# HyperFlex: An SDN Virtualization Architecture with Flexible Hypervisor Function Allocation

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

***Abstract*—Network Virtualization (NV) and Software-Defined Networking (SDN) are both expected to increase the flexibility and programmability of today's communication networks. Combining both approaches may even be a further step towards increasing the efficiency of network resource utilization. Multiple solutions for virtualizing SDN networks have already been proposed, however, they are either implemented in software or they require special network hardware. We propose HyperFlex, an SDN hypervisor architecture that relies on the decomposition of the hypervisor into functions that are essential for virtualizing SDN networks. The hypervisor functions can be flexibly executed in software or hosted on SDN network elements. Furthermore, existing hypervisor solutions focus on data-plane virtualization mechanisms and neglect the virtualization of the control-plane of SDN networks. HyperFlex provides control-plane virtualization by adding a control-plane isolation function, either in software or on network elements. The isolation function ensures that the resources of the control-plane are shared correctly between each virtual SDN network while it also protects the hypervisor resources from resource exhaustion.**

## I. Introduction

Network Virtualization (NV) promises to overcome the current ossification and limitations in today's communication networks [1]. NV allows multiple tenants to share the same physical infrastructure by acquiring virtual resources according to their services' demands. The main drivers for NV include cost reduction, faster network and service deployment (time to market), and higher efficiency in utilizing the network resources. Software-Defined Networking (SDN) is another emerging concept in networking [2]. Decoupling the control-plane from the data-plane of the switches, SDN runs a logically centralized controller in software, namely an SDN controller. Open interfaces and protocols, e.g., OpenFlow (OF), allow remote control of the data-plane SDN switches. Combining SDN and NV introduces network programmability, flexibility and automation resulting in virtual SDN networks (vSDNs).

The functional building blocks of NV are network abstraction, translation, and isolation [1]. In order to virtualize SDN networks, existing solutions introduce a hypervisor layer, or shortly a hypervisor, comparable to virtualization of servers [3]. A hypervisor provides mechanisms that realize the functional building blocks of NV. However, existing hypervisors have limited flexibility. They run either as software or they require special networking hardware, thus, they work only for special-purpose networks. Consequently, in case of different types of networks, such as data center networks, wide-area networks, or enterprise networks, the existing hypervisors are not able to adapt to changing demands.

As SDN decouples the control from the data-plane, an integral part of the performance of SDN is the realization of the control-plane. In particular, for virtual SDN networks (vSDNs), the performance of the control path, i.e., the physical links and the network elements, may have an impact on the performance of the virtual SDN network. As the hypervisor is placed between virtual SDN controllers and virtual SDN networks, the hypervisor itself may influence the performance of the virtual SDN networks. Most existing hypervisors focus on data-plane virtualization only, while less attention has been devoted to the virtualization of the control-plane.

In order to address the shortcomings of existing architectures, we introduce HyperFlex, an SDN hypervisor architecture based on flexible hypervisor function allocation. The design goals of HyperFlex are scalability, flexibility, isolation and protection. In order to provide scalability and flexibility, we analyze the building blocks of NV and how existing solutions realize them. Based on this analysis, HyperFlex relies on the decomposition of the hypervisor into functions that can be hosted on different platforms. The hypervisor functions' realization and placement adapts flexibly to the performance of the target host platform and to the virtual SDN network demands. Thus, HyperFlex can work in different operation modes, which may provide performance gains for different networks. Additionally, to the best of our knowledge, HyperFlex is the first virtualization solution for SDN networks that provides control-plane virtualization.

The remainder of this paper is structured as follows. In Section II, we introduce the HyperFlex architecture and provide detailed information on its possible modes of operation. As a first example for our proposed architecture, we explain how HyperFlex virtualizes the control-plane of SDN networks in Section II. In Section III, we provide a detailed summary of existing state of the art solutions for SDN network virtualization. In Section IV, we show a first prototype implementation of HyperFlex and present results measured in a real SDN testbed. Future work is outlined in Section V. Finally, we draw a conclusion in Section VI.

## II. HyperFlex Design

In this section, we first outline the design challenges of a hypervisor considering control-plane virtualization. Next, we introduce HyperFlex's architecture and its modes of operation.

Finally, we address the control-plane virtualization function and provide examples of its realization in our proposed architecture.

### A. Design Goals

In the following, we point out important aspects that a hypervisor for virtual SDN networks (vSDNs) should fulfill, namely flexibility, scalability, isolation, and protection for the virtual resource and hypervisor functions. These aspects are based on the design goals for NV given in [4] and [5].

*a)* ***Flexibility****:* One of the main properties of NV, in general, is the ability to cope with the network dynamics, where virtual networks provide the flexibility to adapt to the changes in the network, e.g. time-varying traffic [6] or network failure [7], through the migration of virtual nodes or the assignment of virtual link capacities, for instance. In vSDNs, network dynamics are not only to be observed at the physical infrastructure, i.e., SDN data-plane, but also at the tenants' vSDN controllers, i.e., SDN control-plane, as their control-plane can dynamically adapt to the changes in the data-plane, for example by relocating the controllers, to provide the desired overall performance, e.g. control-plane latency. Hence, a vSDN hypervisor layer is also required to provide the flexibility to adapt to the network dynamics at the SDN physical infrastructure as well as the vSDN control layer.

*b)* ***Scalability****:* Each tenant of a vSDN could use its own controller to manage its vSDN topology. A hypervisor architecture for a virtualized infrastructure should provide a high degree of scalability in terms of control-plane access to vSDN topologies, i.e., the physical connection for the tenants to the hypervisor. In large-scale networks, a centralized hypervisor may lead to bottlenecks or to significant waste of network resources. The tenants' controller, for instance, may send messages over long network paths before being discarded due to network over-utilization or hypervisor exhaustion. Additionally, vSDN controllers might be actually managing disjoint sets of physical resources. In such case, having a hierarchical or a distributed hypervisor responsible for the disjoint sets of resources would per se lead to an improved control performance. However, duplicating the software along the hypervisor layer may have two drawbacks. First, it may waste resources due to duplicated instances that are not needed along the whole hypervisor network. Second, it requires resources, i.e., servers, to host the distributed instances.

*c)* ***Isolation and Protection****:* Classical NV requires resource isolation and protection between co-existing virtual networks on the physical data-plane infrastructure. Resource guarantees are important in order to provide the tenants with a reliable operation. Protection means that operations of multiple tenants should not affect each others' performance, i.e., there is no performance degradation due to sharing the network among multiple tenants. Again, in vSDN networks, isolation and protection are required on the physical infrastructure for the data-plane traffic as well as the SDN control-plane. Since the SDN control-plane path, between the vSDN controllers and the physical SDN infrastructure, includes the hypervisor layer, it is important to provide isolation and protection mechanisms for the hypervisor's processing resources among the different vSDN tenants.

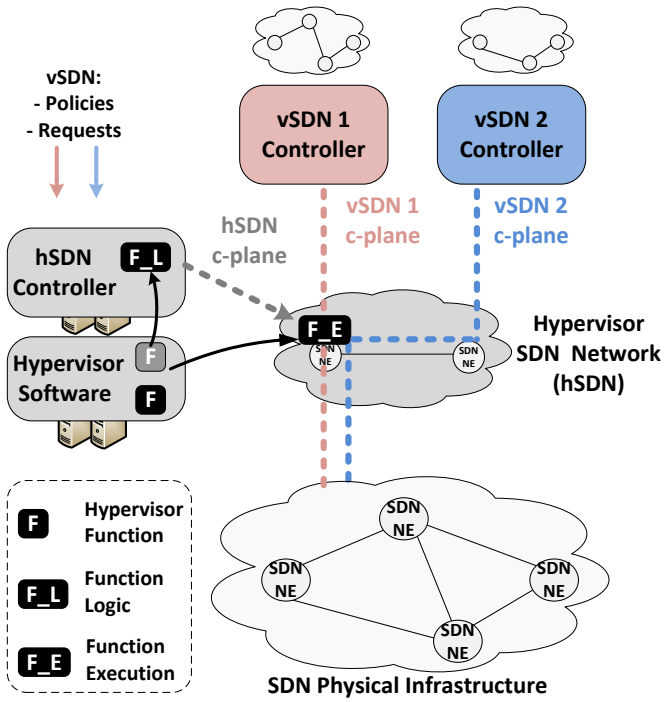



Fig. 1: HyperFlex flexible architecture consisting of hypervisor software, SDN network and controller

### B. HyperFlex Architecture

The first main concept of HyperFlex relies on the decomposition of the hypervisor virtualization functions required to realize vSDN, where having a functional-granularity achieves a more tailored operation. This approach of function decomposition is seen to be more optimal in terms of resource efficiency. If a distributed hypervisor layer is needed, individual virtualization functions could be distributed such that a tailored virtualization layer is achieved, instead of duplicating the whole hypervisor instance along the network.

The second main concept is the hosting platform of the SDN hypervisor layer. The hypervisor functions can be implemented in software. However, since in large-scale networks it is reasonable to assume multiple network elements interconnecting the hypervisor with the tenants' controllers or the physical infrastructure, we propose using the available processing functions, e.g., traffic shapers or packet inspection, on these interconnecting network elements to execute hypervisor functions. Thus, the virtualization layer can be spanned on software as well as SDN network elements, which we call the hypervisor SDN network (hSDN) as illustrated in Figure 1. This hSDN network is operated and controlled by a hypervisor SDN controller. The proposed architecture has the flexibility to allocate the virtualization functions, on a per-function basis, among the software and the SDN network elements of the hypervisor network.

In case a hypervisor function is delegated to the network elements, it needs to be transformed to its corresponding SDN data-plane execution function (F_E) at the hSDN network

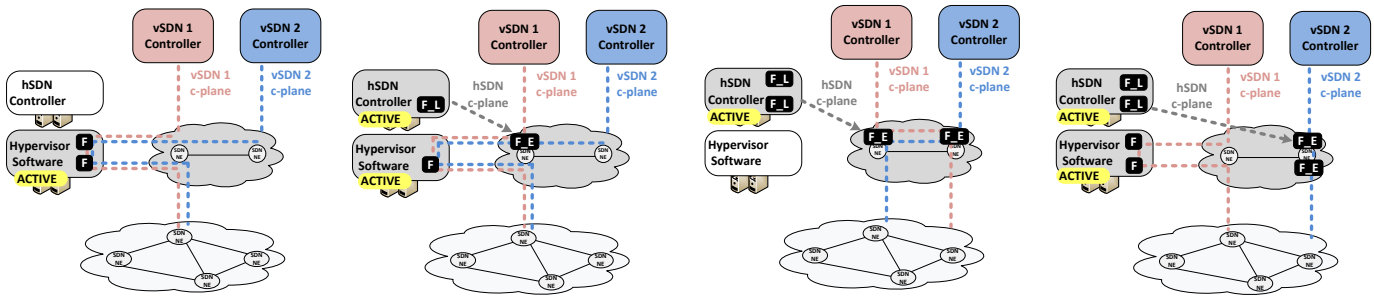



Fig. 2: Alternative modes of operation of HyperFlex

element and a function control logic (F_L) residing at the hSDN controller as shown in Figure 1. Additionally, the hSDN controller is responsible for managing the delegated functions as well as maintaining the connectivity between the vSDN tenants and the physical infrastructure.

A hypervisor consists of multiple virtualization functions that could be potentially delegated or even better suited to heavy-load processing functions at the network elements. As for example, the translation function that transparently maps the vSDN control messages to their physical resources. The translation could be done by the network elements according to their packet inspection capabilities and packet modification performance. Another example is the policy enforcement function, which ensures that a vSDN controller has access only to its acquired virtual resources, or the abstraction function, which determines the granularity of information to be exposed to the vSDN controller. These policies can be translated to rules enforced by the network elements. Additionally, the isolation functions, that slices the data and control-plane resources among the different vSDNs, could benefit from the traffic shaping mechanisms of the network elements.

A main advantage of this proposed architecture is offloading the load of performing the virtualization functions between the software and the hypervisor network elements. This implies that the resources of the servers, hosting the hypervisor software, are extended by the resources of the SDN hypervisor network, which has a direct impact on improving the virtualization performance. It also adds another flexibility dimension by providing the possibility to change dynamically the functions placement during operation, since both options are present, namely the software and the SDN hypervisor network. Such flexibility is crucial, for example, in case the software or the network element is running out of resources and is not capable of offering the required performance for a certain virtualization function. Such function could be offloaded and executed at another software instance or network element.

It is intended to separate the SDN hypervisor network from the SDN physical infrastructure to isolate the virtualization functions from the data-plane, hence eliminate any impact that could be imposed by virtualization on the data-plane performance.

### C. HyperFlex Modes of Operation

*a)* ***Hypervisor in Software****:* The flexibility in HyperFlex's architecture provides the option to operate a classical virtualization layer in software as a centralized or distributed hypervisor. This mode of operation, shown in Figure 2a, is adequate in case a powerful and a wide-spanned servers infrastructure is available to host the hypervisor software. Alternatively, it suffices in case the vSDN tenants are geographically close to the hypervisor or in case of a small-scale SDN physical infrastructure, which allows the performance requirements by the vSDN tenants to be met, i.e., SLAs which include latency or bandwidth guarantees on the data and control-plane. This mode is selected given that the interconnecting network elements can not execute the virtualization functions due to the functions' complexity or the network elements' limited resources.

*b)* ***Function Chaining****:* This mode of operation, shown in Figure 2b, is the intermediary solution between running hypervisor functions solely in software or exclusively on network elements. As it exploits the functional-granularity and the processing resources of the hSDN network, it assigns the virtualization functions to the software or network elements, forming function chains that realize the hypervisor layer. The hSDN controller has to ensure that the correct chains are set along the hSDN network including the hypervisor software. This mode is seen beneficial in case some functions are too complex or not possible to realize by the network elements, e.g., DPI or stateful forwarding. Such virtualization functions are then implemented in software. The function placement decision is based on the available network resources, network elements' capabilities, and vSDN SLA requirements. These parameters can be quite variant, thus making this flexible operation very advantageous.

*c)* ***Hypervisor on Network Elements****:* Realizing the hypervisor functions on network elements enables the full deployment of the hypervisor layer on SDN network elements, as illustrated in Figure 2c. This could help bringing the virtualization functions nearer to the vSDN tenants or the physical infrastructure, hence improving the scalability and performance. It can also be seen as an enabler to achieve infrastructure savings, as servers are only needed to host the hSDN controller. This mode fully exploits the available processing functions in the SDN network elements to host the hypervisor

functions. However, such network elements might not provide all hypervisor functions out of the box, according to their SDN implementation. Therefore, this mode of operation adds the complexity of perhaps modifying the network elements and transforming the hypervisor function to an SDN realization. In addition, this mode adds to the management and control overhead for the transformed functions.

*d) **vSDN Differentiation**:* Another advantage inherited from the flexible operation on a functional-granularity is the possibility to allocate hypervisor functions to vSDN tenants, thus realizing vSDN differentiation. Each hypervisor function can be instantiated on different locations within the network to provide different performance quality for different vSDN tenants. As shown as an example in Figure 2d, vSDN1 is served by the functions running in software while vSDN2 is using the functions hosted by the hSDN network. This elasticity enables more differentiated services and agreements, which impacts the business position of the hypervisor provider.

### D. Control-plane Virtualization

As mentioned in SectionII-B, the hypervisor layer is composed of multiple virtualization functions, forming a function chain, that are traversed by the SDN control-plane communication between the vSDN controllers and the SDN physical infrastructure. The control-plane isolation function is intended to ensure guarantees on each of the virtualization functions shared among multiple tenants and to protect the hypervisor's processing resources, e.g., CPU or memory, from being exhausted by one of the vSDN tenants. If we take the translation function as an example, its resources can be exhausted if a vSDN controller is sending excessive *FLOW_MOD* messages ,downstream to its virtual SDN switches, or if a virtual SDN switch is overloading the translation function with *PACKET_IN* messages, upstream to the controller.

Considering a scenario where the translation function is in software and we target, for instance, to isolate and protect the CPU processing resources for this software. Figure 3 shows the control-plane isolation function realization in software. The first isolation logic module is the admission module which receives control-plane resources requests by each vSDN controller, in the form of requested number of control messages per second from each message type. As the hypervisor's CPU utilization is proportional to the amount of control-plane messages to be processed from each vSDN controller, the admission module contains a CPU utilization as a function of the control messages rate, on which it bases its decision to accept the requests. This information is passed to the message counter module which intercepts the control-plane messages and makes sure that the limit is not yet reached before forwarding to the translation function.

Figure 4 illustrates the function chaining mode, where the translation function is in software while the control-plane isolation function is realized on a network element, that is controlled by the hSDN controller. The admission module plays the same role, however currently at the hSDN controller. The correlation module translates the messages rate into a rate that a network element could enforce, e.g., bits per second or number of packets per second. The rate information is passed to the configuration module. This module configures

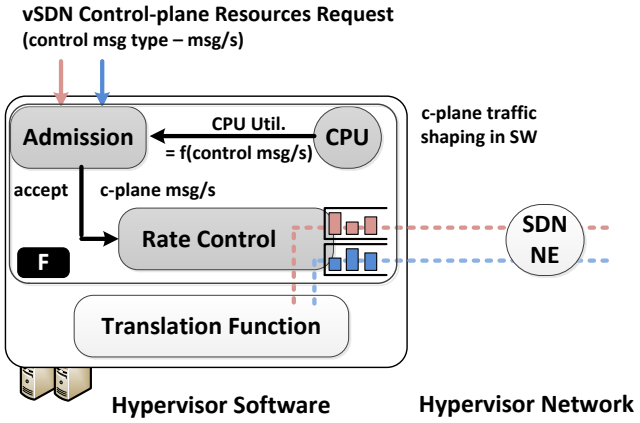



Fig. 3: control-plane isolation function in software

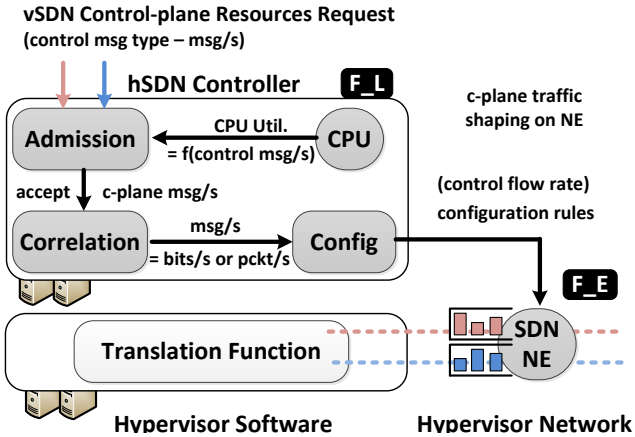



Fig. 4: control-plane isolation function on network elements

the isolation function at the network element, which can be, e.g. a port traffic shaper or OpenFlow meter.

This way the control-plane messages monitoring and isolation would be done by the hypervisor network element before forwarding the messages to the software to be translated, which exploits the processing power of the networking elements. It also protects and improves the utilization of the server, hosting the hypervisor software, as well as the hypervisor network since the control-plane traffic is shaped at the network point of entry according to the assigned control-plane shares of each vSDN controller.

## III. State of the Art

There are several existing hypervisor architectures to provide network virtualization for SDN networks. We are concerned with their proposed architecture, hypervisor platform and isolation mechanisms.

### A. FlowVisor

FlowVisor [3] has been introduced as one of the earliest architectures to provide virtualzation for SDN, based

TABLE I: Comparison between SDN hypervisor solutions concerning their platform and the proposed isolation mechanisms

| | FlowVisor | ADVisor | VeRTIGO | Auto-slice | Carrier-grade | OpenVirteX | **HyperFlex** |
|---|---|---|---|---|---|---|---|
| Hypervisor Platform | software | software | software | software | network elements | software extended by edge network elements | **software and network elements** |
| Isolation Focus | data-plane (flow tables) | – | – | – | data-plane (flow tables/bandwidth) | data-plane (flowspace) | **control-plane (CPU/bandwidth)** |

on OpenFlow. It plays the role of a logically centralized hypervisor layer between the controllers of virtual tenants and the physical SDN infrastructure, although it can also be deployed in a hierarchical fashion. It realizes the network abstraction by providing a mapping between the controller and its allocated virtual network resources. FlowVisor is implemented in software, hence it is a purely software solution. Regarding isolation, FlowVisor provides flow table isolation by maintaining a maximum number of flow entries for each vSDN at their assigned SDN switches. It is also mentioned that FlowVisor has the capabilities to associate certain network QoS guarantees with a virtual slice, e.g., data-plane bandwidth guarantees using the VLAN priority field..

### *B. ADVisor*

Advanced FlowVisor or ADVisor [8] is developed as an extension of FlowVisor, in which a software proxy or an enhanced abstraction unit is added to improve the virtual resources abstraction. Hence, similar to FlowVisor, it is also a software solution. No control-plane isolation mechanisms were proposed by this work.

### *C. VeRTIGO*

The developers of VeRTIGO [9] take the virtual network abstraction a step further. It is also an extension of FlowVisor, hence a software solution as well. VeRTIGO allows the vSDN controllers to select the level of virtual network abstraction that they require. The abstraction feature can range from providing the whole set of assigned virtual resources, with full virtual network control, to abstracting the whole vSDN to a single abstract resource, where the network operation is carried out by the hypervisor while the tenant focuses on services deployment on top. While this feature adds more flexibility in provisioning vSDNs, it also increases the hypervisor complexity and does not address the control-plane isolation limitations of the previous solutions.

### *D. Auto-slice*

AutoSlice [10] thrives to improve the scalability of a logically centralized hypervisor and, thus, it distributes the hypervisor's workload. For that purpose, the physical infrastructure is segmented into non-overlapping SDN domains, while the hypervisor is split into a management module and multiple controller proxies, one for each SDN physical domain. The virtual resources assignment is done by the management module and stored at each proxy, where the proxies in turn carry out the translation of the messages exchanged between the vSDN controllers and the physical infrastructure in their respective domain. It is still a solution targeting software deployment. Regarding isolation, a partial control-plane offloading could be offered by distributing the hypervisor over multiple proxies. However, within each SDN domain, the control-plane isolation problem still persists.

### *E. Carrier-grade Virtualization Scheme*

A distributed virtualization architecture for vSDN has been introduced in [11], by placing translation units in every data-plane SDN switch at the physical infrastructure. Hence, it is solution targeting network elements. A virtualization controller supplies the translation units with the set of policies and rules, which include the assigned label, flow table and port for each vSDN slice. Such distributed architecture aims at minimizing the additional overhead of a logically centralized hypervisor by providing a direct access from the vSDN controllers to the physical infrastructure. However, processing complexity is added to the data-plane physical infrastructure, which jupredizes the data-plane elements in case of overloading its control channel that could result in data-plane performance degradation. Additionally, the data-plane physical switches have to be modified to include a translation unit, which adds a higher complexity and eventually cost. Finally, there are still no isolation or protection functions proposed for the SDN control-plane.

### *F. OpenVirteX*

The authors in [12] tackle the flowspace problem, which is present in all above mentioned solutions. The flowspace problem is simply using flow matching header fields to differentiate between virtual tenants, thus not being able to offer the whole flow matching space to the vSDNs. OpenVirteX places edge switches at the borders of the physical SDN network to re-write the virtual IP addresses, used by the hosts of each tenant, into disjoint addresses to be used within the physical SDN infrastructure. The hypervisor makes sure that the correct address mapping is stored at the edge switches. Hence, the full flowspace could be provided to each tenant. The hypervisor layer is still software and extended by edge switches. Regarding isolation, no isolation schemes are discussed.

Table I shows a summarized comparison between the aforementioned hypervisor solutions regarding their hypervisor platform and their considered isolation mechanisms. Note that alternative solutions, which do not provide a full virtualized SDN, are not considered in our analysis, as for example FlowN [13]. FlowN proposes an architecture with a shared controller among all tenants where network applications of each tenant can be operated on top. In this case, the tenants are restricted to the capabilities of the provided controller by FlowN. Hence, the full programmability of a vSDN is reduced, which is only achieved in case a tenant is able to program and operate its own vSDN controller and virtual SDN switches. It should be further noted here that the above analyzed hypervisor

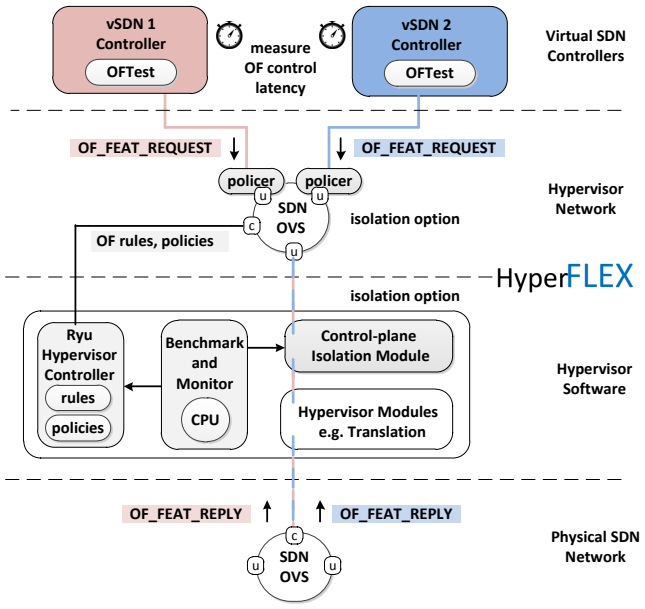



Fig. 5: HyperFlex Testbed Setup

solutions are examples to discuss different proposed architectures and platforms, other hypervisor solutions following the same architecture might be found.

## IV. Evaluation

First measurements in an OpenFlow-based SDN testbed are conducted in order to evaluate a prototype implementation of HyperFlex. In detail, we demonstrate a proof-of-concept of the flexible function allocation and control-plane virtualization. We investigate the performance of different setups for the hypervisor control-plane isolation function providing performance guarantees, which means that there should not be any cross effect between virtual SDN networks. A cross effect may result in performance degradation of one or more virtual SDN networks. Such performance degradation could be, for example, a longer flow setup time due to additional latency on the control paths. We consider the delay of OF control messages as a performance metric for all following evaluations. The overall goal is to guarantee control message delay for each virtual SDN network.

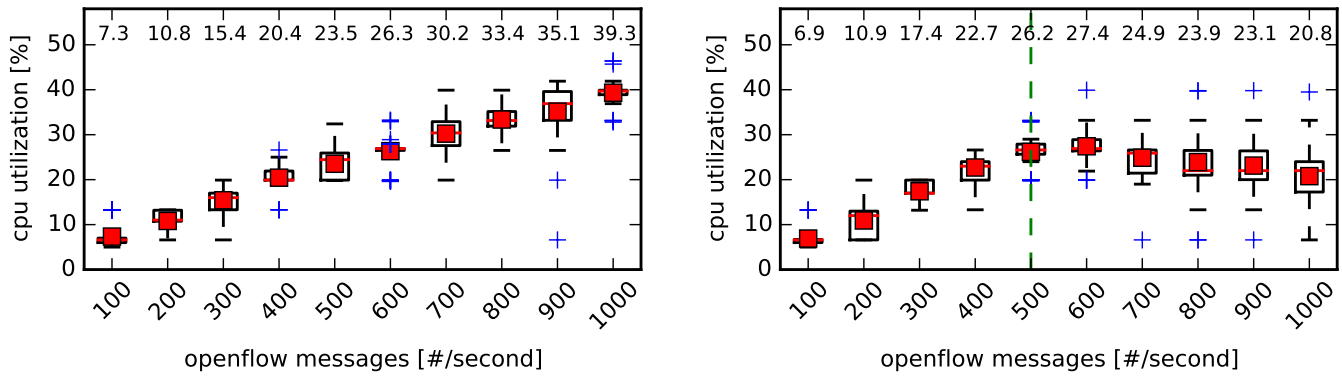
The measurement setup is shown in Figure 5. Two tenant controllers, namely vSDN-C1 and vSDN-C2, are running in a Virtual Machine (VM). For testing purpose, both controllers are based on *OFTest* [14], a framework and test suite for OF switches. *OFTest* is extended to generate an average constant rate of OF messages to test the hypervisor performance and control-plane virtualization under varying control-plane traffic. Both vSDN controllers are connecting to a data-plane switch of the SDN hypervisor network. The data-plane switch is an Open vSwitch (OVS) also running in a VM. An instance of *FlowVisor* [3] is used to provide the core hypervisor functions, e.g., the translation function. To implement the hypervisor controller, a *Ryu* controller [15] is used to control the hypervisor network. Both *FlowVisor* and *Ryu* share one VM. An additional instance of OVS represents the shared physical SDN data-plane network, that is running in another VM.

The measurement procedure is as follows. Both vSDN controllers are sending OF_FEAT_REQUEST messages to the shared data-plane switch and receive OF_FEAT_REPLY messages back. We evaluate the control-plane performance in terms of latency to receive the reply message back from the switch for each request message. All OF messages from both controllers have to be processed by the hypervisor. This produces load on the hypervisor's CPU resources. This CPU load can have a direct impact on the control-plane latency. Thus, the CPU utilization of the hypervisor needs to be monitored and isolated for each vSDN.

Hence, before investigating the performance latency for vSDNs, a first measurement is conducted to measure and analyze the impact of the rate of OF messages on the hypervisor CPU. Thus, the CPU utilization of the VM that hosts the hypervisor software is monitored for two setups. In the first setup, no isolation function is active. In the second setup, the isolation function on the OVS switch of the hypervisor network is activated. The allowed OF message rate is set to 500 messages per second, which corresponds to a traffic rate of 260 *kbps*. Note that this rate holds as long as only one OF message is sent per network packet. In both setups, 100 to 1000 OF_FEAT_REQUESTs per second are sent to FlowVisor. Each run lasts 25 seconds and is repeated 10 times.

Figure 6 shows boxplots for hypervisor's CPU utilization in % according to the rate of OF messages received for both setups. In addition, all figures that are showing boxplots have the corresponding mean values shown as red squares and additionally as values on top. In case no isolation is active, as shown by Figure 6a, the CPU utilization increases linearly with the amount of OF messages sent. The maximum mean value for this setup is 39.3 %. In case of limited CPU resources or shared hypervisor resources, too many network packets containing OF messages can lead to a performance degradation of the hypervisor. In order to avoid CPU over-utilization due to excessive OF messages or packet rate, the network isolation can force the sender of OF messages to aggregate messages. This is additionally marked by the dashed line as shown in Figure 6b. The dashed line illustrates the limit of 500 OF messages. While the CPU utilization linearly increases with the message rate from 100 to 500, the rate limiter installed on the hypervisor network throttles the network rate between 500 and 1000 messages. After reaching a maximum mean value of 27.4 %, the rate limiting even decreases the CPU utilization. Thus, the hypervisor could be protected from over-utilization. In order to additionally illustrate the impact of OF messages on the CPU utilization of the hypervisor, the available processing capacity is limited to 40 % in all following experiments. According to Figure 6, this allows the hypervisor to handle 1000 OF messages per second.

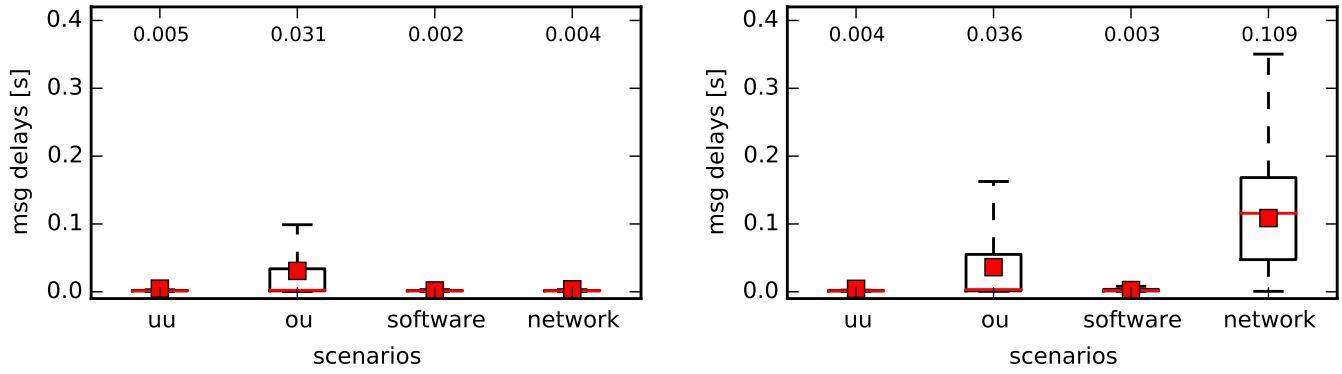
For the investigation of all operation modes of HyperFlex guaranteeing control-plane isolation, vSDN-C1 and vSDN-C2 are each assigned an equal share of 500 OF_FEAT_REQUEST messages per second. The performance of the control-plane isolation is evaluated for the following four scenarios. The configurations of all setups are additionally provided in Table II.




(a) Hypervisor CPU **without** isolation and protection on network

(b) Hypervisor CPU **with** isolation and protection on network

Fig. 6: Hypervisor CPU utilization versus OF messages rate with and without network isolation.




(a) Control-plane latency of vSDN-C1

(b) Control-plane latency of vSDN-C2

Fig. 7: Control-plane latency of vSDN-C1 and vSDN-C2 for four evaluation scenarios: hypervisor CPU under-utilization (uu), CPU over-utilization (ou), isolation in software (software), and isolation on network (network)

TABLE II: Scenario configurations

| | OF messages of vSDN-C1 | OF messages of vSDN-C2 | isolation in software | isolation on network |
|---|---|---|---|---|
| under-utilization (uu) | 500 | 500 | - | - |
| over-utilization (ou) | 500 | 1000 | - | - |
| software | 500 | 1000 | 500 per vSDN-C | - |
| network | 500 | 1000 | - | 260 *kbps* per vSDN-C |

*a)* ***Hypervisor CPU under-utilization****:* In this scenario, each of vSDN-C1 and vSDN-C2 is generating its assigned rate of 500 OF_FEAT_REQUEST messages per second.

*b)* ***Hypervisor CPU over-utilization****:* vSDN-C2 exceeds its assigned rate by sending 1000 OF_FEAT_REQUEST messages per second, while vSDN-C1 keeps sending 500 OF_FEAT_REQUEST messages per second. No isolation mechanisms are applied in this scenario.

*c)* ***Control-plane isolation in hypervisor software****:* vSDN-C2 exceeds its assigned rate in this scenario as well by sending 1000 OF_FEAT_REQUEST messages per second. However, the control-plane software isolation function is activated to monitor the OF messages rate sent by each vSDN controller with a limit of 500 OF_FEAT_REQUEST messages per second. OF messages exceeding the rate limit are dropped.

*d)* ***Control-plane isolation on hypervisor network****:* In this scenario, control-plane isolation is provided by the hypervisor network, i.e. switch. Rate policing is activated at the ingress ports for each vSDN controller. The OF messages rate is transformed to bitrate, where the assigned share of 500 OF_FEAT_REQUEST messages per second corresponds to 260 *kbps*, in case a single OF message is sent per packet.

TABLE III: Loss rate of OF_FEAT_REQUEST messages for vSDN-C1 and vSDN-C2 in each evaluation scenario

| | under-util | over-util | isolation in software | isolation on network |
|---|---|---|---|---|
| vSDN-C1 | 0% | 0% | 5.8% | 0% |
| vSDN-C2 | 0% | 0% | 52.9% | 0% |

All scenarios were run for 25 seconds and repeated 30 times in order to get statistical evidence. Figure 7a and Figure 7b show the control-plane latency of vSDN-C1 and vSDN-C2, respectively, for the four scenarios via boxplots. Both controllers observe a similar average control-plane latency of 4 ms given that the hypervisor's CPU is under-utilized. In case of over-utilization of the hypervisor CPU, which is caused by vSDN-C2 exceeding its assigned control-plane message rate, an up to 6 times higher average control-plane latency is observed for both controllers. This shows the cross effect and impact of vSDN-C2 on the control-plane performance of vSDN-C1, provided that no control-plane isolation is implemented.

With control-plane isolation activated in the hypervisor software, control-plane latency for both controllers is once more similar to the under-utilization case, thus control-plane isolation is achieved. Note that the latency observed by vSDN-C2 is only for the conforming 500 OF messages out of 1000 total messages that are generated per second. The software isolation function, which implements rate limiting on the application layer, i.e., OF layer, drops all OF messages exceeding the limit per second. This results in a loss rate of 52.9% for OF_FEAT_REQUEST messages generated by vSDN-C2, as shown in Table III. Note that there is a minor loss rate of 5.8% observed by vSDN-C1 since the OF message generation is not ideal and might also slightly exceed the assigned 500 OF messages.

In case of control-plane isolation activated on the network element, the control-plane latency of vSDN-C1 is similar to the under-utilization scenario. However, a much higher control-plane latency is experienced by vSDN-C2. Rate limiting on the network element drops TCP packets exceeding the rate limit on the ingress port. In contrast to OF, TCP initiates retransmissions for the dropped packets. Hence, all 1000 OF_FEAT_REQUEST messages are transmitted to the hypervisor and no OF message loss is experienced as shown in Table III. However, retransmissions result in a higher control-plane latency.

## V. Future Work

As for future work, following the concept of decomposing a hypervisor into functions, we are keen on defining metrics and algorithms to dynamically place hypervisor functions over the available resources pool, between software and network elements. The hypervisor functions' placement algorithms should adapt to the virtual SDN networks' requirements, the observed performance by each vSDN, and the properties and current performance of the physical infrastructure. Regarding the control-plane isolation function performance, it is required to be evaluated for different types of OF control traffic, e.g., FLOW_MOD or topology discovery packets. Further evaluation could consider OF control traffic also using UDP, as it is now possible with auxiliary channels since OF 1.3 [16].

We aim to extend our investigation to implement other hypervisor functions, e.g., the abstraction or translation function, as software or their realization on SDN network elements. This would enable us to reach the final goal of a fully adaptable and flexible hypervisor layer.

## VI. Conclusion

In this paper, we propose a hypervisor architecture that decomposes the hypervisor for SDN networks into functions, which can be flexibly placed among servers or SDN network elements, in order to provide a flexible, scalable and performant virtualization architecture. As the hypervisor functions can be flexibly placed among the network, the architecture provides different operation modes that can be used according to the performance guarantees needed for the virtualized SDN network. The architecture also provides control-plane virtualization that ensures control-plane isolation among the virtual SDN tenants while protecting the hypervisor from over-utilization. The realization of the control-plane isolation function is thoroughly discussed as a server-based function or transformed to an SDN-based realization.

A first prototype was implemented and evaluated in order to show the trade-offs between the different operation modes of the proposed hypervisor architecture. Measurement results taken from a real SDN testbed for different utilization scenarios reveal that the flexible hypervisor provides improved isolation between virtual SDN networks of multiple tenants. Furthermore, the architecture is more reliable as the performance protection of hypervisor functions is improved as well.

## References


[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[2] ONF, "Software-Defined Networking: The New Norm for Networks, white paper," April 2012, https://www.opennetworking.org/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[3] R. Sherwood, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, M. Chan, G. Parulkar, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, and M. Kobayashi, "Carving research slices out of your production networks with OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, p. 129, Jan. 2010.

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

[5] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, "Network virtualization: a hypervisor for the Internet?," *IEEE Communications Magazine*, vol. 50, no. 1, pp. 136–143, Jan. 2012.

[6] A. Blenk and W. Kellerer, "Traffic pattern based virtual network embedding," *Proceedings of the 2013 workshop on Student workhop - CoNEXT Student Workhop'13*, New York, USA, 2013, pp. 23–26.

[7] A. Basta, B. Barla, M. Hoffmann, G. Carle, and D. A. Schupke, "Failure coverage in optimal virtual networks," *Optical Fiber Communication Conference*, Anaheim, USA, 2013, pp. 1–3.

[8] E. Salvadori, R. D. Corin, a. Broglio, and M. Gerola, "Generalizing Virtual Network Topologies in OpenFlow-Based Networks," *2011 IEEE Global Telecommunications Conference*, Houston, USA, 2011, pp. 1–6.

[9] R. D. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "VeRTIGO: Network Virtualization and Beyond," *2012 European Workshop on Software Defined Networking*, Darmstadt, Germany, 2012, pp. 24–29.

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

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

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

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

[14] "Oftest." [Online]. Available: https://github.com/floodlight/oftest

[15] "Ryu SDN Framework." [Online]. Available: http://osrg.github.io/ryu/

[16] ONF, "OpenFlow Switch Specifications 1.4," vol. 0, pp. 1–205, Oct. 2013, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.