Online based learning for predictive end-to-end network slicing in 5G networks

EL Hocine Bouzidi*‡, Abdelkader Outtagarts‡, Abdelkrim Hebbar‡, Rami Langar* and Raouf Boutaba§
‡ Nokia Bell Labs, Villarceaux Center - Route de Villejust 91620 Nozay, France
* University Paris-Est, LIGM-CNRS UMR 8049, UPEM, F-77420, Marne-la-Vallee, France
§ University of Waterloo, 200 University Ave. W., Waterloo, ON, Canada
E-mails: el_hocine.bouzidi@nokia.com; {abdelkader.outtagarts, abdelkrim.hebbar}@nokia-bell-labs.com;
rami.langar@u-pem.fr; rboutaba@uwaterloo.ca

***Abstract*—5G networks are expected to provide a variety of services over the same physical infrastructure equipped with a combination of radio and wired transport networks and leveraging network virtualization and Software Defined Networking (SDN). In particular, network slicing is envisioned as a promising solution to enable optimal support for heterogeneous services sharing the same infrastructure. To this end, we design and implement, in this paper, an SDN based architecture for end-to-end network slicing which proactively and dynamically adapts radio slices to the transport network slices. The developed architecture enables the creation, modification and continuity of radio and transport network slices while considering their resource and Quality-of-Service (QoS) requirements. It leverages Machine Learning for predicting radio slices capacities, improving network resource utilization, and predicting congestion within each network slice. We also formulate this network slicing problem as a Linear Program (LP) aiming to minimize the total network delay. Finally, we propose an efficient heuristic algorithm with low time complexity and high estimation accuracy to solve large problem instances. Experimental results using the OpenAirInterface (OAI) platform, FlexRAN, ONOS SDN Controllers and OpenvSwitch demonstrate the efficiency of our approach in terms of guaranteeing low latency and high network throughput.**

***Index Terms*—SDN, Prediction, Cloud-RAN, QoS, ONOS, FlexRAN, Network Slicing**

I. Introduction

The fifth generation (5G) is envisioned to be a multi-service network supporting a wide range of users needs and accommodate a multitude of services, with stringent and heterogeneous requirements. These services require a programmable and a flexible infrastructure, allowing them to share the same Radio Access Network (RAN) and transport network, while guaranteeing the QoS. Software-Defined Networking (SDN) [1] is one of the key emerging technologies for the 5G vision, allowing the providers to control and orchestrate their resources and enable the sharing of the same physical network infrastructure through the concept of Network slicing [2].

Network slicing enables optimal support for heterogeneous services, by running them in an independent Virtual Networks (VNs), on a common shared Physical Network Infrastructure (PNI), such that each VN is allocated a fixed or dynamic portion of the PNI resources. Network slicing is generally applicable in both RAN referred to as radio slicing and transport and Core Network domains, referred to as transport network slicing. Radio slices share radio resources among diverse services (i.e., Internet of Things (IoT), enhanced Mobile BroadBand (eMBB)) by reserving an appropriate portion of bandwidth (i.e., a number of Radio Resource Blocks (RBs)) to be used from that slice for a specific time interval [3]. Transport network slicing consists in creating isolated Packet Networks (PN) composed of physical or virtual switches assigned to a tenant by the network provider that, in turn, owns the corresponding physical resources [4].

Creating an isolated and efficient end-to-end network slice remains challenging, particularly when considering the bandwidth prediction of radio slices to scale in and out the PN slices, as well as congestion inside each PN slice, while taking into account the slices QoS.

To this end, we design and implement, in this paper, an SDN based architecture for end-to-end network slicing which proactively and dynamically adapts radio slices to the transport network slices. The developed architecture enables the creation, modification and continuity of radio and transport network slices while considering their resource and QoS requirements. It leverages Machine Learning for predicting radio slices capacities, improving network resource utilization, and predicting congestion within each network slice. We also formulate this network slicing problem as a Linear Program (LP) aiming to minimize the total network delay. Finally, we propose an efficient heuristic algorithm with low time complexity and high estimation accuracy to solve large problem instances. In addition, we validate our proposal using an experimental Cloud-RAN (C-RAN) prototype, which makes use of OpenAirInterface (OAI) [5], an open-source software implementation of LTE Evolved Packet Core (EPC) [6], the ONOS SDN controller [7], and the Software Defined-RAN (SD-RAN) FlexRAN controller [8].

To the best of our knowledge, we are the first to propose and validate such proactive end-to-end network slicing using an experimental C-RAN and transport testbed.

The main contributions of our paper can be summarized as follows:

- First, we design and implement a Northbound SDN application on top of ONOS and FlexRAN controllers to enable dynamic creation of end-to-end network slices (i.e., eMBB, IoT).

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

- Second, we propose a Machine-Learning based algorithm to predict the capacities of radio slices and adapt them to the PN slices in order to improve network resource utilization, and QoS requirements.
- Third, we formulate the end-to-end network slicing problem as a Linear Program (LP) aiming to minimize the total network delay. We then propose an efficient heuristic algorithm with low time complexity and high estimation accuracy to solve large problem instances.
- Forth, we validate our proposal using an experimental prototype.

The remainder of this paper is organized as follows. Section II presents the related works. In Section III, we discuss the architecture of our framework and the proposed rules placement algorithm. Section IV evaluates the proposed method. We finally conclude this paper in Section V.

II. Related Works

Recently, there has been a dense specific research interest for network slicing on both RAN and Core Network (CN) domains, and its impact on resource management [9] [10].

Most of the literature on Core Network converge towards a common commitment, which consists in separating control plane and data plane of the Core Network EPC nodes (i.e., Mobility Management Entity (MME), Serving Gateway (S-GW), Packet Data Network Gateway (P-GW)). In [11], authors propose new architecture in which they decouple the control plane from the data plane of the S/P-GW by using SDN controllers and GPRS Tunneling Protocol (GTP) based OpenFlow switches to route User Equipment (UE) data. Although using the SDN controller in the Core Network gives more flexibility, the authors do not consider the forwarding of the GTP packets in the S1-U interface, between the RAN and S-GW, where a single path routes all UEs traffic.

In several works, the concept of radio slicing is considered as a RAN sharing issue. Authors in [3] [12] proposed a radio slicing solution for enabling efficient coexistence of eMBB and IoT services, by implementing a northbound SD-RAN application which enables the creation of IoT slices on demand, while considering the QoS requirements.

On the other hand, Much researches leveraging Machine Learning (ML) in QoS-aware routing problem specifically in SDN based networks [13]. Authors in [14], showed the outperformance of the ML technique Long Short-Term Memory (LSTM) on (Auto-Regressive Integrated Moving Average) ARIMA in terms of network routing optimization and congestion prediction based on SDN.

To sum up, all the above works present the following shortcomings: i) the non-consideration of routing the UE packets in S1-U interface based on GTP, ii) the non-consideration of slicing continuity based on the UE connection and finally iii) the non-consideration of traffic prediction when it comes to end-to-end network slicing. Based on these observations, we address in this paper the design and implementation of a system capable of: i) forwarding the UE traffic in multi paths in the S1-U interface, by patching OpenvSwitch [15] to be able to handle GTP traffic, and ii) creating an end-to-end network slices by considering the bandwidth prediction of radio slices to scale in and out the PN slices, as well as network congestion inside each PN slice, while taking into account the flows QoS.

III. SDN-based Dynamic Network Slicing Continuity and Traffic Prediction

In this section, we detail our approach for SDN-based dynamic network slicing continuity and traffic prediction to enable efficient end-to-end slicing. Firstly, we explain the overall system architecture. Thereafter, radio slicing, packet network slicing, end-to-end slicing and proactive end-to-end slicing will be described.

A. SDN-based dynamic network slicing continuity and traffic prediction Framework

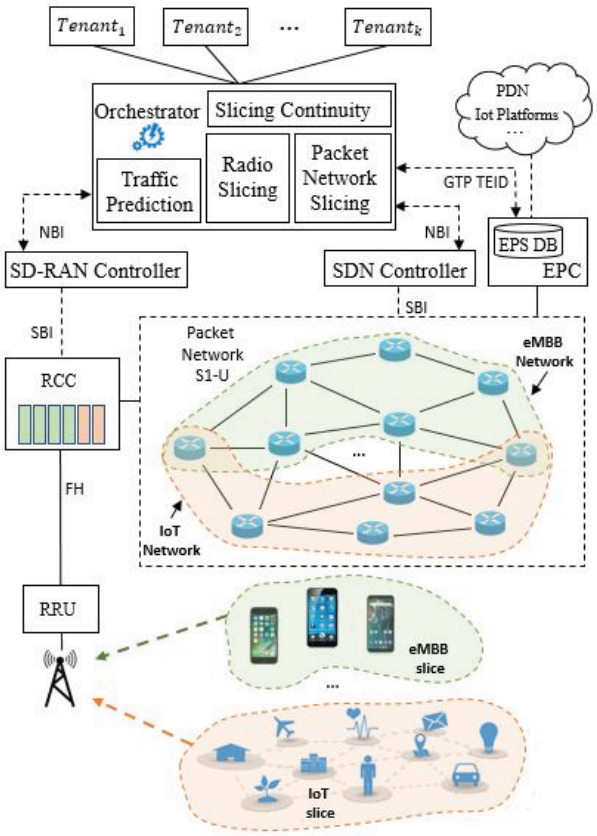

Fig. 1. SDN-based dynamic network slicing continuity and traffic prediction Framework

Fig. 1 presents the system architecture considered in this paper, which consists of two interconnected technology domains. The first is the Radio domain, which provides mobile broadband and IoT services. According to the C-RAN concept [16], the RAN functionalities are split between Remote Radio Units (RRUs) and Radio Cloud Center (RCC). Moreover, we deploy C-RAN following an SD-RAN architecture, in which the RAN control and data planes are separated using a Radio Controller.

The second is the Transport domain, which provides connectivity services to the Radio domain. This domain consists of SDN switches, that we modified to handle the GTP traffic, referred to us Packet Network (PN) that connects the RAN to the Core Network. More specifically it connects the RCC to: a) the MME through the S1-MME interface and b) the S-GW through the S1-U interface, where the traffic can be routed through multiple paths. Moreover, we centralize the control plane using an SDN Controller on top of these devices. The interface between the SDN Controller and the devices is defined by the OpenFlow protocol [17].

On top of the SD-RAN and SDN Controllers an Orchestration layer is deployed, which automatically handles slicing (i.e., creation, modification, continuity, etc.). In what follows, we detail these properties.

B. Radio Slicing

The radio slicing is performed by the orchestration layer module Radio Slicing as shown in Fig. 1. This module communicates with the SD-RAN Controller through the northbound API and dynamically performs slicing based on statistics collected and stored in the RAN Information Base (RIB) in the SD-RAN Controller. The latter in turn sends slicing decisions to the scheduler agent located in the RCC node. In this architecture, the slice creation requests come from the tenants for subscribers located in a specific area. Through the northbound API, the Orchestrator can create radio slices by setting a set of parameters, such as the fraction of RBs that the corresponding slice is allowed to use, and the QoS.

C. Packet Network Slicing

In the current Evolved Packet System (EPS) architecture [6], the traffic flowing from a RCC can be routed only through one single path in the S1-U interface, even if there are multiple paths. Since data are transported with GTP protocol, where the same outer headers (Ethernet, IP and Transport) are added to UE packets, SDN based routing of these packets using the information encapsulated on these layers through different paths is not possible. On the other hand, the UE connections can be identified by a specific field Tunnel Endpoint Identifier (TEID) of the GTP header encapsulated on top of these layers. The idea is thus to deploy a set of bridges capable of routing GTP traffic based on TEID identifiers. To this end, we have extended the open source implementation of OpenvSwitch, to support the GTP forwarding feature. Based on preinstalled GTP rules, the GTP packets can be forwarded to a specific switch ports (i.e., Datapath) and through multiple paths.

A PN slice is hence composed of OpenvSwitwh switches and links capable of routing the traffic flow from a radio slice and ensure continuity, as will be explained in the next section.

D. End-to-End network slicing

Fig. 2 illustrates our proposed procedures of End-to-End network slicing according to the SDN paradigm. It is composed of two main steps. The first one is Slices Creation, in which the tenant starts by creating both radio and PN slices, and associate their informations such as $slice\_id$, the set of OpenvSwitch devices and their corresponding links and ports, to the subscribers International Mobile Subscriber Identity (IMSI) and save them in the centralized Database (EPS DB). The second step happens when a new connection is established (i.e., after UE Initial Attachment, Bearer creation, exchange of UpLink and DownLink TEIDs between RCC and S-GW). In this case, the SD-RAN triggers the Orchestrator of the new connection event. The latter then, associates the connection TEIDs to the already saved informations such as $slice\_id$, and installs remotly through the SDN Controller the corresponding GTP Flow Rules. Then the UE data starts to flow between RCC and S-GW.

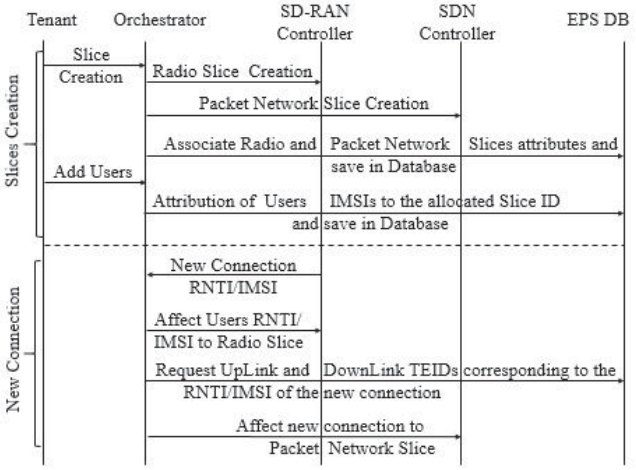

Fig. 2. End-to-End network slicing procedures

E. Proactive end-to-end Slicing

To meet the requested flow's QoS requirements, the sizes of Radio slices must be adjusted proactively and dynamically. To this end, we suppose that, initially, the number of resource blocks that the tenant's radio slice is allowed to use is fixed as a fraction of the whole bandwidth. However, we propose to proactively allocate additional unused resource blocks from other radio slices to the overloaded slice, which exceeds its authorized capacity, by predicting the future evolution of resource blocks requirement, using the Linear Regression method (LR) [18]. On the other hand, these radio slices must be adapted proactively and dynamically to the PN slices. Furthermore, while scaling in/out the PN slices, we distribute the traffic load over the same slice topology. Then, to avoid congestion, we proactively route flows following the paths with less delay and large capacity. To this end, we build on our previous contributions [18] [19], in which we deployed two main modules on the SDN Controller. The first module is the Network Measurement that periodically collects statistics (i.e., latency, throughput, etc.) from the PN slices and stores them in a centralized Traffic Matrix. The second module is the Proactive Forwarding in which we aim to balance the traffic load over the slice topology and predict congestion by rerouting flows to the less delayed paths. This problem can

be defined as a rules placement problem, where the objective is to determine which link to route which flow in order to minimize the total network delay and balance the network load by minimizing link utilization while scaling in/out the PN slice. We formulate this problem as a LP, written as follows:

- Inputs:
  - Network topology: $G(V_i, E_i)$
  - Traffic matrix: TM_i
  - Predicted Traffic Matrix: $\widehat{TM_i}$
  - Matrix of link delay: $D_i = \{d_{(u,v)}\}$
  - Matrix of predicted link delay: $\widehat{D_i} = \{\widehat{d}_{(u,v)}\}$
  - Matrix of link delay thresholds: $TS_i = \{TS_{(u,v)}\}$
  - Size of flow f allocated to (u, v): $S_{(f,u,v)}$
  - Set of flows: $F = \{f_1, f_2, f_3, ..., f_l\}$
- Objective:
  - Minimize $\sum_{(u,v)\in E_i} d_{(u,v)} \times LU_{(u,v)}$
- Constraints:
  - Delay limitation: $\forall (u, v) \in E_i : Max(d_{(u,v)}, \widehat{d}_{(u,v)}) < TS_{(u,v)}$
  - Link capacity limitation: $\forall (u, v) \in E_i : LU_{(u,v)} < MLU_{(u,v)}$
  - Path capacity limitation: $\forall (u, v) \in P$: $MLU_{(u,v)} - LU_{(u,v)} \leq Cap\_Av\_Path(P)$
  - Demand satisfaction: $\forall (u, v) \in E_i, \forall f \in F : \sum S_{(f,u,v)} \leq RSD_i$

$$Scale\_Out_i = \begin{cases} 1 & if\ \widehat{RSD_i} > TSD_i \\ 0 & Otherwise \end{cases}$$

Where $MLU_{(u,v)}$ (Maximum Link Utilization of link (u, v)) denotes the adjusted maximum link capacity in order to avoid congestion. It is defined as follows: $MLU_{(u,v)} = c_{(u,v)} * \theta_{(u,v)}$, where $c_{(u,v)}$ denotes the link capacity, and $0 < \theta_{(u,v)} < 1$ is a constant depending on link characteristics. Also, $LU_{(u,v)}$ determines the utilization of the link (u, v), which is defined as follows: $\forall f \in F : LU_{(u,v)} = \sum (E_{(f,u,v)} \times S_{(f,u,v)})$, where the symmetric Binary matrix $E = \{e_{(f,u,v)}\}$ denotes whether the flow rule f is allocated to the link (u, v) or not. The capacity and delay limitation constraints force each link (u, v) to not exceed its threshold MLU and to not be delayed. The demand satisfaction constraint ensures the PN slice to send or receive the same amount of traffic from the Radio slice and proactively scale in/out the PN slice if the predicted traffic exceeds certain threshold, where TSD_i and $\widehat{RSD_i}$ denote, respectively, the threshold and predicted demand of Radio $Slice_i$.

Solving the proposed LP in the Orchestrator for each incoming flow could be time consuming, as the size of the network and the number of flows increase. Consequently, the computational complexity increases exponentially. Therefore, such approach is not feasible in practice, since it generates high overhead due to the frequent updates of the flow tables.

To cope with this problem, and reduce the computation time and complexity, we propose a simple yet efficient heuristic algorithm, called Congestion prediction and Load balancing Rule placement algorithm (CLR). CLR can efficiently find a feasible and near optimal solution, while minimizing the total network latency and packet loss.

Algorithm 1: CLR Rule placement algorithm

```
1:  procedure CONGEST_REM(G(V_i, E_i), TM_i, TM̂_i)
2:      for all link (u, v) ∈ E_i do
3:          t ← 0
4:          while Max(d_(u,v), d̂_(u,v)) > TS_(u,v)
                    or LU_(u,v) > MLU_(u,v) do
5:              CP ← Get_Congested_Path(u, v)
6:              F ← Sorted_Flows(CP)
7:              R ← F[t].flow
8:              SP ← Sorted_Backup_Paths(R)
9:              for each path p ∈ SP do
10:                 if p can route R then
11:                     Install_Rule(p, R)
12:                     Remove_Rule(CP, R)
13:                     Break
14:             t ← t + 1
15: end procedure
16: procedure LOAD_BALANCING(G(V_i, E_i),TM_i, TM̂_i)
17:     for all flow f ∈ sorted F do
18:         list_P ← Paths that can route f in Slice_i
19:         for all P ∈ list_P do
20:             if Scale_Out_i then
21:                 if mp(P) + f.size < mt then
22:                     Assign f to P
23:                     mt ← Max(LU_(u,v), ∀(u, v) ∈ E)
24:             if Scale_In_i then
25:                 if mp(P) + f.size < MLU(P) then
26:                     Assign f to P
27: end procedure
```

Algorithm 1 shows the detail of the proposed heuristic. It consists of two procedures: CONGEST_REM and LOAD_BALANCING that take as inputs the Network topology $G(V_i, E_i)$ and the current and predicted Traffic Matrix (TM_i and $\widehat{TM_i}$, respectively).

The CONGEST_REM procedure is called to delete a congestion and it works as follows. Upon measuring continuously the current and predicted Traffic Matrix, if a congestion occurs in the PN $Slice_i$, in which the current or predicted link delay is greater than a certain threshold or the link is overloaded, our algorithm finds the flow rule R corresponding to the flow with minimum size in the congested path CP. Then, it sorts all other paths (denoted by SP) by the delay matching the flow rule R (lines 5-8). In this case, the flow rule R must be rerouted to a path in SP (line 9-13). If no path in SP can accommodate the corresponding flow size, the flow is discarded, and our algorithm goes to the next flow in the CP (line 14). On the other hand, the LOAD_BALANCING procedure triggered periodically and when scaling in/out a PN $Slice_i$, and it works as follows. For each flow f in F sorted in descending order, it evaluates possible paths $list\_P$

that can route the flow f. Among these considered paths, we distinguish two situations. The first one, when scaling out the PN $Slice_i$, in which we distribute the flows over the new paths, to this end we minimize the overall LU (line 23). The second one happens when scaling in the PN $Slice_i$, in which we distribute the flows over the original paths while avoiding overloading each link. Note that, $mp(P)$ denotes the maximum path utilization, it is determined as follows: $mp(P) = Max(LU_{(u,v)},\ \forall(u,v) \in P)$. Whereas $MLU(P)$ denotes the maximum threshold determining the maximum tolerable load of all links composing P. It is determined as follows: $MLU(P) = Max(MLU_{(u,v)},\ \forall(u,v) \in P)$. Note that, the action of scaling in/out the PN slice consists in adding or removing a set of network paths.

IV. Performance Evaluation

In this section, we evaluate the efficiency of our proposed approach. We start by presenting our environmental setup. Then, we present the experimental results.

A. Experimental setup

Fig. 3 illustrates the logical building blocks of our framework. In the radio domain, the RCC and RRU nodes are interconnected via a fronthaul interface (FH), and deployed on separated servers by making use of OAI software [5] and Docker container technology [20]. The server hosting the RRU container is connected to an Ettus USRP B210 card [21] via an USB 3.0 interface. The C-RAN infrastructure provides 4G connectivity to two Nokia IoT Gateways, one 4G smartphone and Bandluxe C501 LTE modem. The Nokia IoT Gateways in turn provide Bluetooth connectivity to Bosch XDK110 sensor. Note that, by making use of XDK-Workbench [22] tool, we developed a software for XDK senor that periodically sends Humidity, Pressure, through the IoT Gateway by the means of MQTT protocol [23], to a remote IoT framework. The latter is deployed based on Eclipse Kapua [24] on a remote machine as a cluster of Docker containers. The Packet Network domain consists of a set of OpenvSwitch devices, created by deploying, on a physical machine, a specific OpenvSwitch version that we extended to support GTP traffic. This PN is connected to the Core Network EPC, where the EPC nodes are deployed as a cluster on two servers by the means of Docker technology.

We deployed the FlexRAN controller [8] on top of RCC node, as well as the OpenFlow controller ONOS [7] to control the Packet Network. On top of these two Controllers the Orchestrator is deployed by using Docker. The Radio Slicing, the Packet Network Slicing, the Slicing Continuity, and the Traffic Prediction modules are developed based on Python, which interact with SD-RAN and SDN controllers through the FlexRAN and ONOS northbound APIs, respectively. Furthermore, the statistics needed to enable automation of slicing continuity such as GTP TEID, are extracted from the MongoDB database [25]. It is worth noting that, we implement the Network Measurement module (Latency Measurement and Statistics) as well as the Proactive Forwarding module as cooperating modules for the Java based OpenFlow controller ONOS, based on our previous framework [18] [19].

Noting that, the main prediction method used in the proposed solution is LSTM. In which, we built and trained the LSTM model by using Keras Library [26], where the number of dense layers is 2 and the number of nodes is 42. Furthermore, we saved the trained LSTM model, in such a way that only one step is needed to get the predicted traffic.

As for ONOS and FlexRAN controllers, the radio nodes (i.e., RRU, RCC) and the Core Network nodes (i.e., S-GW, P-GW, MME, Home Subscriber Server (HSS)) are Docker based deployment. They are placed and managed by the containers clustering and scheduling tool Docker Swarm [20].

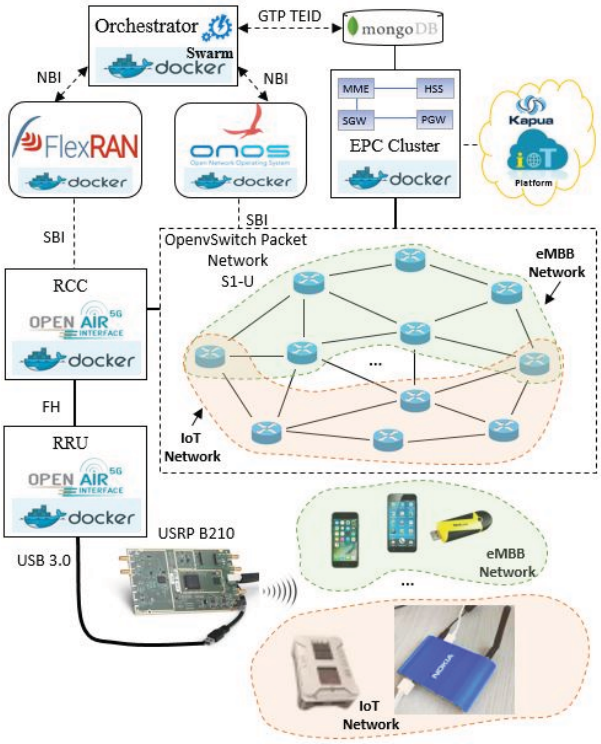

Fig. 3. End-to-End Slicing Prototype

B. Experimental results

In order to evaluate the performance of the proactive and dynamic radio slicing, we compare it to the radio slicing without prediction. Specifically, as shown in Fig. 3, we created two radio slices (i.e., IoT and eMBB). Then, we generated two Iperf [27] traffics from two hosts. The first one is connected to the Bandluxe C501 LTE modem and the second one is connected to the IoT Gateway. Initially, we allocate 40% of the total RBs to the IoT slice and 60% of the total RBs to the eMBB slice. Thereafter, we generated an Iperf traffic from the IoT slice characterized by periodical sessions which can exceed the slice capacity. On the other hand, the traffic generated in the eMBB slice is regular which takes regularly 70% of the RBs allocated to the eMBB slice. Fig. 4-a, shows the amount of RB requested by the IoT slice when all the

bandwidth is allocated to this slice, and the maximum RB that this slice is allowed to use (i.e., 10 RB), during a period of 60 time intervals of 5 seconds. From Fig. 4-b, it can be observed that the throughput is increased while predicting the future behavior of the traffic, and allocating additional RB not used from the other slice. However, the throughput remains stable at certain threshold, by using fixed slices capacities, since there is no dynamicity and the absence of future vision of the traffic evolution, which causes service degradation.

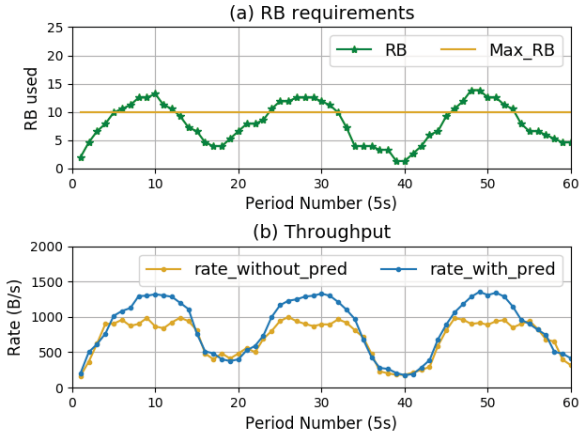

Fig. 4. IoT traffic profile and fixed and dynamic proactive radio slicing

We plot in Fig. 5 the Mean Squared Error (MSE) [14] of the prediction method (LR) used to predict the RB allocation. We can see that LR achieves a good estimation accuracy since the associated MSE values remain very low.

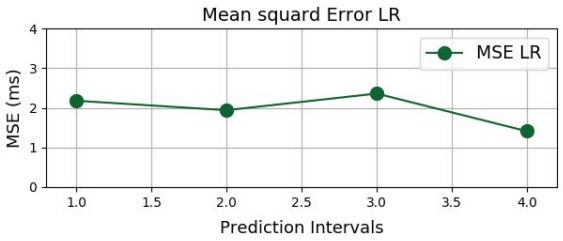

Fig. 5. MSE of LR used to predict RB requirements

In order to evaluate the proposed End-to-End network slicing solution, we propose to run it under two main strategies: i) Static, in which the amounts of resource blocks and the capacities of PN slices are fixed and no scaling in/out, and ii) Zero_touch, where both Radio and PN slices capacities are changed dynamically and proactively. In this latter strategy, we distinguish between the following schemes:

- CLR, which corresponds to the proposed heuristic while using LSTM for Traffic Prediction in PN slices and LR in the corresponding Radio slices.
- CLRv1, which corresponds to CLR while using LSTM for Traffic Prediction in PN slices and no prediction in the corresponding Radio slices.
- CLRv2, which corresponds to CLR while using LR for Traffic Prediction in both PN and Radio slices.
- Hop-count (HC), which is the default routing metric used by ONOS, while using LR for Radio traffic prediction.

Note that, we have added an external emulated IoT and eMBB traffics to inject more traffic to the PN part.

First, we compare in Fig. 6 the prediction accuracy of the different methods used for traffic prediction inside a PN slice. From that figure, we can see that LSTM achieves a good prediction accuracy and performs better than LR, due to its capacity to learn long-term dependencies.

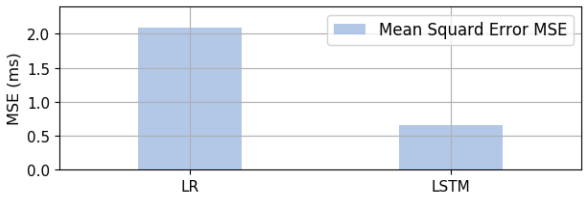

Fig. 6. MSE of CLR using LR and LSTM prediction methods

Fig. 7 plots the end-to-end delay, packet loss and throughput for all schemes (CLR, CLRv1, CLRv2, HC and Static). We can see that the Static approach causes obviously considerable packet loss and decreases the throughput, since the slices capacities either in Radio or PN are fixed in advance. In addition, there is no scaling in/out. On the other hand, the Zero_touch approaches outperform the Static one, since the capacities associated to each slice are dynamic, based on traffic prediction and PN scaling. Specifically, HC outperforms the Static approach since the Radio slicing is dynamic. However, it still causes considerable packet loss and decreases the throughput and latency, since all the flows are forwarded to shortest paths which are not always the optimal ones. On the other hand, the two approaches CLRv1 and CLRv2 improve the network performances in terms of decreasing the packet loss and increasing the throughput. However, a certain level of packet loss is still observed in these two schemes due to the incapacity of LR to predict the future evolution of network traffic in CLRv2 and the absence of traffic prediction in CLRv1. Finally, we can see that CLR outperforms all other schemes, where both the delay and the packet loss are decreased, and the throughput is increased. This is related to the high accuracy of LSTM in predicting network congestion, compared to LR prediction method.

Finally, Fig. 8 plots the maximum link utilization of our scheme CLR under two prediction methods (LSTM and LR) compared to HC. We can see that CLR outperforms HC, since some links and devices are overloaded and experiencing congestion while other are under-utilization in the HC approach. However, when using CLR, the traffic load is balanced, which minimizes the link utilization. Moreover, LSTM shows better performances compared to LR.

V. Conclusion

In this paper, we have addressed the design and implementation of a novel architecture based on SDN and Machine Learning techniques for enabling creation, modification

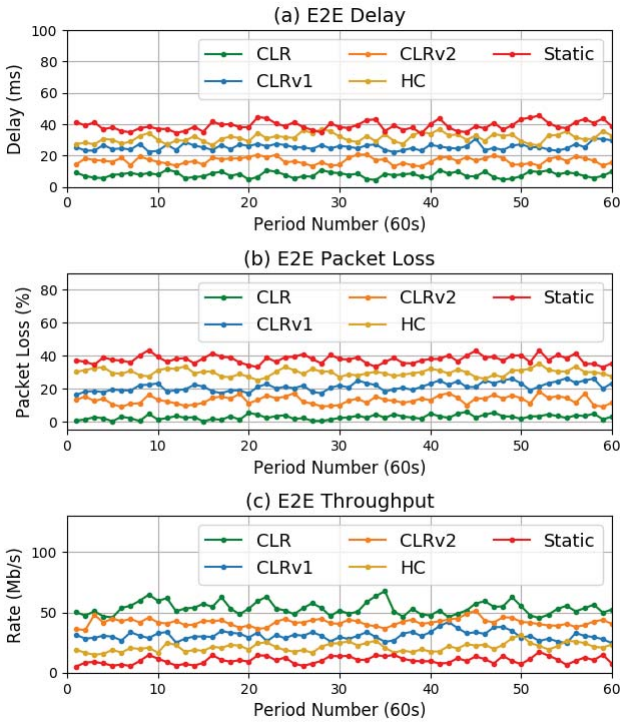

Fig. 7. Delay, Packet Loss and Throughput under CLR based rule placement algorithm with and without prediction

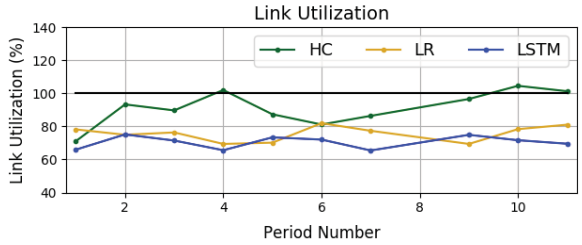

Fig. 8. Maximum link utilization under CLR based rule placement algorithm with and without prediction

and continuity of radio and transport network slices, while considering their performances and QoS requirements. To do so, our solution relies on OAI tool FlexRAN, ONOS and an extended version of OpenvSwitch that we patched to handle GTP packets. Second, the proposed solution enables efficient sharing of RAN and Packet Network resources by proactively adjusting radio slices capacities and adapt them to the corresponding Packet Network slices, then a balancing of traffic load and congestion prevention have been proposed for improving routing and avoiding congestion within each Packet Network slice. To this end, we have formulated the problem as a LP that aims to minimize the total network delay and proposed a Congestion prediction and Load balancing Rule (CLR) placement algorithm to solve it with low time complexity and high estimation accuracy. Experimental results show the feasibility of the proposed solution of handling end-to-end slicing continuity in real-time. Specifically, results show the outperformance of using Machine Learning techniques in real-time on other schemes in terms of increasing throughput and decreasing packet loss and latency. Future work will focus on making use the slicing mechanism in the Core Network.

ACKNOWLEDGEMENT

This work was partially supported by the FUI SCORPION project (Grant no. 17/00464).

REFERENCES

[1] ONF TR-521 Specification, "Sdn architecture," Issue 1.1, Feb. 2016.
[2] A. Mayoral, R. Munoz, R. Vilalta, R. Casellas, R. Martinez, and V. Lopez, "Need for a transport api in 5g for global orchestration of cloud and networks through a virtualized infrastructure manager," *IEEE J. Opt. Commun. Netw.*, vol. 9, no. 1, pp. A55–A62, Jan 2017.
[3] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar, "Dynamic network slicing for 5g iot and embb services: A new design with prototype and implementation results," in *2018 3rd Cloudification of the Internet of Things (CIoT)*, July 2018, pp. 1–7.
[4] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, "Network slices toward 5g communications: Slicing the lte network," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 146–154, Aug 2017.
[5] "Openairinterface simulator/emulator," Tutorial, Jul 2015. [Online]. Available: http://www.openairinterface.org/
[6] "The evolved packet core," Tech. Rep. [Online]. Available: https://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core
[7] Onos. [Online]. Available: http://github.com/opennetworkinglab/onos
[8] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "Flexran: A flexible and programmable platform for software-defined radio access networks," 11 2016, pp. 427–441.
[9] M. R. Raza, M. Fiorani, A. Rostami, P. Ohlen, L. Wosinska, and P. Monti, "Dynamic slicing approach for multi-tenant 5g transport networks," *IEEE J. Opt. Commun. Netw.*, vol. 10, no. 1, pp. A77–A90, Jan 2018.
[10] M. Leconte, G. Paschos, P. Mertikopoulos, and U. Kozat, "A resource allocation framework for network slicing," in *2018 IEEE INFOCOM*, 04 2018, pp. 2177–2185.
[11] A. Ksentini, M. Bagaa, and T. Taleb, "On using sdn in 5g: The controller placement problem," in *2016 IEEE GLOBECOM*, Dec 2016, pp. 1–6.
[12] N. Salhab, R. Rahim, R. Langar, and R. Boutaba, "Machine learning based resource orchestration for 5g network slices," in *2019 IEEE GLOBECOM*, December 2019.
[13] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. Caicedo Rendon, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *JISIS*, vol. 9, 05 2018.
[14] A. Azzouni and G. Pujolle, "A long short-term memory recurrent neural network framework for network traffic matrix prediction," 05 2017.
[15] OpenvSwitch. [Online]. Available: https://www.openvswitch.org/
[16] "How to connect cots ue toai enb via ngfi rru," Tutorial. [Online]. Available: http://gitlab.eurecom.fr/oai/openairinterface5G/wikis/howto-connect-cots-ue-to-oai-enb-via-ngfi-rru
[17] (2015, March) Openflow networking foundation. [Online]. Available: https://www.opennetworking.org
[18] E. H. Bouzidi, D. Luong, A. Outtagarts, A. Hebbar, and R. Langar, "Online-based learning for predictive network latency in software-defined networks," in *2018 IEEE GLOBECOM*, Dec 2018, pp. 1–6.
[19] E. H. Bouzidi, A. Outtagarts, and R. Langar, "Deep reinforcement learning application for network latency management in software defined networks," in *2019 IEEE GLOBECOM*, Dec 2019, pp. 1–6.
[20] Docker. [Online]. Available: https://www.docker.com/
[21] "Usrp b200/b210 specification sheet," Tutorial. [Online]. Available: https://www.ettus.com/product/details/UB200-KIT
[22] "Xdk bosch," Tutorial. [Online]. Available: https://xdk.bosch-connectivity.com/
[23] "Mqtt protocol," Tech. Rep. [Online]. Available: http://mqtt.org/
[24] "Eclipse kapua," Tutorial. [Online]. Available: https://www.eclipse.org/kapua/
[25] "Mongodb," Tutorial. [Online]. Available: https://www.mongodb.com/
[26] keras. [Online]. Available: https://keras.io/
[27] Iperf. [Online]. Available: https://iperf.fr/