation of the low priority slice.

TABLE OF CONTENTS

1 TECHNICAL CONTRIBUTION

1.1 Concept and objectives

1.1.1 Motivation

We are currently observing the softwarization of communication networks, where network functions are translated from monolithic pieces of equipment to programs running over a shared pool of computational, storage, and communication resources [1]. Network softwarisation and network slicing are two important 5G technology enablers. Implementing mobile networks over the commercial datacentres has proven considerable benefits. However, realizing cloud-based mobile networks and serving multiple network slices with different requirements over the same virtualised physical infrastructure are challenging tasks. The dramatic temporal and spatial variation of traffic demands made 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.

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

Slice-aware elastic resource management algorithms consider the QoS requirements, Service Level Agreements (SLAs), and demands 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 whilst increase network's CAPEX and OPEX. For example, as the demands in one network slice increase, more computational resources may be allocated to that network slice and when the demands decrease 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 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 in order to assess if different SLAs are being met across different network slices. This granular KPIs visibility in virtual network infrastructures, allows to trigger decisions that feed the Network Orchestrator to provide just-in-time capacity augmentation, ensuring VNFs have the compute and networking resources they need to meet the performance targets required by the service.

1.1.2 Challenge

The network slicing management and orchestration must be able to allow the requirements over different Network Slice Instances (NSI) to be expressed and enforced. This is mainly achieved through SDN/NFV automation and respecting SLAs agreed between the physical infrastructure provider and its customers (e.g., MVNOs, OTT service providers or vertical industries). Deciding how to efficiently allocate, manage, and control the slice resources in real time is the main challenged addressed by this ORCA experiment.

The goal of elastic resource management algorithms is to serve multiple NSIs over the same physical resources while optimizing the allocation of computational resources to each slice based on its requirements and demands. A high-level view of an algorithm for slice-aware elastic resource management is structured in the following four steps as suggested by [2]:

1) Forming the available resource pool:

In the first step, the algorithm must identify the available physical resources and form the shared resource pool for the serving slices.

2) Estimating the total computational capacity:

In this step, the algorithm maps the total computational capacity to the slice's requirements (e.g., Network Function (NF) processing time as a function of input variables such as the allocated number of Physical Radio Blocks in the RAN).

3) Allocating the available computational resources to different slices:

The algorithm allocates the required computational resource to the NFs of each slice ensuring the total processing time is acceptable. The allocation procedure should consider each SLAs type and slice priorities.

4) Monitoring the network performance and re-allocating the computational resources based on the changes of demands.

The resource management algorithm observes the changes on each slice QoS performance as a result of the changes to the resource demands or resource availability and updates the resource allocation as needed.

The performance of elastic resource management algorithms can be evaluated through the computation of the "degradation function" that characterizes the way in which performance degrades as computation resources diminish. Figure 1 illustrates this concept and the possible gains achieved by elastic resource management algorithms when compared to the standard approach. As it can be seen in Figure 1, performance of elastic algorithms is not degraded by the same relative amount as computational resources are reduced.

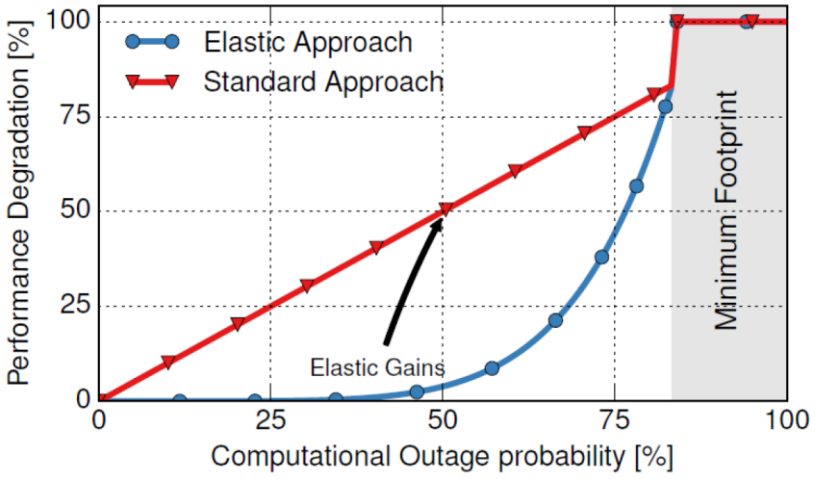

Figure 1 - Example of graceful performance degradation achieved by elastic resource management, when compared to the standard approach [1].

The main objective of this experiment is the validation of elastic resource management algorithms on network slice environments that take advantage of real time monitoring of SLAs based on QoS/QoE metrics.

As each network slice will address a unique vertical need, MNOs will be able to deliver multiple slices to each vertical based on different needs and contexts. The use case in this experiment is looking at an outdoor scenario comprising of two active slices: one belonging to a Public Protection and Disaster Relief (PPDR) MVNO with high throughput and low latency requirements (critical communications),

and the other belonging to an OTT service provider, mainly with traffic with browsing characteristics (delay tolerant – best effort slice). The experiment blueprint for elastic resource management is shown in Figure 2.

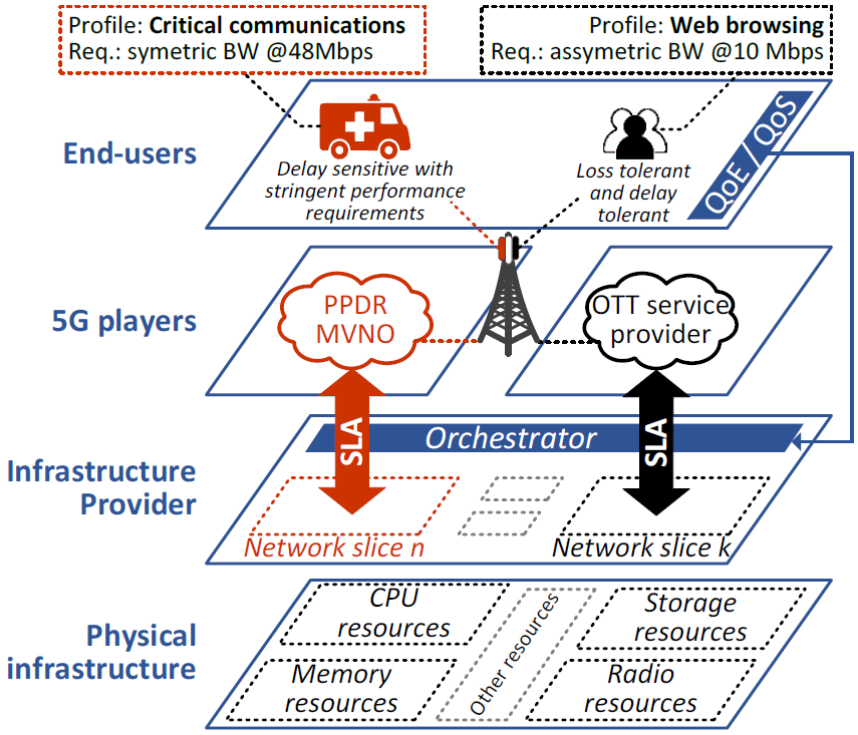

Figure 2 - The experiment blueprint for elastic resource management.

1.2 Technical results and lessons learned

This section contains the technical achievements of the Experiment.

1.2.1 Experiment setup in ORCA facilities

The experiment setup created in IRIS uses four computing nodes, as can be seen in the diagram shown in Figure 3. 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 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 also connected to a B210 USRP through USB3. It implements the UE component of the LTE network.
- Machine C: this is a virtual machine with 2 cores, running an IRIS Ubuntu 16 plain image. Its role is to exchange traffic patterns with the UE through the LTE network, using the iperf tool. This role could be implemented in the physical machine A, but it would consume resources and possibly affect the CPU usage results.
- Machine D: this is also 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 in order to comply with QoS requirements.

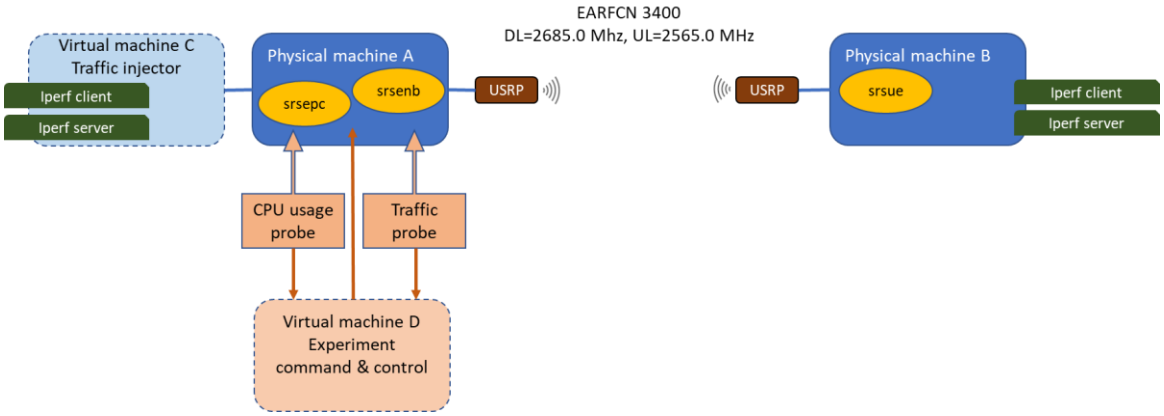

Figure 3 - The experiment setup in IRIS.

The B210 USRPs are configured in single antenna mode, using the LTE EARFCN frequencies: DL=2685.0 MHZ, UL=2565.0 MHz.

Access to each virtual machine is achieved through JFED, as can be seen in Figure 4.

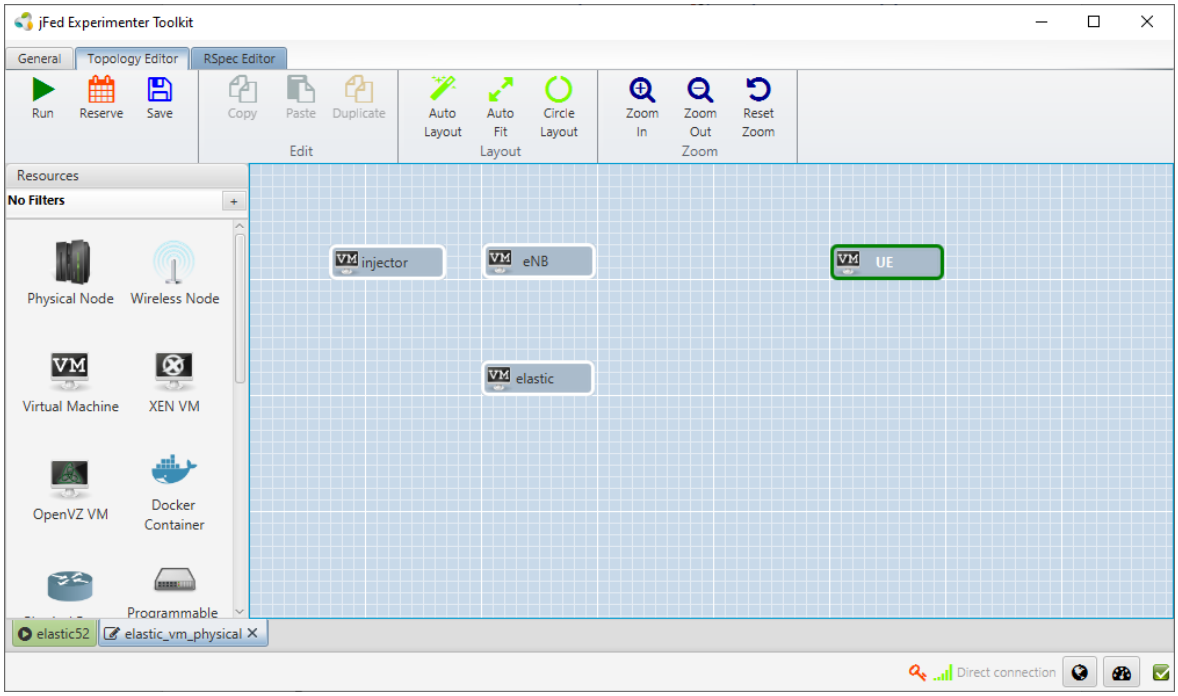

Figure 4 - The experiment setup in JFED.

JFED provides a quick and easy access to each machine through SSH.

1.2.2 A brief description of the srsLTE software

The srsLTE suite [3] is a free and open-source LTE software developed by Software Radio Systems Limited [4]. It comprises the EPC, eNodeB and UE components built upon the srsLTE library, a high-performance LTE library for software-defined radio applications.

Computational efficiency is the most challenging aspect of an SDR application, and the LTE receiver is much more complex than the transmitter [3]. 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.

Figure 5 shows a screenshot of the htop tool on Machine A, with eNodeB running.

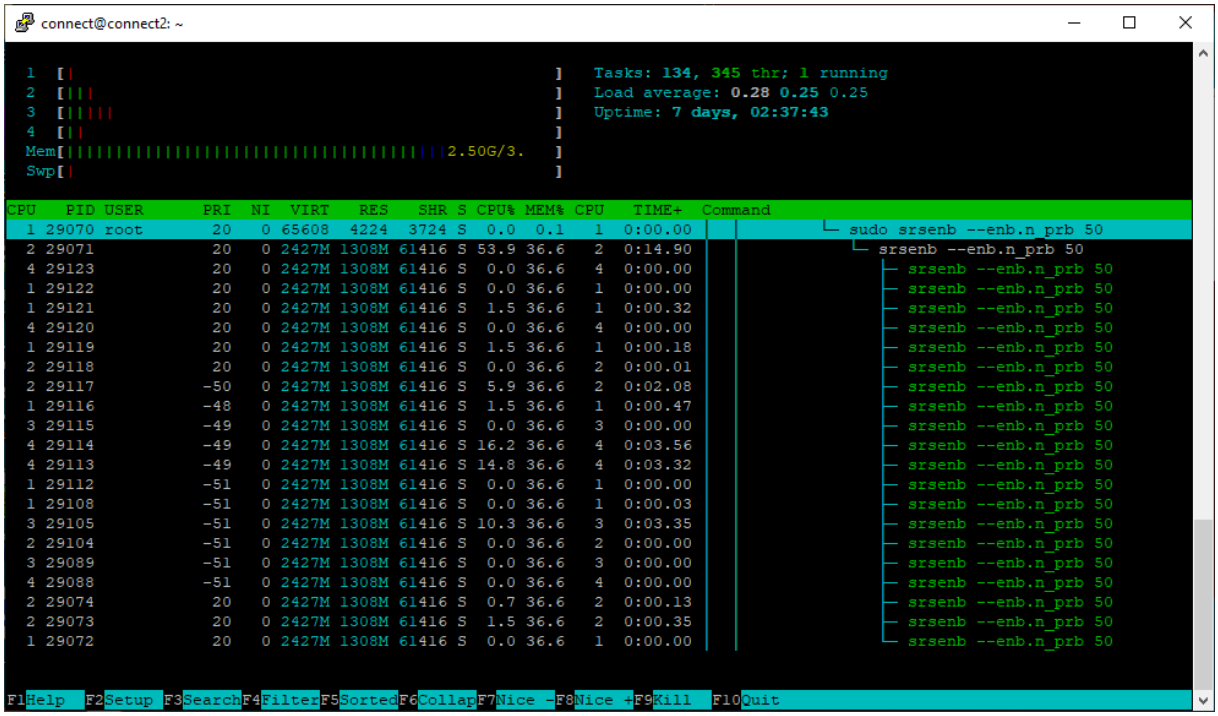

Figure 5 – srs eNodeB processes and threads.

In this specific scenario, the main process contains 20 child threads scattered among the 4 available CPU cores. This software strategy takes advantage of multicore machines and is crucial to support scalability in Cloud RAN environments.

The eNodeB software is highly configurable, 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 maximum MCS index to use in the downlink and uplink channels. The eNodeB component supports 15, 25, 50, 75 or 100 physical resource blocks.

1.2.3 Preliminary tests

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

Specific UDP traffic profiles, which were generated employing the iperf tool, were used to test the maximum throughput for each mode. UDP has been chosen instead of TCP, because preliminary tests with TCP have shown significant temporal variability and average throughput bellow 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; occurrence which 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.

Figure 6 illustrates the UDP traffic profiles that have been used in the initial measurements.

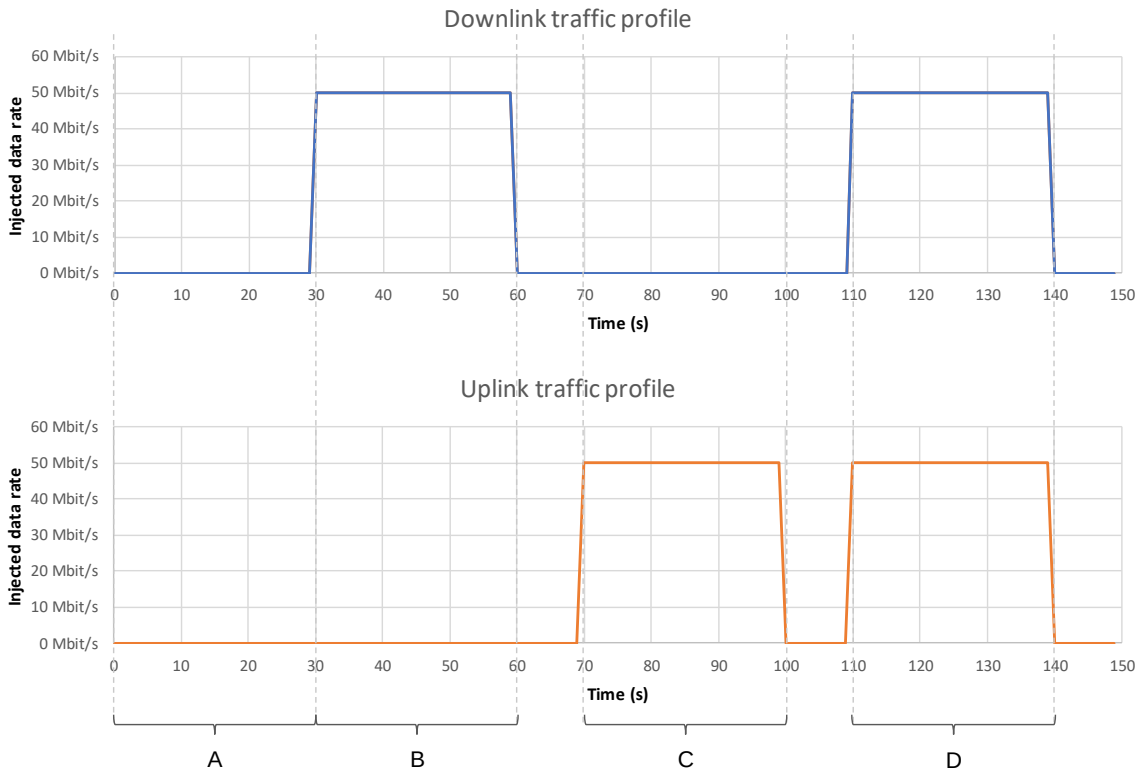

Figure 6 – Traffic profiles used in the initial tests.

Each test takes 150 seconds and comprises four different 30 seconds intervals, identified in Figure 6 as A, B, C and D:

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

Figure 7 illustrates the results of this test, when applied to the PRB75 MCS22 mode. The CPU usage percentage is relative to one core, thus, with four cores this value can reach 400%.

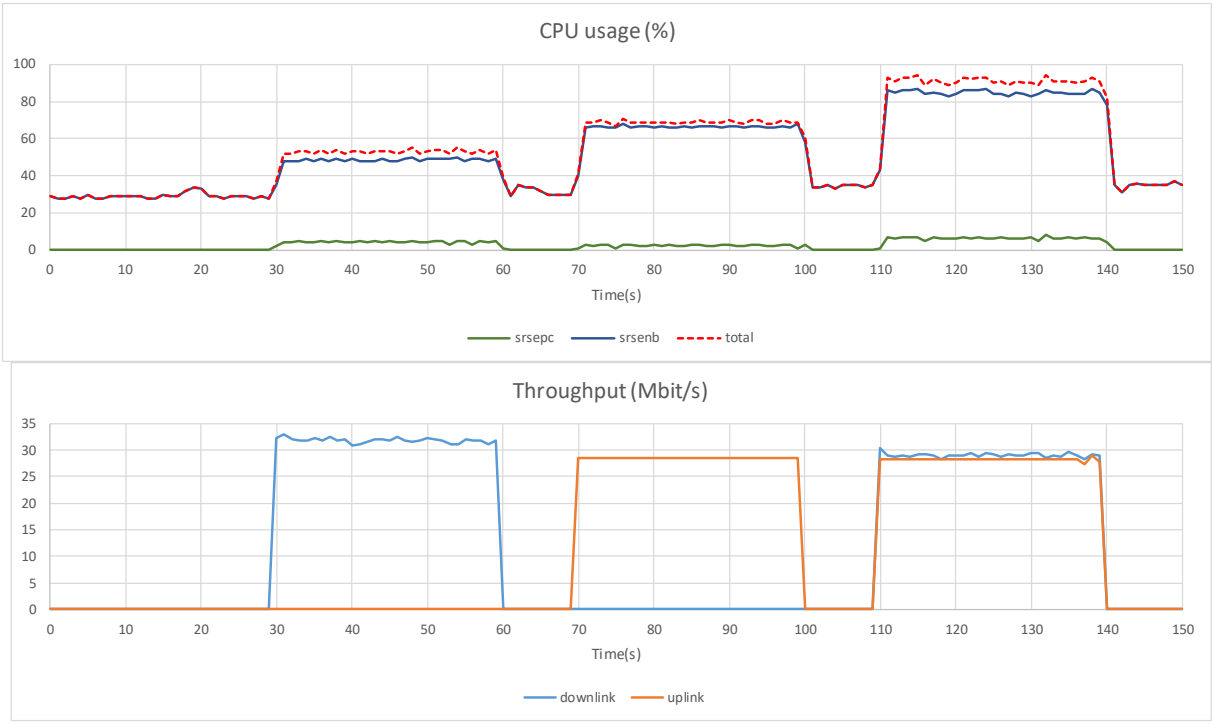

Figure 7 – Test results of mode PRB75 MCS22.

Some interesting aspects, which also have been observed in other mode's tests, should be pointed out:

- The eNodeB process uses significant CPU time, even when there's no traffic in the LTE network.
- In the same situation, the EPC process is almost dormant.
- When traffic is injected into the LTE network, the eNodeB process is responsible for most of the consumed CPU time; the EPC requires little CPU time.
- Uplink traffic requires much more CPU power than downlink traffic. This is explained by the software's authors: "As usual, the LTE receiver is much more complex than the transmitter" [3].
- When there's uplink and downlink traffic at the same time, the total CPU usage is roughly the sum of the extra CPU required by each traffic flow, when tested independently.
- When there's uplink and downlink traffic at the same time, the maximum throughput of each flow (especially the downlink) is slightly lower than the value obtained with just that flow.

This test has been repeated for 20 different eNodeB modes, and the results can be observed in Table 1. Each column represents the following:

- Mode: the eNodeB configuration regarding PRB and maximum MCS. Not every MCS has been tested: only 4 different values that represent very different modulation coding schemes were chosen. 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 Figure 6.
- Uplink max throughput: this is the average throughput observed in the interval C of Figure 6.
- CPU occupancy (idle): the average CPU occupancy during interval A of Figure 6.
- CPU occupancy (max DL): the average CPU occupancy during interval B of Figure 6.

- CPU occupancy (max UL): the average CPU occupancy during interval C of Figure 6.
- CPU occupancy (max DL+UL): the average CPU occupancy during interval D of Figure 6.
- 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.

Table 1 – A summary of the initial test results.

Mode		Downlink max throughput (Mbit/s)	Uplink max throughput (Mbit/s)	CPU occupancy % (Idle)	CPU occupancy % (Max DL)	CPU occupancy % (Max UL)	CPU occupancy % (Max DL + UL)	downlink factor (alpha) CPU% per Mbit/s	uplink factor (beta) CPU% per Mbit/s
PRB	Max mcs								
15	10	2.2 Mbit/s	1.6 Mbit/s	26.2	35.8	34.6	38.1	4.34	5.15
	17	4.4 Mbit/s	3.1 Mbit/s	18.3	37.4	24.2	43.4	4.38	1.95
	22	6.5 Mbit/s	4.3 Mbit/s	22.5	37.7	36.4	48.8	2.32	3.19
	auto	8.5 Mbit/s	4.7 Mbit/s	21.4	37.5	35.2	44.5	1.90	2.95
25	10	3.9 Mbit/s	3.3 Mbit/s	24.1	24.7	26.9	30.0	0.17	0.84
	17	7.5 Mbit/s	6.2 Mbit/s	25.7	26.7	27.7	35.3	0.14	0.33
	22	10.9 Mbit/s	8.8 Mbit/s	28.0	28.8	32.4	44.8	0.07	0.51
	auto	13.8 Mbit/s	10.3 Mbit/s	29.9	33.9	38.8	42.8	0.29	0.87
50	10	7.7 Mbit/s	7.7 Mbit/s	22.2	34.0	40.3	47.7	1.53	2.35
	17	14.8 Mbit/s	14.3 Mbit/s	24.0	40.5	45.0	59.3	1.11	1.47
	22	21.2 Mbit/s	20.0 Mbit/s	21.9	40.2	51.9	63.8	0.86	1.50
	auto	23.6 Mbit/s	22.3 Mbit/s	24.0	41.1	62.3	67.3	0.73	1.72
75	10	11.5 Mbit/s	11.1 Mbit/s	33.4	44.3	51.8	64.2	0.95	1.65
	17	22.4 Mbit/s	20.1 Mbit/s	28.9	48.7	64.4	82.6	0.88	1.76
	22	31.8 Mbit/s	28.6 Mbit/s	29.2	53.0	69.1	91.2	0.75	1.39
	auto	34.6 Mbit/s	33.2 Mbit/s	34.1	57.4	88.3	91.2	0.67	1.63
100	10	15.4 Mbit/s	16.5 Mbit/s	53.8	75.1	97.9	115.9	1.38	2.67
	17	29.8 Mbit/s	30.9 Mbit/s	53.2	81.1	117.3	142.9	0.94	2.07
	22	41.2 Mbit/s	44.2 Mbit/s	55.9	94.5	132.9	172.7	0.94	1.74
	auto	42.9 Mbit/s	47.7 Mbit/s	53.7	88.9	155.4	159.9	0.82	2.13

The alpha and beta factors were calculated from the maximum throughput values and the respective increase in CPU usage. They are useful to estimate the CPU usage of a specific mode, given the expected downlink and uplink data rates.

These factors can be applied to discretionary throughputs only if the CPU usage increases linearly with the data rate. Thus, in order to validate this hypothesis, a test with a UDP ramp profile (linearly increasing throughput) has been designed and implemented. Figure 8 shows the results obtained in the test, which provide evidence that indeed the CPU usage varies linearly with the throughput.

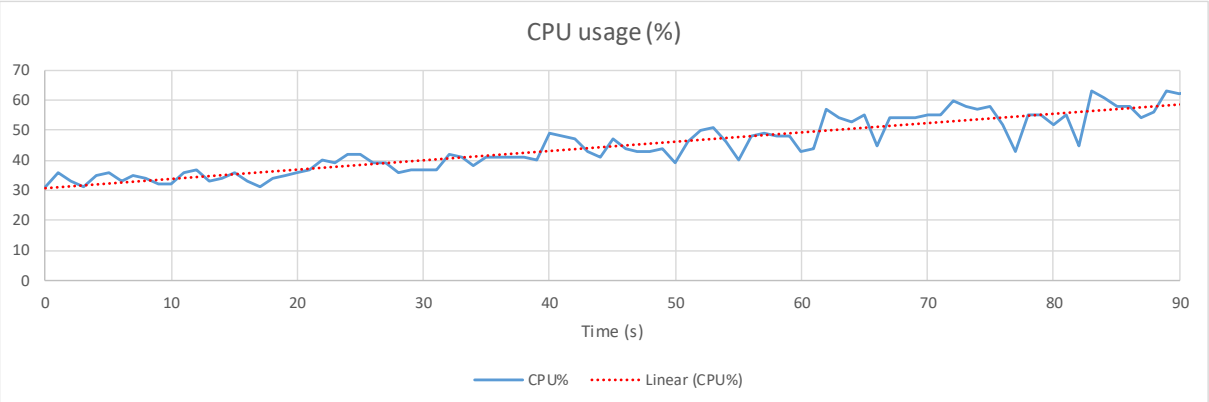

Figure 8 – CPU usage versus ramp traffic profile.

Thus, 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.

1.2.4 Other preliminary tests

In order to refine the ELASTIC algorithm, one particular question had to be addressed: when the MCS index increases, what is the impact of this complexity rise on CPU usage? A specific test has been envisioned to answer this question: the test scenario was injected with a constant 10 Mbit/s UDP traffic profile for 90 seconds, and the CPU usage was recorded. The test has been repeated with MCS 10, 17, 22 and auto, all with PRB75.

Figure 9 shows the results for the downlink flow. Surprisingly, it seems that higher MCS values don't require more CPU power to be implemented.

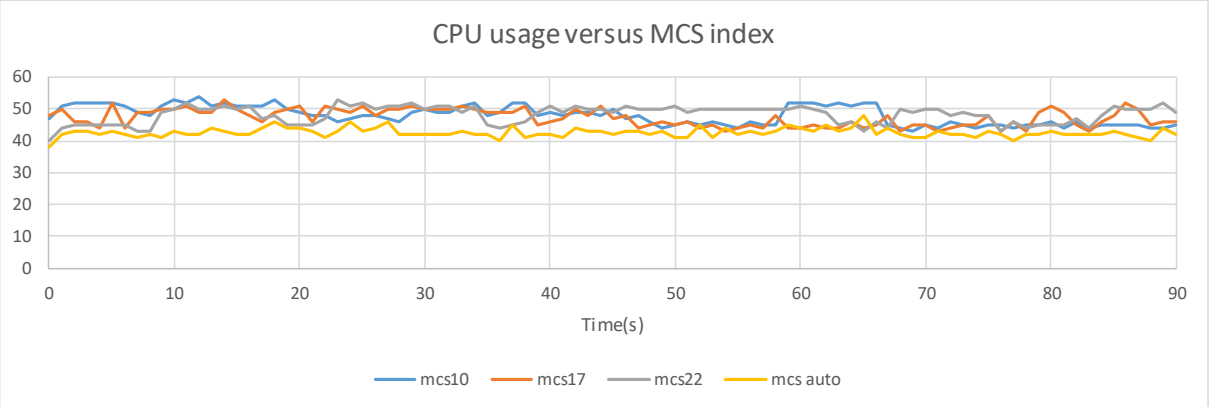

Figure 9 – Downlink: CPU usage versus MCS index.

Figure 10 shows the results of the same test, but now for the uplink flow. The conclusion is the same: there's no evidence to support the idea that higher MCS values require more CPU power.

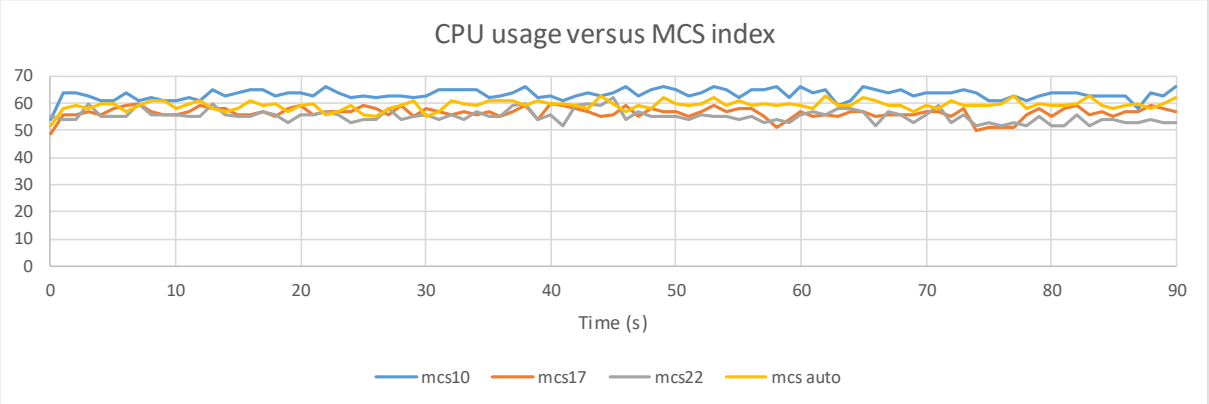

Figure 10 – Uplink: CPU usage versus MCS index.

However, in Table 1 a noticeable increase in CPU usage is seen as the MCS value increases, especially in the uplink flow. This is explained by the higher throughput that superior MCS values permit and not by the modulation scheme itself.

1.2.5 The ELASTIC algorithm

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.

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, resources are allocated from the OTT slice. Figure 11 illustrates this algorithm.

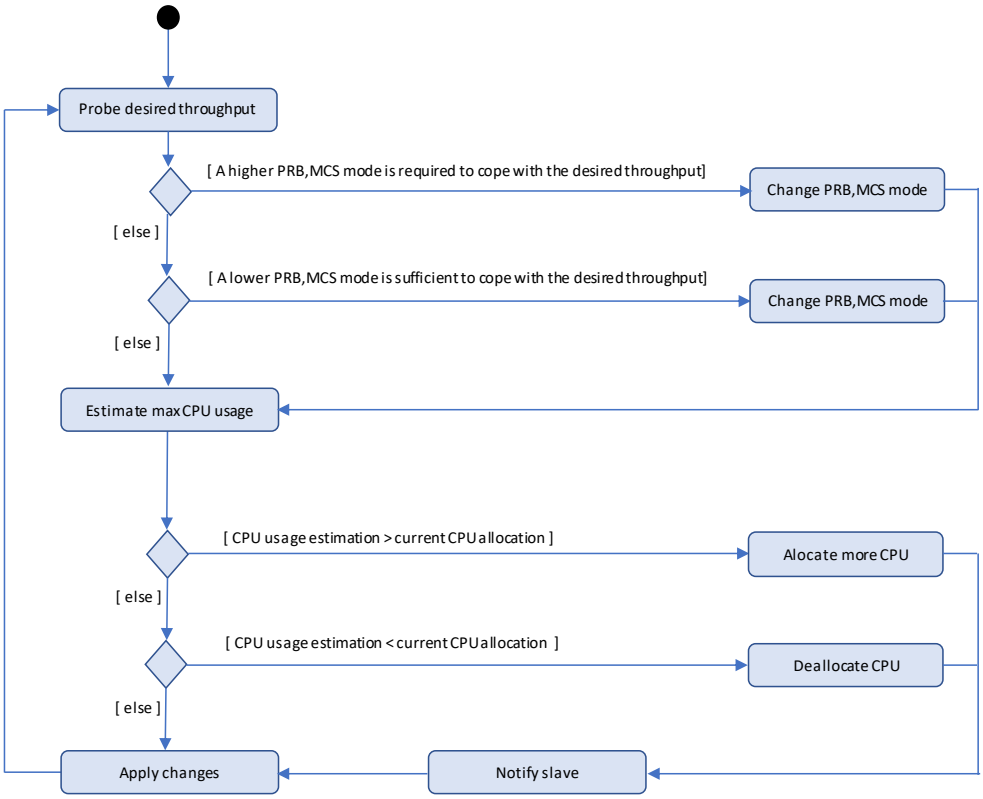

Figure 11 – The ELASTIC master algorithm.

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 reduced as well. Figure 12 shows how it works.

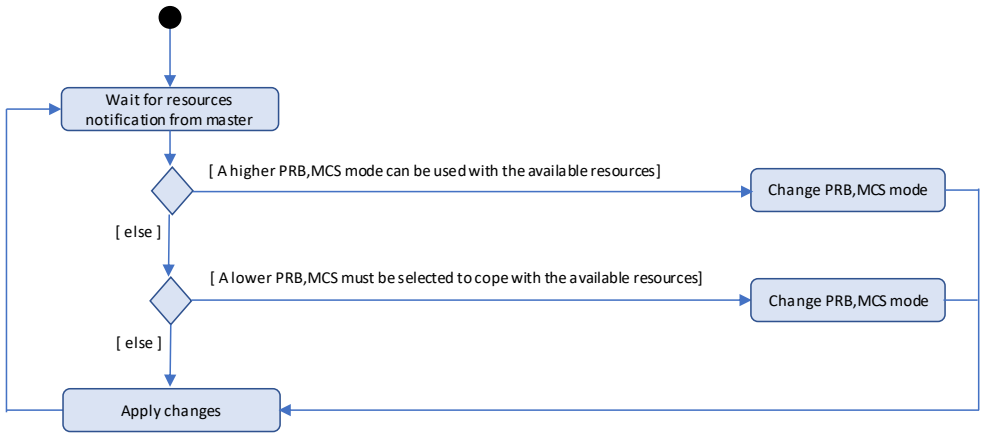

Figure 12 – The ELASTIC slave algorithm.

The OTT slice adaptation to the new amount of available resources by 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 don't deplete the available resources.

The decision about which PRB/MCS mode to use under specific CPU availability is based on a look up table, which is shown in Table 2.

Table 2 – OTT slice adaptation look up table.

	PRB	down MCS	up MCS	max down thr	max up thr	max cpu %
100%	75	auto	auto	34.6 Mbit/s	33.2 Mbit/s	91.2
90%	75	auto	10	34.6 Mbit/s	11.1 Mbit/s	75.8
80%	75	auto	10	34.6 Mbit/s	11.1 Mbit/s	75.8
70%	50	auto	auto	23.6 Mbit/s	22.3 Mbit/s	67.3
60%	50	auto	10	23.6 Mbit/s	7.7 Mbit/s	59.3
50%	25	auto	auto	13.8 Mbit/s	10.3 Mbit/s	42.8
40%	25	auto	22	13.8 Mbit/s	8.8 Mbit/s	38.3
30%	25	10	10	3.9 Mbit/s	3.3 Mbit/s	27.6

The table has been statically populated with values obtained from the preliminary tests described in section 1.2.3, thus, it is tailored for this specific scenario. 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.

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 in the same scenario over time. However, in different scenarios, the ELASTIC algorithm remains the same, only the lookup table would have to be changed.

1.2.6 ELASTIC testing

The ELASTIC algorithm has been tested against 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 the CPU resources are initially equally shared 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, auto.

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

1.2.7 ELASTIC behaviour with TCP traffic

The first TCP test measured the difference in throughput when downloading at full speed. Traffic was generated with iperf in TCP mode, for 100 seconds. Figure 13 shows the results.

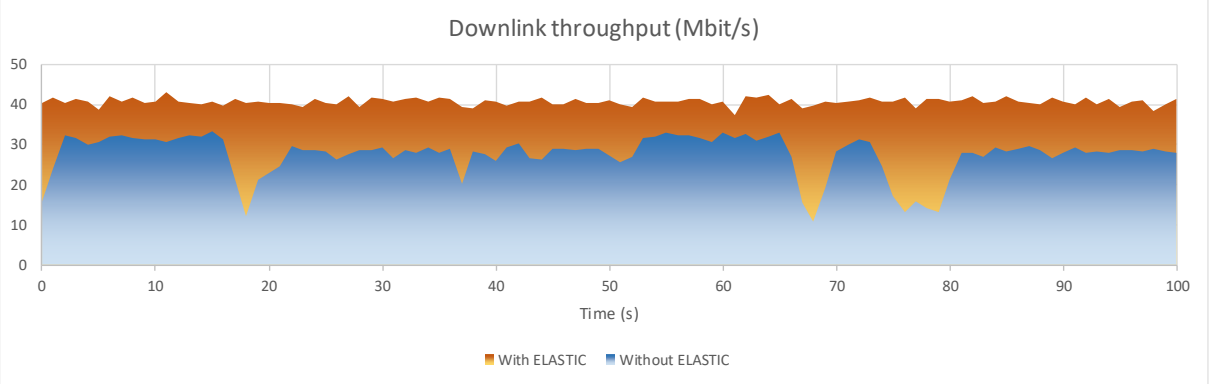

Figure 13 – Downlink throughput comparison.

As can be seen in the chart, the throughput is substantially higher when ELASTIC is active. This is explained by the switching to the PRB100 mode instead of keeping the default PRB75 mode. Table 3 contains a summary of the average throughput and the performance gain with ELASTIC.

Table 3 – Downlink results.

Average throughput	
Without ELASTIC:	27.5 Mbit/s
With ELASTIC:	40.7 Mbit/s
Improvement:	48.0%

The next test is similar to the previous one, but the flow direction is now uplink. The results are shown in Figure 14 – Uplink throughput comparison.Figure 14.

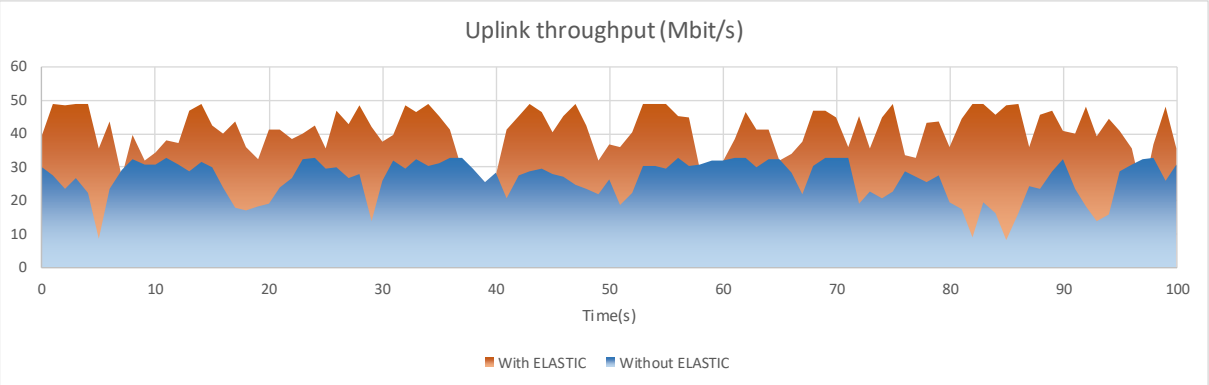

Figure 14 – Uplink throughput comparison.

The uplink throughput is more irregular, but the performance gains are also evident. During the test, at some points a peak throughput of almost 50 Mbit/s has been reached. Table 4 shows that the average gain with ELASTIC reached 55.6%, a very significative improvement.

Table 4 – Uplink results.

Average throughput	
Without ELASTIC:	26.3 Mbit/s
With ELASTIC:	40.9 Mbit/s
Improvement:	55.6%

Finally, the last test involved measuring performance gains with TCP full speed traffic in the downlink and uplink flows simultaneously. The results of this test can be seen in Figure 15.

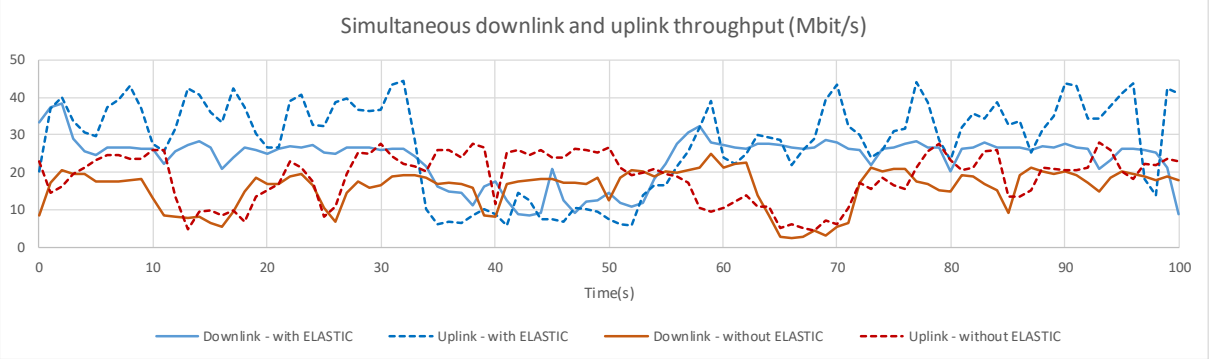

Figure 15 – Simultaneous downlink and uplink throughput comparison.

It's also obvious that the ELASTIC algorithm allows superior throughputs in this scenario. However, there's a time interval (from t=32 to t=58) where the ELASTIC mode performed worse than the normal mode. This can be explained by sporadic signal quality degradation that led to lost packets and consequently to a TCP congestion window size reduction. A new try could have been done and more favourable results could be obtained, but this kind of events is so frequent in this scenario that it deserves to be documented.

Table 5 – Downlink & uplink results.

Average throughput	Downlink	Uplink
Without ELASTIC:	15.9 Mbit/s	18.8 Mbit/s
With ELASTIC:	23.8 Mbit/s	28.2 Mbit/s
Improvement:	49.8%	49.8%

As can be seen in Table 5, when downloading and uploading simultaneously, the average downlink and uplink figures are substantially lower than the values of the previous tests. This may be explained by an increase in lost packets and latency due to the increased computational burden. Nevertheless, even with these lower figures, ELASTIC still reveals considerable gains.

1.2.8 ELASTIC behaviour with UDP traffic

An UDP test has also been conceived in order to evaluate the behaviour of ELASTIC under rapid changing traffic bursts of different throughputs, both in the downlink and uplink flows. Figure 16 illustrates the traffic profile that has been created to submit ELASTIC to a range of different challenges, namely UDP bursts of different throughputs (in downlink and uplink), simultaneous operation in downlink and uplink and sudden bursts in one direction when it is already operating with high throughput on the other direction.

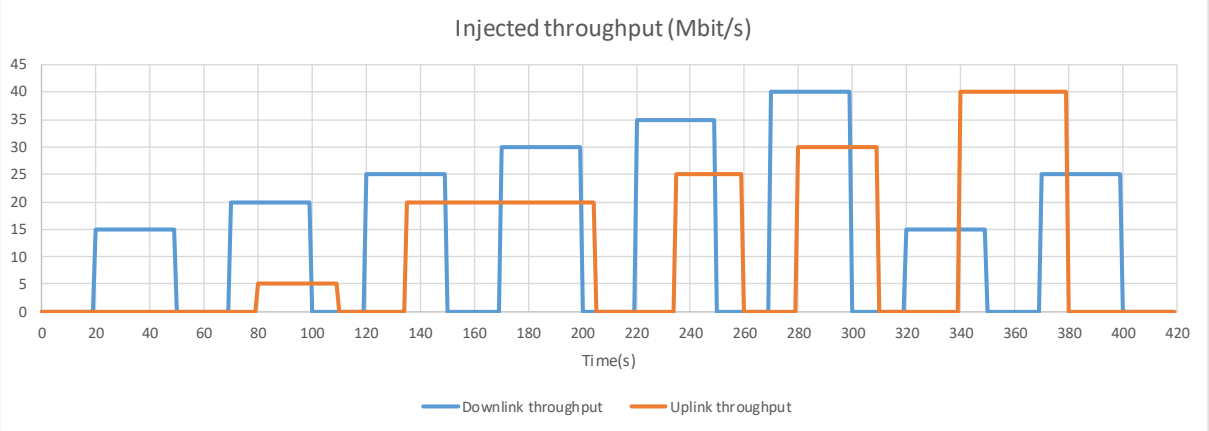

Figure 16 – UDP burst traffic profile.

In the first round, only the downlink flow has been tested, without and with ELASTIC. The results of this test can be observed in Figure 17

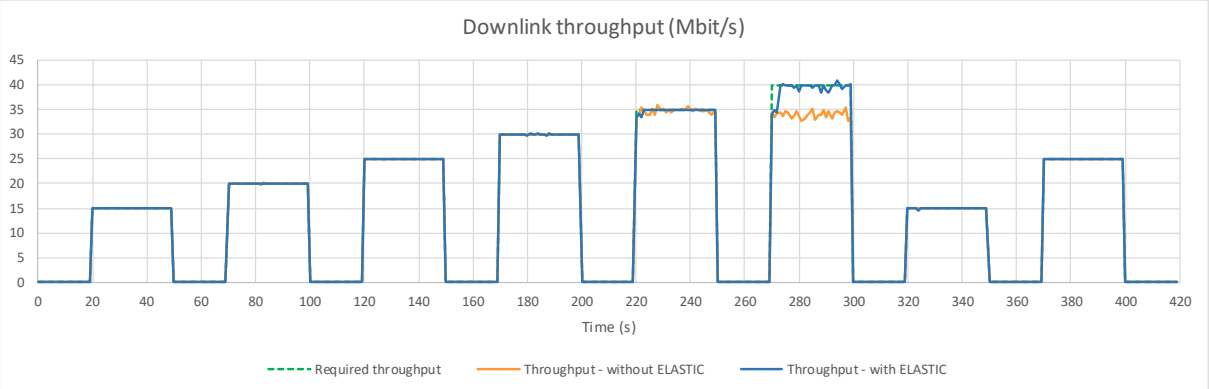

Figure 17 – UDP burst downlink results.

In this test scenario, both the modes (without and with ELASTIC) were able to cope with the requested traffic profile, except in the period from t= 270 s to t=300 s. In this time interval, the ELASTIC mode performed better, with a gain of 15.5% as can be seen in Table 6

Table 6 – UDP downlink results.

Average throughput	
Without ELASTIC:	34.0 Mbit/s
With ELASTIC:	39.2 Mbit/s
Improvement:	15.5%

The second round is similar to the first one; only the flow direction has been changed. Figure 18 shows the results of this test.

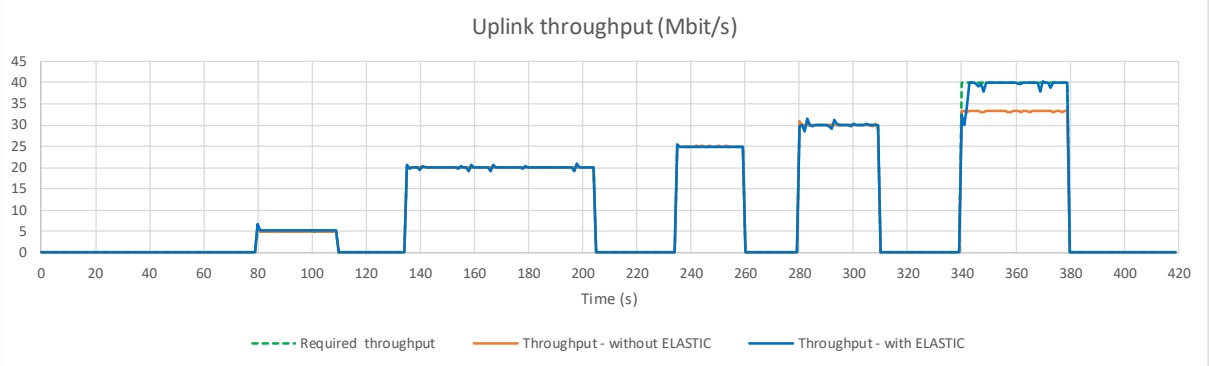

Figure 18 – UDP burst uplink results.

Like in the previous test, both modes were able to meet the desired throughput most of the time, except for the most demanding interval, between t=340 s and t=380 s. In this period, the ELASTIC mode performed 18.2% better, as can be seen in Table 7.

Table 7 – UDP uplink results.

Average throughput	
Without ELASTIC:	33.2 Mbit/s
With ELASTIC:	39.2 Mbit/s
Improvement:	18.2%

Finally, the third round explores the most demanding test: downlink and uplink at the same time. The results from this test can be seen in Figure 19 and Figure 20. These results were split in two charts, to simplify the comparison between the ELASTIC mode and the normal mode in downlink and uplink separately.

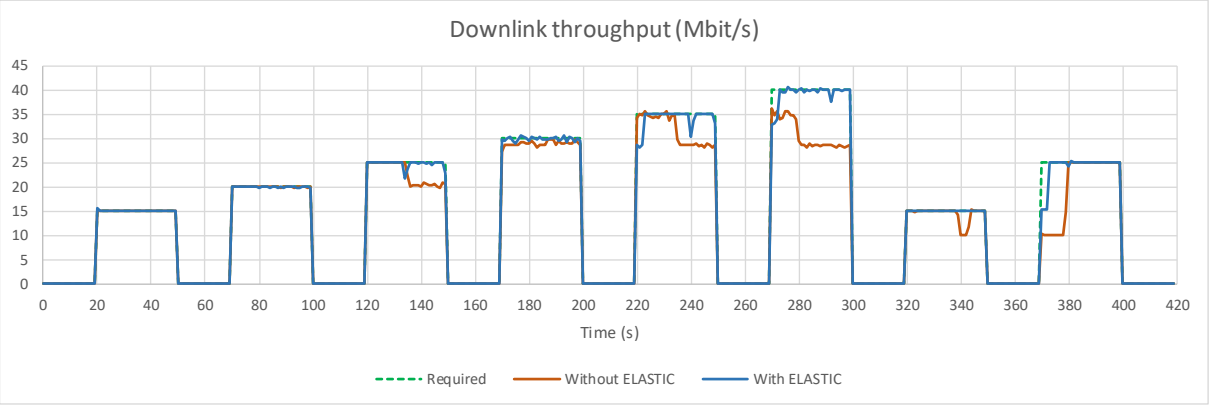

Figure 19 – UDP burst downlink/uplink - downlink results.

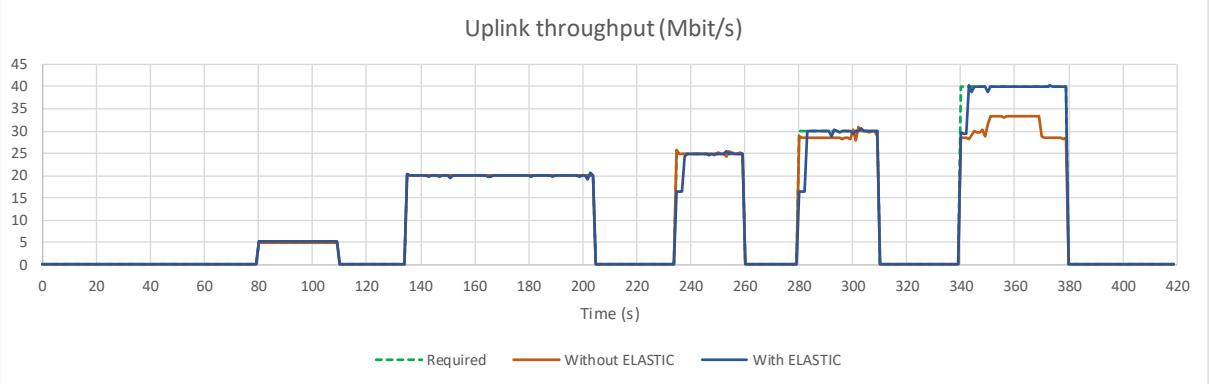

Figure 20 – UDP burst downlink/uplink - uplink results.

These charts 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 downlink and uplink.

1.2.9 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 the eNodeB process has been temporarily switched to higher PRB and MCS values, which requires more than the initial 100% available CPU capacity to work correctly. Figure 21 shows the CPU usage measured during the test, which corroborates this behaviour.

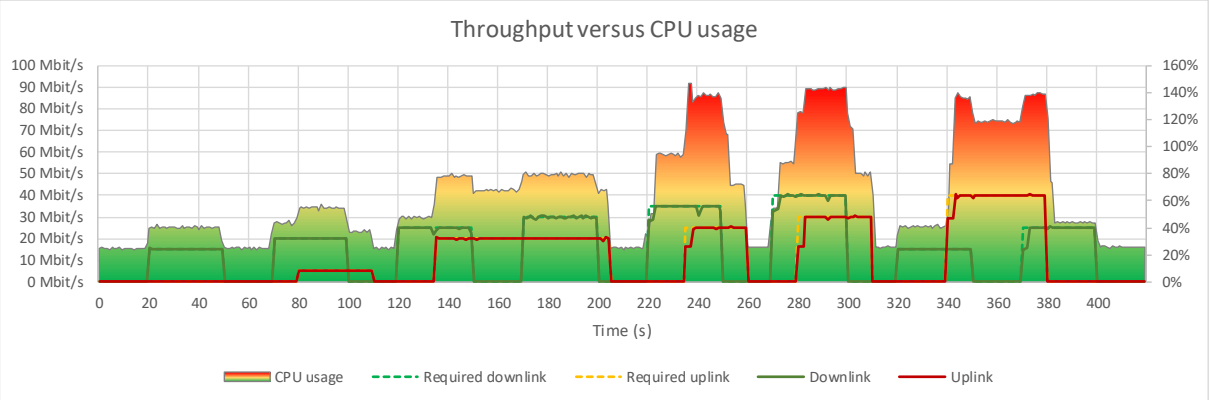

Figure 21 – ELASTIC UDP burst downlink/uplink throughput versus CPU usage.

It's evident 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 well above 100%.

Figure 22 supports a brief analysis of the measured CPU usage.

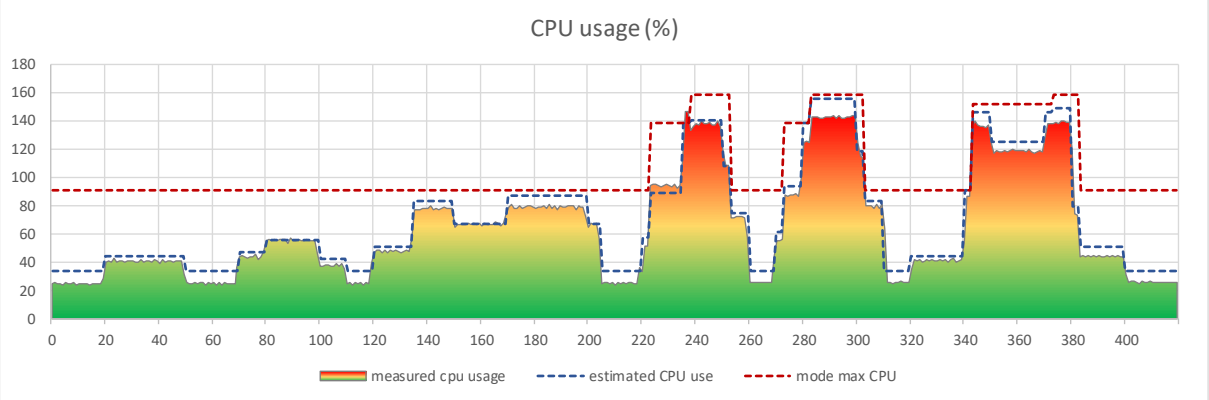

Figure 22 – ELASTIC UDP burst downlink/uplink – CPU usage analysis.

The gradient coloured area represents the measured CPU usage, as seen in the previous chart. The blue dotted line is the CPU usage estimation, calculated with the formula and values explained in section 1.2.3. Notice the similarity between the estimated and measured CPU usage. Finally, the red dotted line represents the maximum CPU usage of the chosen PRB/MCS mode at a specific time. Under low throughput this value is 91.2%, which corresponds to the maximum CPU usage of the default mode PRB 75, MCS auto, auto.

Each time the ELASTIC algorithm needs to switch to a PRB/MCS mode that requires more CPU resources than the initial value for the priority slice, it must allocate that CPU resources from the OTT slice. In this use case, the initial CPU allocated for each slice has been 100%, thus, in this particular test, the allocated CPU power from the OTT slice has the shape of the pattern illustrated in Figure 23.

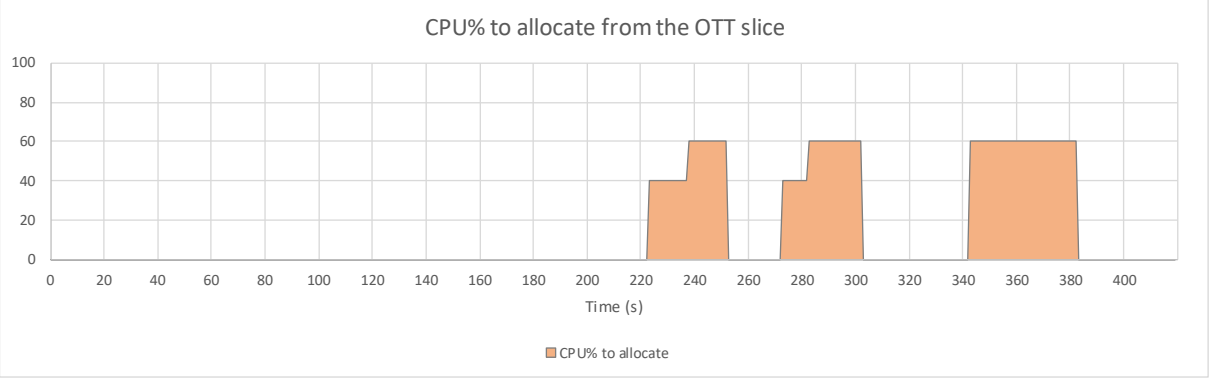

Figure 23 – ELASTIC UDP burst downlink/uplink – CPU allocation from the OTT slice.

The requested CPU value results from the new mode max CPU usage (the red dotted line of Figure 22), rounded to the next 10.

1.2.10 Consequences on the OTT slice

Whenever the ELASTIC algorithm requests CPU resources from the OTT slice, it must adapt to the new diminished available CPU power, changing the 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's absolutely necessary to switch to PRB/MCS values that can work correctly with the reduced 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 test performed on the high priority slice, the CPU allocation of Figure 23 has been recorded and replayed later in the OTT slice, when it was downloading

and uploading TCP traffic at full speed. Figure 24 illustrates the results which were obtained in this test.

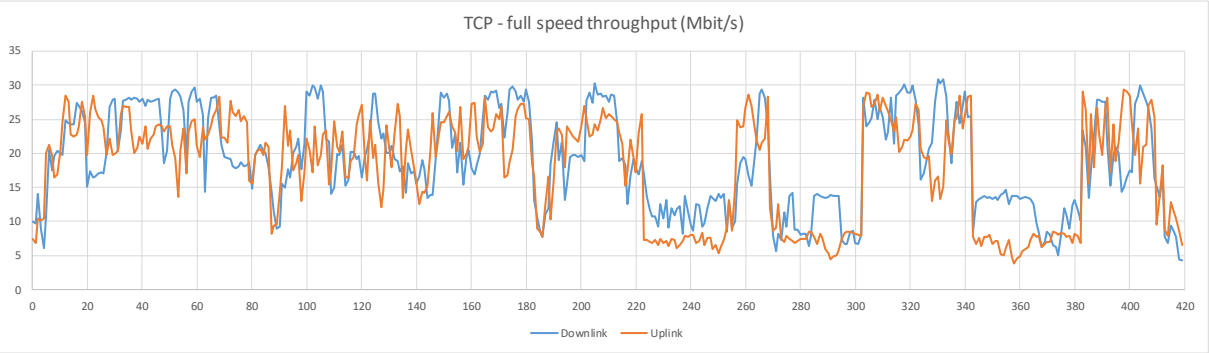

Figure 24 – OTT adaptation to resource starving – throughput view.

The reduction of throughput during the periods of CPU depletion is noticeable. Furthermore, it's also evident the priority given to the downlink flow by the ELASTIC algorithm, by choosing higher MCS values for this flow. This is a design feature, since in OTT slices most of the traffic flows in the downlink direction. Moreover, downlink traffic requires less CPU power than uplink traffic.

Figure 25 presents 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.

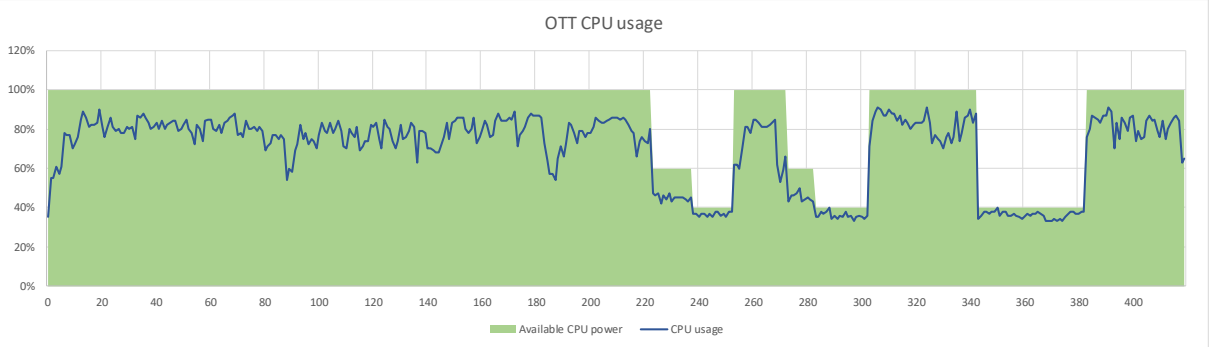

Figure 25 – OTT adaptation to resource starving – CPU usage view.

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.

1.3 Impact

Network slicing has born as an emerging business to operators (infrastructure owners), by allowing them to sell the customized slices to various tenants at different prices. In order to provide better-performing and cost-efficient services, network slicing involves resource management across the radio and cloud infrastructure.

In this context, this experiment made a substantial contribution to Allbesmart's expertise concerning resource management of cloud RAN slices, knowledge that is crucial to leverage its QoS/QoE analytics framework - UXPERT to the SDN/NFV domain in order to feed resource management algorithms and contribute towards the optimization of network slicing deployments.

Thanks to the ORCA facility we were able to substantially increase our knowledge about cloud RAN technologies and test resource management algorithms using radio equipment that otherwise would be beyond our reach.

2 FEEDBACK TO ORCA

2.1 Testbeds/hardware/software resources used

Please indicate what is used in your experiment or extension in the table below. Please describe in additional paragraph in case the used facility/resource is not listed in the tables.

TESTBEDS	Required (Yes/No)
w.iLab.t (Heterogeneous wireless testbed @ imec, Ghent, Belgium)	
IRIS (Software Defined Radio testbed @ TCD, Dublin, Ireland)	Yes
ORBIT (20 x 20 radio grid testbed @ Rutgers University, New Jersey, US)	
IMEC portable testbed	
TUD macro scale testbed (Macro scale testbed @ TUD, Dresden , Germany	
KU Leuven testbed (KU Leuven @Leuven, Belgium)	

SDR HARDWARE PLATFORMS	Number of nodes required
Nutaq ZeptoSDR	
Nutaq picoSDR	
PicoZed Xilinx Zynq®-7000 SoC	
USRP B200-mini	
USRP E310	
USRP N210	
USRP X310	
USRP 2920	
USRP 2921	
USRP RIO 2942R	
USRP RIO 2943R	
USRP RIO 2952R (+ GPS)	
USRP RIO 2953R (+ GPS)	
USRP RIO 2953R (+ EBD)	
WARPv2	
Xilinx ZC706 Evaluation Kit - Zynq® 7000 SoC + AD FMCOMM radio frontend	
ZedBoard Xilinx Zynq®-7000 SoC	

ZedBoard Xilinx Zynq®-7000 SoC + AD FMCOMM radio frontend	
BB – NI PXI 7975 Module	
BB – NI PXI 7965 Module	
FE – NI PXI 5644	
FE – NI PXI 7976R	
USRP B210	2

SOFTWARE TOOLS	Required (Yes/No)
IRIS Software Radio (SRSLTE)	Yes
GNU Radio	
NI LabVIEW	
Xilinx Vivado Design Suite v2015.4 for RFNoC related development	
Xilinx Vivado Design Suite v2016.2 and Analog Device AD9361 HDL Reference Design	
Xilinx Software Development Kit (SDK)	
NI LabVIEW Full Duplex	
NI LabVIEW Massive MIMO	
NI LabVIEW Communications System Design Suite based GFDM flexible transmitter	
LabVIEW Communications LTE Application Framework	
LabVIEW Communications 802.11 Application Framework	
NI L1-L2 API	
ns-3 network simulator (LTE + WiFi module)	
Time-Annotated Instruction Set Computer (TAISC)	
Generic Internet-of-Things ARchitecture (GITAR)	

2.2 Feedback on the usage

This section collects feedback information, please use a scale of 1 to 5 to give answer where rating is expected, 1 being extremely unsatisfied and 5 being perfectly satisfied.

2.2.1 Feedback on the testbed and experimentation tools

Please share your experience regarding the testbed usage, such as whether it is easy to get acquainted with the testbed:

- How did you experience the learning curve regarding using the ORCA facility? **5**
- How do you rate the documentation provided for the testbeds supported in ORCA? **4**
- How did you experience the experimentation tools, such as jFed, or software-defined radio related toolkits. **5**
- Did you make use of all requested testbed infrastructure and hardware resources, as specified in your Open Call proposal? If not, please explain. **yes**
- Did you have enough time to conduct your Experiment/Extension in the testbed(s) offered by ORCA? **yes**
- Were the results of your Experiment/Extension below / in line with / exceeding your initial goals and expectations? **exceeding**
- What were the hurdles / bottlenecks? What could not be executed? Was this due to technical limitations? In case you experienced technical limitations, please specify them. **There were some problems running srslte in IRIS, but they were solved**
- How was your experience with particular experimentation tools, such as jFed or remote access of other software toolkits? **5**

2.2.2 Feedback on the interactions and communications

Please share your experience with respect to the administrative process, patron communication, and support received from the ORCA consortium:

- How do you rate the level of work for administration / feedback / writing documents / attending conference calls or meetings compared to the timeframe of the Experiment/Extension? **5**
- How was your experience concerning the communication and support of your Patron? **5**
- Is there any other kind of support that you would expect from the patron, which is not available today? **Not at all, he has been extremely supportive and helpful.**

2.2.3 Main added value and what is missing?

ORCA was extremely useful to support this experiment, by allowing our company to have remote access to equipment and resources, namely USRPs and computational nodes, that are usually beyond our reach. The way how different network scenarios can be easily created within JFED, even interconnecting nodes of distinct testbeds, has been perceived as a major advantage of ORCA.

There are no shortcomings to report.

3 PROMOTIONAL MATERIALS

Experimental validation of resource management algorithms for elastic network slicing (ELASTIC)

allbesmart

Goals of the experiment

The main objective of this experiment is the validation of elastic resource management algorithms able to serve multiple Network Slice Instances (NSI) over the same physical resources while optimizing the allocation of computational resources to each slice based on its requirements and demands.

The experiment deploys a use case on top of the IRIS testbed that provides 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 MVNO Public Safety service with high throughput and reduced latency requirements and the other emulating an OTT service provider (delay tolerant – best effort slice). A resource management algorithm is implemented and evaluated in terms of performance gains when operating under computational resource limitations.

Main challenges

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

Experiment setup

The experiment setup created in IRIS uses four computing nodes, as can be seen in the diagram shown in Figure 1. Machine A implements the EPC and eNodeB components of the LTE network, while machine B contains the UE component. Machine C is used to exchange traffic patterns with the UE through the LTE network, using the iperf tool. Finally, machine D implements the ELASTIC algorithm: it receives traffic and CPU usage data from the two probes and determines the actions to perform in order to comply with QoS requirements.

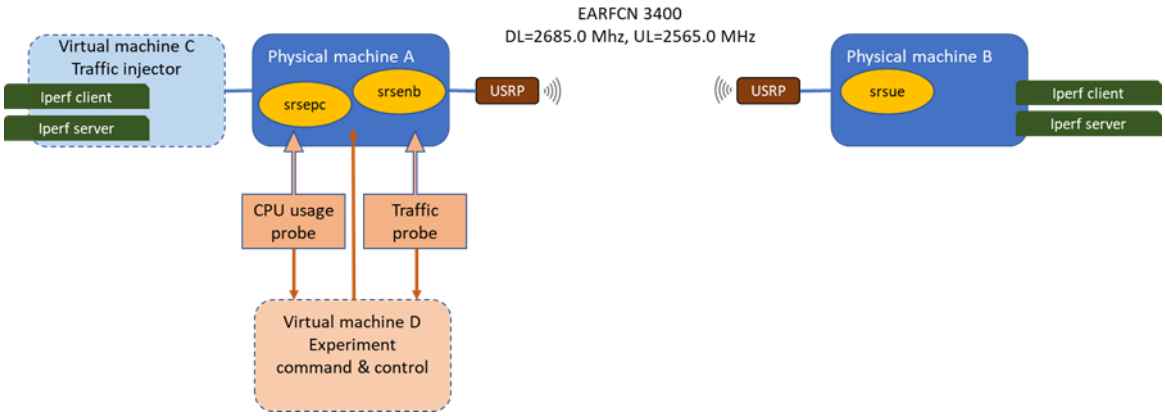

Figure 1 - Experiment setup in IRIS.

The two B210 USRPs are configured in single antenna mode, using the LTE EARFCN frequencies: DL=2685.0 MHZ, UL=2565.0 MHz. Access to each virtual machine is achieved through JFED, using SSH terminal sessions.

Main results

The ELASTIC algorithm proved to be very effective to increase the TCP throughput of the high priority slice if more CPU resources are required to comply with stringent QoS requirements. Testing revealed gains of 48% in downlink, 55,6% in uplink and 49,8% in simultaneous downlink and uplink. Figure 2 shows the results of the TCP downlink test.

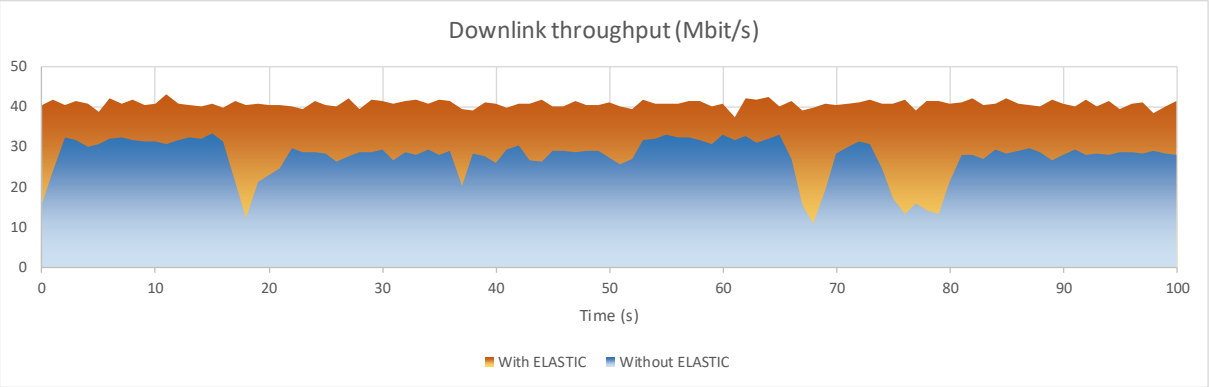

Figure 2 – TCP download throughput comparison.

ELASTIC was also successful dealing with UDP traffic bursts, even with high throughput demand in both directions at the same time. Figure 3 illustrates how ELASTIC deals with UDP traffic bursts and its impact on CPU usage.

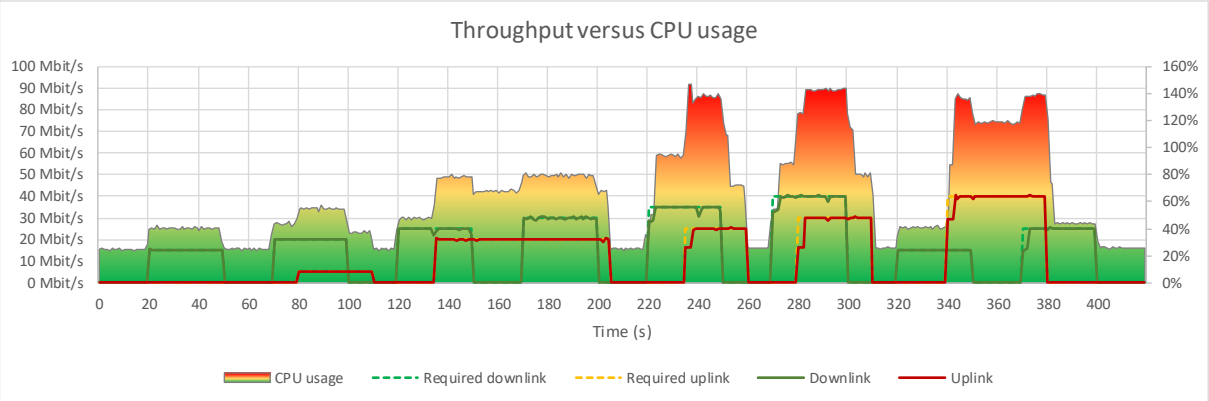

Figure 3 – How ELASTIC handles UDP traffic bursts.

Conclusions

When two competing cloud RAN LTE slices are implemented over the same computational infrastructure, optimized management of computational resources is an effective instrument to ensure that the high priority slice can cope with demanding QoS requirements under shortage of computational resources. The ELASTIC algorithm implemented in this experiment proved to be effective to increase the performance of the priority slice without disrupting the operation of the low priority slice.

Feedback

ORCA was extremely useful to support this experiment, by allowing our company to have remote access to equipment and resources, namely USRPs and computational nodes, that are usually beyond our reach. The way how different network scenarios can be easily created within JFED, even interconnecting nodes of distinct testbeds, has been perceived as a major advantage of ORCA.

Quote

“Thanks to the ORCA facility we were able to substantially increase our expertise on cloud RAN technologies and test resource management algorithms using radio equipment that otherwise would be beyond our reach.”

REFERENCES

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

[2] D. M. Gutierrez-Estevez, "The Path Towards Resource Elasticity for 5G Network Architecture", 2018.

[3] Ismael Gomez-Miguelez et al., "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*, 2016.

[4] "Software Radio Systems," [Online]. Available: https://www.softwareradiosystems.com/about-us/. [Accessed 25 10 2019].