An adaptive network slicing for LTE Radio Access Networks

Pedro H. A. Rezende and Edmundo R. M. Madeira
Institute of Computing
University of Campinas (Unicamp)
Campinas, São Paulo, Brazil
Emails: pedrohenrique@lrc.ic.unicamp.br, edmundo@ic.unicamp.br

***Abstract*—5G mobile systems are envisioned to satisfy the service requirements from a diversity of vertical industries. Network Slicing, which is a promising technology to be integrated into 5G systems, enables multiple virtual networks to be created on top of a physical substrate. These multiple virtual networks (or network slices) are tailored according to the users' needs. The consolidation of multiple technologies, such as SDN and NFV, provides all the elasticity, programmability and modularity necessary to manage network slices. In this paper, we present a Slice Optimizer component as an extension to LTE's evolved NodeB to realize the concept of network slicing on LTE Radio Access Networks. This proposed component communicates with an SDN Controller to receive information regarding the network slices and adapts the slices according to the network state. Simulations were performed to validate the Slice Optimizer and highlight the benefits that can be achieved with our proposal, such as the improvement of user's QoS experience due to a more efficient use of network resources.**

***Index Terms*—Network Slicing, 5G, LTE, SDN, QoS provisioning.**

I. Introduction

Upcoming 5G mobile systems are intended to satisfy the needs of a variety of vertical industries such as healthcare, media, manufacturing and automotive. Each vertical industry has its own service requirements, including ultra-low latency, high mobility and always-on connectivity. As a consequence, the "one-size-fits-all" architectural approach existent in today's networks is incapable to address this diversity of vertical markets.

One of the key technologies expected to be integrated into 5G systems is Network Slicing [1]. Network Slicing allows multiple virtual networks to be created on top of a common physical substrate, being mutually independent, instantiated on-demand and with independent management. These virtual networks (or network slices) are tailored to meet the needs of each vertical market. Software Defined Networks (SDN) [2] and Network Functions Virtualization (NFV) [3] are two enablers of Network Slicing. They provide all the elasticity, programmability and modularity necessary to manage network slices. It is expected that SDN and NFV will play a big role on future 5G mobile systems.

The contributions of this paper are twofold. First, we introduce Slice Optimizer (SO), a novel component, embedded in LTE's evolved NodeB (eNB), responsible to realize the concept of network slicing in the LTE downlink channel. Second, we show that the SO can dynamically reassign resources to network slices according to the network state, improving the Quality of Service (QoS) perceived by the users. The SO receives information regarding network slices from an Orchestrator, which is embedded in an SDN Controller. LTE networks serve as a cornerstone for the development of upcoming 5G mobile systems. The implementation and simulations to validate the SO are performed in the Network Simulator 3 (NS-3)[4], and the results confirm the effectiveness of our proposal.

The remainder of this paper is organized as follows. Section II presents some background concepts on Wireless Software Defined Networks and LTE Downlink Scheduling. Section III shows some related work. Section IV shows our system architecture and the proposed extension, whereas Section V details their implementation. Section VI presents the considered scenario and the results derived from simulations. Finally, Section VII concludes the paper.

II. Background

This section introduces Wireless Software Defined Networks and provides an overview of LTE Downlink Scheduling.

A. Wireless Software Defined Networks

Software Defined Networks (SDN) propose the separation between two planes in IP Networks, the first is responsible for controlling the network and the second is responsible for forwarding flows according to rules enforced by the first. In this way, the hardware belonging to the forwarding plane becomes simplified, performing a reduced number of functions. The control plane is consolidated in a software component, named controller, facilitating policy enforcement and network configuration. The controller communicates via its Southbound Application Programming Interface (API) with devices composing the forwarding plane. The controller can offer services to higher-level components such as network applications through its Northbound API.

The efforts on SDN so far have mostly focused on wired network. However, wireless networks can also benefit from SDN. Such benefits include: improving end-user connectivity, localization and QoS enforcement; increasing the security of the wireless network; and helping to allocate communication channels as well as to reduce the interference among access points [5].

B. LTE Downlink Scheduling

The LTE packet scheduler is responsible for allocating radio resources to the user equipments (UEs). The scheduler operates at the MAC layer of the eNB. The frame structure of the downlink air interface contains ten subframes of 1 ms each, which is also the transmission time interval (TTI) [6].

In each subframe, the scheduler makes the scheduling decision based on some information such as Radio Link Control (RLC) buffer status, QoS received by upper layers and the Channel Quality Indicator (CQI) reported by the UE. QoS constraints can be provided by the Evolved Packet System (EPS)

bearer, which is associated with a QoS Class Identifier (QCI). To maximize the UEs throughput, the Adaptive Modulation and Coding (AMC) block selects the proper modulation and coding scheme (MCS). The Hybrid Automatic Repeat Request (HARQ) is responsible for the retransmission of lost packets.

III. Related Work

In [7], the authors propose a slicing scheme to slice an LTE network into many virtual networks. These virtual networks are held by different Service Providers (SP) and the minimum amount of resources blocks allocated to each SP is defined by a service contract. The authors in [8] propose a joint resource provisioning and admission control scheme with the goal to maximize the total rate of network slices based on their users' channel state information. The main difference between our paper and these two works is that we rely on an SDN architecture.

In [9], the authors expose a network slicing management and orchestration framework. The proposed framework can be employed by network operators to automate, deploy and configure multiple infrastructure resource domains. The authors in [10] propose a framework responsible to perform the negotiation, selection and assignment of slices to end users in 5G networks. However, none of these works validated their proposed frameworks through simulation or experiments.

The authors in [11] present CellSlice, which is a system responsible to realize the concept of network slicing in the RANs of WiMax technology. The work proposed in [12] aims to virtualize the infrastructure of WiFi networks, allowing the creation of multiple virtual WLANs. Each virtual WLAN offers different types of service. Different from [11] and [12], in our work we focus on the LTE technology.

In this paper, we propose a component responsible to perform network slicing on the RAN of LTE networks. The proposed component receives information from an SDN Controller. We validated our component using different number of UEs and network policies; and we analyzed four different QoS metrics in the experiments: delay, jitter, packet loss and throughput.

IV. System Architecture and Proposed Extension

In this section, we present our system architecture and our proposed component, called Slice Optimizer (SO), as an extension to the eNB. Fig.1 depicts our system model consisting of two major elements, the SDN Controller and the eNB. The eNB has three interfaces, one wireless interface, which provides the wireless connectivity between the eNB and the UEs, and two wired interfaces, connecting the SDN Controller and the Evolved Packet Core (EPC) to the eNB. In this paper, the SDN Controller is constrained to communicate only with one or more eNBs and, therefore, no communication exist between the SDN Controller and the LTE EPC. The Controller is only responsible for the slicing aspects and, therefore, the EPC is still responsible for the security, billing and so forth.

The SDN Controller is responsible for the life cycle management and control of the slices, such as their creation, instantiation, orchestration and deactivation. For the instantiation of new slices, templates could be used for most common type of services [13]. These templates would be available to mobile users via the Controller's Northbound API, being accessed by a User Application.

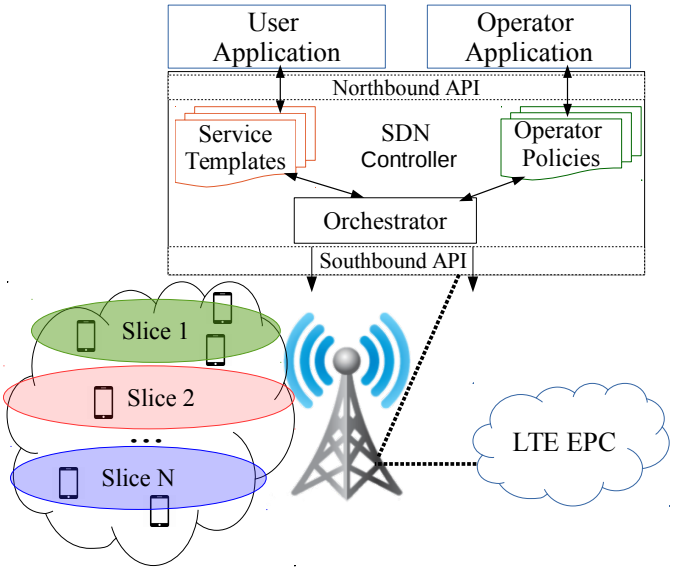


Fig. 1: System Architecture with some slices instantiated.

The Operator Application is used by the network operators to define policies. The prioritization of real-time services like remote surgery over non-real-time services is an example of such policies. Different from the traditional approach, the SDN Controller through its Orchestrator module is responsible for allocating the air resources to UEs. The Orchestrator relies on the operators' policies and on templates chosen by the end users to allocate the radio resources to each slice. In our proposal, the resources are allocated to each slice in subframes, meaning that all the resource blocks in a subframe have to be assigned to a single slice.

The Orchestrator can allocate more than one subframe to a slice per frame. Moreover, since a frame contains ten subframes, the orchestrator can allocate at maximum ten slices per frame. The resource allocation process executed in the orchestrator produces as an output an Allocation Array (AA) of size 10, where the index is the subframe number and the value is the slice identifier. The AA has size 10 because a frame is composed of 10 subframes. The orchestrator can reallocate these resources and create a new AA periodically if, for example, a new policy is added by the network operator.

After allocating the resources to slices, the orchestrator sends a message containing the AA and a set of slices descriptors to the SO via a southbound control protocol like OpenFlow, NetConf or FlexRan [14]. The slice descriptor consists of a set of fields representing the slice: the slice identifier; a set of tuples (IMSI, QCI) to differentiate types of traffic (flows) from the same UE, where IMSI is the International Mobile Subscriber Identity used to identify the UE, and the QCI stands for QoS Class Identifier; and the slice priority. The slice priority is defined according to policies determined by the network operator.

After receiving the message sent by the SDN Controller's Southbound API, the SO is responsible to process this message and store the AA and the slices descriptors into the Slices' Storage. The role of the SO is to select the best slice to be scheduled in the current subframe, and not necessarily choose the slice defined in the AA. The best slice is the one defined in the AA for the current subframe whenever this slice has data to transfer; otherwise, the best slice is the one with highest

priority that has data to transfer. In such manner, the SO can reduce the wastage of downlink channel resources that could happen if, for example, there is a sudden change of the network state or whenever the SDN Controller does not apply a proper policy to allocate the radio resources. Moreover, relying on the SO to select the best slice according to the network state releases the SDN Controller from creating new AAs frequently, which could give some scalability problems in the case that the SDN Controller needs to reallocate radio resources for hundreds or thousands of eNBs at the same time.

Fig.2 illustrates the SO along with the communication of entities relevant for downlink MAC scheduling. The basic information for the operation of SO are: a set of slices descriptors; an AA representing which slice is allocated to each subframe; the current subframe and the buffer status of each tuple (IMSI, QCI). The buffer status represents the amount of data remaining to be transferred and it is updated by the RLC entity whenever there is new data to be transferred.

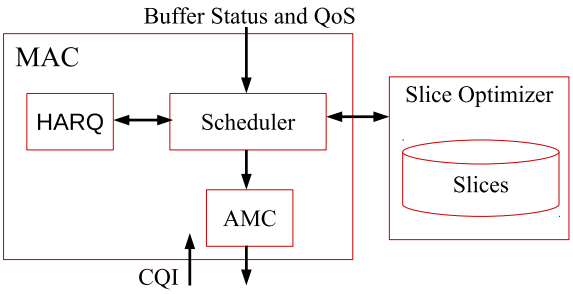


Fig. 2: Slice Optimizer and entities relevant for downlink MAC scheduling.

V. Implementation Details

In this section, we provide the implementation details of the Slice Optimizer along with the SDN Controller in NS-3. NS-3 is a discrete-event network simulator in which the simulation core and models are implemented in C++ language. We also used the embedded NS-3 LTE module [15]. To implement our proposal, we have created three C++ classes, each of them representing the SDN Controller, Orchestrator and the Slice Optimizer. NS-3 does not have any protocol implemented for the communication between the SDN Controller and the eNB. For this reason, all the communications between these three classes are just C++ function calls. In our implementation, a slice is a set of flows having the same application type.

The SDN Controller class provides two external APIs, one to receive policies from the Operator Application as well to offer Service Templates to the User Application; and another API to communicate with the Slice Optimizer. Currently, there are only three templates implemented: VoIP, video (streaming) and file transfer; which means we have only three slices, one for each application type. There are two policies implemented, one to prioritize VoIP over video and another to prioritize video over VoIP. The prioritized traffic receives 5 subframes while the other gets 4 from the 10 subframes available in each frame. In both policies, the file transfer is treated as the lowest priority traffic, receiving only 1 subframe.

The Orchestrator is a module residing in the SDN Controller responsible for the creation of the AA based on policies and requested service templates; and for sending the AA created as well as the slices descriptors to the SO. The creation of AA by the Orchestrator is relatively simple. First it verifies the requested templates with the aim to know the slices that have UEs registered to them. If all slices have UEs registered, the operator checks its policies, giving 5 subframes to the slice that has the highest prioritized traffic, 1 subframe to the file transfer slice and 4 subframes to the remaining slice. The orchestrator uses a function to randomly distribute these slices in the AA, always respecting the number of subframes to each slice. If the VoIP slice has no requested templates, the video slice receives 9 subframes of the AA; and vice-versa. If the VoIP and video slices do not have any requested templates, the file transfer slice receives 10 subframes of the AA.

Before discussing the SO implementation details, it is important to mention that the NS-3 simulator does not fully support per-flow scheduling mechanism. This means that even if more than one EPS bearer is defined per UE, only one is used. As a consequence, in our implementation we considered only one EPS bearer per UE, which means the IMSI is enough to verify the buffer status of each UE and, therefore, only the IMSI value from the tuples (IMSI, QCI) is going to be provided to the scheduler. Nevertheless, the QCI will be utilized later when the NS-3 simulator fully implement per-flow scheduling mechanism. Furthermore, NS-3 provides several downlink packet schedulers; however, we only implemented the communication between the SO and the Proportional Fair (PF) packet scheduler. The PF uses the Radio Network Temporary Identifier (RNTI) to identify UEs while the SO and the Controller use the IMSI. In this case, we have implemented a function in the PF to translate these RNTIs to IMSIs and vice-versa.

The SO, which is a class embedded in the eNB, provides two APIs, one to receive messages from the SDN Controller, and another to communicate with the LTE scheduler. The Slices Storage is just a C++ Map used to store the slice identifier as key and the slice descriptor as value.

Traditionally, in every subframe, the LTE scheduler is responsible for allocating resource blocks to UEs. In our implementation, the scheduler, before allocating resource blocks to the UEs, requests the SO which slice will be scheduled in this subframe. The SO, in turn, selects the best slice to be scheduled in this subframe according to Algorithm 1.

The Algorithm 1 receives from the scheduler the following arguments: an integer representing the current subframe ($subFrame$) and a set containing the buffer status of all UEs ($bufferStatusSet$). In Line 2, the slice's identifier representing the slice defined in AA ($allocationArray$) in this current subframe is assigned to $sliceID$. In Line 3, the $sliceID$ is used as a key in $sliceStorage$, which in turn returns the corresponding slice descriptor ($actualSliceDesc$). The for loop from lines 5 – 8 returns to the scheduler the UEs (IMSIs) from $actualSliceDesc$ if at least one UE from $actualSliceDesc$ has data waiting to be transferred. Otherwise, the algorithm goes to Line 9.

The variables $maxTraffic$ and $maxPriority$ represent respectively the sum of traffic to be transferred and the priority of the best slice so far. The for loop from lines 11 – 24 iterates over all slices, except the one defined in the $allocationArray$, to select the best slice. In every iteration, a slice descriptor is chosen ($sliceDesc$). If its priority, represented as $slicePriority$, is lower than the $maxPriority$, the algorithm proceeds to the next slice descriptor. Otherwise, the algorithm sums the traffic waiting to be transferred in all UEs

belonging to the corresponding $sliceDesc$, and assigns this sum to $sumTraffic$ (Lines 16 – 18). If the $sumTraffic$ is equal to 0, the algorithm proceeds to the next slice descriptor. Otherwise, if this $sumTraffic$ is higher than $maxTraffic$, or whenever $SlicePriority$ is higher than $maxPriority$, the algorithm goes to Lines 22-24. In these lines, the best slice will be updated by the current $sliceDesc$. Then, the algorithm proceeds to the next for-loop iteration. After iterating over all slices descriptors, Algorithm 1 returns to the scheduler a set containing the UEs (or IMSIs) belonging to the best slice.

As can observed by Algorithm 1, the best slice is the one defined in AA for this subframe whenever this slice has data to be transferred; otherwise, the best slice is the one which has highest priority and data waiting to be transferred. After receiving the set, the scheduler according to its own scheduling policy allocates the resource blocks only to this set.

Algorithm 1 Select Slice to be scheduled.

```
1: procedure SELECTSLICE(subFrame, bufferStatusSet)
2:     sliceID ← allocationArray[subFrame]
3:     actualSliceDesc ← slicesStorage[sliceID]
4:     dataTraffic ← 0
5:     for each IMSI in actualSliceDesc.ues do
6:         dataTraffic ← bufferStatusSet[IMSI]
7:         if dataTraffic > 0 then
8:             return actualSliceDesc.ues
9:     maxTraffic ← 0
10:    maxPriority ← 0
11:    for each sliceDesc in slicesStorage − {actualSliceDesc} do
12:        slicePriority ← sliceDesc.Priority
13:        sumTraffic ← 0
14:        if slicePriority < maxPriority then
15:            continue
16:        for each IMSI in sliceDesc.ues do
17:            dataTraffic ← bufferStatusSet[IMSI]
18:            sumTraffic +← dataTraffic
19:        if sumTraffic = 0 then
20:            continue
21:        if sumTraffic >= maxTraffic or slicePriority >
   maxPriority then
22:            maxPriority ← slicePriority
23:            maxTraffic ← sumTraffic
24:            bestSliceDesc ← sliceDesc
25:    return bestSliceDesc.ues
```

VI. Performance Evaluation

In this section, we evaluate the performance of our implementation using two network slicing schemes: static and dynamic. In the static slicing, the Orchestrator populates the AA and sends it to the SO, which in turn is obligated to provide the scheduler the slice defined by the AA in the current subframe, even if there is no data to transmit on the slice in this subframe. In the dynamic slicing (Algorithm 1), the Orchestrator also populates the AA and sends it to the SO, but the SO only provides to the scheduler the slice defined by the AA in the respective subframe if the slice has data to transfer; otherwise, the SO assigns to the scheduler the highest priority slice that has data to transfer.

A. Simulation Scenario

The simulation scenario is composed of a single cell, one eNB, one EPC, one SDN Controller, one traffic generator and several users receiving VoIP, FTP and Video traffic. The scenario has three slices. Slices 1, 2 and 3 are for VoIP, FTP and Video traffic, respectively. There are several UEs, varying from 10 to 40, with increments of 10. Each UE receives traffic from only one application. The traffic generator, which sends traffic to the eNB via the EPC, is responsible for generating the data traffic to be delivered to the UEs. The wired connection between the traffic generator and the EPC has bandwidth of 100Gb/s and latency of 1 ms. The VoIP traffic is mapped to QCI 1 (Guaranteed Bit Rate - Conversational Voice), the FTP traffic is mapped to QCI 9 (Non-Guaranteed Bit Rate - Video (Buffered Streaming) or TCP-Based Application) and the Video traffic is mapped to QCI 4 (Guaranteed Bit Rate - Non-Conversational Video (Buffered Streaming)). Table I outlines the traffic model employed in the simulation.

TABLE I: Traffic Model.

Application	VoIP	FTP	Video
Slice	1	2	3
Description	G.711[1]	TCP Bulk[2]	MPEG 4[3]
BitRate	64 Kbps	N/A	525 Kbps
QCI	1	9	4
Percentage of UEs	60%	10%	30%

[1] On/Off Model. OnTime = 0.352; OffTime = 0.650
[2] Defined in: http://www.nsnam.org/doxygen/tcp-bulk-send_8cc_source.html.
[3] Trace-based. We used the N3Talk trace, which is available in http://www-tkn.ee.tu-berlin.de/research/trace/ltvt.html.

Two different operator policies were evaluated. One prioritizing the Video slice over the VoIP slice, producing an $AllocationARRAY_A$ (AA_A); and another policy that prioritizes the VoIP slice over the Video slice, creating an $AllocationARRAY_B$ (AA_B). The policy was not changed during the simulation, which means the SDN Controller did not create a new AA throughout the simulation. To simulate a suburban scenario, we used the Kun 2600 MHz Propagation Loss Model and a fading model generated with the *fading-trace-generator.m* script, which is provided by the NS-3. The HARQ process is enabled. Table II summarizes the main configuration parameters used in this paper.

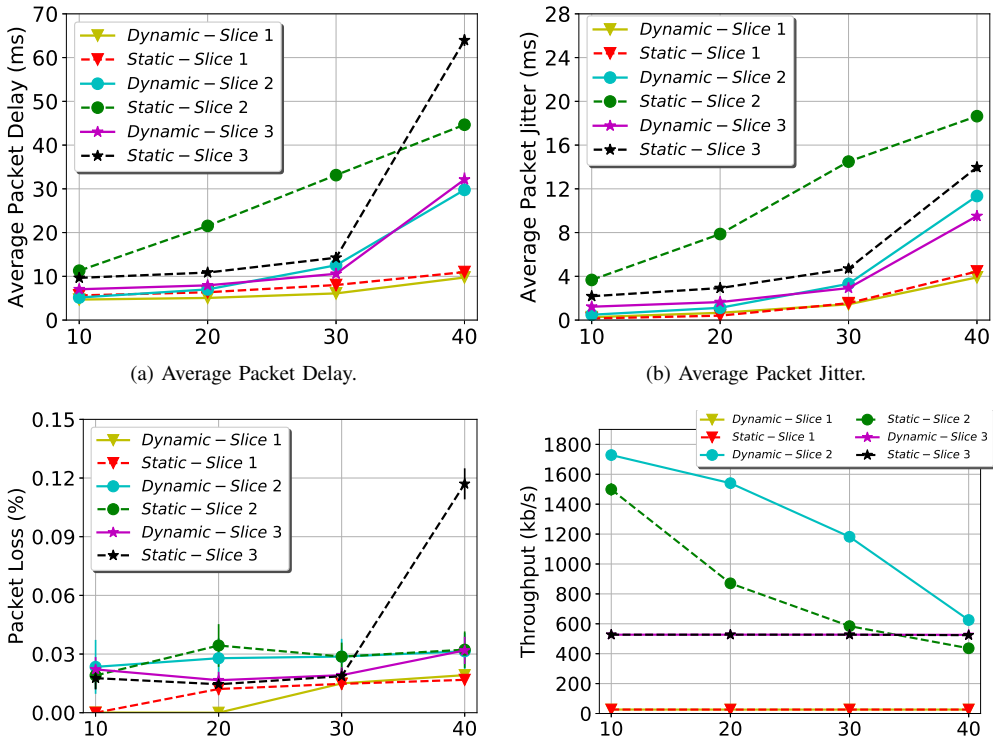
TABLE II: Simulation Parameters.

Parameter	Value
System Type	Single Cell
Cell Radius	100 m
System Frequency	2655 MHz
Propagation Model	Kun 2600 MHz Propagation Loss Model
eNB TX power	30 dBm
UE TX power	10 dBm
System Bandwidth	5 MHz (25 Physical Resource Blocks)
MAC scheduler	Proportional Fair
RLC Mode	Unacknowledged Mode (UM)
RLC Buffer Size	30 Kbytes
UE Speed	3 Km/h
Allocation $ARRAY_A$	$(3, 1, 2, 1, 3, 3, 1, 3, 1, 3)$
Allocation $ARRAY_B$	$(1, 1, 2, 1, 3, 3, 1, 3, 1, 3)$
Number of Replications	34
Simulation Duration	120 s

B. Simulation Results

The figures presented in this section show average values for four QoS metrics: delay, jitter, packet loss and throughput. We performed 16 experiments, one for each combination of AAs, slicing scheme and number of UEs. The 95% confidence intervals are presented in the figures. Notice that they are small.

Figure 3 exposes the average QoS values per UE for AA_A. The AA_A prioritizes the video slice over the VoIP slice. Figures 3(a) and 3(b) depict respectively the average packet delay (APD) and average packet jitter (APJ) per UE. As can be seen in these figures, the dynamic scheme considerably reduces the APD and the APJ for slices 2 and 3 in comparison to the static scheme, not mattering the number of UEs in the network. Also, the dynamic scheme slightly decreases the APD and the APJ for slice 1. The dynamic scheme provides better results because subframes not used by a slice are given



(a) Average Packet Delay.

(b) Average Packet Jitter.

(c) Average Packet Loss.

(d) Average Throughput.

Fig. 3: Average QoS values per UE in each Slice as a function of the number of UEs for AA_A.

to another slice. Therefore, packets will be served faster by the scheduler when the dynamic scheme is employed, reducing the time a packet will stay in the RLC buffer and, consequently, decreasing the APD.

Similarly, since packets are served faster by the scheduler when the dynamic scheme is used, the jitter will decrease as there will be a reduction of the delay variation. For the scenario with 40 UEs, the APDs for slice 2 and slice 3 are respectively 44 ms and 64 ms using the static scheme, while the APDs for the same slices in the dynamic scheme are 30 ms and 32 ms, respectively. Likewise, for the same scenario, the APJ decreases around 38% for slice 2 and 30% for slice 3 when the dynamic scheme is employed.

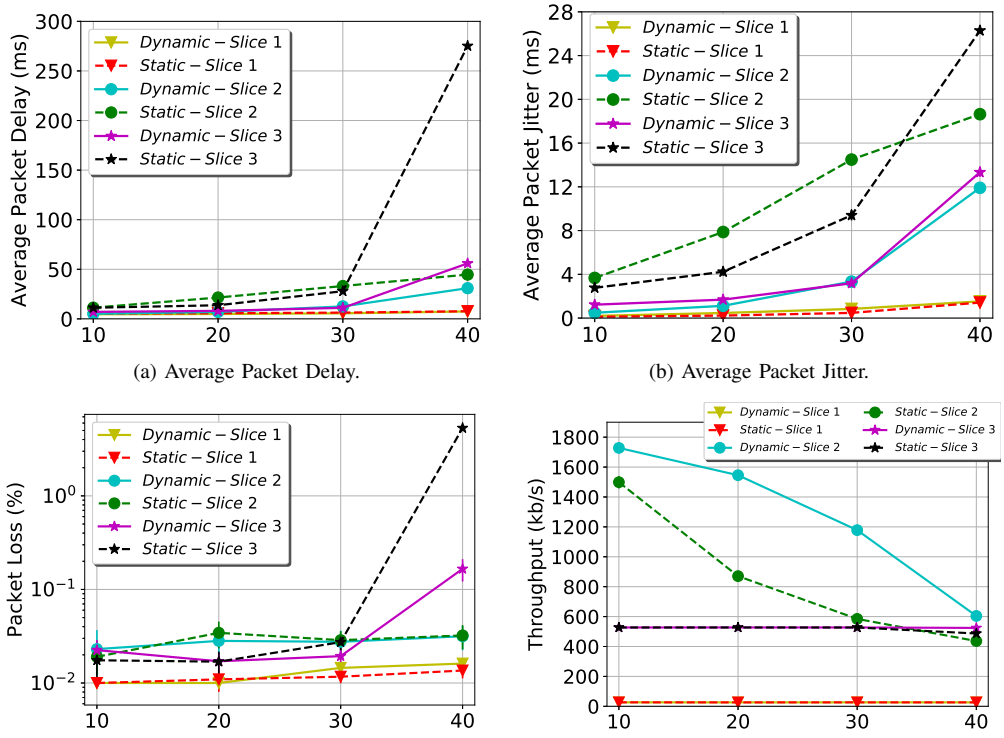
Figure 3(c) depicts the average packet loss (APL) rate per UE. The packet loss rate is less than 0.15% for all slices, not mattering the slicing scheme used and the number of UEs in the network. As shown in Figure 3(a), the APD of slice 3 using static scheme is considerable. However, even with this APD, the packets rarely fill completely the RLC buffers and, therefore, only a small number of packets are being dropped by the RLC entity.

In Figure 3(d) the average throughput (AT) per UE is presented. As can be seen in the figure, in slices 1 and 3 there is no considerable difference in using static or dynamic schemes. Nevertheless, in slice 2, the AT increases in all scenarios when the dynamic scheme is employed. This happens, because the TCP traffic always tries to get more resources and, as a consequence, when the SO realizes that the slices 1 and 3 do not have data waiting to be transmitted in the current subframe, the SO allocates the subframe to the FTP traffic, increasing its throughput. In the scenario with 40 UEs, using dynamic slicing, the AT per UE in slice 2 is 38% higher than the AT per UE in the same slice using static slicing.

Figure 4 presents the average QoS values per UE for AA_B. The AA_B prioritizes the VoIP slice over the video slice. Figures 4(a) and 4(b) expose the APD and the APJ per UE, respectively. Both figures are similar to Figures 3(a) and 3(b); however, in Figures 4(a) and 4(b), the APD and APJ of slice 3 increases substantially when the static scheme is employed, especially in the scenario with 40 UEs. This happens, because the slice 3 has only four subframe in policy AA_B, while the same slice has five subframes in policy AA_A. Therefore, in the static scheme using policy AA_B, the scheduler serves a lower amount of packets from slice 3 and, consequently, the APD and APJ greatly increases. On the other hand, using the dynamic scheme, the SO allocates unused subframes to slice 3, reducing the APD and APJ. In the scenario with 40 UEs, the APD and APJ of slice 3 in the static scheme are respectively 5 and 2 times higher in comparison to the same scenario in the dynamic slicing.

In Figure 4(c) the APL per UE is illustrated. The figure is in logarithmic scale. The figure is similar to Figure 3(c); nevertheless, in the scenario of 40 UEs, the APL greatly increases in slice 3 using the static scheme, reaching 5.5%. The reason of this increase is due to the high APD in slice 3, as showed in Figure 4(a). Since packets are waiting a long time to be scheduled in slice 3 using the static scheme, the RLC buffers become full and, consequently, packets are dropped. In contrast, in the dynamic scheme, packets are served faster by the scheduler and therefore the RLC buffers do not become full, since unused subframes are used by the slices.

Figure 4(d) presents the AT per UE. The main difference between this figure and Figure 3(d) is that in the scenario of 40 UEs, using the dynamic scheme, the AT of UEs belonging to slice 3 is 8% higher. This happens, because a good amount of packets from slice 3 are dropped in the scenario of 40 UEs



(a) Average Packet Delay.

(b) Average Packet Jitter.

(c) Average Packet Loss.

(d) Average Throughput.

Fig. 4: Average QoS values per UE in each Slice as a function of the number of UEs for AA_B.

when the static scheme is employed, as shown in Figure 4(c).

As can be seen by the results in this section, the dynamic slicing implemented in the SO gives a better QoS experience to users in comparison to the static slicing, in both policies evaluated and all scenarios. The dynamic slicing outperforms the static scheme since the former uses more efficiently the radio subframes than the latter. In this way, the SO can adapt to the network state and, even if the network operator does not apply a proper policy in the network, the SO can still offer a good QoS experience to end users.

VII. Conclusion

In this paper, we have proposed an extension to LTE's eNB called Slice Optimizer aiming to realize the concept of network slicing in LTE networks. The Slice Optimizer is responsible to feed the eNB's scheduler with network slicing information. The Slice Optimizer can reassign resources to network slices according to the network state, not relying solely on information provided by the SDN Controller. Simulations show that the Slice Optimizer adapts to the network state, using more efficiently the network resources and, consequently, improving users' QoS experience. As future work, we plan to integrate virtualized network functions into the slices and perform simulations with varying traffic conditions.

Acknowledgment

The authors would like to thank CAPES for the Ph.D. grant. This work is part of the INCT of the Future Internet for Smart Cities (CNPq 465446/2014-0, CAPES 88887.136422/2017-00 and FAPESP 2014/50937-1).

References

[1] J. Ordonez-Lucena *et al.*, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, May 2017.

[2] D. Kreutz *et al.*, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[3] R. Mijumbi *et al.*, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[4] "ns-3 network simulator." [Online]. Available: https://www.nsnam.org/

[5] C. Chaudet and Y. Haddad, "Wireless Software Defined Networks: Challenges and opportunities," in *2013 IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems (COMCAS 2013)*, Oct 2013, pp. 1–5.

[6] O. Grndalen *et al.*, "Scheduling Policies in Time and Frequency Domains for LTE Downlink Channel: A Performance Comparison," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 4, pp. 3345–3360, April 2017.

[7] M. I. Kamel, L. B. Le, and A. Girard, "LTE Wireless Network Virtualization: Dynamic Slicing via Flexible Scheduling," in *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)*, Sept 2014, pp. 1–5.

[8] S. Parsaeefard *et al.*, "Joint resource provisioning and admission control in wireless virtualized networks," in *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, March 2015, pp. 2020–2025.

[9] A. Devlic *et al.*, "NESMO: Network slicing management and orchestration framework," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 1202–1208.

[10] V. K. Choyi *et al.*, "Network slice selection, assignment and routing within 5G Networks," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct 2016, pp. 1–7.

[11] R. Kokku *et al.*, "CellSlice: Cellular wireless resource slicing for active RAN sharing," in *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, Jan 2013, pp. 1–10.

[12] K. Guo, S. Sanadhya, and T. Woo, "ViFi: Virtualizing WLAN Using Commodity Hardware," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 18, no. 3, pp. 41–48, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2721896.2721905

[13] N. Nikaein *et al.*, "Network Store: Exploring Slicing in Future 5G Networks," in *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*, ser. MobiArch '15. New York, NY, USA: ACM, 2015, pp. 8–13.

[14] X. Foukas *et al.*, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16, 2016, pp. 427–441.

[15] "Ns-3 lte-epc network simulator (lena)." [Online]. Available: http://networks.cttc.es/mobile-networks/software-tools/lena/