GANSO: Automate Network Slicing at the Transport Network Interconnecting the Edge

José Takeru Infiesta*, Carlos Guimarães*, Luis M. Contreras†, Antonio de la Oliva*
*Universidad Carlos III de Madrid, Spain
†Transport & IP Networks, Telefónica I+D / Global CTIO Unit, Madrid, Spain
Email: jtinfiesta@gmail.com, cmagalha@pa.uc3m.es, luismiguel.contrerasmurillo@telefonica.com, aoliva@it.uc3m.es

***Abstract*—5G and Edge computing are two technologies set to impose a paradigm shift from today's traditional networking solutions. In particular, transport networks, which connect distinct computing infrastructures, must guarantee a wide range of performance requirements from coexisting network services. 5G network slicing enables such capability by providing the flexibility to support multiple and isolated virtual networks over the same and shared infrastructure. This paper introduces the *GST And Network Slice Operator* (*GANSO*) framework for automating the creation of network slices over SDN architectures, focusing on transport networks interconnecting Edge data centers. To characterise the type of network slice to be deployed, it uses Generic network Slice Templates (GSTs). Initially, five GST attributes are implemented in a proof-of-concept prototype, namely through configurable *User Data Access* and *Rate Limit* parameters. It is then validated in a scenario considering the instantiation of network slices over the transport network for different virtual applications hosted across the edge-to-cloud continuum.**

***Index Terms*—Network Slicing, GST, SDN, Transport, Edge**

I. Introduction

The concept of multiple edge data centers has been introduced to overcome the limited resources and computing power of single and isolated edge instances. By exploiting the locally distributed edge data centers, usually co-located with the Radio Access Network (RAN) (e.g., base stations and aggregation points), the workload required by different network services can be balanced among the resources of collaborative computing nodes. Multiple deployment arrangements (e.g., hierarchical- and mesh-based) can be employed to connect the different (edge) data centers together. Transport networks (e.g., access network, metropolitan area network and core network) play a key role in their interconnection and, consequently, network slicing at the transport network – the transport slice – is essential to glue all the virtual applications of a given network service while providing guarantees with respect to their key performance indicators (KPIs).

Network slicing is expected to become the new manner of providing services on telecom networks, mostly pushed by the introduction of 5G services which are natively built on top of slices. Through network slicing, telecom operators can allocate partitions of a physical infrastructure (including computing, storage and networking resources) to different vertical customers which run their respective services as if the specific partition was a dedicated network.

The network slices are defined end-to-end (E2E) comprising different network segments such as access, core and transport. In the case of 5G, the 3GPP [1] considers an overarching 3GPP Management System that interacts with different domain-specific controllers per segment. The network slices are requested to the 3GPP Management System by means of templates used to specify their concrete characteristics according to the requirements of each particular vertical customer. The GSMA has defined a generic blueprint that may be used by any kind of vertical customer to specify its needs. Such a blueprint is known as Generic network Slice Template (GST) [2], containing 35 parameters on its latest version. The vertical customer, when specifying the values for this generic blueprint, forms what is known as a NEtwork Slice Type (NEST) that is the actual input to the 3GPP Management System. From an operational point of view, the telecom provider might support both Standardised- and Private-NEST. While Standardised-NEST pertain to those elaborated in a normalised way by distinct organisations (e.g., 5G-ACIA in the case of Industry 4.0), Private-NEST are directly customised and created by the telecom provider.

At the transport network segment, the NEST is processed to extract requirements that directly apply to the creation and instantiation of the transport slice. This consists on identifying parameters that have a direct or an indirect impact on the selection of transport resources to form the E2E slice connectivity. Some attributes can be directly translated into transport network requirements [3] (e.g., "slice throughput"), while others have indirect implications (e.g., "latency from (last) UPF to the application server") with respect to the location of the application (e.g., edge or central cloud) and the resources to be committed in the transport network to reach it. Thus, from the perspective of the transport network control entities it is essential to account on mechanisms for assisting on such translation, resulting on the instantiation of the transport network slice.

This paper focuses on that provisioning of network slices in the transport segment based on the concept of slice templates. It proposes the *GST And Network Slice Operator* (GANSO), a framework developed to automate the provisioning of network slices over Software Defined Networking (SDN) infrastructures (e.g., for connection between edge data center(s), centralized and/or public clouds). This framework leverages on GSTs to define the network slice type and that are then translated and mapped into the network resources and parameters required for its instantiation. Section II introduces related work linked

to the concept and purpose of network slicing. Section III presents the proposed framework, whose validation, through a proof-of-concept prototype, and initial results are shown in Section IV. Finally, Section V provides concluding remarks and future work.

II. Background and Related Work

In this section the relevant concepts leveraged on this work are briefly recapped and framed under their relevance to automate the provisioning of network slicing. Afterwards, the existing state of the art linked to the concept and purpose of network slicing automation is identified.

A. Software Defined Networking (SDN)

Software Defined Networking (SDN) [4] decouples the data and control planes, logically placing the latter in a centralised *controller* node as opposed to traditional networks in which both are embedded within forwarding devices. While this controller decides how to forward packets, forwarding devices are left out to perform the configured actions. Such approach has paved the way for a reliable and fast network programmability which is achieved through two interfaces: (i) *Northbound API*, which provides applications with an abstract view of the network so that the control plane can be easily configured; and (ii) *Southbound API*, which enables the controller to communicate with the data plane elements.

By doing so, SDN can contribute to the automation of the provisioning of network slices as it allows the desired network behaviour to be conveyed to the controller which, in turn, is responsible for enforcing said behaviour onto the underlying forwarding devices as well as enabling network resources to be distributed among different network slices.

B. GST and NEST

The concept of Generic network Slice Template (GST) and NEtwork Slice Types (NEST) [2] was developed to aid in the automation of the process of designing and instantiating network slices for network operators. Whereas both are used to describe a network slice the main difference between the two is that while the former models a network slice by containing the names of the attributes that characterise it, the latter defines the actual values of these attributes for a particular network slice. NESTs can then be interpreted by the network operator so that it can instantiate network slices that comply with the specifications of its requester.

By doing so, GST and NEST can contribute towards the automation of the provisioning of network slices as both define a common set of generic attributes that can convey the communication requirements of different network slices.

C. Network Slicing of Transport Network

Network slicing is a concept introduced by the *3rd Generation Partnership Project* (*3GPP*) organisation's Release 15 for the standardisation of 5G. The physical infrastructure is virtualised such that its resources can be distributed among logical independent networks that simultaneously serve different customers with specific requirements.

This concept is pushing towards programmable transport networks where different network services can be accommodated and coexist simultaneously. Such design is being leveraged by several SDN/NFV-based E2E network slicing solutions. 5G-TRANSFORMER [5] is an example of such approach that aims to bring the network slicing paradigm into SDN/NFV-based mobile transport networks by provisioning and managing slices tailored to specific requirements. Another example is proposed in [6] where a slice management and orchestration framework is defined. The former work defines a layered concept where higher layers provide an abstraction from the lower layers, with the top layer providing a set of interfaces to allow the vertical to monitor and operate the slice. The latter work proposes a control, management and orchestration platform that forwards requests to the respective controllers that implement the data path on the underlying SDN-switches. Other works [7][8][9] also leverage on the usage of SDN mechanisms to configure the data plane of network slices.

SliMANO framework [10] is proposed as a plug-in based framework that automated instantiation of E2E network slices over the different network segments. Requests are made through a custom-made *Northbound API* and, later, translated by the SliMANO framework to the necessary network operations. With respect of network slicing at the transport network, its interaction with the network controller (e.g., an SDN controller) is limited to the (re)configuration of the datapath, not defining or implementing any mechanism to define or enforce the requirements of the network slice.

Proponents of the previous work already exploit, although limited, the use SDN to implement slicing on the transport network. The previous works either consider custom-made interfaces to define the requirements of the network slices or focus only on the provisioning of the data plane forwarding for the network slice disregarding any of their requirement. This work advances on the state of the art by leveraging on: (i) the use of GST and NEST as a uniform and accepted characterization of a network slice; and (ii) the use of SDN at the transport network not only to configure the data path for a given network slice but also to enforce its requirements.

III. GANSO: GST And Network Slice Operator

The *GST And Network Slice Operator* or *GANSO* (which is Spanish for 'goose') is a proof of concept framework developed as a means to automate the creation, deployment and instantiation of network slices over Software Defined Networking (SDN) infrastructures. This framework also offers a set of management mechanisms to aid the network administrator on its interaction with the underlying network.

A. GANSO Framework Overview

Due to the flexibility and the degree of network personalisation provided by SDN, GANSO has been developed to instantiate network slices over architectures that follow this networking paradigm. As such, it exploits many features enabled by SDN, like reliable and fast network programmability.

As depicted in Figure 1, GANSO is provided as an application residing on the SDN Application Layer. Using the *Northbound API*(*s*) exposed by the underlying controller(s), it is able to convey its requests for network slices, along with their requirements, to the SDN Control Plane Layer. In turn, the controller translates the requested network behaviour into rules that are understood by the forwarding devices that perform packet forwarding, being implemented in the SDN Data Plane Layer through a *Southbound API*.

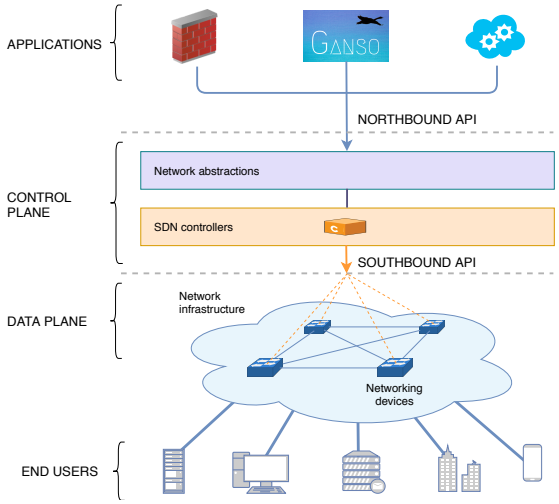

Fig. 1. GANSO Placement at the SDN Architecture Planes

For the time being, GANSO relies upon Open Network Operating System (ONOS) — an open source controller proposed by the Open Networking Foundation (ONF) — to represent the control plane of a SDN network, as it allows network configuration with high availability, scalability and performance through strong abstractions. Nevertheless, GANSO is flexible to be integrated with different controllers through extensions to use their *Northbound API*.

Being implemented as an ONOS application, GANSO takes advantage of the *Northbound API* defined by ONOS to directly instruct the controller to create and instantiate network slices. Specifically, GANSO communicates with the REST API exposed by ONOS which bases the exchange of information on the JSON format. Therefore, a request for a network slice is interpreted by GANSO and translated into a JSON-formatted description of the network slice's behaviour. Subsequently, it is sent through POST requests to the ONOS REST API.

Upon receiving the JSON data, the controller reads its content and sets the rules that enforce the requested behaviour in the underlying forwarding devices through the *Southbound API*, thus creating a network slice that fulfils and delivers the user's network requirements. For this purpose, ONOS uses the OpenFlow protocol [11] and, therefore, forwarding devices must be compliant with OpenFlow (e.g., *Open vSwitch*) in order to enable their communication with the controller and the configuration of desired network behaviour.

When OpenFlow is used in an SDN infrastructure, the controller essentially configures the flow tables of each forwarding device, inserting, removing and modifying flow entries that serve as rules that specify how a packet matching a given criteria must be processed. Thus, GANSO also relies on OpenFlow for the creation of the different flow entries in the forwarding devices that match the network slice's attributes configured at the GANSO application through GSTs/NESTs.

B. Network Slice Request Workflow

The creation and instantiation of a new network slice (whose workflow is represented in Figure 2) is initiated by the user by issuing a request towards the GANSO application (step 1). Whenever a new request is received, GANSO provides a GST form to be filled by the user (step 2) and expects a NEST to be returned with the desired network slice configuration and requirements (step 3). Upon receiving the NEST, GANSO triggers the instantiation of the requested network slice in the underlying network (step 4).

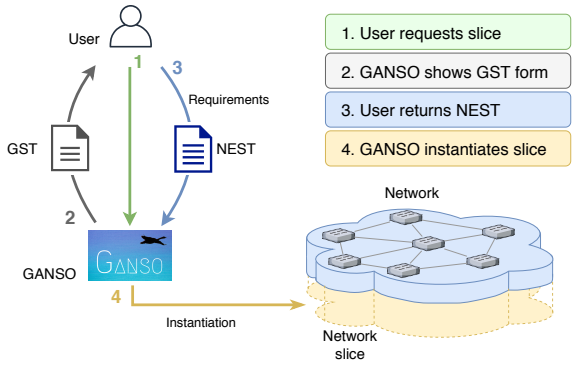

Fig. 2. GANSO - Network Slice Request and Instantiation Workflow

Throughout this process GANSO executes a set of internal procedures (Figure 3). First, it reads the values of the attributes configured in the NEST, mapping them into an XML file that contains the information of the network slice and that is subsequently stored. Alternatively, GANSO allows users to directly upload their own NEST XML provided that these are properly formatted. The NEST XMLs are then translated into JSON format (as referred in the previous Section) and processed such that the corresponding network slice is instantiated.

C. Supported (GST) Attributes

Currently, GANSO implements, as a proof-of-concept, two attributes that can be set while creating the network slices: *User Data Access* and *Rate Limit*. The decision to implement these attributes over others was based on the fact that they not only showcase different behaviors that are expected to be supported by slices but also because these are parameters typically requested by customers. Additionally, these can be directly mapped into five GST attributes: *User Data Access*, *Maximum Downlink Throughput per UE*, *Maximum Uplink Throughput per UE*, *Guaranteed Downlink Throughput per UE* and *Guaranteed Uplink Throughput per UE*.

- **User Data Access:** this parameter is declared as mandatory in the GST definition of a network slice and it is

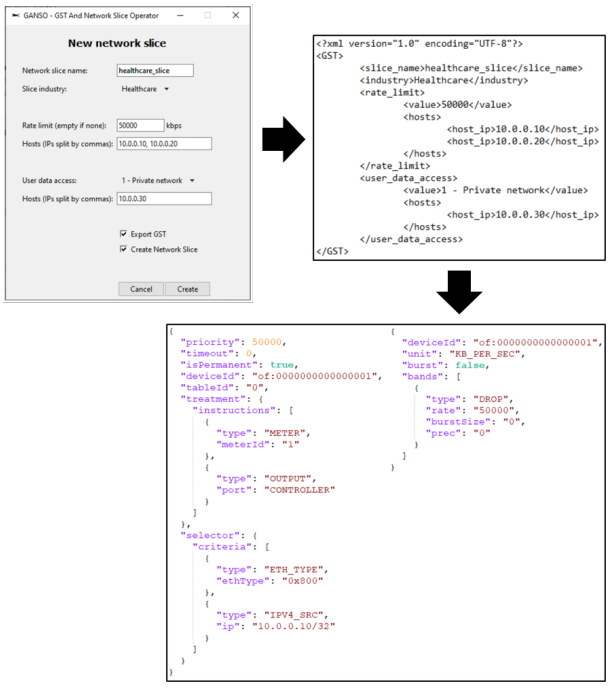

Fig. 3. GANSO - From GST to JSON Format

used to define the level of connectivity allowed within the network slice (i.e., how the network slice handles the user data) through three options: (i) *Direct Internet access*, where devices can directly access the Internet; (ii) *Termination in the private network*, where devices can only communicate in the private network; and (iii) *Local traffic*, where no connection whatsoever is allowed (i.e., all data traffic stays local and devices do not have access to the Internet or private network). Thus, when filling a GST, users are able to choose one of these options, with the default value being *Direct Internet access*. When the *Termination in private network* option is selected, GANSO creates two rules per device (or subnet): (i) one that enforces forwarding of packets sent by the specified device (or subnet) to destinations belonging to the same network; and (ii) one for dropping packets sent by the specified device (or subnet) to destinations which are external to the network. Finally, when the attribute has value *Local traffic* only one rule that drops all packets sent by a specified device (or subnet) is required.

- **Rate Limit:** this attribute enables defining the maximum throughput, either in downlink or uplink, allowed to a given device (i.e., UE), as described by the *Maximum Downlink Throughput per UE* and *Maximum Uplink Throughput per UE* attributes in the GST. In addition, by leveraging on this same attribute, GANSO is also able to guarantee that the requested throughput, either for downlink or uplink, is provided when enough bandwidth is available, as defined by the *Guaranteed Downlink Throughput per UE* and *Guaranteed Uplink Throughput per UE* attributes in the GST. When filling the GST, a GANSO user can set this limit in *Kbps*, with the default value being *none*. To enforce the *Rate Limit* in the underlying network, GANSO creates flow and meter entries in the forwarding devices to drops all incoming packets whose throughput surpasses the configured limit.

D. Integration with ONOS and OpenFlow

As described in the previous sections, GANSO is implemented as an application that relies on ONOS and the OpenFlow protocol to configure the underlying SDN network.

1) Bootstrap and Communication: Upon startup, GANSO connects to the controller of the network and learns which forwarding devices are under the domain of the controller. This is an essential step during its bootstrap so that GANSO can identify to which devices the POST requests must be sent when rules are created. As mentioned before, GANSO communicates and configures ONOS through its REST API which understands data formatted in JSON. Thus, for each flow and meter entry to be installed in the forwarding devices via OpenFlow, a POST request that describes said entry element needs to be defined and sent to the controller. Furthermore, one request per flow entry must also be sent for each specified device (or subnet) and each forwarding device that sits in the network. The IP address of the device (or subnet) will be used as the matching criteria for packets in the flow entry such that isolation and multiplexing of slices are provided.

It should be noted that in OpenFlow-enabled devices, reading and processing rules follows a strict order. Packets are sequentially compared to flow entries from highest to lowest priority on each available flow table. This process stops when a packet matches any flow entry or the last entry of the last flow table is reached. When a packet matches the criteria set by a flow entry, the instructions associated to it are applied. Thus, when creating network slices the priority of the attributes must be taken into consideration, otherwise the expected behavior is not correctly implemented in the forwarding devices.

2) Attribute Implementation: An example of the ingress packet processing pipeline configured in a forwarding device is shown in Figure 4, where different configurations for both *User Data Access* and *Rate Limit* attributes are depicted.

Since *User Data Access* attribute defines the connectivity level allowed to a given device (or subnet), flow entries to implement this parameter must have a higher priority than the ones to implement the *Rate Limit* attribute.

User Data Access: Flow entries of *User Data Access* attribute are allocated in *Flow Table 0*. When the *Local traffic* option is set, the flow entries must be defined with the highest priority and with the defined action to drop all matching packets (i.e., UDA rule 2). If the *Termination in the private network* is set, flow entries must be second in priority with defined action to forward the packets to be matched by flow entries in *Flow Table 1* (i.e., UDA rule 1). Lastly, if the default value *Direct Internet access* is set no additional flow entries are required (i.e., ONOS forwarding rules).

Rate Limit: The rules for the *Rate Limit* attribute, which have lower priority, are contained in *Flow Table 0* (i.e.,

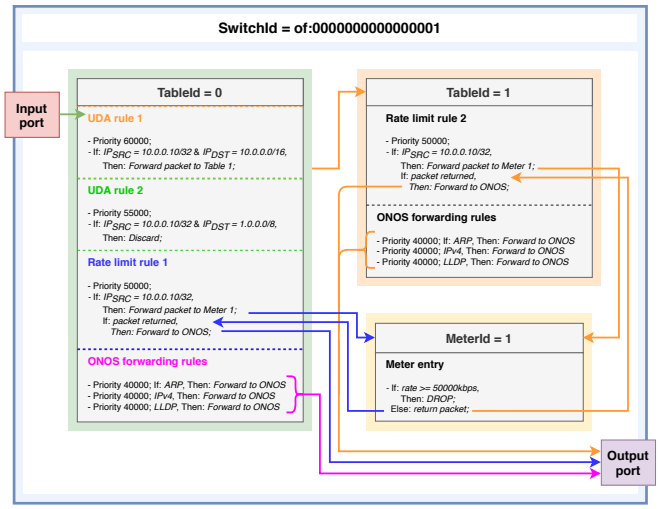

Fig. 4. Example of the Packet Processing Pipeline

Rate limit rule 1) if the *User Data Access* attribute has been configured with the default *Direct Internet access* option. If *User Data Access* is set to another mode, *Rate Limit* flow entries must exist in *Flow Table 1* too (i.e., Rate limit rule 2) so that the attribute is processed. These flow entries forward matching packets to the meter entries that are configured to drop all packets when the throughput is over a given threshold.

IV. Validation Results

This section provides an assessment of the proposed GANSO framework through its implementation and testing over Mininet, as the emulator of an SDN network topology, and the ONOS controller, as the SDN controller managing the network. A set of experiments were conducted to validate not only the correct implementation of the supported GST attributes but also to showcase the automation of the configuration and instantiation of network slices over SDN infrastructures through the GANSO framework.

A. Experiment Network Topology

The validation presented in this section has been carried over an emulated network (as depicted in Figure 5), composed by three edge nodes (i.e, e_1, e_2, e_3), one core data center (i.e., c_1), two Open vSwitch switches (i.e., s_1 and s_2) that support OpenFlow and an ONOS controller. These are interconnected through virtual links configured with a bandwidth of 10 Mbps. Moreover, this network is connected to the Internet through a NAT which enables connectivity towards external hosts (i.e., Cloud DC). Virtual applications (i.e., vApps) are hosted on the edge nodes, core and cloud data centers, as depicted in Figure 5.

B. Experiment 1: User Data Access

The User Data Access attribute defines how the network slice should handle user data: locally only, private network, open to Internet (e.g, for far edge, local edge(s) or public cloud connections). As such, in the following experiment, *ping* command is used to test the different levels of connectivity allowed within the network slice. Specifically, $vApp_1$ (hosted on edge e_1) issues ICMP request messages to both $vApp_5$ (hosted on the cloud) and $vApp_2$ (hosted on edge e_2). The obtained results are presented in Figure 6.

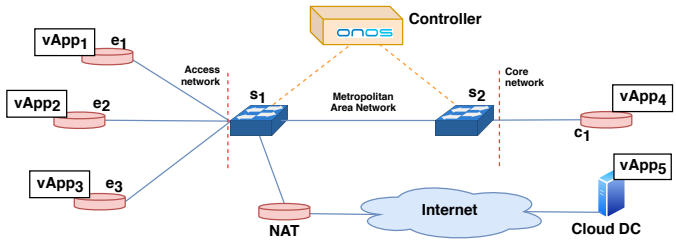

Fig. 5. Experiment topology

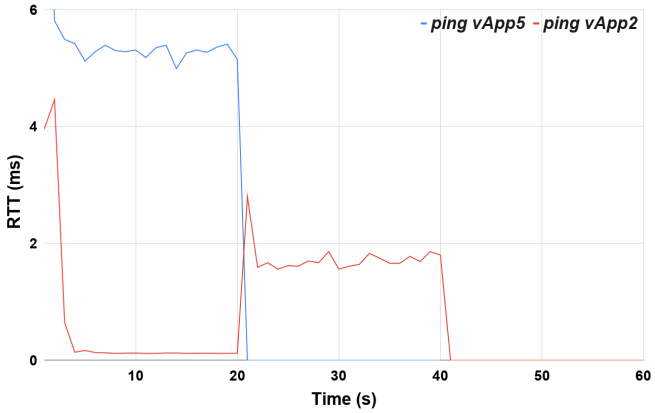

Fig. 6. User Data Access Experiment

Initially, the network slice corresponding to $vApp_1$ is configured with *User Data Access* attribute set to *Direct Internet access*. Both request messages are delivered to their destinations (i.e., $vApp_5$ and $vApp_2$), with the correspondent replies being successfully received by $vApp_1$ (average round-trip time of 5.29ms and 0.119ms, respectively).

At time 20 seconds, the network slice is reconfigured with the *User Data Access* attribute set to *Termination in the private network*. As expected, while connectivity to the $vApp_5$ was lost, connectivity to $vApp_2$ remained active but the round-trip time of the ICMP messages increased to 1.690ms. As additional flow entries are introduced in the switches, packets from $vApp_1$ need to be matched against these new entries which increases the processing time within the switch.

Finally, at time 40 seconds, the network slice is reconfigured again with the *User Data Access* attribute set to *Local traffic*. As this option sets the traffic as local only, the connectivity towards $vApp_2$ is also lost.

C. Experiment 2: Rate Limit

The *Rate Limit* attribute defines the amount of bandwidth allocated to a given network slice. As such, in this experiment *iperf* is used to generate constant traffic flows, while the

Rate Limit attribute is reconfigured on different network slices. Specifically, $vApp_1$, $vApp_2$ and $vApp_3$ act as *iperf* clients while $vApp_4$ acts as an *iperf* server. During the experiment, the following *Rate Limits* are configured on each network slice:

- **E_0 (t = 0s):** No *Rate Limit* set to any network slice.
- **E_1 (t = 19s):** $vApp_1$'s network slice set to 5.5 Mbps.
- **E_2 (t = 39s):** $vApp_2$'s network slice set to 1.5 Mbps.
- **E_3 (t = 59s):** $vApp_3$'s network slice configured with with *User Data Access* set to *Local traffic*.

The obtained results are presented in Figure 7.

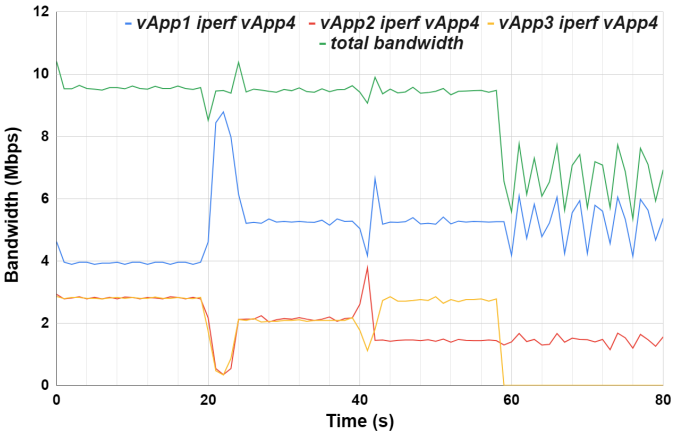

Fig. 7. Rate Limit Experiment

At E_0, $vApp_1$, $vApp_2$ and $vApp_3$ are able to transmit at 4 Mbps, 3 Mbps and 3 Mbps, respectively. As *Rate Limit* is not set to any network slice, this allocation is done by Mininet.

At E_1, the throughput of $vApp_1$ increases to 5.5 Mbps, as opposed to the other two vApps which see their throughput decrease to around 2.25 Mbps. This behaviour is due to the *Rate Limit* also acting as guaranteed throughput when there is enough bandwidth available to satisfy the request.

At E_2, the opposite behavior is seen, where the *Rate Limit* imposes the maximum throughput allowed to $vApp_2$. This releases bandwidth which is taken by $vApp_3$.

Moreover, upon reconfiguration of the *Rate Limit* in both E_1 and E_2 a brief surge in the throughput is observed after which it gets stabilised around the configured threshold.

Finally, upon E_3, the throughput of $vApp_3$ goes to 0 Mbps as it has only local connectivity, but the other two vApps do not take advantage of the released bandwidth due to the configured *Rate Limit*. Although the average throughput of the latter vApps remains near the set *Rate Limit*, it becomes less stable because packets from $vApp_3$ are only dropped at switch s_1, imposing an additional packet processing.

V. Conclusions

The instantiation of E2E services, which might comprise virtual applications distributed across multiple edge data centers (and centralized and/or public clouds), require network slicing to be applied into different network segments. Transport networks are one of these segments, as they interconnect the different components that compose a given service. This paper proposes the GST And Network Slice Operator (GANSO) framework, developed to automate the customisation and creation of network slices in the transport network, namely over SDN infrastructures. GST attributes are used to characterise a type of network slice and mapped towards the network resources and parameters required to its instantiation over the underlying network. *User Data Access* and *Rate Limit* attributes are implemented in a proof-of-concept prototype and validated through a set of experiments that focused on the instantiation of network slices, over the transport network, for different virtual applications.

Future work will focus on the support of the remaining GST attributes and a fully network slice lifecycle management. These extensions, together enhancements on traffic prediction, will enable an intelligent and automated transport slice management, so that network slices can be dynamically reconfigured to make an efficient use and allocation of the available resources without degradation of the deployed services.

Acknowledgments

This work has been (partially) funded by H2020 EU/TW 5G-DIVE (Grant 859881) and H2020 5Growth (Grant 856709). It has been also funded by the Spanish State Research Agency (TRUE5G project, PID2019-108713RB-C52PID2019-108713RB-C52 / AEI / 10.13039/501100011033)

References

[1] 3GPP, "Management and orchestration; Provisioning," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 28.531, Mar. 2020, version 16.5.0.

[2] GSMA, "Generic network Slice Template, Version 3.0," May 2020.

[3] L. M. Contreras, S. Homma, and J. A. Ordonez-Lucena, "Considerations for defining a Transport Slice NBI," Internet Engineering Task Force, Internet-Draft draft-contreras-teas-slice-nbi-02, Jul 2020.

[4] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[5] A. de la Oliva, X. Li, X. Costa-Perez, C. J. Bernardos, P. Bertin, P. Iovanna, T. Deiss, J. Mangues, A. Mourad, C. Casetti, J. E. Gonzalez, and A. Azcorra, "5G-TRANSFORMER: Slicing and Orchestrating Transport Networks for Industry Verticals," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 78–84, 2018.

[6] P. K. Chartsias, A. Amiras, I. Plevrakis, I. Samaras, K. Katsaros, D. Kritharidis, E. Trouva, I. Angelopoulos, A. Kourtis, M. S. Siddiqui, A. Viñes, and E. Escalona, "SDN/NFV-based end to end network slicing for 5G multi-tenant networks," in *2017 European Conference on Networks and Communications (EuCNC)*, 2017, pp. 1–5.

[7] I. Afolabi, M. Bagaa, T. Taleb, and H. Flinck, "End-to-end network slicing enabled through network function virtualization," in *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2017, pp. 30–35.

[8] D. Giatsios, K. Choumas, P. Flegkas, T. Korakis, and D. Camps-Mur, "SDN implementation of slicing and fast failover in 5G transport networks," in *2017 European Conference on Networks and Communications (EuCNC)*, 2017, pp. 1–6.

[9] D. Giatsios, K. Choumas, P. Flegkas, T. Korakis, J. J. A. Cruelles, and D. C. Mur, "Design and Evaluation of a Hierarchical SDN Control Plane for 5G Transport Networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[10] F. Meneses, M. Fernandes, D. Corujo, and R. L. Aguiar, "SliMANO: An Expandable Framework for the Management and Orchestration of End-to-end Network Slices," in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, 2019, pp. 1–6.

[11] Open Networking Foundation (ONF), "OpenFlow Switch Specification, Version 1.5.1," Mar. 2015.