Chapter 1
Network Virtualization and Network Hypervisors

Andreas Blenk[1,2] and Wolfgang Kellerer[1]

[1] *Chair of Communication Networks, Technical University of Munich, 80333, Munich, Arcisstr. 21, Germany*
[2] *Faculty of Computer Science, University of Vienna, Vienna, Währinger Str. 29, Austria*

*Corresponding Author: Wolfgang Kellerer; wolfgang.kellerer@tum.de

Abstract: Modern communication networks, have to deal with dynamically changing requirements of emerging applications. In order to support such heterogeneous requirements on the same substrate network concepts for network abstraction and network programmabiluty are essential to offer applications individually tailored views on the network. Network virtualization is the basic concept to provide an abstraction of a substrate network that can be offerend to a network application provider. Here, a network hypervisor realizes the level of indirection between substrate network and virtual network(s). Additional programmability can be offered through the concept of Software Defined Networking (SDN). SDN provides a network abstraction based on a centralized view on a network to program individual features such as forwarding rules inside a network. This chapter gives an overview over the networking concept network virtualization at the example of the virtualization of SDN-based networks. The combination of

network virtualization and Software Defined Networking provides full network abstraction and adaptability. We detail the basic features of a network hypervisor: abstraction and isolation and provide the example of FlowVisor as a prominent network hypervisor example in the state of the art.

Keywords: Network Virtualization, Network Hypervisors

1.1. Introduction

Communication networks such as the Internet, data center networks or enterprise networks have become a critical infrastructure of our society. Although these communications networks and their protocols have been a great success, they have been designed for providing connectivity in a best-effort manner. However, given the current shift away from human-to-human communication toward machine-to-machine communication, e.g., in the context of (distributed) cloud computing or Cyber-Physical Systems (CPSs), designing networks for best-effort transmission is no longer sufficient. The reasons are manifold: future applications like Internet-of-Things or robotics require communication networks providing Quality-of-Service (QoS) guarantees and predictable performance while they are sharing the same underlying network infrastructure. Whereas traditional communication networks have been planned and operated by humans, resulting in a rather slow operation and update, modern applications require fast and automatic changes to new requirements as those of future networking concepts such as 5G [Agyapong et al., 2014].

Indeed, traditionally it has been assumed that communication networks serve applications with homogeneous network resource requirements not changing over time. However, today's application requirements fluctuate on time scales from minutes to milliseconds and are possibly highly diverse with respect to their required network resources [Benson et al., 2010, Erman and Ramakrishnan, 2013, Garcia-Dorado et al., 2012, Gehlen et al., 2012]. For example for CPSs, communication networks must serve

latency-critical control loops where resource adaptations must be put into effect within millisecond timescales. Despite of their different requirements, applications typically share the same physical infrastructures. As a consequence, they rely on the same protocol stack. However, communication network infrastructures with their current protocol stacks lack adequate mechanisms to handle changing application requirements in a timely manner. Hence, today's communication networks lack the flexibility in providing efficient resource sharing with a high level of adaptability, needed to support demands with diverse network resource requirements changing over time. Overall, this results in a performance that is far from perfect for both the network operator and the network users.

Two paradigms, namely Network Virtualization (NV) [Anderson et al., 2005] and Software-Defined Networking (SDN) [McKeown et al., 2008], are expected to cope with those requirements for flexible network resource sharing and adaptability. Whereas NV is seen as a key enabler to overcome the ossification of the Internet by introducing flexible resource sharing [Anderson et al., 2005], SDN introduces a new way of flexibly programming the shared resources at runtime [McKeown et al., 2008].

NV abstracts physical resources of Infrastructure Providers (InP) and enables tenants, i.e., Service Providers (SP), to use virtual resources according to their users' demands. Due to NV, InPs and SPs can control physical and virtual resources respectively in a dynamic and independent manner. To gain the highest efficiency out of virtualized networks, InPs need mechanisms that quickly provide (virtual) network resources in a predictable and isolated manner. On the other side, SPs should be able to flexibly request and control their resources with high degree of freedom. Hence, NV opens a new path towards communication systems hosting multiple virtual networks of SPs.

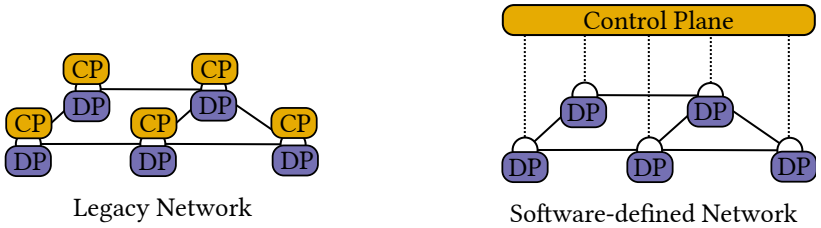
SDN decouples control planes of network devices, such as routers and switches, from their data planes. Using open interfaces such as OpenFlow [McKeown et al., 2008], SDN provides new means of network programmability [McKeown et al., 2008]. With networks being completely programmable, SDN can realize Network Operat-

ing Systems (NOSs) integrating new emerging concepts; NOSs can be tailored to application-, service-, and user-specific demands. As an example, NOSs can integrate raising mechanisms from the research field of Artificial Intelligence (AI). This might lead to future NOSs, and communication networks that self-adapt to unforeseen events, e.g., based on knowledge that is inferred at runtime from the behavior of network users, network topologies, or the behavior of network elements.

Combining NV and SDN offers the advantages of both worlds: a flexible and dynamic resource acquisition by tenants through NV and a standardized way to program those resources through SDN. This is called *the virtualization of software-defined networks*, leading to the existence of multiple Virtual Software-Defined Networks (vSDNs) sharing one infrastructure [Sherwood et al., 2009, 2010, Al-Shabibi et al., 2014a,b]. With both paradigms, it is expected that multiple vSDNs coexist while each one is individually managed by its own NOS. The combination makes it possible to implement, test, and even introduce new NOSs at runtime into existing networking infrastructures.

Like in computer virtualization where a hypervisor manages Virtual Machines (VMs) and their physical resource access [Barham et al., 2003], a so-called virtualization layer realizes the virtualization of SDNs [Sherwood et al., 2009]. The virtualization layer assigns, manages, and controls the physical network resources, while coordinating the access of virtual network tenants. The virtualization layer in SDN-based networks is realized by one or many network hypervisors [Koponen et al., 2014]. They implement the control logic needed for virtualizing software-defined networks. They act as proxies between the NOSs of tenants and the shared physical infrastructure, where the vSDN networks reside. Due to their key position, a deep understanding of design choices and performance implications of network hypervisor implementations is of significant importance. Without this understanding, network operators cannot provide SPs with guaranteed and predictable network performance - a critical obstacle for the success of combining NV and SDN.

In this book chapter, we introduce background needed to understand the concept of network virtualization and the SDN network hypervisors concept, i.e., NV and SDN.



(a) Legacy network where the control plane (CP) and the data plane (DP) are integrated into a device.

(b) Software-defined network where the control plane (CP) is decoupled from the data plane (DP) of the devices.

Figure 1.1: Comparison of legacy and software-defined network.

Furthermore, we provide an outline of the defining characteristics of network hypervisors. Finally, we briefly survey the first SDN network hypervisor, namely FlowVisor.

1.2. Background

1.2.1. Software-Defined Networking (SDN)

In legacy networks, the control plane is tightly coupled into the same device as the data plane. Fig. 1.1a shows an example network where the control plane is distributed among the devices. The control plane is responsible for control decisions, e.g., to populate the routing tables of Internet Protocol (IP) routers for effective packet forwarding. Accordingly, in case of distributed control planes, an overall network control is established through the operation of distributed network operating systems. As distributed operating systems may belong to different stakeholders, a common agreement on the available functions and protocols is always needed in case of adaptations. This, however, may hinder the innovation of communication networks: in order to let operating systems cooperate, basic agreements need to be established first. Achieving such agreements might, however, be a time intensive and long taking task.

In order to overcome such problems, SDN decouples the control plane from the

data plane, which allows a centralized logical control of distributed devices [McKeown et al., 2008]. Fig 1.1b illustrates an example where the control is decoupled from the devices. The control plane logic is centralized in the SDN controller, which operates the SDN switches. The centralized control maintains the global network state, which is distributed across the data plane devices. While the data plane devices still carry out the forwarding of data packets, the centralized control now instructs the data plane devices how and where to forward data packets.

SDN defines the Data-Controller Plane Interface (D-CPI)[1] to program distributed networking devices. The D-CPI is used between the physical data plane and the (logically centralized) control plane. The connection between SDN devices and SDN controllers is referred as *control channel*. To establish connections through control channels, SDN switches still need to implement logics, so-called agents, which receive and execute commands from the external control plane. To provide a common operation and control among heterogeneous SDN devices, instruction sets that abstract the physical data plane hardware are needed. The most most popular development is the OpenFlow (OF) protocol [McKeown et al., 2008].

SDN controllers are written in software, which can be implemented through a variety of programming languages, e.g., C++ or Python. This hardware independence is expected to bring faster development and deployment of networking solutions. One of the first SDN controllers has been NOX [Gude et al., 2008]. Many SDN controllers have followed: e.g., Ryu [ryu], ONOS [Berde et al., 2014], Beacon [Erickson, 2013], OpenDayLight [OpenDaylight, 2013]. Being implemented in software, SDN further allows to freely design control plane architectures among the whole spectrum from one central entity to a completely distributed control plane, such as in traditional networks [Tootoonchian and Ganjali, 2010, Koponen et al., 2010].

SDN applies a match and action paradigm to realize packet forwarding decisions. The SDN controller pushes instruction sets to the data plane, which include a match

[1]This interface has also been known as the *Southbound* interface earlier.

and action specification. In SDN, match specifications define a flow of packets, i.e., network traffic flow. A match is determined by the header values of packets of a network flow. For example, OF defines a set of header fields including, e.g., the Transmission Control Protocol (TCP) header and the IP header. An SDN controller instructs SDN switches to match on the specified fields and apply actions. The main actions are forward, drop, or modify network packets. Each version of the OF specification extended the set of header fields that can be matched on as well as the available actions.

An SDN switch stores the instructions on how to handle network packets in one or multiple flow tables. The size for storing flow table entries is a defining characteristic of an SDN switch. Flow tables can be implemented either in hardware or software. Current OF switches use Ternary Content Addressable Memories (TCAMs) for hardware tables, which are fast but costly and limited. In contrast, software tables provide more space but are slower than hardware tables in storing and complex matching of network flows [Kuźniar et al., 2015]. Accordingly, for a predictable isolation, such resources also need to be allocated properly.

In order to ease the development of new networking applications as well as the control of software-defined networks, controllers provide Application-Controller Plane Interfaces (A-CPIs) [Open Networking Foundation (ONF), 2014b,c]. Networking applications, like firewalls or load balancers, reside in the application control plane and use such A-CPIs, . However, networking applications can be developed upon the provided functionality of the controllers' specific A-CPIs. Accordingly, while networking application designers and operators can again freely develop in any programming language, they are dependent on the A-CPI protocol of the respective controller, e.g., the REST API of ONOS [Berde et al., 2014] in OF-based networks. Unfortunately, no common instruction set for the A-CPI has been defined yet.

This book chapter targets network hypervisors for OpenFlow (OF)-based SDN networks. Hence, this section introduces background information on the OF protocol, its implementation aspects and message types, which are defined by the Open Networking Foundation (ONF) in different versions Open Networking Foundation (ONF) [2009,

2011a,b, 2012, 2013, 2014a]. Moreover, it introduces several SDN network hypervisors.

1.2.2. OpenFlow Protocol

As OF is so far the most prominent and accepted realization for SDN-based networks, many OF controllers, switches, and network hypervisors exist. Accordingly, when discussing OF-based vSDNs, background information on the OF protocol be provided for a basic understanding of OF-based network hypervisors.

OpenFlow components.. In an SDN network, one *controller* manages multiple *OF switches*: controllers and switches build the end-points of the OF protocol. The switches are connected via multiple *OpenFlow (OF) control channels* with the controller.

An OF switch typically has one control channel for one controller; auxiliary (parallel) control channels are possible, e.g., to improve redundancy. The OF specification does not specify the control channel to be an out-band network, i.e., dedicated switches for the control traffic, or an in-band one, where control traffic is transmitted through the managed OF switches.

Literature sometimes calls the entity that implements the OF specification on the switch side the *OpenFlow (OF) agent* [Kreutz et al., 2015]. Given the IP address of the controller, OF agents initiate TCP connections to the controller via the OF control channel interface. The controller's IP address is normally pre-configured on switches before starting network control and operation. OF messages, i.e., commands, statistics, and notifications are sent through the control channels.

OpenFlow messages.. OF defines three message types: *Controller-to-Switch*, *Asynchronous*, and *Symmetric*. The controller initiates *Controller-to-Switch* messages: e.g., to request features or to send a packet out on the data path of the switch. Only switches send asynchronous messages. Switches use these messages to report network events to controllers or changes of the their states. Both controllers or switches can

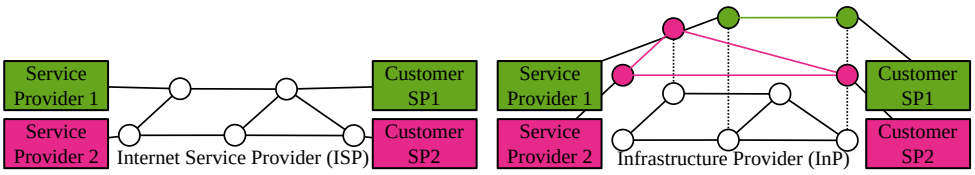
send *Symmetric* messages. They are sent without solicitation. The following message are typically used in OF-based networks. *Asynchronous* messages:

- `OFPT_PACKET_IN`: Switches send `OFPT_PACKET_IN` messages to controllers to transfer the control of the packet. They are either triggered by a flow entry (a rule that specifies to send `OFPT_PACKET_IN` messages) or by a table-miss (when no rule can be found and the switch is then sending `OFPT_PACKET_IN` messages as its default behavior).

Controller-to-Switch messages:

- `OFPT_FEATURES_REQUEST` and `OFPT_FEATURES_REPLY`: This message request/reply pattern exchanges the main information on switch identities and on switch capabilities. A controller normally requests features once when a new control channel connection is established.
- `OFPT_FLOW_MOD`: This message modifies flow table entries; it adds, modifies, or removes flows from tables.
- `OFMP_PORT_STATS`: The controller sends this message to request statistics about one or many ports of the switch. Statistics can be about received or transmitted packets and bytes etc.
- `OFPT_PACKET_OUT`: A controller uses this message type to send a packet out through the datapath of a switch. The controller sends this message, for instance, to discover topologies by using the Link Layer Discovery Protocol (LLDP).

The next section provides a brief introduction on network virtualization. Afterwards, we connect both worlds, i.e., SDN and NV through the notion of network hypervisors.



(a) Traditional role: SP providing access to service providers and customers. SPs cannot impact network decision and resource allocation.

(b) NV business model: SPs request virtual networks. Customers connect to services via virtual networks.

Figure 1.2: Traditional business model with SP versus NV business model with INP.

1.3. Network Virtualization

NV has its roots in the virtualization of computers. Virtualization of computers has been one of the main drivers for the deployment of data centers and clouds [Goldberg, 1974, Li et al., 2010, Sahoo et al., 2010, Douglis and Krieger, 2013, Smith and Nair, 2005, Zhang et al., 2014]. Inspired by this successful development, NV has initially been investigated for testbed deployments [Anderson et al., 2005, Turner and Taylor, 2005, Feamster et al., 2007]. The idea of sharing physical networking resources among multiple tenants or customers has then been transferred to communication networks serving production network traffic: NV is seen as the key enabler for overcoming the ossification of the Internet [Anderson et al., 2005, Feamster et al., 2007, Turner and Taylor, 2005]. As the idea of virtual networks is not new in general (e.g., VLAN defines layer 2 virtual networks), different network virtualization definitions and models have been proposed [Chowdhury and Boutaba, 2008, Bari et al., 2013, Casado et al., 2010, Koponen et al., 2015, 2014], e.g., based on the domain (data centers, wide area networks) they target.

NV has led to new business models, which are seen as main drivers for innovation for communication network technologies. One business model for network virtualization is introduced in [Chowdhury and Boutaba, 2009], as briefly introduced in the following. Fig. 1.2 compares the traditional roles with the NV roles. Traditionally, an

Internet Service Provider (ISP) provides Internet access for its customers towards a Service Provider (SP), such as Google, Netflix, Amazon, etc., which host their services in data centers (Fig. 1.2a). In the business models of network virtualization, as illustrated in Fig. 1.2b, the traditional role of an Internet Service Provider (ISP) of managing and operating networks is split into an SP role and an InP role [Feamster et al., 2007]. The SP's role can be enriched with network control. They become the operators of virtual networks. Thus, SPs can use their knowledge about their services and applications to implement advanced network control algorithms, which are designed to meet the service and application requirements. It is then the task of the InP to provide virtual networks to the SPs. SPs (tenants) might even create virtual networks by requesting resources from multiple InPs.

Generally, virtualization is construed differently among networking domains [Chowdhury and Boutaba, 2009]. Hence, different networking domains use varying technologies to realize virtual networks. For instance, VXLAN [Mahalingam et al., 2014], GRE [Farinacci et al., 2000], or GRE's NV variant NVGRE [Garg and Wang, 2015] are used in data centers to interconnect virtual machines of tenants. Techniques such as Multiprotocol Label Switching (MPLS) [Xiao et al., 2000, Rosen et al., 2001] create logically-isolated virtual networks on the IP layer based on tunneling technologies, while potentially relying on specialized networking hardware.

Full network virtualization comprises all physical resources that are needed to provide virtual networks with guaranteed and predictable performance. The ability to program virtual networks, e.g., by using SDN, is a further important key aspect of (full) network virtualization [Koponen et al., 2014]. Taking a look at Virtual Local Area Network (VLAN)-based virtualization without the ability of programming, tenants have no opportunity to instruct switches to make traffic steering decisions. However, to fully benefit from NV opportunities, tenants should obtain virtual network resources, including full views of network topologies and allocated networking resources, involving link data rates and network node resources, such as Central Processing Unit (CPU) or memory.

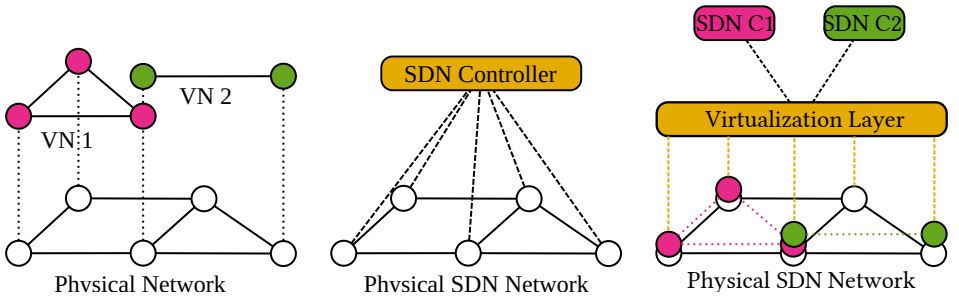
Providing isolated and programmable virtual networks has manifold advantages: first, network operators can design, develop, and test novel networking paradigms, without any constraints imposed by the currently deployed protocols or (Internet) architecture [Anderson et al., 2005]. Second, network systems that are designed to the demands of the served applications or users do not suffer from the overhead of unused network stacks or protocols. Furthermore, NV is seen as a key to provide predictable (guaranteed) network performance [Ballani et al., 2011]. As a consequence, SPs should be enabled to offer new services over existing infrastructures much faster with higher flexibility, i.e., ease to adapt their networks to changing user and service demands [Keller et al., 2012]. One way to offer this kind of feature is to use so-called network hypervisors, as introduced in the next section.

1.4. SDN Network Hypervisors

SDN network hypervisors implement the virtualization layer for virtualizing SDN networks. They provide the main network functions for virtualization of SDN networks. In this section, we explain how to virtualize SDN networks through a network hypervisor and its virtualization functions. We highlight the main functions that need to be implemented towards being compliant with the introduced abstraction demands of NV.

1.4.1. From Software-Defined Networks to Virtual Software-Defined Networks

Combining NV and SDN provides tenants the advantages of both concepts, i.e., flexible resource sharing by acquiring virtual networks, and programmability of the network resources by using SDN. With the introduced programmability of SDN, NV offering



(a) Virtual Networks (b) SDN Network (c) Virtualized SDN Network

Figure 1.3: Comparison of virtual networks, SDN network, and virtual SDN networks. Dotted lines on Fig. 1.3a indicate the embedding of the virtual nodes. Dashed lines in Fig. 1.3b illustrate the control connections between the SDN controller and the physical SDN network. Black dashed lines in Fig 1.3c show the connection between tenant controllers and the virtualization layer, while dashed colored lines show the connections between the virtualization layer and the physical SDN network. Dotted lines between virtual nodes on the physical network indicate the virtual paths between them.

virtual resource programmability towards tenants can now be put into effect [Jain and Paul, 2013]. Accordingly, NV is seen as one killer application of SDN [Drutskoy et al., 2013, Feamster et al., 2014], that is, it provides the programmability of virtual network resources. The result of combining NV and SDN are vSDNs sharing the same infrastructure.

Figure 1.3 illustrates the differences between virtual networks, software-defined networks, and virtual software-defined networks. Figure 1.3a shows the traditional view of virtual networks. Two virtual networks are hosted on a substrate network. The dashed lines illustrate the location of the virtual resources. The interconnection of the virtual nodes is determined by the path embedding or routing concept of the InP. A clear way how a tenant can control and configure its virtual network is not given. SDN provides one option for providing virtual network resource control and even configuration to tenants. Figure 1.3b illustrates how an SDN controller operates on to top of a physical SDN network. The SDN controller is located outside of the network elements. It controls the network based on a logically centralized view. Figure 1.3c shows the combination of NV and SDN. A virtualization layer is responsible

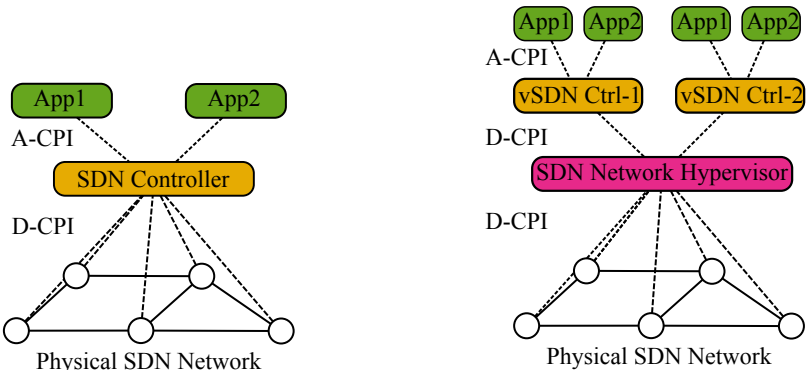
for managing the physical network. Besides, the virtualization layer orchestrates the control access among SDN controllers (here SDN C1 and C2) of tenants. As an example, tenant 1 (VN 1) has access to three network elements while tenant 2 (VN 2) has access to two network elements. Note the virtualization layer in the middle that is shared by both tenants.

1.4.2. SDN Controllers versus SDN Network Hypervisors

By adding a virtualization layer, i.e., a network hypervisor, on top of the networking hardware, multiple vSDN operating systems are alleviated to control resources of the same substrate network. This concept has been proposed by [Sherwood et al., 2009, 2010]. The network hypervisor interacts with the networking hardware via the D-CPI through an SDN protocol, e.g., OF. In case of NV, the hypervisor provides on top the same D-CPI interface towards virtual network tenants. This feature of the hypervisor to interface through multiple D-CPI with multiple virtual SDN controllers is seen as one of the defining features when virtualizing SDN networks.

Fig. 1.4 illustrates the difference between SDN networks and vSDNs in terms of their interfaces to tenants. In SDN networks (Fig. 1.4a), network applications are running on top of an SDN controller. The applications use the A-CPI of the controller to connect and communicate with the SDN controller.

For vSDNs, as depicted in Fig. 1.4b, the network hypervisor implements the virtualization layer. The tenants now communicate again via a D-CPI with the network hypervisor. Still, on top of the tenant controllers, applications communicate via the controllers' A-CPI interfaces during runtime. The hypervisor acts as a proxy: it intercepts the control messages between tenants and the physical SDN network. The hypervisor acts as the SDN controller towards the physical SDN network. It translates the control plane messages between the tenant SDN controllers and the physical



(a) Applications interact via A-CPI with SDN Controller. SDN controller interacts via D-CPI with physical SDN network.

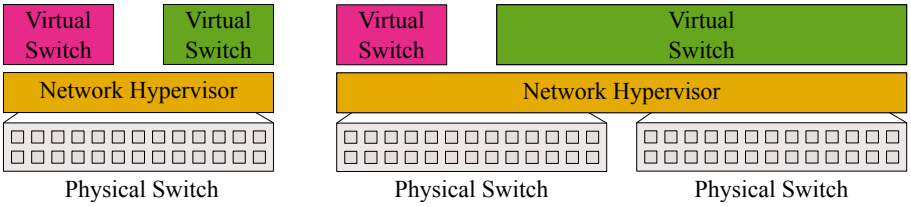
(b) Comparison of SDN and vSDNs, and respective interfaces. The tenant controllers communicate through the SDN network hypervisor with their virtual switches.

Figure 1.4: Comparison of SDN and Virtual Software-Defined Network (vSDN), and respective interfaces.

SDN network. Message translation is the main functional task a hypervisor has to accomplish.

1.4.3. SDN Network Hypervisors: Virtualization Tasks and Functions

Next to translating messages, SDN network hypervisors face many tasks when virtualizing SDN networks: they need to grant access to tenants, isolate the virtual networks on the data plane, avoid interference on the control plane, guarantee predictable network operation, grant adaptation capabilities etc. In the following, we outline the tasks of a network hypervisor to abstract (virtualize) SDN networking resources and to create isolated virtual SDN networks.



(a) Switch Partitioning (b) Switch Partitioning & Aggregation

Figure 1.5: Comparison between switch partitioning and switch partitioning & aggregation. Fig. 1.5a shows a physical switch partitioned into two virtual switches. Fig. 1.5b illustrates partitioning and aggregation. Here, the right virtual switch represents an aggregation among two physical switches.

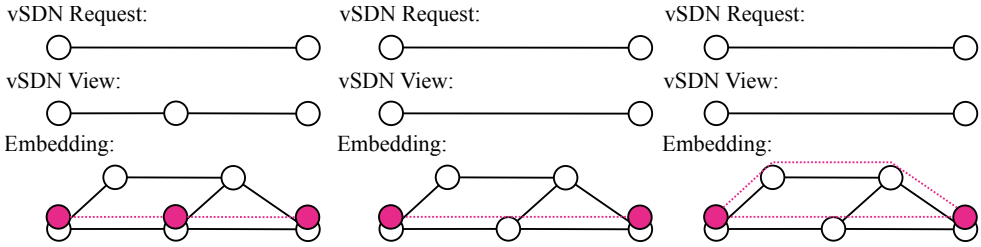
1.4.3.1. Abstraction

Overall, many tasks are affected by the realization of the main feature that hypervisors need to offer — the abstraction feature. Abstraction means "the act of considering something as a general quality or characteristic, apart from concrete realities, specific objects, or actual instances" [Corsini, 2002]. As abstraction is seen as a fundamental advantage of NV and SDN [Casado et al., 2014, 2010, Douglis and Krieger, 2013], an SDN network hypervisor should be able to abstract details of the physical SDN network. The degree of abstraction of the network representation determines also the *level of virtualization* [Chowdhury and Boutaba, 2008], which is provided by a network hypervisor. The available features and capabilities are directly communicated by a hypervisor towards the tenants.

Three SDN network abstraction features are seen as the basic building blocks of a virtualization layer for SDN: topology abstraction, physical node resource abstraction, and physical link resource abstraction.

Topology Abstraction

Topology abstraction involves the abstraction of topology information, i.e., the information about the physical nodes and links that tenants receive as their view of the topology. The actual view of tenants is defined by the mapping of the requested



(a) View abstraction with a 1-to-N link mapping and 1-to-1 node mapping, which involves the intermediate node. No path abstraction and no path splitting.

(b) View abstraction with 1-to-N link mapping and 1-to-1 node mapping, which does not involve the intermediate node. Path abstraction and no path splitting.

(c) Link abstraction with 1-to-N link mapping and 1-to-1 node mapping, which does not involve the intermediate nodes. Path abstraction and path splitting.

Figure 1.6: Comparison between link abstraction procedures. On top the requested virtual network. In the middle the provided view based on the embedding on the bottom.

nodes/links to the physical network and the abstraction level provided by the virtualization layer. Generally, we define the mapping of a virtual node/link to many physical nodes/links as a "1-to-N" mapping. A virtual node, for instance, can span across many physical nodes. In case a tenant receives a "1-to-N" mapping without abstraction, he has to do additional work; the tenant has to implement the forwarding of network packets on intermediate nodes by himself. When regarding links, while a tenant requests only a virtual link between two nodes, he receives a view also containing intermediate nodes to be managed by the tenant. In case nodes and links are mapped to only one physical instance, we call this a "1-to-1" mapping.

The provided information about nodes involves their locations and their interconnections through links. A virtual node can be realized on one ("1-to-1") or across many physical nodes ("1-to-N"). Fig. 1.5 illustrates the two cases. Fig. 1.5a shows an example where a switch is partitioned into multiple virtual instances. Each virtual switch is running on one physical switch only. As an example for node aggregation, i.e., where two physical instances are abstracted as one, a tenant operating a secure SDN network wants to operate incoming and outgoing nodes of a topology only. Thus, a physical

topology consisting of many nodes might be represented via "one big switch" [Monsanto et al., 2013, Chowdhury and Boutaba, 2008, Casado et al., 2010, Jin et al., 2015]. As illustrated in Fig. 1.5b, the right green switch is spanning two physical instances. The network hypervisor aggregates the information of both physical switches into one virtual switch.

Similar, different mapping and abstraction options for virtual links or paths exist. Physical links and paths are abstracted as virtual links. Realizations of virtual links can consist of multiple hops, i.e., physical nodes. A tenant's view might contain the intermediate nodes or not. An example where intermediate nodes are not hidden from the view is shown by Fig. 1.6a. The request involves two nodes and a virtual path connecting them. As the physical path realization of the virtual link spans an intermediate node, the view depicts this node. Fig. 1.6b shows a view that hides the intermediate node. Besides, a virtual path can also be realized via multiple physical paths, which is illustrated by Fig. 1.6c. For this, the infrastructure needs to provide path splitting techniques [Yu et al., 2010].

Physical Node Resource Abstraction

In virtual networks, CPU and memory are usually considered as the main network node resources. For SDN networks, memory is additionally differentiated from the space to store matching entries for network flows, i.e., flow table space. While flow table space is only used for realizing the match-and-action paradigm, memory might be needed to realize a network stack (e.g., a tenant NOS on a device in case of a distributed virtualization architecture) or for realizing network management functionality (e.g., a byte counter for charging). CPU resource information can be abstracted in different ways. It can be represented by the number of available CPU cores or the amount of percentage of CPU. Similar, tenants can receive a concrete number or partitions of memories. Flow table resources involve, e.g., the number of flow tables or the number of TCAMs [Pagiamtzis and Sheikholeslami, 2006, Panigrahy and Sharma, 2002]. For

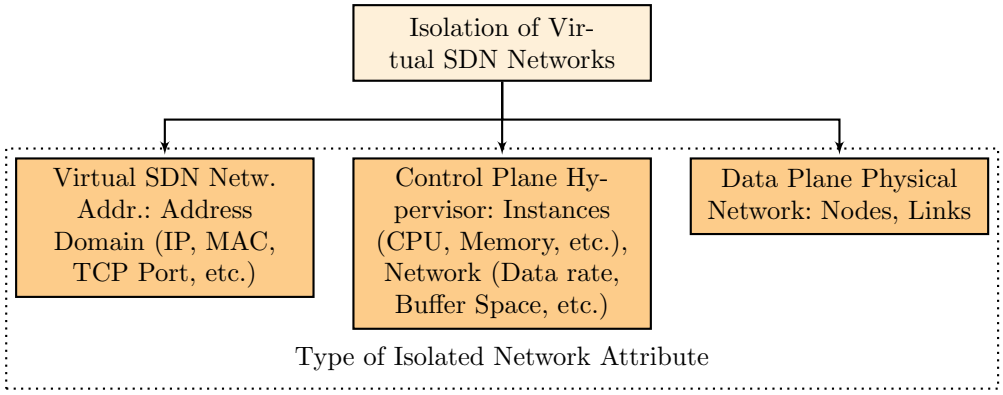


Figure 1.7: Network hypervisors isolate three network virtualization attributes: control plane, data plane, and vSDN addressing.

instance, if switches provide multiple table types such as physical and software tables, network hypervisors need to reserve parts of these tables according to the demanded performance. Again, based on the level of abstraction, the different table types (software or hardware) are abstracted from tenants' views in order to lower operational complexity for them.

Physical Link Resource Abstraction

Physical link resources involve the data rate, the available queues, the queue attributes such as different priorities in case of a priority queuing discipline, as well as the link buffers. Tenants might request virtual networks with delay or loss guarantees. To guarantee delay and upper loss bounds, substrate operators need to operate queues and buffers for the tenants. That is, the operation of queues and buffers is abstracted from the tenant. However, tenants might even request to operate queues and buffers themselves. For instance, the operation of meters, which rely on buffers, is a fundamental feature of recent OF versions. To provide tenants with their requested modes of operations, like metering, substrate operators need to carefully manage the physical resources.

1.4.3.2. Isolation

Network hypervisors should provide isolated virtual networks for tenants while trying to perceive the best possible resource efficiency out of the physical infrastructure. While tenants do not only demand physical resources for their operations, physical resources are also needed to put virtualization into effect, e.g., to realize isolation. Physical resources can be classified into three main categories as depicted in Fig 1.7: the provided addressing space, the control plane and the data plane resources.

vSDN Addressing Isolation

With virtualized SDN networks, tenants should receive the whole programmability of an SDN network. This means, tenants should be free to address flows according to their configurations and demands. Accordingly, techniques need to provide unique identification of the flows of different tenants. In a non-virtualized SDN network, the amount of addressable flow space is limited by the physical infrastructure attributes, i.e., the type of network (e.g., layer 2 only) and the used protocols (e.g., MPLS as the only tunneling protocol). The available headers and their configuration possibilities determine the amount of available flows. In virtualized SDN networks, more possibilities are available: e.g., if the vSDN topologies of two tenants do not overlap physically, the same address space in both vSDNs would be available. Or if tenants request lower OF versions than the deployed ones, extra header fields added by higher OF versions could also be used for differentiation. For instance, OF 1.1 introduced MPLS that can be used to distinguish tenants who request only OF 1.0.

Control Plane Isolation

In SDN networks, it is well known that the control plane performance affects the data plane performance [Tootoonchian and Ganjali, 2010]. In addition, execution platforms (i.e., their CPU, memory, etc.), which host SDN controllers, directly influence the control plane performance [Kuzniar et al., 2014, Rotsos et al., 2012, Tootoonchian

and Ganjali, 2010, Tootoonchian et al., 2012]. For instance, when an SDN controller is under heavy load, i.e., available CPUs run at high utilization, OF control packet processing might take longer. As a result, forwarding setups on switches might be delayed. On the other hand, also the resources of the control channels and switches can impact the data plane performance. In traditional routers, the routing processor running the control plane logic communicates with the data plane elements (e.g., Forwarding Information Base (FIB)) over a separate bus (e.g., a PCI bus). In SDN, controllers are running as external entities with their control channels at the mercy of the network. If a control channel is currently utilized by many OF packets, delay and even loss might occur, which can lead to delayed forwarding decisions. On switches, so called agents manage the connection towards the external control plane. Thus, switch resources consumed by agents (node CPU & memory, and link buffer & data rate) can also impact the performance of the control plane.

The overall performance perceived by tenant controllers is determined by many factors: the physical network, i.e., node and link capabilities; the processing speed of a hypervisor being determined by the CPU of its host; the control plane latency between tenant controllers and hypervisors is determined by the available data rate of the connecting network. What comes in addition is the potential drawback of sharing resources: performance degradation due to resource interference. Resource interference happens, for instance, when misconfigured controllers overwhelm virtualization infrastructures with too many control messages. Without isolation, the overload generated by a tenant can then degrade the perceived performance of other tenants, which degrades the data plane performance. To provide predictable and guaranteed performance, a resource isolation scheme for tenants needs to carefully allocate all involved resources at the data and the control plane.

Data Plane Isolation

The main resources of the data plane are node CPUs, node hardware accelerators, node flow table space, link buffers, link queues, and link data rates. Node resources need to be reserved and isolated between tenants for efficient forwarding and processing of the tenants' data plane traffic. On switches, for instance, different sources can utilize their CPU resources: (1) generation of SDN messages, (2) processing data plane packets on the switches' CPUs, i.e., their "slow paths", and (3) switch state monitoring and storing [Sherwood et al., 2009]. As there is an overhead of control plane processing due to involved networking operations, i.e., exchanging control messages with the network hypervisor, virtualizing SDN networks requires an even more thorough allocation of all involved resources. Besides, the utilization of the resources might change under varying workloads. Such variations need also be taken into account when allocating resources.

In order to successfully accomplish all these tasks, many challenges need to be solved in different research areas: e.g., architecture design of hypervisors providing predictable and guaranteed performance as well as solving algorithmically hard resource allocation problems for provisioning isolated virtual network resources.

1.5. A Network Hypervisor example: FlowVisor

FlowVisor (FV) [Sherwood et al., 2009] has been the first hypervisor for virtualizing OF-based software-defined networks, enabling sharing of SDN networking resources between multiple SDN controllers. In this section, we will briefly describe how FlowVisor addresses some of the aforementioned attributes of network hypervisors. For a more comprehensive elaboration and comparison of different network hypervisor implementation, we refer to our survey in [Blenk et al., 2016].

Architecture. FV is a software network hypervisor and can run stand-alone on any commodity server or inside a virtual machine. Sitting between the tenant SDN controllers and the SDN networking hardware, FV processes the control traffic between tenants from and to the SDN switches. FV further controls the view of the network towards the tenants, i.e., it can abstract switch resources. FV supports OF 1.0 Open Networking Foundation (ONF) [2009].

Flowspace. FV defines the term *flowspace*. A flowspace for a tenant describes a possibly non-contiguous sub-space of the header field space of an OF-based network. Flowspaces between tenants should not overlap; therefore, FV guarantees isolated flowspaces for tenants. If tenants try to address flows outside their flowspaces, FV rewrites the packet headers. If packet headers cannot be rewritten, FV sends an OF error message. FV distinguishes shared from non-shared switches between tenants for flowspace isolation. For shared switches, FV ensures that tenants cannot share flowspaces, i.e., packet headers. For non-shared switches, flowspaces can be reused among tenants as those are physically isolated.

Bandwidth Isolation. While OF in its original version has not provided any QoS techniques for data plane isolation, FV realized data plane isolation by using VLAN priority bits in data packets. Switches are configured out-of-band by a network administrator to make use, if available, of priority queues. FV, then, rewrites tenant rules to further set the VLAN priorities of the tenants' data packets [Sofia, 2009]. The so called VLAN Priority Code Point (PCP) specifies the 3-bit VLAN PCP field for mapping to eight distinct priorities. As a result, data packets can be mapped to the configured queues; hence, they receive different priorities.

Topology Isolation. FV isolates the topology in a way that tenants only see the ports and switches that are part of their slices. For this, FV edits and forwards OF messages related to a specific slice per tenant only.

Switch CPU Isolation. In contrast to legacy switches, where control does not need an outside connection, OF switches can be overloaded by the amount of OF messages they need to process. The reason is that the processing needed for external connec-

tions adds overhead on switch CPUs. In detail, OF agents need to encapsulate and decapsulate control messages from and to TCP packets. Thus, in case a switch has to process too many OF messages, its CPU might become overloaded. In order to ensure that switch CPUs are shared efficiently among all slices, FV limits the rate of messages sent between SDN switches and tenant controllers. For this, FV implements a software-based message policing (briefly software isolation). This message policing needs to be specified and installed with the respective slice configuration.

Flow Entries Isolation. To efficiently realize the match-plus-action paradigm, SDN switches store match-plus-action instructions in flow tables. In case tenants demand Gigabit transmissions, this lookup needs to perform in nanoseconds. A special memory that operates in such timescales is TCAM [Fang Yu et al., 2004]. However, due to its cost, TCAM space is limited. FV reserves each tenant parts of the table space for operation. For this, FV keeps track of the used table space per tenant. Tenants cannot use table space of other tenants in case they demand more table space. FV sends tenants, which exceed their capacities, an indicator message telling that the flowspace is full.

Control Channel Isolation. To distinguish between vSDNs, each vSDN is assigned its distinct transaction identifier. If tenants use the same identifier, FV rewrites the OF transaction identifiers to make them distinct again. It further modifies controller buffer accesses and status messages to put isolation into effect.

1.6. Summary

In this cahper we have described the necessity of network abstraction to address hetereogeneous application requirement on the same network substrate. The concept of network virtualization provides an abstraction of the network topology and of the network node and link resources to offer an individual network view, a virtual network, to a network application provider. Several virtual networks may run on the same net-

work substrate and have to be isolated from each other. A network hypervisor as an intermediary layer takes care of providing the abstracted view and of the isolation. Further network abstraction for adaptation of (virtual) networks is provided by the concept of Software Defined Networking (SDN). SDN supports a cetralized view on a network to program e.g. individual forwarding rules. As the combination of network virtualization and SDN realizes full network abstraction and programmability, this chapter has used this combination to illustrate and explain network virtualization. Without loss of generality, the network virtualization concept based on the described hypervisor concepts can also be applied without SDN. However, when dealing with network virtualization, it is important to understand both concepts and the difference between network hypervisor and SDN controller, in particular.

Bibliography

RYU SDN framework, http://osrg.github.io/ryu.

Patrick Kwadwo Agyapong, Mikio Iwamura, Dirk Staehle, Wolfgang Kiess, and Anass Benjebbour. Design considerations for a 5G network architecture. *IEEE Commun. Mag.*, 52(11):65–75, 2014. ISSN 01636804. doi: 10.1109/MCOM.2014.6957145.

Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, William Snow, and Guru Parulkar. OpenVirteX: a network hypervisor. In *Proc. USENIX Open Netw. Summit (ONS)*, pages 1–2, Santa Clara, CA, March 2014a.

Ali Al-Shabibi, Marc De Leenheer, and Bill Snow. OpenVirteX: Make your virtual SDNs programmable. In *Proc. ACM Workshop on Hot Topics in Softw. Defined Netw.*, HotSDN '14, pages 25–30. ACM, August 2014b. ISBN 978-1-4503-2989-7. doi: 10.1145/2620728.2620741. URL http://doi.acm.org/10.1145/2620728.2620741.

Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38(4):34–41, April 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.136.

Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *Proc. ACM SIGCOMM*, SIGCOMM '11, pages 242–253. ACM, August 2011. ISBN 978-1-4503-0797-0. doi: 10.1145/2018436.2018465. URL http://doi.acm.org/10.1145/2018436.2018465.

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proc. ACM Symp. on Operating Systems*, SOSP '03, pages 164–

177. ACM, 2003. ISBN 1-58113-757-5. doi: 10.1145/945445.945462. URL http://doi.acm.org/10.1145/945445.945462.

Md Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: A survey. *IEEE Commun. Surveys & Tutorials*, 15 (2):909–928, 2013. ISSN 1553-877X. doi: 10.1109/SURV.2012.090512.00043.

Theophilus Benson, Aditya Akella, and David a. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. ACM SIGCOMM IMC*, page 267, New York, New York, USA, November 2010. ACM Press. ISBN 9781450304832. doi: 10.1145/1879141.1879175. URL http://portal.acm.org/citation.cfm?doid=1879141.1879175.

Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, William Snow, Guru Parulkar, Brian O'Connor, and Pavlin Radoslavov. ONOS: Towards an open, distributed SDN OS. In *Proc. ACM Workshop on Hot Topics in Softw. Defined Netw.*, pages 1–6, Chicago, Illinois, USA, August 2014. doi: 10.1145/2620728.2620744.

Andreas Blenk, Arsany Basta, Martin Reisslein, and Wolfgang Kellerer. Survey on Network Virtualization Hypervisors for Software Defined Networking. *IEEE Commun. Surveys & Tutorials*, 18(1):655–685, 2016. ISSN 1553-877X. doi: 10.1109/COMST.2015.2489183. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=7295561http://ieeexplore.ieee.org/document/7295561/.

Martín Casado, Teemu Koponen, Rajiv Ramanathan, and Scott Shenker. Virtualizing the network forwarding plane. In *Proc. ACM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, PRESTO '10, pages 8:1–8:6. ACM, November 2010. ISBN 978-1-4503-0467-2. doi: 10.1145/1921151.1921162. URL http://doi.acm.org/10.1145/1921151.1921162.

Martin Casado, Nate Foster, and Arjun Guha. Abstractions for software-defined networks. *Communications of the ACM*, 57(10):86–95, September 2014. ISSN

00010782. doi: 10.1145/2661061.2661063. URL http://dl.acm.org/citation.cfm?doid=2661061.2661063.

N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54:862–876, 2008.

N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *IEEE Commun. Mag.*, 47(7):20–26, July 2009. ISSN 0163-6804. doi: 10.1109/MCOM.2009.5183468. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5183468.

R. J. Corsini. *The Dictionary of Psychology*. Psychology Press, 2002.

Fred Douglis and Orran Krieger. Virtualization. *IEEE Internet Computing*, 17(2):6–9, 2013. doi: 10.1109/MIC.2013.42.

Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, 17(2):20–27, 2013.

David Erickson. The beacon openflow controller. In *Proc. ACM Workshop on Hot Topics in Softw. Defined Netw.*, HotSDN '13, pages 13–18. ACM, August 2013. ISBN 978-1-4503-2178-5. doi: 10.1145/2491185.2491189. URL http://doi.acm.org/10.1145/2491185.2491189.

Jeffrey Erman and K.K. Ramakrishnan. Understanding the super-sized traffic of the super bowl. In *Proc. ACM SIGCOMM IMC*, IMC '13, pages 353–360. ACM, 2013. ISBN 978-1-4503-1953-9. doi: 10.1145/2504730.2504770. URL http://doi.acm.org/10.1145/2504730.2504770.

Fang Yu, R.H. Katz, and T.V. Lakshman. Gigabit rate packet pattern-matching using TCAM. In *Proc. IEEE ICNP*, pages 174–183, October 2004. doi: 10.1109/ICNP.2004.1348108.

Dino Farinacci, Tony Li, Stan Hanks, David Meyer, and Paul Traina. Generic

routing encapsulation (gre). RFC 2784, RFC Editor, March 2000. URL http://www.rfc-editor.org/rfc/rfc2784.txt. http://www.rfc-editor.org/rfc/rfc2784.txt.

Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *ACM SIGCOMM Computer Commun. Rev.*, 37(1):61, January 2007. ISSN 01464833. doi: 10.1145/1198255.1198265. URL http://portal.acm.org/citation.cfm?doid=1198255.1198265.

Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN: An intellectual history of programmable networks. *ACM SIGCOMM Computer Commun. Rev.*, 44 (2):87–98, April 2014.

Jose Luis Garcia-Dorado, Alessandro Finamore, Marco Mellia, Michela Meo, and Maurizio M. Munafo. Characterization of ISP Traffic: Trends, User Habits, and Access Technology Impact. *IEEE Trans. on Netw. and Serv. Manage.*, 9(2):142–155, June 2012. ISSN 1932-4537. doi: 10.1109/TNSM.2012.022412.110184. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6158423.

Pankaj Garg and Yu-Shun Wang. Nvgre: Network virtualization using generic routing encapsulation. RFC 7637, RFC Editor, September 2015.

Vinicius Gehlen, Alessandro Finamore, Marco Mellia, and Maurizio M. Munafò. *Uncovering the Big Players of the Web*, pages 15–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-28534-9. doi: 10.1007/978-3-642-28534-9_2. URL https://doi.org/10.1007/978-3-642-28534-9_2.

Robert P Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, 1974.

Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Commun. Rev.*, 38(3):105–110, 2008.

Raj Jain and Subharthi Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Commun. Mag.*, 51(11):24–31, November 2013.

Xin Jin, Jennifer Gossels, Jennifer Rexford, and David Walker. CoVisor: A compositional hypervisor for software-defined networks. In *Proc. USENIX Symp. NSDI*, NSDI'15, pages 87–101. USENIX Association, May 2015. ISBN 978-1-931971-218. URL http://dl.acm.org/citation.cfm?id=2789770.2789777.

Eric Keller, Dushyant Arora, Soudeh Ghorbani, Matt Caesar, and Jennifer Rexford. Live Migration of an Entire Network (and its Hosts). *Princeton University Computer Science Technical Report*, pages 109–114, 2012. doi: 10.1145/2390231.2390250. URL http://dl.acm.org/citation.cfm?doid=2390231.2390250ftp://ftp.cs.princeton.edu/techreports/2012/926.pdf.

T. Koponen, M. Casado, P.S. Ingram, W.A. Lambeth, P.J. Balland, K.E. Amidon, and D.J. Wendlandt. Network virtualization, US Patent 8,959,215, February 2015.

Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *Proc. USENIX Conf. OSDI*, OSDI'10, pages 351–364. USENIX Association, October 2010. URL http://dl.acm.org/citation.cfm?id=1924943.1924968.

Teemu Koponen, Keith Amidon, Peter Balland, Martin Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. Network virtualization in multi-tenant datacenters. In *Proc. USENIX Symp. NSDI*, NSDI'14, pages 203–216. USENIX Association, April 2014. ISBN 978-1-931971-09-6. URL http://dl.acm.org/citation.cfm?id=2616448.2616468.

Diego Kreutz, Fernando MV Ramos, PE Verissimo, C Esteve Rothenberg, Siamak

Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proc. IEEE*, 103(1):14–76, 2015.

Maciej Kuzniar, Peter Peresini, and Dejan Kostic. What you need to know about SDN control and data planes. Technical report, EPFL, TR 199497, 2014.

Maciej Kuźniar, Peter Perešíni, and Dejan Kostić. *What You Need to Know About SDN Flow Tables*, pages 347–359. Springer International Publishing, Cham, 2015. ISBN 978-3-319-15509-8. doi: 10.1007/978-3-319-15509-8_26. URL https://doi.org/10.1007/978-3-319-15509-8_26.

Yunfa Li, Wanqing Li, and Congfeng Jiang. A survey of virtual machine system: Current technology and future trends. In *Proc. IEEE Int. Symp. on Electronic Commerce and Security (ISECS)*, pages 332–336, July 2010. doi: 10.1109/ISECS.2010.80.

M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. RFC 7348, RFC Editor, August 2014. URL http://www.rfc-editor.org/rfc/rfc7348.txt. http://www.rfc-editor.org/rfc/rfc7348.txt.

Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Commun. Rev.*, 38(2):69–74, 2008.

Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing Software-Defined Networks. In *Proc. USENIX Symp. NSDI*, pages 1–13, Lombard, IL, April 2013. USENIX Association. ISBN 978-1-931971-00-3. URL https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/monsanto.

Open Networking Foundation (ONF). OpenFlow Switch Specifications 1.0 (ONF TS-001), December 2009. https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf.

Open Networking Foundation (ONF). OpenFlow Switch Specifications 1.1.0 (ONF TS-002), February 2011a. https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf.

Open Networking Foundation (ONF). OpenFlow Switch Specifications 1.2 (ONF TS-003), October 2011b. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf.

Open Networking Foundation (ONF). OpenFlow Switch Specifications 1.3.0 (ONF TS-006), October 2012. https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

Open Networking Foundation (ONF). OpenFlow Switch Specifications 1.4 (ONF TS-012), October 2013. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

Open Networking Foundation (ONF). OpenFlow Switch Specifications 1.5.0 (ONF TS-020), December 2014a. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

Open Networking Foundation (ONF). SDN Architecture Overview, Version 1.0, ONF TR-502, June 2014b. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.

Open Networking Foundation (ONF). SDN Architecture Overview, Version 1.1, ONF TR-504, November 2014c. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf.

OpenDaylight. A linux foundation collaborative project, 2013. URL http://www.opendaylight.org.

Kostas Pagiamtzis and Ali Sheikholeslami. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE J. Solid-State Circuits*, 41(3): 712–727, 2006.

Rina Panigrahy and Samar Sharma. Reducing TCAM power consumption and increasing throughput. In *Proc. IEEE High Perf. Interconnects*, pages 107–112, August 2002. doi: 10.1109/CONECT.2002.1039265.

E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. RFC 3031, RFC Editor, January 2001. URL http://www.rfc-editor.org/rfc/rfc3031.txt. http://www.rfc-editor.org/rfc/rfc3031.txt.

Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W. Moore. *OFLOPS: An Open Framework for OpenFlow Switch Evaluation*, pages 85–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-28537-0. doi: 10.1007/978-3-642-28537-0_9. URL https://doi.org/10.1007/978-3-642-28537-0_9.

Jyotiprakash Sahoo, Subasish Mohapatra, and Radha Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Proc. IEEE Int. Conf. on Computer and Netw. Techn. (ICCNT)*, pages 222–226, April 2010. doi: 10.1109/ICCNT.2010.49.

Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick Mckeown, and Guru Parulkar. FlowVisor: A network virtualization layer. Technical report, OpenFlow Consortium, 2009.

Rob Sherwood, Jad Naous, Srinivasan Seetharaman, David Underhill, Tatsuya Yabe, Kok-Kiong Yap, Yiannis Yiakoumis, Hongyi Zeng, Guido Appenzeller, Ramesh Johari, Nick McKeown, Michael Chan, Guru Parulkar, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, and Masayoshi

Kobayashi. Carving research slices out of your production networks with OpenFlow. *ACM SIGCOMM Computer Commun. Rev.*, 40(1):129–130, January 2010.

James E Smith and Ravi Nair. The architecture of virtual machines. *IEEE Computer*, 38(5):32–38, 2005.

Rute C Sofia. A survey of advanced ethernet forwarding approaches. *IEEE Commun. Surveys & Tutorials*, 11(1):92–115, 2009.

A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *Proc. USENIX Wkshp. on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 10–10, Berkeley, CA, USA, April 2012. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2228283.2228297.

Amin Tootoonchian and Yashar Ganjali. HyperFlow: A distributed control plane for OpenFlow. In *Proc. USENIX Internet Network Management Conf. on Research on Enterprise Netw.*, INM/WREN'10, pages 3–3. USENIX Association, April 2010. URL http://dl.acm.org/citation.cfm?id=1863133.1863136.

J.S. Turner and D.E. Taylor. Diversifying the Internet. In *Proc. IEEE Globecom*, volume 2, pages 6 pp.–760, December 2005. doi: 10.1109/GLOCOM.2005.1577741.

Xipeng Xiao, Alan Hannan, Brook Bailey, and Lionel M Ni. Traffic engineering with MPLS in the internet. *IEEE Network*, 14(2):28–33, 2000.

Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with DIFANE. *ACM SIGCOMM Computer Commun. Rev.*, 40 (4):351–362, October 2010.

Zhaoning Zhang, Ziyang Li, Kui Wu, Dongsheng Li, Huiba Li, Yuxing Peng, and Xicheng Lu. VMThunder: fast provisioning of large-scale virtual machine clusters. *IEEE Trans. Parallel and Distr. Systems*, 25(12):3328–3338, December 2014. ISSN 1045-9219. doi: 10.1109/TPDS.2014.7.