SDN managed Network Slicing in Mobile Backhaul

Alexander Hefele
Technische Universität Dresden
Dresden, Germany
alexander.hefele@tu-dresden.de

Jose Costa-Requena
Aalto University
Department of Communication and Networking
Espoo, Finland
jose.costa@aalto.fi

***Abstract*—This paper provides a solution for ultra reliable low latency communication (URLLC) module for the 5G mobile backhaul. This proposal considers network slicing based on Software Defined Networking (SDN) to deploy URLLC communications. This paper presents the design of the modules and algorithms that implement the network slicing functionality. It will be illustrated by applying it to a simulated Mobile Backhaul environment.**

***Index Terms*—SDN, 5G Mobile Backhaul, Network Slicing, Traffic Monitoring, Latency Measurement, Traffic Engineering, URLLC**

I. Introduction

The fast growth of the data consumption in mobile networks due to the wide spread of smartphones and tablets is raising concerns about bandwidth and resources consumption.

Therefore, mobile operators require a solution that will allow efficiently managing the network capacity and optimizing the usage of available resources and reduce the Operative Expenses (OPEX). In order to address this demand new technologies such as Network Slicing and SDN based network management are proposed.

Network Function Virtualization (NFV) allows the mobility of network nodes which are virtualized and can be deployed in any location of the network on top of a virtualization platform such as KVM, VMware, Openstack. NFV together with SDN can be used to deploy orchestrator for optionally managing the Mobile Backhaul could be a solution.

The following sections will outline a suggestion for a monitoring and packet forwarding algorithm within the Mobile Backhaul to support Network Slicing utilizing SDN. After introducing the current status of the research about SDN controllers and traffic monitoring algorithms in section II. Section III introduces to the concept of Network Slicing embedded in the Mobile Backhaul, whereas Sections IV and V describe more detailed the URLLC module's underlying monitoring and packet forwarding algorithms. In addition the functionality will be shown and validated in a Mininet [1] simulation environment outlined in section VI. Drawbacks caused by the Mobile Backhaul architecture and the OpenFlow protocol will be evaluated in section VII, while giving suggestions for a possible change in the OpenFlow protocol functionalities. The paper will be concluded by summarizing the key findings and outlining possible future improvements in section VIII.

II. State of the Art

A network supporting SDN enabled Network Slicing requires OpenFlow capable switches. Their behavior is determined by a controlling entity using the OpenFlow protocol. Thus the performance of the entire network is highly depending on the controller. The recent developments in the field of SDN controllers will be outlined in the following paragraphs and a reasoning for the controller choice for this project is given. Additionally the recent developments in SDN based network traffic monitoring is given.

A. SDN Controller

Several different controllers have been proposed over the past few years. When considering different papers, that are evaluating those frameworks [2]–[5], it shows that none of the initially developed controller frameworks are considered anymore in evaluations and comparisons. NOX, NOX-MT, Beacon, and Maestro which, for instance, where considered for a performance comparison by Tootoonchian et al. in [2] are by now replaced by OpenDayLight [6], ONOS [7], RYU [8], Floodlight [9] and POX [10].

In 2014 Khondoker et al. [3] compared various of those frameworks by applying criteria such as interfaces, platform support, REST API, productivity, documentation, modularity, OpenFlow support etc. As a result they considered RYU the most suitable controller frameworks for common implementations, mainly due to its good OpenFlow support. OpenDayLight however, which back then was founded 9 months prior to the publication of the paper, performed quite well in the comparison.

In [4] Salman et al. once again compared the performance of common controllers by focusing on the impact of thread count and switch number. Along with old frameworks such as NOX and Maestro they also considered RYU, OpenDayLight and the even more recently released ONOS which is principally designed for carrier networks. In their results they outline OpenDayLight one of the prosperous full-featured controller supporting a wide range of applications such as data centers. RYU on the other hand was considered to be very suitable for research applications, but due to its lack of modularity compared to ONOS and OpenDayLight and the inability of running cross-platforms it is considered to have a limited range of real market applications.

Mamushiane et al. [5] understand their contribution as an extension of the work by Salmon et al., described above. They are applying similar metrics, more focusing on the effect of network load. Furthermore they are using the most recent releases with up-to-data features. In terms of scalability the results show that ONOS and OpenDayLight perform very well under heavy load and high numbers of switches. The results show that OpenDayLight from a feature based and ONOS from a overall performance based evaluation perspective have the best results. Due to the low latency performance of RYU, it is suggested for delay sensitive applications.

Regarding the controller used for this project RYU was chosen. Taking the high delay sensitivity into consideration, it is very suitable for the given project. Disadvantages compared to more recently developed controllers such as the scalability for bigger, more distributed networks, can be disregarded due to the nature of the project as a proof of concept.

B. *Traffic Monitoring*

With the introduction of OpenFlow 1.3 [11] SDN controllers include features to enable traffic monitoring for individual switches. The controller can request statistics and applying meter bands. In particular the following sets can be obtained:

- Individual flow statistics
- Aggregate flow statistics
- Flow table statistic
- Port statistics
- Queue statistics for a port
- Group counter statistics
- Meter statistics

Prior work regarding monitoring covers general approaches as well as concrete frameworks implemented in different controllers, while considering scalability and implementational constraints.

Tootoonchian et al. did a quite early study [12] on traffic monitoring utilizing queried statistics. As a result they present OpenTM which is a real-time measurement tool without packet sampling. Within their study they are focusing on monitoring traffic on a predefined route. Their main focus is allocated in the right polling pattern for the switches. After evaluating different suggestions, they conclude in evenly distributed statistic queries being the best performing solution.

After the introduction of meter bands in Openflow 1.3 Hamad et al. [13] considered those within the application of traffic measurement to enable traffic engineering. In general they took a similar approach as the previously presented work by Tootoonchian et al. But instead of only focusing on a specific route they are estimating the available bandwidth of each link in the network by querying meter statistics of individual flows.

More recent work such as the implementation of *PANORAMA* by Gangwal et al. [14] focuses on the implementational challenges of a link specific network monitoring. *PANORAMA*, which is a real-time, birds-eye view monitoring tool designed for POX, obtains the link data transfer rate, link loss and link delay by applying algorithms suggested by Adrichem et al. in [15].

III. Network Slicing in 5G Mobile Backhaul

The concept of Network Slicing incorporates the isolation of network functionalities to deliver different features to different network slices. Users can be therefore assigned to their individually required network slice such as for instance high bandwidth or low latency. Using static network slices by modifying VLAN identifiers (VID) in the medium access layer, MPLS in the network layer or DSCP/ToS in the transport layer is already used in the Mobile Backhaul. Introducing Network Slicing based on SDN to the Mobile Backhaul enables operators to not depend on static configurations applied to one of the fields above anymore. Instead User Equipment (UE) can be assigned to their dedicated network slice dynamically.

URLLC however requires immediate modifications to its network slice to provide the necessary features to the user such as low latency and high reliability. Especially the later requires a fast adaptation of the network slice in case of congestion, which cannot be ensured by static configurations. Thus SDN is inalienable for URLLC communication within the Mobile Backhaul.

The evolved NodeB (eNB), the base station (BS) in a 5G system, uses TEIDs as unique identifiers included in the GTP encapsuling. To be able to distinguish uplink from downlink traffic, the EPC assigns a different TEID in the GTP encapsuling. Thus every UE is identified by two different TEIDs and the information whether this user should be assigned to the URLLC group is given by the EPC. An overview of the structure is given in figure 1.

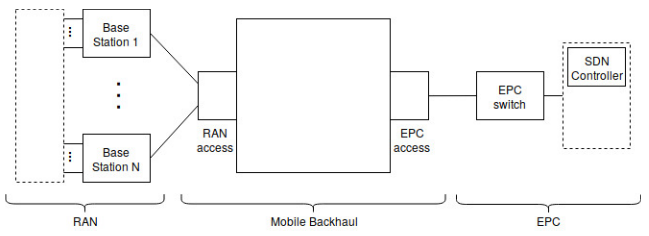

Fig. 1. Structural diagram of the Mobile Backhaul including the EPC and RAN

As already stated above, dynamic Network Slicing requires a SDN controller. In this suggestion the controller's modules are hosted in the EPC, communicating with the Mobile Backhaul's Switches via the OpenFlow Protocol. Figure 2 shows the suggested modules of the URLLC application including its depending application *Switches*.

Switches itself is located within the RYU framework at /ryu/ryu/topology/switches.py and is mainly responsible for topology discovery in the respected network utilizing LLDP packets. Additionally those packets are used to obtain the individual link latency which is further elaborated in section IV-C.

The URLLC application is hosting the modules for traffic monitoring and packet forwarding. Those are executed in a threaded environment providing additional measurement and monitoring functionalities, described more detailed in section IV, as well as URLLC and non-URLLC traffic forwarding in the Mobile Backhaul network, described in section V.

The previously obtained measurement data is saved in a MongoDB database to be available for other modules and can be used for path determination. Additionally the data can be accessed outside the application for visualization purposes.

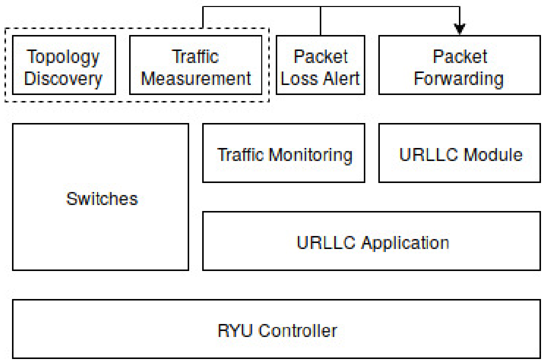

Fig. 2. Architectural overview for controller application

IV. Traffic Monitoring

The suggested traffic measurement and monitoring implementation includes the following unidirectional measurements for every individual link in the Backhaul network:

- residual bandwidth
- link packet loss
- link delay

Those metrics and other calculated values based on the measurements, such as delay variation, are used to determine the optimal path for URLLC. The algorithms for the monitoring follow the suggestions provided by Gangwal et al. [14] and Adrichem et al. in [15]. The monitoring can be classified among passive (residual bandwidth, link packet loss) and active measurements (link delay). Detailed information on the implementation of the individual measurement scheme will be outlined within the next paragraphs.

A. Residual Bandwidth

The residual bandwidth ($B_{residual}$) is calculated with the following approach:

$$B_{residual} = B_{available} - B_{used} \tag{1}$$

The available bandwidth ($B_{available}$) is hardware specific and needs to be provided to the controller within the environment of a configuration file. The used bandwidth (B_{used}) on the other hand is the measured metric and is obtained by iteratively querying the switches port statistics and applying equation 2 for every unidirectional link's outgoing port.

$$B_{used} = \frac{D^{tx}_{cur} - D^{tx}_{prev}}{t_{cur} - t_{prev}} \tag{2}$$

Where D^{tx} represents the number of the port's transmitted bytes and t denotes the time of port observation.

B. Link Packet Loss

Similar to the residual bandwidth estimation the link packet loss (LPL) measurement applies a passive measurement scheme by utilizing port statistics. In order to acquire the LPL between switch 1 ($s1$) and switch 2 ($s2$) in percent, the following equation is applied.

$$LPL^{s1,s2}_{\%} = \left[1 - \frac{n\_packet^{s2}_{rx}}{n\_packet^{s1}_{tx}}\right] * 100 \tag{3}$$

Where n_packet denotes to the port statistic of received (rx) or transmitted (tx) packets at the designated switch.

C. Link Delay

Compared to the previous two measurement schemes link delay (LD) cannot be acquired by passively requesting statistics from individual switches. Active measurement by inserting probes into the network has to be much rather applied and is based on two individual measurements. First the round trip time (RTT) of the probe from the controller to the first switch of the respected link to its second switch and back to the controller. In order to obtain the desired link delay, the individual controller-switch delay has to be determined as well and subtracted from the RTT. Thus individual LD is obtained according to the following equation:

$$LD = RTT - \left(\frac{SCD_1 + SCD_2}{2}\right) \tag{4}$$

Where SCD denotes to the switch's switch-controller delay.

From an implementational perspective a probe can by any Ethernet packet created by the controller and handled accordingly by the switches. When using RYU as a controller it is helpful to use the built-in packet libraries such as the *EchoRequest* for the switch-controller delay, since the packet is automatically sent back by the desired switch. The delay itself is obtained by saving a timestamp when sending the packet and comparing it with the timestamp of the *EchoReply*. For the RTT measurements LLDP packets, that are used to discover the network topology, can be utilized. Per default LLDP packets don't carry timing information, therefore the *Switches* application, hosting the topology discovery, was manipulated to store timestamp data when sending a LLDP packet. After receiving the packet in the controller the RTT is calculated and stored in the application's port information field. Thus the URLLC application can request this information after every topology discovery and apply it to the locally maintained port statistics.

D. *Measurement Improvements*

Due to statistical variations in the sampled measurement values a sliding window was applied to ensure lower variance while at the same time ensuring a quick response time to sudden changes in specific links. For the passive and the individual active measurements a window size of 4 was applied. Whereas the LD which is a combination of RTT and SCD was once again averaged with a window size of 6.

E. *Visualization*

To visualize measurement data the Python framework Dash [16] developed by Plotly is used. It provides data visualization functionalities that are displayed by a web browser. With the present implementation it can be distinguished between Backhaul and non-Backhaul links to be plotted. While at the same time individual links can be still selected and deselected. The visualized data includes:

- used bandwidth
- link latency
- link packet loss
- switch-controller delay

Figure 3 shows a sample plot of the link latency within the Backhaul network measured for the last 40 seconds. The underlying network is simulated in Mininet and has the same topology as the Backhaul network used for the experiments described in section VI.

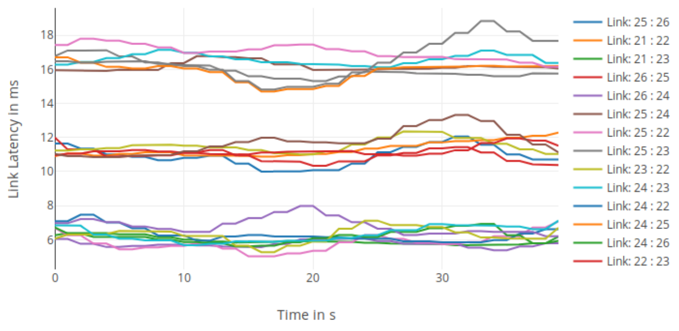

Fig. 3. Latency in Backhaul network without external user traffic

V. URLLC Packet Forwarding

Since the user identification is done with both TEIDs a UE data base has to be maintained in the controller based on those. Therefore they need to be extracted at the dedicated entities and the VLAN representing the URLLC network slice needs to be maintained and applied to those UE, which will be explained in the following paragraphs.

A. *TEID Extraction and Priority Assignment*

The TEID extraction along with the VID assignment is done in the individual BS and the *EPC switch* connected to the EPC itself. Before receiving a packet from the UE, there are not any flows, except the LLDP and default controller forwarding, installed in the switches.

Regarding the packet forwarding two different protocols have to be processed by every entity of the network:

- ARP for host discovery
- GTP used for URLLC and non-URLLC

Whereas a GTP packet is assumed to be a UDP packet with the destination port 2152 and the specific GTP header including the TEID.

When receiving in packet from an unknown UE, this device has to be registered and is therefore forwarded to the controller. The obtained registration information is:

- UE MAC address
- TEID assigned by BS
- TEID assigned by EPC
- VID assigned to UE

After registering the UE's MAC address and the TEID assigned by the BS, necessary flows are installed in the BS which ensure packet forwarding to the Backhaul network. For the ARP traffic basic forwarding to the *RAN access*, the EPC and the UE based on the MAC address is installed in the BS and the *EPC switch*. In the initial phase of the registration the information whether the UE is in group of URLLC users cannot be obtained since the EPC is assigning those priorities. Thus every UE's initial packet is forwarded as if the UE would be in the group of non-URLLC users.

In order to finish the UE's registration in the controller, the first response packet from the EPC needs to be fetched in the *EPC switch*. This is achieved by implementing a flow resulting in controller and *EPC access* forwarding. Thus the UE's communication is still allocated in the default group, but the TEID information sent by EPC can be extracted in the controller and the registration process is done.

Along with previously exchanged information about URLLC specific TEIDs the controller now changes the VID assignment in the BS and the *EPC switch* and therefore assigns the UE to the URLLC group. Regardless the priority of the UE the previously installed flow in the *EPC switch* is modified towards *EPC access* forwarding without considering the controller anymore.

B. *URLLC Path Maintainance*

Obtaining the right path in the Backhaul network after reaching a predefined packet loss rate is done by utilizing the Dijkstra algorithm. The individual link latency is used by the Dijkstra algorithm after excluding the congested link from the graph. Figure 4 shows the possible result for that calculation. The dashed line denotes the congested link.

In order to achieve a higher reliability two disjoint paths, installed in the Backhaul network, are necessary. This secondary, disjoint URLLC path is determined by another Dijkstra path calculation excluding the congested path as well as the

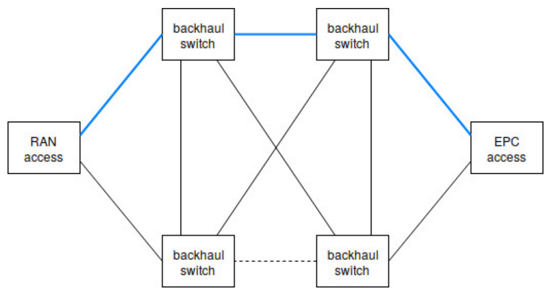

Fig. 4. Primary path obtained by Dijkstra algorithm

previously obtained primary path. A possible result of that calculation is given in figure 5. Due to the reduced amount of edges in the graph the path latency of the secondary path cannot be lower than the primary path's latency.

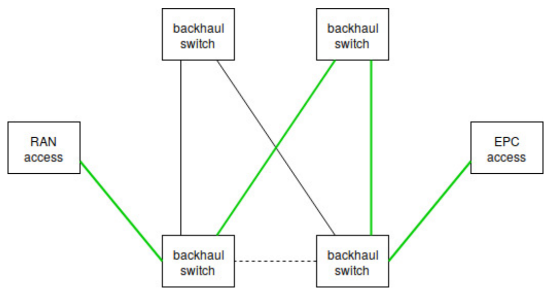

Fig. 5. Secondary path obtained by Dijkstra algorithm

After detecting a rise in packet loss for the currently selected primary URLLC path, the controller changes the URLLC group's VID. This directs traffic via the secondary path to provide enough time for the controller to fetch the latest measurement data and calculate a new path based on those latency measurements. After recalculating the new URLLC path, the controller deploys it in the Backhaul network and removes the previously used URLLC trajectory. Thus the URLLC group is assigned the new primary path with the lowest latency in the network. After reducing the graph to extract the secondary URLLC path, it is calculated and installed in the switches.

VI. System Validation

Objective of this paper is to evaluate the opportunity of enhancing the Mobile Backhaul's performance by introducing SDN enabled Network Slicing. In the previous sections III, IV and V individual algorithms were explained and the topology illustrated. Within this section the performance of the Mobile Backhaul will be elaborated by applying it to a experimental setup which emulates the BS's and EPC's behavior in a Mininet environment.

The legend in the measurement plots provided in this section uses its own notation. A link is described by two switches separated by a column. A switch is labled by two digits, whereas the first digit denotes to the switch's location in the network and the second labels the switch according to the indexing in figure 5. The location index 2 denotes to the switch being allocated in the Mobile Backhaul.

A. Topology Description

For the experiment a virtual machine provided by Mininet [1] runs Mininet 2.2.2 and the RYU controller version 4.32 @cef24da9. Referring to figure 1 in section V the Backhaul is emulated by the network of switches shown in figure 6.

The topology is built using the following Modules. Hosts such as the UEs and the EPC utilize the module *Hosts* found with the Mininet Python framework at *mininet.node*. Every host is configured with a unique MAC and IP address within the emulated network. The *mininet.node* location applies as well to the controller being a *RemoteController*. The links in between switches are emulated by the module *TCLink* found at *mininet.link*. The advantage of using *TCLink* over the basic *Link* is that specifications for available bandwidth, latency, packet loss, etc. can be made. For the present topology changes were made exclusively to the latency since that is the metric used for determining the new path. All other values are initialized within the *TCLink* module according to their defaults. Links that do not belong to the Mobile Backhaul are configured without any latency.

The marked red path in this figure shows the default path for non-URLLC with its individual link delays.

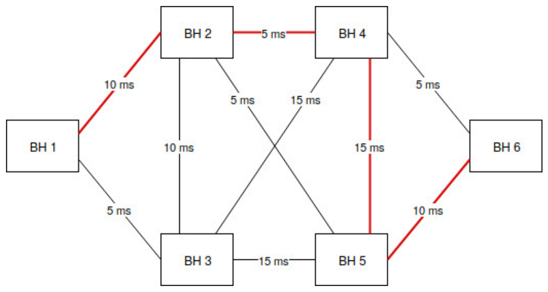

Fig. 6. Mobile Backhaul used for experiment with default path

Testing the functionality of the Mobile Backhaul requires UEs as well as emulated behavior of the eNB and the EPC, illustrated in figure 7. A UE as well as the EPC are implemented as hosts that are providing sockets for sending and receiving data. A host is creating a GTP packet with a host specific TEID and random 1 kB payload that is forwarded to the emulated BS in 100 ms intervals. The EPC listens to incoming packets, changes the TEID and sends them back to the UE it received the packet from.

B. Test Case Description

In the present test environment the default path, given in figure 6, is initialized to be BH$[1-2-4-5-6]$. Using the latest measurement values, the primary URLLC path is determined to be BH$[1-2-4-6]$ and the secondary path is BH$[1-3-2-5-6]$. Both paths are shown in figure 8,

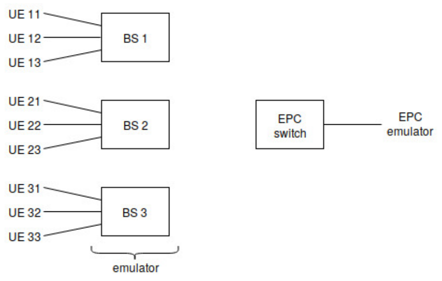

Fig. 7. RAN and EPC emulation used for experiment

where the blue and green trajectory denote the primary and secondary URLLC path.

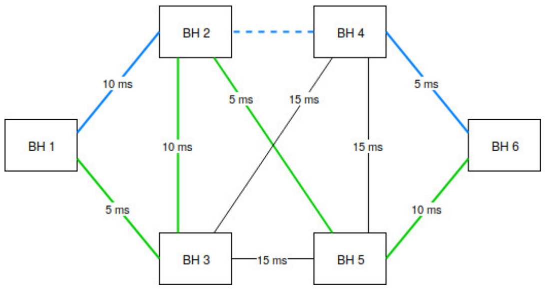

Fig. 8. Primary (blue) and secondary (green) URLLC path

Thus when congesting the link BH$[2-4]$, illustrated by the dashed line in figure 8, the default path as well as the primary URLLC path are affected and the URLLC group should receive a new path for the Mobile Backhaul.

For congesting the network the built-in Linux tool Iperf [17] is used. Therefore a dedicated Iperf-host is connected to every Backhaul switch (BH 2, 3, 4, 5), which installs individual forwarding flows for Iperf generated traffic during the URLLC Module initialization. Every Iperf host provides a running Iperf server which acts as a traffic drain, started by the bash command:

iperf -s -u &

Where the options *-s* and *-u* specify a server expecting UDP packets.

Congesting the link BH$[2-4]$ is therefore done by sending Iperf traffic from BH 2 to BH 4. The Iperf client, acting as a source, is initiated by the bash command:

iperf -c 10.0.24.1 -u -b 100m -t 40

Where the option *-c* and *-u* specifies the client, sending UDP packets, with the IP address of the dedicated server. *-b* and *-t* denotes the sent bandwidth of 100 Mbit/s and 40 seconds duration of the traffic injection.

The measured packet loss rate and bandwidth in the Backhaul links are given in the figures 9 and 10. Additionally to the individual link status measurements, the RTT for URLLC and non-URLLC UEs will be evaluated. The advantages of the proposal from the users' perspective is given in figure 13.

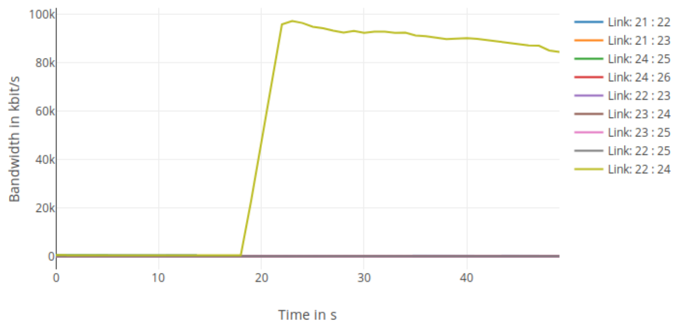

Fig. 9. Bandwidth measurement after congesting with Iperf

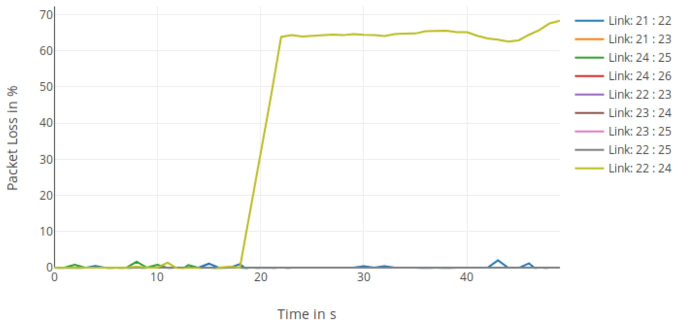

Fig. 10. Packet loss rate measurement after congesting with Iperf

C. Results Description and Evaluation

Figure 9 clearly shows the start of the Iperf congestion with an increased measured bandwidth in the BH$[2-4]$ link. Due to the limited bandwidth of the virtualized links a sudden increase of the link's packet loss rate, shown in figure 10, is detected.

After detecting a high packet loss on the link that belongs to the URLLC path and removing that link from the graph, a new route has to be calculated and assigned to the URLLC group's VLAN. According to recently measured link latency BH$[1-2-4-6]$ and BH$[1-2-5-6]$ were calculated for the new primary and secondary URLLC paths. The newly installed forwarding rules are given in figure 11.

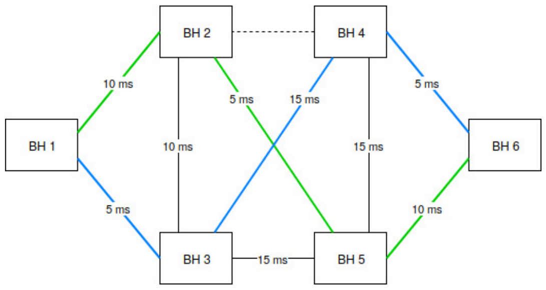

Fig. 11. New primary (blue) and secondary (green) URLLC path

For validating those new forwarding paths being applied to the URLLC group, after detecting high packet loss, bandwidth measurements are used. Therefore figure 12 shows the measurements already given in figure 9 but with a differently scaled ordinate axis. The link BH$[1-2]$, for instance, previously carried the non-URLLC and the URLLC traffic. After redetermining the URLLC path BH$[1-3]$ is used to forward URLLC traffic. This results into a reduced bandwidth, visualized in figure 12, in BH$[1-2]$, since it is now only carrying non-URLLC traffic. BH$[1-3]$ on the other hand will be used for URLLC traffic and therefore has an increased measured bandwidth.

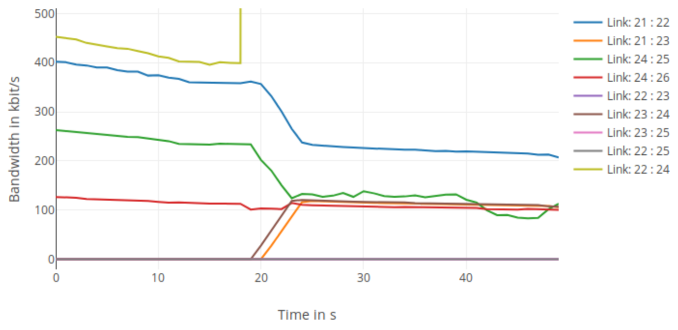

Fig. 12. Bandwidth measurements of non-congested links

The functionality of the mechanism outlined in the paragraph above is also shown in the individual user's RTT measurement. In figure 13 the effects for a dedicated URLLC and non-URLLC UE are shown, while congesting the same link as in the previous measurement. It clearly proofs the good performance of path determination algorithm since the URLLC UE is not affected by the congestion, whereas a very high RTT is measured for the non-URLLC UE during the congestion by Iperf.

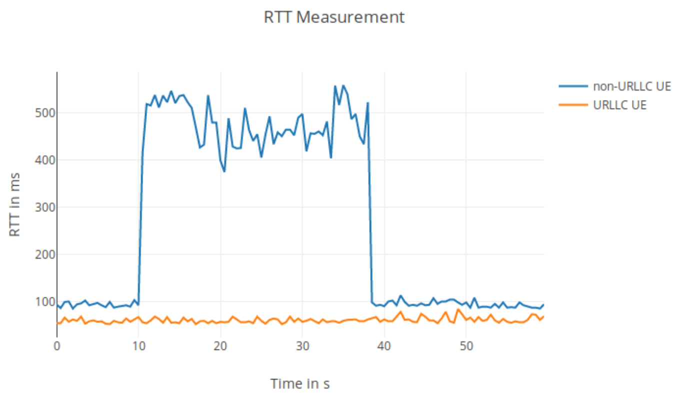

Fig. 13. RTT measurements for a URLLC and a non-URLLC user

VII. Drawbacks and Possible Solutions

The topology description in section VI-A shows already a drawback of the current solution. In the present deployment, the GTP encapsulation is done by the UE itself. This does not correspond to a real deployment, in which the TEID assignment and therefore the GTP encapsulation in ensured by the eNB. Since TEIDs cannot be used as a matchable identifier within OpenFlow, the UE's MAC address is used as a unique identifier for maintaining a UE data base in the controller and matching accordingly in the *EPC switch* would not be available.

Thus, for a real deployment, a different matchable identifier has to be used for the controller to allocate a UE in its dedicated user group. A solution for that problem is to integrate GTP parsing and therefore TEID matching in the OpenFlow standard. When using a OpenFlow capable eNB the priority information would be exchanged with the controller situated in the EPC during the UE registration process, illustrated in figure 14.

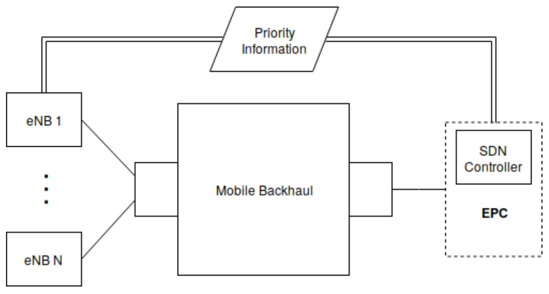

Fig. 14. Exchange of priority information between eNB and EPC

Based on the exchanged priority information, the controller would instruct the eNB to directly assign the URLLC's VID to the GTP frame created in this entity, when receiving packets from that UE. Thus the controller would be only involved during the registration process and no additional packets have to be processed by the controller afterwards.

Generally speaking the user group identification is not entirely based on the VLAN identifier. Instead of matching for a specific VID in the Backhaul the ToS field could be used as well by assigning a specific QCI depending on the the UE being a URLLC member.

VIII. Conclusion

This paper has outlined the possibility of using SDN to enable Network Slicing in the Mobile Backhaul.

It suggests algorithms to measure and monitor necessary link specific data in a network with OpenFlow capable switches by introducing active and passive measurement technologies. The obtained data is utilized to determine reliable and low latency paths that can be assigned to a specific group of UEs, referred to as URLLC group. The suggested controller application is tested by congesting specific links in the present network. After detecting the congestion the URLLC group's packets are redirected on a different path that is not congested, which proves the capability of SDN managed Backhaul networks to adjust to topology changes by assigning alternative paths.

References

[1] Mininet. [Online]. Available: http://mininet.org/
[2] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," vol. 54, 04 2012, pp. 10–10.

[3] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, Jan 2014, pp. 1–7.

[4] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, April 2016, pp. 1–6.

[5] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular sdn controllers," in *2018 Wireless Days (WD)*, April 2018, pp. 54–59.

[6] Open Daylight. [Online]. Available: https://www.opendaylight.org/

[7] Open Network Operating System (ONOS). [Online]. Available: https://onosproject.org/

[8] RYU. [Online]. Available: https://github.com/osrg/ryu

[9] Project Floodlight. [Online]. Available: http://www.projectfloodlight.org/

[10] POX. [Online]. Available: https://github.com/noxrepo/pox

[11] *OpenFlow Switch Specification*, Open Networking Foundation Std. ONF TS-023, Rev. 1.3.5, 03 2015.

[12] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *Passive and Active Measurement*, A. Krishnamurthy and B. Plattner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 201–210.

[13] D. J. Hamad, k. G. Yalda, and I. T. Okumus, "Getting traffic statistics from network devices in an SDN environment using OpenFlow," 09 2015.

[14] A. Gangwal, M. Conti, and M. S. Gaur, "Panorama: Real-time bird's eye view of an OpenFlow network," in *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, May 2017, pp. 204–209.

[15] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–8.

[16] Dash. [Online]. Available: https://dash.plot.ly/

[17] Iperf. [Online]. Available: https://iperf.fr/