Network Orchestrator for QoS-enabled Service Function Chaining in reliable NFV/SDN infrastructure

M. Gharbaoui, S. Fichera, P. Castoldi
Scuola Superiore Sant'Anna,
Pisa, Italy
{m.gharbaoui, s.fichera, castoldi}@santannapisa.it

B. Martini
CNIT
Pisa, Italy
barbara.martini@cnit.it

***Abstract*— The promise of SDN/NFV to offer enhanced functionalities to the service provider networks is now tangible as both technologies have started being steadily investigated and experimentally demonstrated. However, a number of research challenges have not been fully addressed yet. More specifically, a limited interest has been devoted on how to assure reliability and QoS performance while data traverse service chain paths. In this work, we present an SDN orchestrator that periodically evaluate the availability status of the network and, accordingly, adapt service chain paths to recover from congestion events and to preserve QoS performance thus avoiding SLA violations. A prototype is presented along with a performance evaluation of the orchestrator in terms of improved data transfer capabilities against the cost of path adaptations and the controller overhead.**

Keywords—SDN, NFV, orchestration, ONOS, QoS, reliability

I. INTRODUCTION

In the context of 5G, users expect services are provisioned anywhere, anytime and through any access technology. In order to address such demanding user expectations, dynamic resource provisioniong mechanisms need to be effectively put in place to assure service pervasiveness, to minimize network/data delivery delays and to maximize service availability [1][2]. In this scenario, Network Function Virtualization (NFV) and Software-Defined Networking (SDN) emerged as the breaking technologies for the telecommunication industry since they enable on-demand provisioning mechanisms and scalability solutions typical of cloud computing in Telco deployments [3][4]. On the one hand, NFV allows middlebox services (e.g., firewall, NAT, DPI) to be provisioned as virtual appliances in a cloud and to be handled as independent service components (i.e., service functions) that can be flexibly composed to provide dynamically established sequences of data processing capabilities (i.e., dynamic service function chains) and, thus, to differentiate packet treatment to data flows based on service requirements (e.g., latency-sensitive multimedia data flows enforced to pass through traffic accelerators). In NFV deployments, SDN effectively provides programming abstractions that can be exploited for the dynamic steering of data traffic along the path of service function chains (i.e., service chain paths) [5][6].

The promise of SDN/NFV to offer enhanced functionalities to the service provider networks is now tangible as both technologies have started being steadily investigated and experimentally demonstrated. Indeed, in [7] a proof-of-concept is described that experimentally demonstrates the feasibility of a convergent SDN/NFV deployment for self-adapting service chaining aiming at dynamically addressing customers' SLA. Several works in the literature show that the adoption of SDN/NFV, possibly in combination with an orchestration layer, provides increasing flexibility and scalability to dynamic service chaining [8]. Such works either provide SDN-based solutions for steering packets across service functions [9][10][11] or address orchestration of cloud/NFV services over SDN networks while meeting service requirements [12][13][14][28]. Furthermore, few research works have tackled QoS support in SDN frameworks. [23] enriched OpenFlow (OF) protocol with some QoS capabilities while [24] proposed to find the best path according to predefined constraints such as end-to-end delay. Several solutions have also been proposed to support the dynamic reconfiguration of service graphs, compose VNF chains and provide middlebox traffic steering [25][26][27]. However, a number of research challenges have not been addressed yet. In particular, most of the approaches in the literature related to service function chaining neither involve network QoS, nor address load balancing functionality, nor consider the availability status of the network (e.g., link capacity usage) that might affect the actual QoS performance (e.g., throughput) experienced by service chain data. Indeed, QoS degradations may be possible due to concurrent usage of network resources (i.e., switches, links) by admitted service chains that inject many traffic flows in the network. Definitely, traffic-engineered service chaining solutions are required in SDN/NFV infrastructures to address reliability of service chain paths and to provide QoS assurance to service data [8].

As an approach to address the aforementioned challenges, in this work we present an SDN orchestrator aiming to achieve reliable and QoS-enabled service chain paths in SDN/NFV infrastructures. More specifically, the proposed SDN orchestrator performs an orchestration process at the network infrastructure level by collecting monitoring data and, in case of congestion events or QoS degradations, activates recovery mechanisms through

978-1-5090-6008-5/17/$31.00 ©2017 IEEE

redirection of (part of) service chain paths. This orchestration process adaptively regulates the use of network resources (i.e., switch and links) and, accordingly, dynamically arranges service chain paths to maximize the available data throughput. Moreover, since the balanced usage of resources does not imply to strictly address QoS requirements, the orchestration also detects deviations of actual data flow throughput from the SLA and puts in place service chain path redirections to preserve QoS performance. Previous works of authors addressed the architectural aspects of a NVF/SDN infrastructure with orchestration capabilities aligned with SOA principles [15] and in combination with context-awareness features [16][9]. Moreover, in [17] we present a preliminary implementation of an SDN orchestrator using a Floodlight controller for adaptive service function path provision dealing with only congestions at the network node level without addressing QoS-awareness as this work does.

A prototype of the SDN orchestrator is presented which has been developed as an application operating on top of an ONOS controller and realizing a set of orchestration policies aiming at offering different level of guarantees (i.e., service availability and/or QoS assurance). The evaluation of the SDN orchestrator is provided in terms of increased amount of transmitted data that is achieved with the use of the proposed SDN orchestrator. The cost of the proposed orchestration solution is also shown in terms of time spent for path redirections and in terms of controller overhead.

II. SDN Orchestrator Design

This section describes the main building blocks of the SDN orchestrator which allows for a high-available and QoS-enabled provisioning of service chain paths, thus offering a reliable NFV/SDN infrastructure.

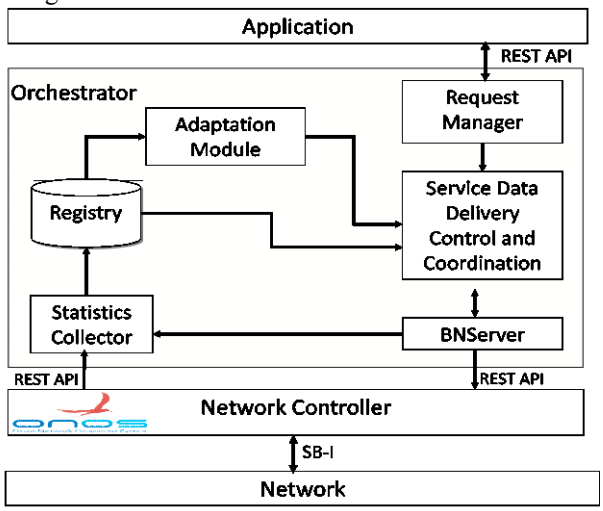

Fig. 1. SDN Orchestrator: Building Blocks

The SDN orchestrator mainly consists in an orchestration logic that is developed as an application running on top of the ONOS platform [18] to provision programmable and adaptive service chain paths. The SDN orchestrator leverages on the controller capabilities to monitor the status of the network and accordingly arrange service chain paths while preserving predetermined QoS requirements. ONOS has been chosen due to its NorthBound Interface (NB-I) which offers a simple way to directly interact with the controller through a REST API. This way, in principle the SDN orchestrator can flexibly operate over multiple ONOS controllers thereby addressing scalability. Moreover, the SDN orchestrator exposes at NorthBound a RESTful interface which enables the upper layer applications to directly ask for service chain paths using an intent-based semantics, i.e., through prescriptive rather then descriptive directives requiring technology-specific low-level details. The provisioning details are then handled by the SDN orchestrator which maps them into specific ONOS-compliant queries that will then be translated into appropriate flow entries. Upper layer applications could be either NFV orchestrators complying to ETSI MANO directives [19] or orchestration functions in Service Delivery Platforms (e.g., according to the IEEE NGSON specifications [16]). In [17] we specified a REST API for the SDN orchestrator according to the NEtwork MOdeling (NEMO) Language specifications [20].

Fig. 1 presents the main software building blocks of the SDN orchestrator and the interactions among them. The main components are the following:

- *Request Manager* which exposes a REST API to Applications. It handles requests for the setup of service chain paths between specified source and destination endpoints while ensuring that a specified sequence of Virtual Network Functions (VNFs) is traversed by the traffic flow. Both endpoints and VNFs are specified as IP addresses. The request involves composite paths consisting in a sequence of path segments to be individually provisioned in a coordinated way by leveraging the *Service Data Delivery Control and Coordination* component.
- *Service Data Delivery Control and Coordination:* decomposes the requested service chain path into an ordered sequence of path segments, coordinates the provisioning actions of each path segment through proper mapping into ONOS-compliant queries while leveraging the *BNServer*.
- *Adaptation Module:* is in charge of triggering the *Service Data Delivery Control and Coordination* component for the re-provisioning of (part of) service chain paths throughout different set of switches (i.e., redirection) as soon as a degradation event (e.g., congestions of one or more switches, decrease of data throughput under a specified threshold) is detected using operational data stored in the Registry. The decision for redirections is triggered according to specified orchestration policies aiming at offering different level of guarantees (i.e., service availability and/or QoS assurance).
- *Statistics Collector:* obtains the operational status data of service chain paths after processing the set of collected OF statistics retrieved leveraging the *BNServer*. The operational status data are stored in the *Registry* and consist in (i) throughput on per-chain (i.e., per-flow) basis, and (ii) switch throughput computed from per-port byte counters.
- *BNServer:* is in charge of interacting with the base controller functions through appropriate APIs. This component leverages the ONOS controller functions to handle the low-level directives (i.e., OF messages) for installing the forwarding rules throughout the network and collect traffic statistics at switches.

- *Registry:* contains descriptive and operational data on network nodes and service chain paths. More specifically, it contains (i) a list of the available VNFs instances with related descriptive information (i.e., type, network location in terms of IP prefix, Data Path ID and port of the switch they are connected to); (ii) descriptive information about the service chain paths and each comprised segment established throughout the network (e.g., IP addresses of end-points, identifier and port number of intermediate switches); (iii) operational information about the load (i.e., throughput) of switches and the actual QoS performance (i.e., data throughput) of service chain data flows evaluated using monitoring data collected by the *Statistics Collector*.

III. Orchestration Policies

When a request arrives for the setup of a service chain path, the *Request Manager* aims at finding a set of switches capable of satisfying the request requirements (i.e., source and destination endpoints, VNFs to be traversed). Once the service chain path setup is performed, the *Statistics Collector* engine processes the retrieved OF statistics to obtain the throughput information on per-flow and on per-switch basis. Such mechanism allows to adapt the active service chain paths while recovering from service degradations due to switch congestions or QoS violations. In fact, a high throughput will cause the buffer to overflow causing a loss of packets not only for the affected flow but also for the other flows traversing the switch. To this purpose, we have considered four different orchestration policies that aim at offering different level of guarantees (i.e., service availability and/or QoS assurance).

A. Network-aware Flow Redirection (NFR)

The *Adaptation Module* periodically collects from the *Registry* the throughput of all the switches and the list of currently active paths. For each switch that is connected to a cloud platform deploying a VNF, if the throughput is higher than a given threshold, the switch is considered as overloaded. Every active service chain path traversing an overloaded switch is then deleted and the orchestrator tries to redirect it throughout other switches, if available. This policy assures a regulated usage of switches thereby recovering from congestions and maximizing the available data throughput at the nodes [17].

B. QoS-aware Flow Redirection (QFR)

The *Adaptation Module* periodically receives from the *Registry* the throughput performance of the service chain data flows. If the throughput value is detected to be below the threshold specified in the SLA, the flow is deleted and redirected along another service path. This policy assures that the QoS performance are preserved in line with SLA during the entire duration of the established service chain.

C. Network and QoS-aware Flow Redirection (NFR+QFR)

This policy combines the two previous ones. Both the switches and flows are considered. In case of either switch overload or flow throughput under a given threshold, the flow path redirection is performed.

D. No policy

Once a request arrives, the *Adaptation Module* selects the sequence of the switches connected to the cloud platforms randomly. The only adopted constraint is that a switch can be traversed only once by a given flow. After the path provisioning, no performance checks are carried out, neither redirections.

IV. Performance Evaluation

In order to evaluate the performance of our proposed orchestration policies we use Mininet [21], an emulated environment that allows us to deploy the Abilene network topology [22]. The topology is composed of 11 OF switches and 13 end hosts. A subset of the switches is connected to emulated cloud platforms that deploy the VNFs while the remaining switches are simply transit switches. For the selection of the appropriate switch, we exploit the Dijkstra's shortest-path algorithm to select the closest instance for each service specified in the chain. In this algorithm, the switches are associated with no weight. Moreover, all the end hosts in the topology are randomly chosen to behave as a source/destination of the service delivery path requests. It is assumed that VNFs performing the same type of network functions are present in each cloud platform. Mininet connects to the ONOS controller which communicates with our orchestrator through a REST API. The interaction between the orchestrator and ONOS englobes the research for routes between two given endpoints, the setup/delete of the data delivery paths and the statistics collection. The throughput values of switches and flows are derived from periodic polling of monitoring data from switches and from established flows. Even though this approach is not scalable, it has been adopted in this phase of orchestration deployment where the feasibility of the solution is under investigation.

TABLE I. Parameters

Parameters	Value
Number of switches	11
Number of end hosts	13
Number of requests per iteration	100
Number of iterations	5
Requests inter-arrival time	30s
Threshold overloaded switches	2 Gb/s
Threshold QoS	500 Mb/s
ΔT – Flows statistics	120s
ΔT – Switches statistics	320s

We compare the performance of our orchestration policies with respect to the baseline (no policy case). To this purpose, we send 100 requests generated randomly according to a Poisson process. Each request has an exponentially distributed duration referred to as service time in the following. The source and destination end hosts are uniformly distributed. A summary of set-up parameters is shown in Table I.

1) Average number of transmitted bytes

For every data delivery path, once established, Mininet uses the *iperf* tool to send traffic (10 Mb/s) for the whole duration of the service time. The statistics collector periodically asks for the current number of transmitted bytes of every active flow in the network. The plotted values are

relative to the data collected at the port of the last traversed switch. The average number of transmitted bytes is calculated as the average of all the transmitted bytes collected for all the active flows.

Fig. 2 plots the average number of transmitted bytes as a function of the average service time. As expected, results show that by increasing the service duration, the average number of transmitted bytes increases for all the policies. By comparing the behavior of the policies, we notice that both NFR and QFR improve the performance with respect to the baseline. This is due to the fact that by detecting the congested switches or the compromized flows, respectively, the orchestrator through path redirections avoids the degradation of the performance and then increases the average number of transmitted bytes. Moreover, by increasing the load, NFR starts to behave better than QFR. In fact, collecting per-switch statistics allows the orchestrator, in case of a congested switch, to improve in one operation the performance of all the flows traversing that switch. For this reason, in the NFR+QFR, the combination of the two policies gives almost the same results as NFR. Finally, at high loads, QFR and NFR present almost the same results since almost all the switches become congested which prevents the orchestrator from redirecting the traffic.

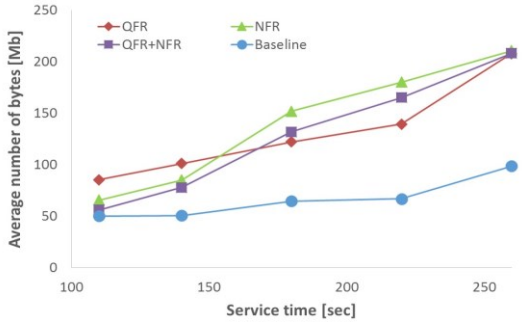

Fig. 2. Average number of transmitted bytes

2) Redirection time:

The redirection time corresponds to the time spent in redirecting the flows in case of the presence of one or more overloaded switches and/or the requested QoS is not met for a flow. It includes the time necessary for deleting the flow(s), looking for a new route(s) and setting up the flow entries in the switches along the new path.

Fig. 3 plots the average redirection time for the three policies with respect to the baseline where this time is evidently null. Results show that QFR presents a lower redirection time with respect to NFR. In fact, in NFR the redirection time corresponds to the deletion of all the flows traversing the overloaded switch, the research of new routes, and the setup of flow entries for all of them, while in QFR, the redirection operation is performed only for one single flow. In the NFR+QFR case, the combination of the two policies further increases the redirection time, especially at high loads where the orchestrator spends more time in trying to find available switches for redirecting the traffic.

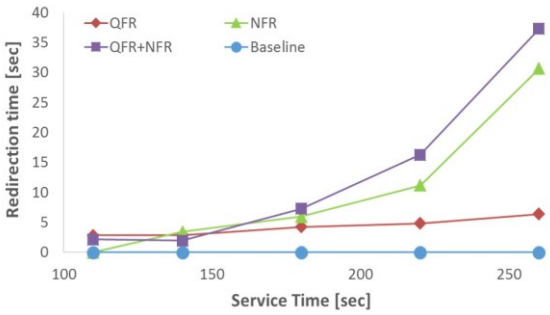

Fig. 3. Average redirection time

3) Controller overhead

The controller overhead corresponds to the cost generated by the redirection operations on the controller in terms of number of exchanged messages. When this number increases, the control channel might be congested and the controller might start discarding packets which lowers the performance.

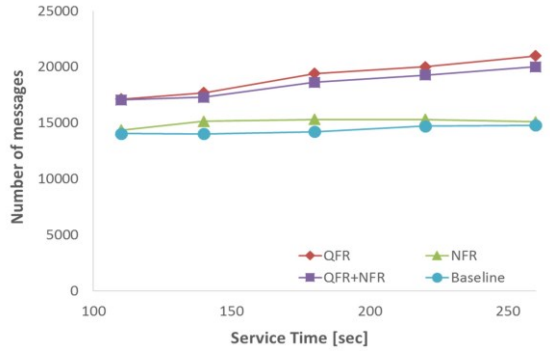

Fig. 4. Controller overhead

Fig. 4 plots the overhead as a function of the service time for the three policies. Results show that QFR presents the highest number of exchanged messages since the interaction with the controller is performed for every flow (e.g., delete, statistics collection, setup of flow entries) to detect QoS degradations. In the NFR case, the number of exchanged messages is much lower (around 25%) since it is possible to aggregate many operations within one interaction if they belong to the same device (e.g., delete all the flows of a switch, get the statistics for one switch). This number is almost equal to the number of exchanged messages in the baseline case where no policies are applied. In fact, when the network is congested, more switches become overloaded which makes the redirection operation harder and even impossible in some cases. When the deletion of the flows and their redirection is not performed and the interaction between the orchestrator and the controller is limited to the statistics collection. Finally, in the NFR+QFR case, the combination of the two policies further increases the overhead. However, the number of messages in this case is lower than the sum of the number of messages in the separate two policies. In fact, in the QFR+NFR policy, since the statistics intervals of the two policies are different, deleting all the flows of an overloaded switch and redirecting them will lead to respecting the QoS

and then decreasing the interaction with the controller. On the other hand, redirecting one compromised flow might minimize the overloaded switches situations which also alleviates the overhead.

V. Conclusion

This paper presented an SDN orchestrator aiming at the adaptive provisioning of high-available and QoS-assured service chain paths in SDN/NFV infrastructures. While periodically evaluating the availability status (i.e., load) of switches, the SDN orchestrator is able to address a regulated usage of those to achieve reliable service chain paths. Moreover, by also evaluating the actual QoS performance of data flows, the SDN orchestrator is able to detect possible SLA violations and put in place recovery actions to preserve QoS performance according to SLAs. The performance of the proposed orchestration policies has been evaluated in terms of improved data transfer capabilities against the cost of path adaptations and of controller overhead. Results show that the SDN orchestrator improve the reliability of the SDN/NFV infrastructure through higher available service chain paths while also addressing QoS requirements. More specifically, the NFR orchestration policy allows to address a regulated usage of resources leading to higher performance in terms of transmitted data with minimized burden to the controller but at the cost of higher required time for path redirections due to the aggregated handling of a bundle of flows as a switch get overloaded. However, the increment of redirection time is significant only at higher service time when the whole network is congested. On the other hand, the QFR policy achieves the QoS assurance of flows with slightly lower gain in terms of amount of transmitted data and minimized time required for redirections since the actions are carried out on a single. In flow. In contrast, the burden to the controller is higher since the interaction with the controller is performed on per-flow basis. In the NFR+QFR policy, the gain in terms of transmitted data is significant and comparable with the NFR case. This policy also assure QoS performance in line with SLA throughout the duration of the service chains. However, this comes at the cost of both increased redirection time and number of messages exchanged with the controller.

As future works we plan to investigate how to consider various traffic classes with different QoS requirements and how to improve the scalability of the monitoring through spatial or temporal correlations of a subset of data (i.e., polling from a subset of switches, estimations on a subset of temporal samples).

Acknowledgment

This work was partially supported by the ICT14 H2020 5G Exchange (5GEx) innovation project (671636) and by the LASH-5G project approved within the 1st Call of the Fed4Fire+ project.

References

[1] J. Sánchez, et al. "Softwarized 5G networks resiliency with self-healing," in Proc. of 5GU, Akaslompolo, 2014.

[2] B. Martini, et al, “Latency-aware composition of virtual functions in 5G,” Proc. of NetSoft, London, 2015.

[3] J. Soares et al., "Toward a telco cloud environment for service functions," in IEEE Communications Magazine, Feb. 2015.

[4] Q. Duan, et al. "A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing," IEEE TNSM, December 2012.

[5] B. Han, et al. “Network function virtualization: Challenges and opportunities for innovations,” IEEE *Communications Magazine, 2*015.

[6] B. Nunes, et al. “A survey of software-defined networking: Past, present, and future of programmable networks,” IEEE *Communications Surveys Tutorials, Third 2014.*

[7] F. Callegati, et al, "SDN for dynamic NFV deployment," in IEEE Communications Magazine, October 2016.

[8] A. M. Medhat; et al. "Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges," in IEEE Communications Magazine , vol.PP, no.99, pp.2-9, October 2016

[9] B. Martini, et al. "SDN controller for context-aware data delivery in dynamic service chaining,", Proc. of NetSoft, London, April 2015

[10] Ying Zhang et al., "StEERING: A software-defined networking for inline service chaining," 2013 21st IEEE International Conference on Network Protocols (ICNP), Goettingen, 2013, pp. 1-10.

[11] Z. Qazi et al., “Practical and Incremental Convergence between SDN and Middleboxes”, Open Network Summit, Santa Clara, CA, 2013

[12] A. Csoma et al., “ESCAPE: Extensible Service Chain Prototyping Environment using Mininet, Click, Netconf and POX,”, ACM SIGCOMM Computer Commu. Rev., vol. 44, no. 4,2015, pp 125-26

[13] K. Giotis, et al., "Policy-based orchestration of NFV services in Software-Defined Networks," Proc. of NetSoft, London, 2015

[14] F. Callegati, et al., "Dynamic chaining of Virtual Network Functions in cloud-based edge networks," Proc. of NetSoft, London, 2015

[15] B. Martini, et al, “A Service-Oriented Approach for Dynamic Chaining of Virtual Network Functions over Multi-Provider Software-Defined Networks”,. Future Internet 2016, 8, 24.

[16] F. Paganelli, et al., “Context-aware service composition and delivery in NGSONs over SDN,” IEEE Communications Magazine, Aug. 2014.

[17] A. A. Mohammed, et al., "SDN controller for network-aware adaptive orchestration in dynamic service chaining," Proc. of NetSoft, 2016.

[18] http://onosproject.org/

[19] ETSI, “Etsi gs nfv-man 001, network functions virtualisation; management and orchestration (nfv),” Tech. Rep.

[20] Y. Xia, et al.,, “Nemo (network modeling) language,” Working Draft, IETF Secretariat, Internet-Draft draft-xiasdnrg-nemo-language-03, October 2015.

[21] http://mininet.org/

[22] “Abilene network,” https://en.wikipedia.org/wiki/Abilene Network.

[23] [OpenFlow Consortium, “OpenFlow switch specification, version 1.0.0,” Dec. 2009. [Online]. Available: http://www.openflow.org/documents/ openflow-spec-v1.1.0.pdf

[24] K. Jeong, et al., “Qos-aware network operating system for software defined networking with generalized openflows.” in Network Operations and Management Symposium (NOMS). IEEE/IFIP, 2012.

[25] Eder J. Scheid, et al., “Policy-based dynamic service chaining in Network Functions Virtualization,” IEEE Symposium on Computers and Communication (ISCC), 2016.

[26] Z. A. Qazi, et al., “SIMPLE-fying Middlebox Policy Enforcement Using SDN,” in Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, ser. SIGCOMM ’13. New York, NY, USA: ACM, 2013.

[27] A. Csoma, et al., “ESCAPE: Extensible Service Chain Prototyping Environment Using Mininet, Click, NETCONF and POX,” in Proceedings of the ACM Conference on SIGCOMM. ACM, 2014.

[28] W. Cerroni, et al., “Cross-layer resource orchestration for cloud service delivery: A seamless SDN approach,” Computer Networks, 2015.