Towards a Dynamic SDN Virtualization Layer: Control Path Migration Protocol

Arsany Basta, Andreas Blenk, Hassib Belhaj Hassine, Wolfgang Kellerer
Chair for Communication Networks
Department of Electrical and Computer Engineering
Technische Universität München, Germany
Email: {arsany.basta, andreas.blenk, hassib.belhaj, wolfgang.kellerer}@tum.de

***Abstract*—Virtualization of software defined networks enables tenants to bring their own controller and manage their virtual resources with the full programmability provided by Software Defined Networking (SDN). Distributed SDN hypervisors are proposed to provide an efficient platform for the virtualization of physical SDN networks. They address the issue of scalability that a centralized hypervisor could suffer from. As virtualization provides the possibility to change virtual SDN networks on run time, a hypervisor layer needs efficient mechanisms to dynamically adapt to the changing requirements. Existing proposals provide only a static configuration setup for their distribution of the hypervisor instances. However, in order to satisfy the dynamics of virtual SDN networks, management protocols are needed to support dynamic changes. In this paper, we propose a control path migration protocol for distributed hypervisors. Such protocol is needed to support the dynamic adaptation of the virtualization layer. Our protocol is providing the missing procedure that allows a dynamic change of control connections between virtual SDN networks and the tenants' controllers, respectively. We provide a proof of concept implementation for our proposal. Through measurements in a real testbed setup, we show that our protocol is efficient in terms of control latency overhead and provides transparency to the controllers of the virtual SDN networks.**

I. Introduction

Network Virtualization (NV) is a fundamental ingredient of future network architectures. With NV, the physical network resources, e.g., switches, are sliced into virtual networks (slices). For example, as proposed in [1] and [2] for the future 5G mobile network, slicing of the physical network resources allows the management of virtual network resources individually according to the demands of the network traffic. Furthermore, virtual networks provide the capability to cope with the dynamics in Network Function Virtualization (NFV) use cases, where virtual slices can dynamically interconnect virtual network functions according to the functions' demands.

Software Defined Networking (SDN) is simplifying the way to change the behavior of a network as a whole, i.e., to change the traffic steering dynamically based on a global state. Accordingly, the virtualization of SDN networks is one fundamental feature that can improve the performance of communication networks. While NV provides network resources via slices, SDN can improve the network performance through advanced control mechanisms within a virtual network slice.

An established concept for virtualization of SDN networks is to introduce a hypervisor layer, or shortly a hypervisor. The hypervisor operates as an intermediate layer between SDN controllers and their particular virtual SDN network. The hypervisor provides the functionality that is needed to slice the physical SDN network into multiple virtual SDN networks. It manages and controls the access of the individual SDN controllers to their virtual SDN networks, e.g., consisting of multiple virtual SDN switches. It translates network messages between SDN controllers and their slices, and isolates the data plane and control plane traffic of the slices from each other in order to avoid cross effects between them.

Different implementation concepts for the hypervisor have been proposed. In favor of achieving a scalable hypervisor, there have been several implementation concepts that rely on a distributed architecture, e.g., FlowN [3]. Such a distributed concept is needed, e.g., if the SDN controllers of the slices are also distributed among a network, as it might be the case for wide area networks or for distributed cloud networks. Furthermore, a distributed hypervisor can adapt more efficiently to changing requirements, e.g., in case slices are added and removed or in case demands of slices are changing over time. Changing requirements may also lead to varying load on the distributed components. In order to adapt to changing requirements, a distributed hypervisor requires mechanisms that allow a dynamic and efficient adaptation of the hypervisor virtualization layer.

Distributed hypervisors provide the capability to spread hypervisor instances, which are hosting the functions realizing the virtualization, among the network. The instances connect the SDN controllers with their slices, i.e., they establish control connections between SDN controllers and their virtual SDN networks. As control connections are associated with load, efficient mechanisms to change the load distribution among the instances are necessary in order to provide an efficient virtualization. One possibility to adapt the load could be the migration of control connections between the instances. Besides load balancing, energy efficiency could be another operation target. As less hypervisor instances may provide a better energy efficiency, instances need to be turned on and off dynamically. Again, active control connections would need to be migrated. Although these examples are illustrating an important goal of hypervisor implementations, existing distributed SDN hypervisors do not provide any mechanisms that support the dynamic operation of the distributed instances and their control connections.

In this paper, we propose a control path migration protocol for distributed SDN hypervisors, which are composed of multiple hypervisor instances. Such protocol is needed to allow

978-3-901882-77-7 © 2015 IFIP

distributed hypervisors to dynamically change virtual SDN networks. With our protocol, control connections between tenants' controllers and virtual SDN networks can be changed dynamically. We define a control connection as the connection from the virtual SDN network, i.e., virtual SDN switch, to the SDN controller through a hypervisor instance. The protocol supports the migration of OpenFlow (OF) control connections. In detail, the control connections of one hypervisor instance can be migrated to another hypervisor instance with negligible control latency overhead at run time. In addition to the migration protocol, we also propose an architecture for a dynamic SDN virtualization layer. We implement our protocol for an SDN network, that uses OpenFlow to demonstrate a proof of concept. The measurement results are quantified in terms of control plane performance overhead. To summarize, our contributions include the following:

- An architecture that provides a dynamic SDN virtualization layer
- A protocol for control path migration among distributed SDN hypervisors
- A proof of concept implementation of our proposed OpenFlow-based protocol

The remainder of this paper is structured as follows. In Section II, we provide a review for the existing state of the art for distributed SDN hypervisors and protocols for SDN control management. In Section III, we introduce the considered architecture for a dynamic distributed SDN hypervisor layer. Section IV provides a detailed explanation of the proposed protocol that provides an online management for virtual SDN control slices. In Section V, we show a first prototype implementation of the protocol and we present measurements from a real testbed. Finally, we draw conclusion in Section VI.

II. Related Work

In the context of SDN virtualization, there are several existing hypervisors that propose a distributed hypervisor layer, e.g., Carrier-grade [4], FlowN [3] and Auto-slice [5]. There are also hypervisors that support the operation of both a centralized and a distributed hypervisor layer, e.g., FlowVisor [6], OpenVirteX [7] and HyperFlex [8]. However, the control paths between the SDN controllers, hypervisor instances and SDN switches are often presented as a static configuration that has to be decided at the initialization of the network. The dynamic migration of the control path has not been addressed by existing SDN virtualization solutions.

For non-virtual SDN networks, there are several architectures that propose distributed SDN controllers. In [9], ONOS is presented. ONOS is a distributed SDN operating system that follows the steps of distributed controllers such as ONIX [10]. Their focus is on obtaining and maintaining network abstractions in a distributed SDN controllers layer. The dynamic assignment of SDN switches to controllers has not been addressed. Another distributed SDN controllers solution has been introduced in Pratyaastha [11]. It considers the applications state to establish the control path between SDN switches and distributed controllers, however, no dynamic change in the control path is considered.

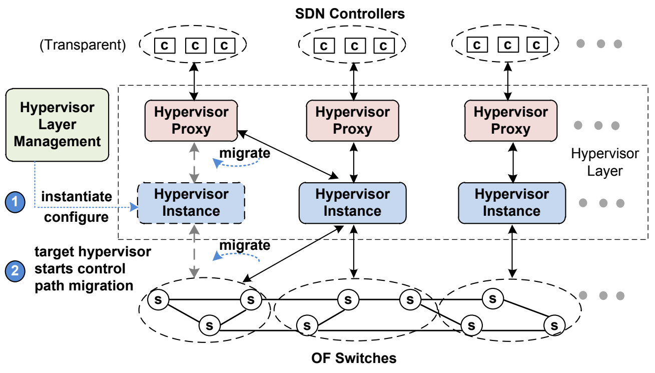

Fig. 1. Distributed SDN hypervisor layer, which includes hypervisor instances, management and proxies.

There have been few recent attempts to address this limitation in non-virtual SDN networks, e.g., ElastiCon [12]. ElastiCon introduces a mechanism to change the active control connections from one controller to another, dynamically on run time. The mechanism defines the message exchange required to achieve the control path migration.

However, the existing procedures cannot be applied to virtual SDN networks directly. In virtual SDN networks, there is an add-on complexity as the control connection spans hypervisor instances as well. Thus, new protocols are needed in the context of virtual SDN networks to enable the dynamic migration of the whole control connection, i.e controller to hypervisor as well as hypervisor to switch connections. The coordination between both migrations needs to be additionally defined and evaluated. In this work, we introduce an OF-based protocol that enables the migration of active control connections for virtual SDN networks. Our aim is to enable a dynamic and scalable SDN virtualization layer.

III. Dynamic SDN Virtualization Layer

In this section, we outline a virtualization architecture that provides a dynamic distributed SDN hypervisor layer. We consider a hypervisor layer that dynamically adapts to changes in the network state. The changes in network state can be in terms of, e.g., over utilization for the resources of a hypervisor instance [8]. It can be also in terms of new requirements from the virtual SDN networks, e.g., requested control latency. The adaption is realized by instantiating or terminating hypervisor instances and using our proposed protocol to achieve the migration of the active control connections. The hypervisor layer is comprised of three main entities: 1) hypervisor instances 2) hypervisor proxies and 3) hypervisor layer management, as illustrated in Figure 1.

The hypervisor instances are the controllers of the physical SDN network, which can be considered as the main workforce in the hypervisor layer. They implement the functions needed for SDN virtualization, e.g., topology abstraction, policy translation, control and data paths isolation [13]. The hypervisor functions are extended in our work to include a migration function that executes the proposed protocol.

We aim at providing a dynamic virtualization layer that adapts its structure without interrupting or even notifying the tenants' SDN controllers, i.e., changes in the hypervisor layer are transparent to the virtual SDN slices. For this purpose, we add hypervisor proxies on top of the hypervisor instances. The hypervisor proxies would act as an interface between the hypervisor layer and the SDN controllers. In case direct control connections are established between the hypervisor instances and the SDN controllers, a change in the control path would require the SDN controller to establish an OF control connection with the target hypervisor and to close the existing OF control connection towards the initial instance. This would mean that the dynamic scaling of the hypervisor layer would not be transparent anymore to the tenants' controllers. Moreover, it might lead in many cases to control path performance issues depending on the controller's supported OF version and implementation. For instance, if the controller does not support OF auxiliary connections that are introduced in OF v1.3.0 [14], a control connection interruption would occur during the hypervisor migration. Thus, we place a proxy that maintains the control connection with the SDN controllers and has a connection to the running hypervisor instances. The proxy contains forwarding policies which specify how to forward the control messages from the SDN controllers to the designated hypervisor, and vice versa.

Finally, a management entity is needed to make the decision of adding and removing hypervisor instances based on the network state information that it collects from the network, i.e., performance of the switches and running hypervisor instances, or based on the requirements that it receives from the SDN controllers. The decision includes the location of a new hypervisor instance that has to be added or the identifier of the instance that should be terminated. Once a decision is made, the management starts the instantiation or termination of the hypervisor instances. The management is also required to install the configuration at the target hypervisor, which contains the addresses of its assigned proxies, controllers and switches. The target hypervisor uses this configuration to trigger and start the migration protocol, i.e., establishing the connections to the configured proxy and switches.

Figure 1 illustrates an example where a decision has been made by the management entity to add a hypervisor instance to the SDN virtualization layer. The additional hypervisor instance is instantiated through the management entity. The instantiated hypervisor instance uses the proposed protocol to take over a set of OF switches, i.e., physical network domain, and a set of SDN controllers. In other words, the instantiated hypervisor triggers the migration of the control path, i.e., connections to designated proxy and switches.

IV. Control Path Migration Protocol

In this section, we present our proposed protocol to enable a dynamic migration of the control path for virtual SDN networks, i.e., between SDN controllers, hypervisors and OF switches. The protocol is composed of three phases: 1) initialization, 2) controller path migration and 3) switch path migration. The procedure is illustrated in Figure 2.

A. Initialization Phase

In non-virtual SDN networks, the control connection is defined by the OF specification to be initialized by the OF switch. The controller IP address is configured on each OF

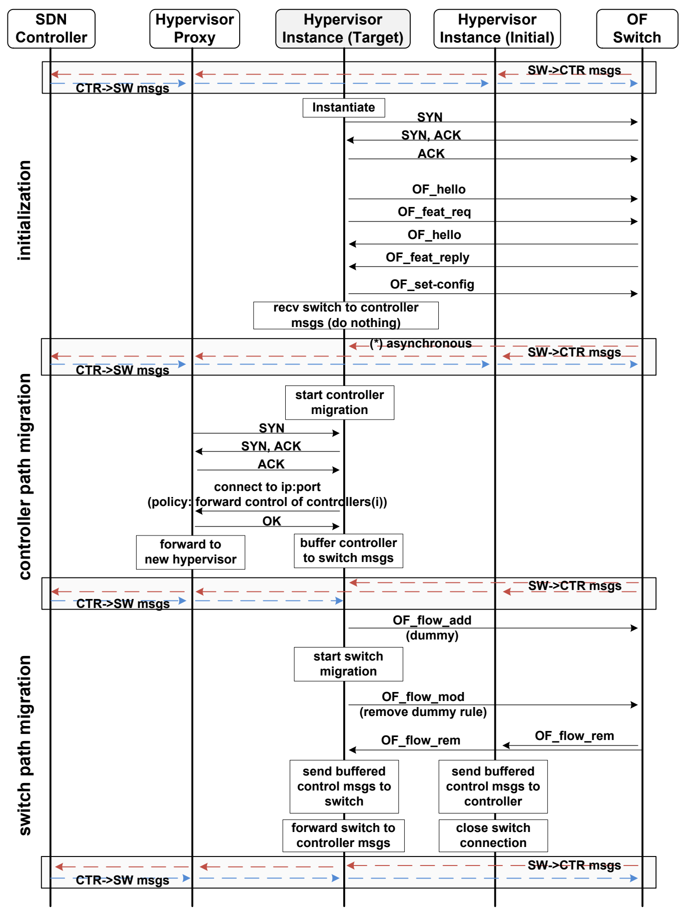
SDN Controller
Hypervisor Proxy
Hypervisor Instance (Target)
Hypervisor Instance (Initial)
OF Switch
SW->CTR msgs
CTR->SW msgs
initialization
Instantiate
SYN
SYN, ACK
ACK
OF_hello
OF_feat_req
OF_hello
OF_feat_reply
OF_set-config
recv switch to controller msgs (do nothing)
(*) asynchronous
SW->CTR msgs
CTR->SW msgs
controller path migration
start controller migration
SYN
SYN, ACK
ACK
connect to ip:port (policy: forward control of controllers(i))
OK
forward to new hypervisor
buffer controller to switch msgs
SW->CTR msgs
CTR->SW msgs
switch path migration
OF_flow_add (dummy)
start switch migration
OF_flow_mod (remove dummy rule)
OF_flow_rem
OF_flow_rem
send buffered control msgs to switch
send buffered control msgs to controller
forward switch to controller msgs
close switch connection
SW->CTR msgs
CTR->SW msgs

Fig. 2. The message exchange and states of the control path during migration from an initial to a target hypervisor instance.

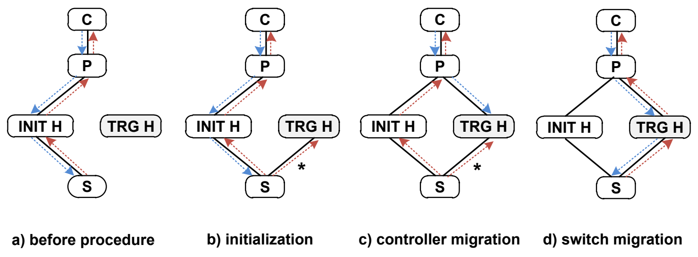

Fig. 3. Phases of control path migration from an initial to a target hypervisor instance. Connections: solid black. Switch to controller messages: dashed red. Controller to switch messages: dashed blue. (*): asynchronous OF messages.

switch, where each switch attempts to initiate the connection to its configured controller. In the context of SDN virtualization, hypervisors act as controllers for the OF switches. In case of a dynamic on-demand instantiation of hypervisors, switches require to know the IP addresses of the target hypervisors. The first option is to install a pre-determined set of addresses for hypervisors at each switch. This acts as a limitation as it sets an upper bound to the maximum number of distributed hypervisors that can be instantiated. It can also induces a waste of switch resources in case of a high number of active hypervisor instances since the switches have to store this set of possible addresses of hypervisors. It also requires a trigger during run time to inform the switch about the active hypervisor that the switch should use.

Therefore, we propose a novel method to use the listening ports of OF switches. Almost all available protoype and commercial switches, e.g., Open vSwitch, CPqD, HP, Brocade and Pica8, implement a listening port that can be used to initialize the control connection from the hypervisor. That means the switch-controller TCP connection initialization is the counter opposite of the OF specification. The target hypervisor instance is configured by the cnetralized hypervisor management. This configuration includes the switch IP addresses and their respective listening ports to which it needs to initiate the connection. After the connection establishment, the OF_Hello and configuration procedures follow the OF specification.

Once the target hypervisor is connected to the OF switch, it acts as a second controller to the switch as illustrated in Figure 3b compared to Figure 3a. This means that asynchronous OF messages,i.e., not in a request-reply form, e.g., OF_Pckt_in, would be also sent to the target hypervisor as well. However, at that phase, there is no connection yet between the target hypervisor instance and the proxy, i.e., towards the controller. Hence, the target hypervisor can simply discard the received messages from the switch momentarily till the whole procedure is finished.

B. Controller Path Migration Phase

After the initialization phase, the target hypervisor instance initiates a TCP connection to the designated proxy from its configuration. The proxy addresses are pre-configured in the hypervisor instance by the management. The hypervisor informs the proxy of its assigned controllers. Consequently, the proxy installs policies to forward the control messages of those SDN controllers to the target hypervisor instance as shown in Figure 3c. An example adaption of the forwarding policy at the proxy can be as follows. (remove: connection controller C1 $\Leftrightarrow$ connection hypervisor H1) and (forward: connection controller C1 $\Leftrightarrow$ connection hypervisor H2). Since controller path migration is executed before the switch migration, the target hypervisor is required to buffer controller to switch messages till the switch migration is completed.

C. Switch Path Migration Phase

The final phase to complete the control path migration is the switch path migration. We have developed a protocol that uses OF messages to do the switch migration, that can be used starting from OF v1.0 [15]. Switch migration is initialized with the target hypervisor sending a dummy OF_flow_add message to the switch. This Flow_add message can be defined from OF flowspace fields which are not utilized by the controllers and are known by all hypervisor instances, e.g., OF cookie field, which is a controller issued ID and has 64 bits.

Afterwards, the target hypervisor starts switch path migration by sending an OF_flow_mod to the switch that deletes the dummy flow from the switch. The reason for this is that deletion of a flow dictates an OF switch to send an OF_flow_rem to all its assigned controllers, and hence the initial as well as the target hypervisors would receive it. In this way, the initial hypervisor can be notified of the switch migration and withhold the responsibility for that switch. Meanwhile, the target hypervisor instance would know that it can take over the switch, thus meaning the control path migration completion. The target hypervisor then starts to send the buffered control message to the switch and continues normal operation as illustrated in Figure 3d.

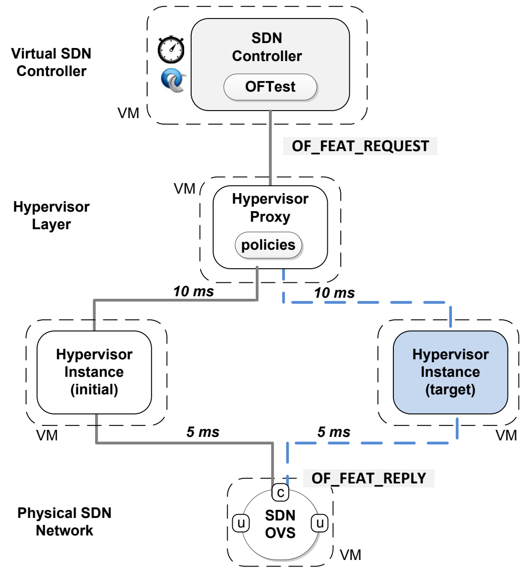

Fig. 4. Evaluation setup for the control path migration protocol.

V. Implementation and Evaluation

We have conducted an initial evaluation of a prototype system that has been developed in a real SDN testbed. The evaluation setup can be seen in Figure 4. The setup includes an established virtual SDN network, with an SDN controller and an OF switch. The control connection between the switch and the controller is assigned to the initial hypervisor instance. We evaluate the scenario where a new hypervisor instance, i.e., target hypervisor, is instantiated. The target hypervisor triggers our proposed protocol to take over the assignment of the SDN controller and switch by migrating the control connection, i.e., hypervisor to switch as well as hypervisor to proxy connections.

For the hypervisor layer, we have implemented a hypervisor instance in Python. The hypervisor implementation uses OpenflowJ Loxi library [16], which is an open source library. This library provides an OF protocol API that can be used for OF message parsing and generation. The library can support OF v1.0 up to v1.3. The hypervisor implementation also includes the proposed protocol. Additionally, a hypervisor proxy has been implemented in Python. It contains the forwarding policies, i.e., mapping, for the connections to the SDN controller and hypervisor instances.

The physical SDN network is realized as an Open vSwitch (OvS) [17], running an OF v1.2 and controlled by the implemented hypervisor. We use the listening port 6634 on the OvS in the initialization phase of the migration protocol. The SDN controller is realized by OFTest [18], a test suite framework

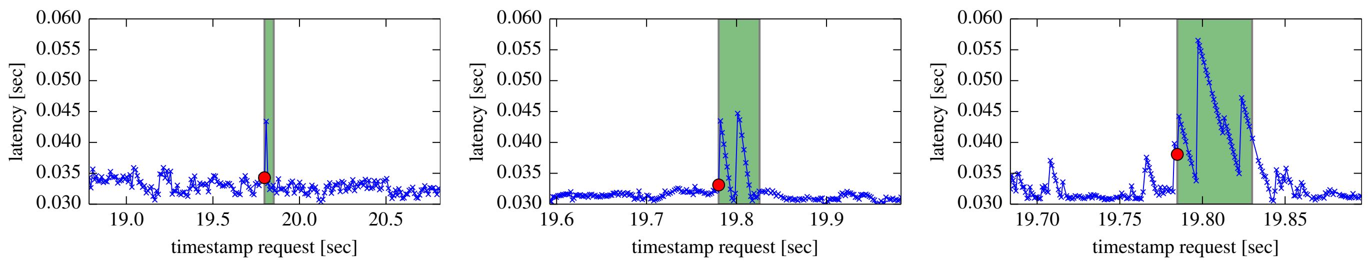

Fig. 5. Time-series for three runs with different OF message rate 100, 500, 1000. Red dot shows the start of the migration procedure. The highlighted area shows the duration of the migration protocol and the affected OF messages.

for OF switches. The default OFTest can only be configured to generate a total number of OF message, without controlling the message sending rate. We have extended OFTest extensively to generate an average constant sending rate of OF messages to evaluate the impact of the assignment protocol on the control plane performance. Taking control latency as a performance metric, we use OFTest to generate OF_feat_request messages at an average rate and we measure the latency to receive the corresponding OF_feat_reply back at OFTest, by using message xids which are unique transaction identifiers used to match requests to replies.

Throughout each run, we instantiate a new hypervisor instance and trigger the migration protocol. Hence, we can evaluate the impact of the migration on the control plane performance in terms of control latency. Each of the system components runs on a separate virtual machine (VM). In order to emulate a more realistic system, we have added an artificial latency of 10 millisecond on the link between the proxy and the hypervisor instances. In addition, a latency of 5 millisecond has been added on the link between the hypervisor instances and the switch, as shown in Figure 4. For evaluation, we iterate over an OF sending message rate from 100 requests/sec to 1000 requests/sec. For each rate, we repeat the evaluation run 40 times. Each run has a duration of 40 seconds.

Figure 5 illustrates the impact of the control path migration protocol on individual packets under different rates. Figures 5a, 5b and 5c show the time series for a sample run with an OF message sending rate of 100, 500 and 1000 requests per second, respectively. We identify the start of the migration protocol with the xid of the last OF_feat_request sent from the SDN controller to the initial hypervisor instance. The end of the protocol is marked by the xid of the first OF_feat_reply sent from the target hypervisor instance to the SDN controller. This indicates the duration needed to complete the migration protocol and the number of affected OF packets as well.

For one evaluation run, we observe an increase in the control latency during the migration execution compared to normal operation. During the migration execution, the maximum, i.e., peak, control latency goes to 43 milliseconds at 100 requests per second (Figure 5a). At 1000 requests per second (Figure 5c), the maximum control latency goes to 57 milliseconds. Hence, the maximum increase in control latency can be noted to be proportional to the OF rate on the control plane up to 1000 requests per second. The same observation can be made for the number of affected packets by the control path migration, where it increases in a proportion to the OF

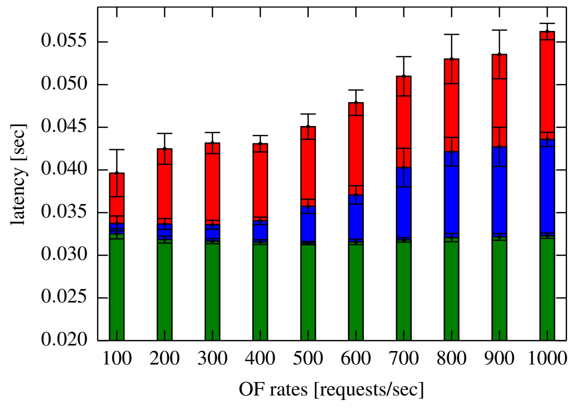

Fig. 6. Latency of OF packets during migration process. Green (bottom) bars show the average latency, blue (intermediate) bars show the average latency of affected packets, red (upper) bars show the average of the maximum latency with standard deviation of all runs. For all bars, the standard deviation is added via black error bars.

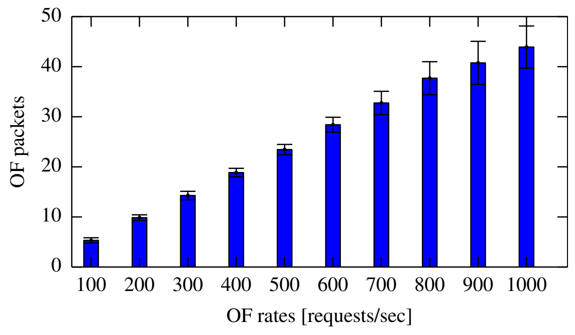

Fig. 7. Number of affected OF packets during migration process. The error bars indicate the standard deviation of the average values for all runs.

control rate. We consider the number of affected packets as an indicator to the operational duration in which the control channel is affected by the migration to another hypervisor. It is important to note that there is no packet loss (0%) due to migration for all evaluated OF rates. This implies that we could achieve a control path migration that is transparent to the SDN controllers in terms of control packet loss.

In Figure 6, we demonstrate statistical evaluation for the control latency over 40 runs for all marked packets affected by the migration. The green (bottom) bars show the average and standard deviation of the control latency in normal operation. This is the average of the 50 packets before the migration starts and of the 50 packets after the migration finished. The average and standard deviation of the control latency during migration is represented via the blue (intermediate) bars. Finally, we calculate the average and standard deviation of the maximum latency reached during migration, which is illustrated via red (upper) bars.

The first observation is that the average control latency in normal operation, i.e., green bars, is not affected by the control rate. The average control latency during migration, i.e., blue bars, confirms the correlation between the increase in the control latency during migration and the OF control rate. The same observation can be shown for the average of maximum control latency, i.e., red bars, during control path migration.

This can be explained as follows. During the migration, the protocol starts with the controller path migration, afterwards it initializes the switch path migration. All SDN controller to switch messages are buffered at the target hypervisor till the switch migration is completed, then it sends all buffered control messages to the switch. Thus, the number of affected, i.e, buffered, packets during migration increases with increasing the control rate as shown in Figure 7. However, note that the number of affected packets during migration is negligible compared to the total number of packets during operation. For example, at 1000 requests per second and for a run duration of 40 seconds, the average number of affected packets is 45 packets out of a total of 40000 packets, which is only 0.1%.

VI. Conclusion

In this paper, we introduce an SDN virtualization architecture that provides a scalable and dynamic distributed SDN hypervisor layer. The virtualization layer should adapt to changes in network state and virtual SDN requirements. In order to support this dynamic changes, we propose an OpenFlow-based control path migration protocol for distributed SDN hypervisors, that can adapt the virtualization layer without interrupting the virtual SDN networks, i.e., transparent.

A prototype has been implemented for the introduced protocol and an evaluation system has been setup on a real testbed. We have evaluated the impact of our proposed control path migration protocol on the performance of the virtual SDN control plane through measurements of the control latency with varying OF message sending rates. Our evaluation shows that the control latency and the number of packets affected during migration are proportional to the OF rate. We have observed no control packet loss for all OF rates. Additionally, we could observe that the number of packets affected by the migration can be negligible compared to the total umber of packets in normal operation. Hence, we conclude that the dynamic migration of the control path in the virtualization layer has no significant impact on the virtual SDN slices, thus transparent to their operation.

For future work, we plan to extend our evaluation to cover several other performance evaluations. The impact on the control plane performance from the frequency of executing the control path migration, i.e., how many control migration events are executed over time, can be evaluated. We are also keen to observe the control plane performance in case the control paths of several switches and controllers are migrated simultaneously. Additionally, it would be valuable to evaluate the overall latency including the management configuration time. Finally, evaluating the impact of migration with higher OF control rates can show more insights.

Acknowledgment

This work is part of a project that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 647158 FlexNets).

References

[1] 5G Initiative Team, "NGMN 5G Initiative White Paper," Feb. 2015, https://www.ngmn.org/uploads/media/NGMN-5G-White-Paper-V1-0.pdf.

[2] Mobile and wireless communications Enablers for the Twenty-twenty Information Society (METIS), "Final report on architecturer (Deliverable D6.4)," Feb. 2015, https://www.metis2020.com/wp-content/uploads/deliverables/METIS-D6.4-v2.pdf.

[3] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," *IEEE Internet Computing*, vol. 17, pp. 20–27, 2013.

[4] P. Skoldstrom and W. John, "Implementation and Evaluation of a Carrier-Grade OpenFlow Virtualization Scheme," *Second European Workshop on Software Defined Networks*, pp. 75–80, Berlin, 2013.

[5] Z. Bozakov and P. Papadimitriou, "AutoSlice: automated and scalable slicing for software-defined networks," *Proceedings of the ACM conference on CoNEXT student workshop*, pp. 3–4, New York, 2012.

[6] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[7] A. Al-Shabibi, M. D. Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar, "Openvirtex: A network hypervisor," *Open Networking Summit (ONS)*, Santa Clara, 2014.

[8] A. Blenk, A. Basta, and W. Kellerer, "Hyperflex: An sdn virtualization architecture with flexible hypervisor function allocation," *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 397–405, OTTAWA, 2015.

[9] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed SDN OS," *Proceedings of the third workshop on Hot topics in software defined networking (HotSDN)*, pp. 1–6, New York, 2014.

[10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." *Proceedings of the 9th USENIX conference on Operating systems design and implementation Article (OSDI)*, pp. 1–6, Berkeley, 2010.

[11] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: an efficient elastic distributed SDN control plane," *Proceedings of the third workshop on Hot topics in software defined networking (HotSDN)*, pp. 133–138, New York, 2014.

[12] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: an elastic distributed sdn controller," *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, pp. 17–28, New York, 2014.

[13] N. M. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, pp. 862–876, 2008.

[14] ONF, "OpenFlow Switch Specifications 1.3.0," Oct. 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[15] ONF, "OpenFlow Switch Specifications 1.0.0," Oct. 2009, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.

[16] "Openflowj loxi OpenFlow Library." [Online]. Available: https://github.com/floodlight/loxigen/wiki/OpenFlowJ-Loxi

[17] "Open vSwitch (OvS)." [Online]. Available: http://openvswitch.org/

[18] "OFTest." [Online]. Available: https://github.com/floodlight/oftest