Middleware implementation for RYU SDN Controller to manage switches in a C-RAN scenario

Lucas Nóvoa[1], Virgínia Tavares[1], Cleverson Nahum[1], Silvia Lins[2] and Aldebaro Klautau[1] *

[1]LASSE - 5G IoT Research Group
Federal University of Pará (UFPA), Belém - PA, Brazil

[2]Ericsson Research
Kista, Sweden

(lucas.pinto,virginia.tavares)@itec.ufpa.br, silvia.lins@ericsson.com

(cleversonahum,aldebaro)@ufpa.br

Abstract. *With the advent of 5G, more stringent application requirements were imposed to the cellular networks. The adoption of Software Defined Network (SDN) technology in the transport network enables more dynamic network control, suitable for several real-time operations and use cases present in 5G deployments. Implementing testbed for various transport network scenarios is not trivial due to the high costs involved, especially with respect to hardware in more complex network topologies. This work provides a low-cost alternative that facilitates complex transport network topologies implementation in real testbeds. It adopts Mininet software for transport network emulation and implements a middleware that facilitates the control of flows and routes as well as the automatic recognition of any topology. In networking research domain, the implemented middleware contributes to the simplification of switches management in software defined networks scenarios.*

1. Introduction

In 5G networks, the Centralized Radio Access Network (C-RAN) architecture plays a strategic role towards cost-efficiency [Kitindi et al. 2017], splitting base station functions between the Base Band Unit (BBU) and the Remote Radio Head (RRH). The fronthaul link is responsible for the BBU-RRH interconnection, while the backhaul link connects BBU nodes to the core network. The fronthaul and backhaul links are part of the transport network which, motivated by cost reduction and more efficient usage of its resources, converges to packet switching deployments instead of using overprovisioned/dedicated links. In this sense, the Software-Defined Network (SDN) is a key-enabler for centralized transport network management for both the fronthaul and backhaul, facilitating the adaptation of network parameters based on end-application requirements. SDN helps centralized

*This work was supported in part by the Innovation Center, Ericsson Telecomunicações S.A., Brazil, CNPq and the Capes Foundation, Ministry of Education of Brazil. L. Nóvoa, V. Tavares, C. Nahum and A. Klautau contributions include both development of the testbed and concepts, they are with LASSE - 5G Group, Federal University of Pará, Belém, Brazil (e-mails:{lucas.pinto, virginia.tavares}@itec.ufpa.br {cleversonahum, aldebaro}@ufpa.br). S. Lins contributions are theoretical concepts and he is with Ericsson Research, Ericsson AB, Sweden.

management in features like traffic control, throughput analysis, delay measurement, and queue priority.

Testbeds play an important role in 5G transport networks and SDN-related research. Taking into consideration some of the testbeds available in the literature as [Muñoz et al. 2017, Rostami et al. 2017], it is possible to perceive that transport network complexity directly increases the testbed implementation costs, since more complex transport networks demands more network resources like real switches and routers interconnected to represent the targeted topology. Another negative aspect is the limited flexibility to deploy different transport network topologies since it demands physical rearrangement of switches/routers connections.

Targeting a low-cost platform to provide easier and more flexible transport network implementation in 5G testbeds, this work proposes the Transport Network Testbed (TNT), composed by an emulated transport network for both fronthaul and backhaul using the Mininet software and the implementation of a middleware to facilitate the route/flow control and the acquisition of information about the transport network through the RYU, that is a SDN controller to manage and control applications [Ryu 2021]. A related work used as a base for the creation of this middleware was [Nahum et al. 2020]. There are many different SDN Controllers as Open Network Operation System (ONOS), OpenDayLight (ODL), OpenKilda, Faucet, and RYU. The main reason to choose RYU instead of the other is that the RYU provides a well-defined API for developers, allowing to change how the components are managed and configured [Baskoro et al. 2019]. There are no works related to some software implementation in the research and industrial field to improve the SDN in a 5G scenario.

This work's structure is as follows: Section II shows the implemented Transport Network Testbed (TNT) in this section, the testbed big picture is provided, and the traffic redirection logic is explained. Section III focuses on the flow redirection and how the middleware manages the communication with the RYU SDN Controller. Section IV presents the results obtained using the middleware. Finally, section V concludes the proposed work where the emphasis is given on the middleware usage by the network manager and the low cost of the middleware.

2. The implemented Transport Network Testbed

The Transport Network Testbed enables the implementation of a C-RAN [Kitindi et al. 2017] scenario with emulated fronthaul and backhaul using Mininet software [Mininet 2021], deploying a SDN transport network with traffic management that can be easily connected to external applications as artificial intelligence-based agents. In a C-RAN architecture, a remote radio head (RRH) communicates with baseband unit (BBU) through the fronthaul, and BBU communicates with the core network through the backhaul. In this work, emphasis is given to the implemented middleware between the SDN controller API and the external applications, using Mininet as the framework for the emulated transport network topology. For mobile network functions deployment at RAN and core networks, it was used the Connected AI testbed proposed in [Nahum et al. 2020]. The TNT architecture works as an extension to the Connected AI testbed. The TNT implements a middleware to facilitate the communication with the SDN controller API and performs switch nodes management through configuration

files and API requests. It can be used to control the routing process, the data flow and to obtain network Key Performance Indicators (KPIs). The TNT code is available on Github [LASSE 2021] with all the use cases explored along this work.

2.1. Architecture Overview

Figure 1 depicts an example of a network architecture implemented using the TNT over the Connected AI testbed. A C-RAN architecture is implemented using a RRH and a BBU to perform eNB processing. In this architecture, all elements inside the dotted rectangle are executed on the machine that runs the Mininet software, responsible for implementing a middleware between the network manager and the SDN controller. In this figure, Mininet emulates the transport network of the backhaul connection, where all depicted switches are emulated. Each 5G network service is performed inside a Docker container and orchestrated with Kubernetes [Kubernetes 2021], using Calico as the container network interface as described in [Nahum et al. 2020]. The machine that executes the Mininet software is responsible for implementing the topologies of switches and virtual hosts, managing the network backhaul using the developed middleware and executes the SDN controller.

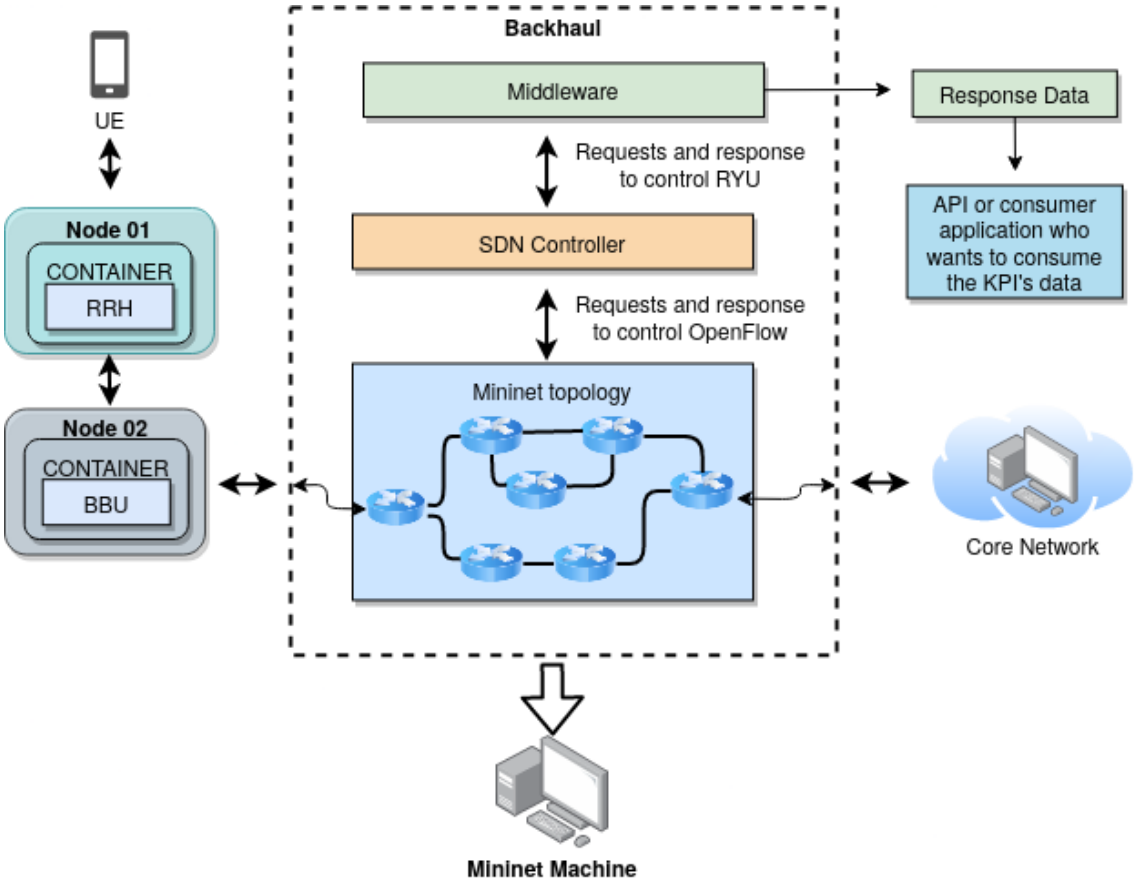

Figure 1. Architecture Overview

It is important to emphasize that Mininet's switches and virtualized host architecture do not have bidirectional communication with a host machine. Hence, the hosts are not reachable from external networks, and it is not possible to forward the backhaul

traffic over the emulated topology using the Mininet default implementation. Therefore, it is necessary to create traffic redirection rules for the containers running into the testbed applications to communicate with the Mininet virtualized switches as they were a non virtualized machines.

We implemented virtual ethernet interfaces into the Mininet machine to create a link between its ethernet interface and the virtual interface of hosts and switches emulated in Mininet to enable bidirectional communication between them in a way similar to [Nahum et al. 2020]. Once this link is created, a rule is applied to the BBU and core network containers forcing their traffic to be forwarded to the Mininet machine instead of making a direct communication as made by default. When the backhaul traffic is finally redirected to the Mininet machine, the forward of the machine ethernet interface to the emulated host virtual interface makes the backhaul traffic pass through the emulated topology with all switches and routers that users previously defined into Mininet deployment scripts.

One drawback of the method used to create links between the machines and the Mininet virtual hosts presented in [Nahum et al. 2020] is its low flexibility, which hinders the definition of new emulated topologies since all of them should manually implement the links between the machines and virtual hosts. In this work, the implemented middleware is responsible for identifying any Mininet topology set by the user, and automatically creates these virtual links between the Mininet machine interface, configuring the forward rules at the containers and machines. Therefore, the user does not need to make any manual configuration, but only to use a configuration file which provides the IPs of containers and Mininet virtual hosts, required for link instantiation for traffic forwarding.

3. Flows Redirection

Once the middleware is running, and the routes are already configured, the processing pipeline works as follows: the implemented code works as a middleware for RYU and external applications, and RYU works as an API for the OpenFlow protocol that is responsible for editing the forward tables on each switch. The changes implemented in the RYU configuration file allow RYU to maps all switches with their links, ports, and adjacent switches. The mapped information is stored in a textfile that operates as an interface between programs to analyze the routes and obtain the networks' KPIs.

In Figure 2, one of the API request options is presented, where the middleware decreases significantly the number of requests needed to the external applications to create a new flow into the Mininet topology. So, the TNT middleware simplifies the communication between a consumer application and the RYU SDN controller. In this figure, steps 1 and 2 are executed every time the initial script rises. These steps are done to clean any previous rule in all switches and to create the flow table 0 to send the IP to table 1 and ARP to table 2. After that, the processes depicted are all executed by input entries required by OpenFlow to manage the switches. Once the application performs the step 3, the middleware routine is started. The middleware receives the consumer application request and executes steps 4 to 8.

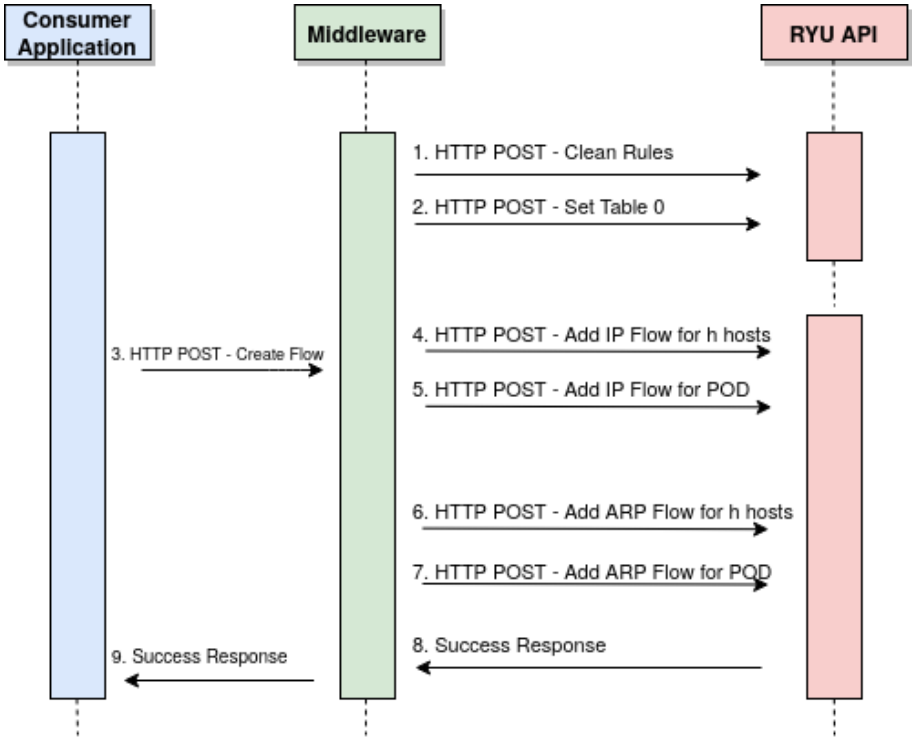

Figure 2. Ryu REST flow diagram

4. Results

This work implements three experiments to test some use cases of the TNT. First, it is compared the number of requests to implement all flow tables in the simple scenario, using the middleware and RYU. Second, it is performed a route change algorithm during the time, whether it is showed that the analysis does not interfere in the communication between the machines. Third, the delay measurement was used to create a dynamic algorithm, whether the delay is a trigger for changing the route. The setup for build the use cases that the middleware, Mininet and RYU SDN Controller attends is composed of one computer with a Core i5-8400 and CPU@2666 MHz Quad-Core processor with an operational system Ubuntu 18.04 LTS, 5.4.0-73-generic kernel and a 8 GB of RAM.

Figure 3 presents three different backhaul topologies, where each topology has different routes and weights. The adjacency matrices shown in this graph representation, contain the representation of the connections between the switches. The rows i and columns j represent if there is a direct link between switch i and switch j. These matrices represents the switches communication and which routes and flow are possible to be set.

The middleware's initial script is responsible for choosing the default path between the two endpoint switches and creates each matrix showed in Figure 3. The middleware script allows the user to manage the traffic through the switches routes. Therefore, to perform the storage of switches' connection, it is necessary to use the RYU program to get the dictionary of a switch, adjacency switch, and port that connect both switches. The data is stored in a configuration file, and the middleware creates this matrix for getting the default path between the hosts. In most Linux interfaces and switch configurations, it is possible to set one route at a time. However, using the OpenFlow table managed by the

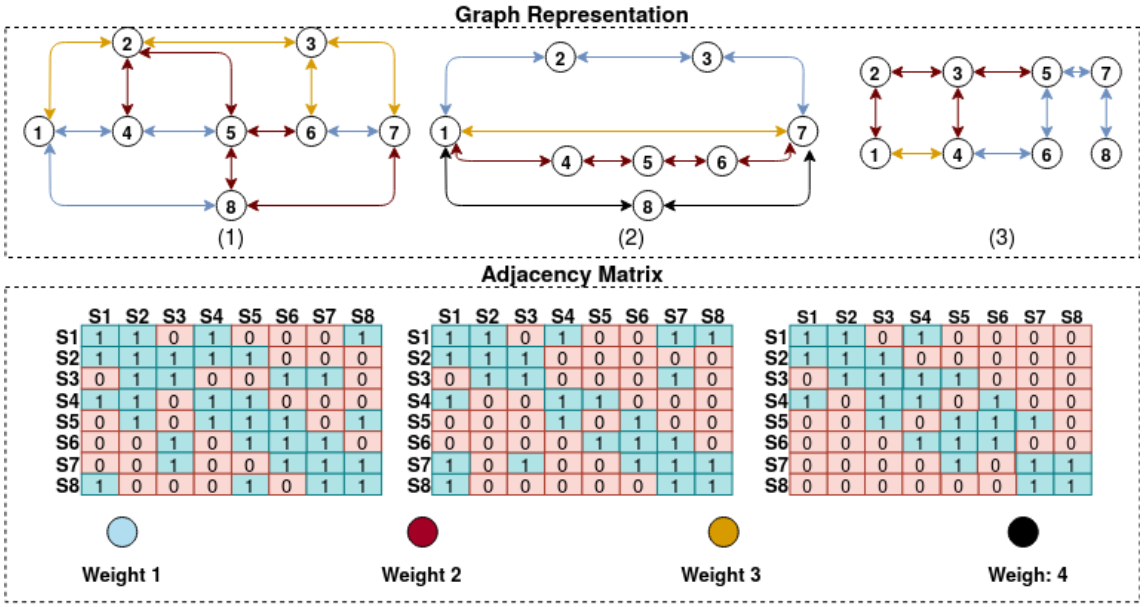

Figure 3. Graph Representation

middleware and the input that the data comes, it is possible to provide a simple path such as passing through switches 1-4-5-6-7 or more complex paths such as passing through the same switch more than one time for different ports. Therefore, a packet coming from the same source towards the same destination could have different treatments depending on the port on which the switch dispatches the data. The different treatments allow the user to make dynamic configurations.

The weights seen in the graph representation indicate the delay that the switch adds to the route. Weight 1 is the shortest delay, and weight 3 is the most extended delay of the analyzed topology. These different delays are set to simulate a scenario of different possibilities for several use cases; for example, a UE that is sending data to the internet can use one of the available paths in Mininet topology. Also, the UE can use the path with higher delay to verifies the effects of a lag on the network or use the path with lower delay to simulate an application that needs a real-time request. These weights differ from architecture to architecture to attend different cases. For the second topology, the weight 2 has a delay of 14ms, while that same weight in the first topology has a delay of 30 ms. No matter the number of switches in the topology, a network manager with access to the Middleware developed can use different weights and paths to get the UE requirements' best traffic configuration.

One of the best ways for configuring the switches is by creating a table that handles with IP and another that handles with ARP; after that, the flow redirection can be easily configured. Therefore, to create communication like that, many requests have to be made in each switch of the topology. The requests for small topology (with 3 or 4 switches) is performed with no difficulty. Otherwise, the number of requests can be a problem for the network manager for an extensive topology. The number of requests for certain numbers of switches with and without the use of Middleware for each topology is shown in Figure 4. In this figure, the x-axis represents the number of switches in the Mininet topology, and the y-axis represents the number of requests to change one route in all

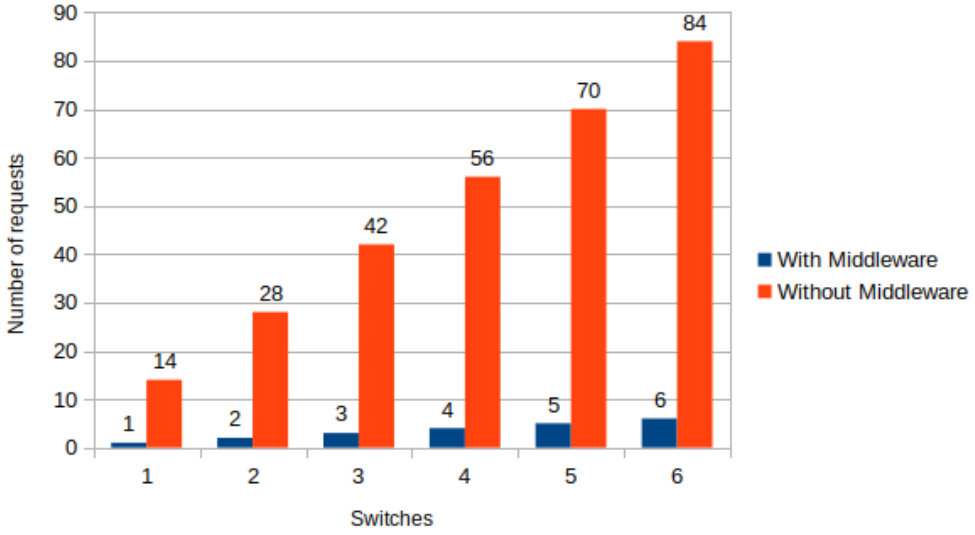

Figure 4. Request graphic according the number of switches

switches. The number of requests that a network manager has to send in the RYU API to create a simple route is represented by the orange bar and it has fourteen requests per switch. Therefore, the middleware enables a great reduction in the number of requests , decreasing the complexity to perform changes in the topology routes.

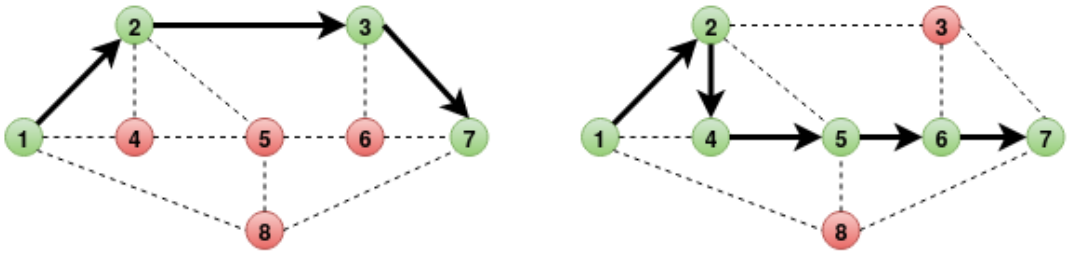

Figure 5. Traffic flow through the switches in Mininet topology

Using the middleware, the network manager has to send just one request for each switch to configure the IP and ARP rules in Tables 1 and 2 and the flow redirection. Therefore, for a topology of six switches, if the network manager chooses not to use the middleware, the number of requests to create the forward for IP and ARP equals eighty-four. Hence, the middleware reduces the number of steps that the network manager has to perform, for the six switch's case, with middleware, the consumer application needs just one request. It is possible to create dynamic configuration using the middleware to configure all switches. For example, whether the consumer application performs all settings initially by applying the forward packet rules using the middleware, an external agent only has to send one request to generate a new route.

In Figure 5 it is perceived that the topology already has a path configured (the green switches). However, all inactivated red switches are configured by the consumer application that performs all settings initially by applying the forward packet rules using

the middleware. For example, the behavior of switch number four is configured in such a way that if a packet comes from switch two, he will forward the traffic to the port connected with switch number five. Once the behavior of the switch is already configured, this will allow an external agent to only sends one request to generate an activated new path.

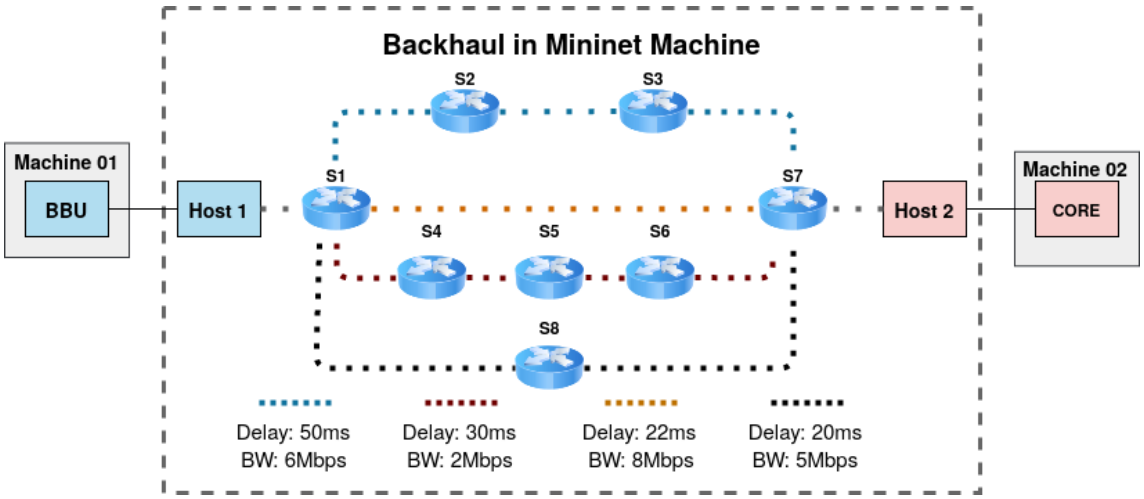

Figure 6. The third topology overview

The third topology proposed for fronthaul is showed in Figure 6. This is a simple topology where there are five possible routes. Each dotted between a switch, and another represents links configured in the Mininet file itself, which means that this is a use case of the user developing the application. All configurations as creating several switches, the links, the latencies between the links, and the bandwidth between switches will be left according to the user's need.

For the real-time route change test, the communication was initiated between the UE and the core of the network for one minute, resulting in a throughput graph over time for four different routes as shown in Figure 7. During these 60 seconds, the route change was performed without interrupting the two hosts' data transfer. During this interval, different length of bandwidth was requested. When the value of the throughput passing through the route is higher than the bandwidth value, e.g., if the bandwidth value is 5 Mega and the consumer application is sending 7 Mbit/s, the network manager has to change the traffic for another route.

In the first 15 seconds, route one (1-2-3-7) was used to forward the UE data to the core network. In the interval of T=15 to T=20, the middleware performs a route change, resulting in a transition gap of curves. Both curves in this transition decay in the same proportion during the change from one route to another. This pattern is repeated for the other two remaining routes. The graph also shows us that the middleware can change route without problem in the communication between the machines, allowing, for example, an Machine Learning Agent to use this middleware in real-time applications for differents multipath routing user cases, like shortest path, load balancer, and static queues.

The consumer application can use the middleware's RESTful API for the delay test to choose when getting the delay response instead of executing it every second for each switch using the Ryu Events call. The delay control can help the test with the lag

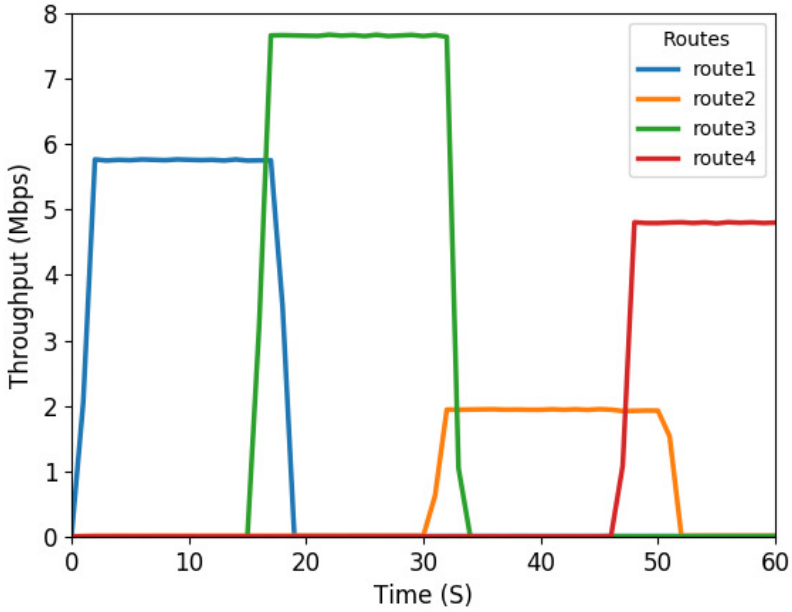

Figure 7. Throughput over the time

problem and possible solutions like changing the route responsible for the lag. Therefore, to demonstrate how the consumer application can perform this delay control, this work will focus on just one link in the forward lines.

During one minute, traffic between BBU and the network's core suffered four variations, one every 15 seconds. The network manager performed these variations to simulate a scenario of instabilities during this time. The delay measurement also has a probe that verifies when the delay is more significant than 45 ms. Whether the probe verifies that the application exceeded the limit value, this probe changes the route once this switch is compromised with a lag. By contrast, once the application has a lag, the algorithm has to return for this route if the latency decreases to a value lower than 45 ms. The Figure 8 shows the route's change with the delay at the same time.

In the interval of 20 to 40 seconds, it is perceived that the topology has the main route with a lag. The consumer application use the second route with a bandwidth of 8 Mega, while the probe does not allow us to use the main route. When the probe perceives that the delay has returned to a value lower than 45 ms, another change of route is performed once the main route it is operating normally.

In the first interval, the delay was in the range of 11 ms until 24 ms, which means that the application can have traffic passing through the route, including this switch, without a problem. The instant of time t=15 seconds, the delay increases until the value of 72 ms. With this increase, the probe perceives that the delay of the analyzed switch exceeded the value of 45ms, so a request is sent to change the route for one route with no compromised switches. The middleware has to use this probe for every switch in the network. It is perceived that the throughput does not change the route in the second that the delay breaks the limit. This lateness in the route's change is because rootthe probe sends the request in t=15 seconds, but it took time until the change occurs.

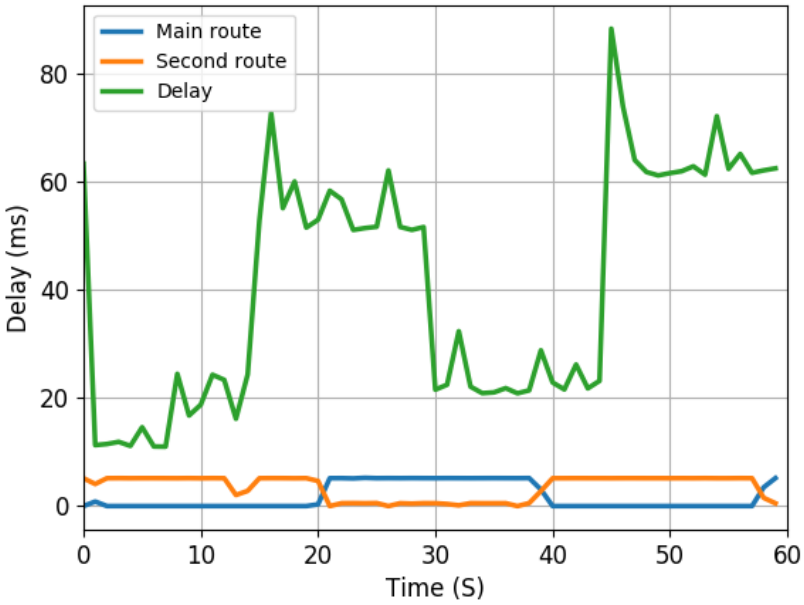

Figure 8. Throughput and Delay analsis

In the interval of 20 to 40 seconds, it is perceived that the topology has the main route with a lag, so this route can not pass the backhaul traffic. The consumer application have to use the second route with a bandwidth of 8 Mega, while the probe does not allow us to use the main route. When the probe perceives that the delay has returned to a value lower than 45 ms, another change of route is performed once the main route it is operating normally.

The probe does not interfere with traffic, been used simultaneously during the BBU communication with the network's core. Therefore, a consumer application can control the traffic according to the delay variation, changing or returning to the same route every time a route reaches some limit value.

This testbed proved to be flexible, allowing the implementation of different topologies for Fronthaul and Backhaul, allowing changes in routes during runtime, which means that the same topology can attend to differents UE application requirements. The solution also proved to be low-cost and very useful for research, specially when several scenarios need to be tested. The SDN controller's complexity is abstracted, allowing the user to adopt the proposed software according to their needs, having their effort to manage the OpenFlow switches being significantly reduced.

5. Conclusion

This paper presented the transport network testbed (TNT) to implement a low-cost transport network to the Frothaul and Backhaul in mobile networks using Mininet. Furthermore, the developed middleware is responsible for creating links between any Mininet topology and mobile network functions, mainly to the backhaul and fronthaul as demonstrated by this work, enabling the traffic control and other SDN functions. There were two results in this paper: throughput over time and Delay analysis. The first result demon-

strated that the proposed middleware could change routes from one switch to another and measure the number of requests performed by the RYU API and the middleware. The second result used a probe to measure a limit value for the delay using the created middleware; Once the probe perceived that a delay value was achieved, the middleware executes a route's change.

Therefore, it was demonstrated how a network manager or an external application could take advantage of the simplification provided by the implemented software, creating traffic redirect, delay analysis, and reducing the effort to manage all switches. Otherwise, there is an actual direct result that is the computation cost of the proposed work. Accordingly, the implementation of TNT is fully low-cost, once all software used was OpenSource, and their specification is a simple CPU and memory requirement. Hence, a simple computer can run and simulate this transport control in backhaul or fronthaul with Mininet, RYU, and the implemented middleware. Finally, this work must emphasize that the easy deployment of this scenario instead of configuring real cisco switches is also an essential contribution of this proposed work.

References

Baskoro, F., Hidayat, R., and Wibowo, S. B. (2019). Comparing lacp implementation between ryu and opendaylight sdn controller. In *2019 11th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–4.

Kitindi, E. J., Fu, S., Jia, Y., Kabir, A., and Wang, Y. (2017). Wireless Network Virtualization With SDN and C-RAN for 5G Networks: Requirements, Opportunities, and Challenges. *IEEE Access*, 5:19099–19115.

Kubernetes (2021). Kubernetes: Production-Grade Container Orchestration. [Online]. Available: `https://kubernetes.io/` Accessed on 2021-03-27.

LASSE, T. (2021). Transport Network Testbed. [Online]. Available: `https://github.com/lasseufpa/transport-network-testbed` Accessed on 2021-03-27.

Mininet (2021). Mininet: An Instant Virtual Network on your Laptop (or other PC). [Online]. Available: `http://mininet.org/` Accessed on 2021-03-27.

Muñoz, R., Nadal, L., Casellas, R., Moreolo, M. S., Vilalta, R., Fàbrega, J. M., Martínez, R., Mayoral, A., and Vílchez, F. J. (2017). The adrenaline testbed: An sdn/nfv packet/optical transport network and edge/core cloud platform for end-to-end 5g and iot services. In *2017 European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE.

Nahum, C. V., Nóvoa, L., Tavares, V. B., Batista, P., Lins, S., Linder, N., and Klautau, A. (2020). Testbed for 5g connected artificial intelligence on virtualized networks. *IEEE Access*, 8:223202–223213.

Rostami, A., Ohlen, P., Wang, K., Ghebretensae, Z., Skubic, B., Santos, M., and Vidal, A. (2017). Orchestration of ran and transport networks for 5g: An sdn approach. *IEEE Communications Magazine*, 55(4):64–70.

Ryu (2021). Ryu API. [Online]. Available: `https://ryu.readthedocs.io/en/latest/api_ref.html` Accessed on 2021-03-23.