Security Analysis of Network Virtualization Implemented with Open Virtual Network (OVN)

A Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Engineering

in

Computer Science & Engineering

at

Department of Computer Science & Engineering

Jadavpur University, Kolkata

by

Soumya Banerjee

(Registration Number- 154156 of 2020-21)

(Class Roll Number- 002010502032)

(Examination Roll Number- M4CSE22032)

Under the Guidance of

Prof. Chandan Mazumdar

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University, Kolkata - 700032

July, 2022

JADAVPUR UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate of Recommendation

This is to certify that the thesis entitled **"Security Analysis of Network Virtualization Implemented using Open Virtual Network"** has been satisfactorily carried out by **Soumya Banerjee (University Registration Number: - 154156 of 2020-21, Class Roll Number: - 002010502032, Examination Roll Number: - M4CSE22032)**. It is a bona-fide record of workcarried out under my guidance and supervision and be accepted in partialfulfillment of the requirement for the degree of **Master of Engineering in Computer Science and Engineering** from the **Department of ComputerScience & Engineering, Faculty of Engineering and Technology, Jadavpur University**, Kolkata - 700032.

Prof. Chandan Mazumdar
(Thesis Supervisor)

Department of Computer Science and Engineering
Jadavpur University, Kolkata, West Bengal, India - 700032

Prof. (Dr.) Anupam Sinha

HOD, Department of Computer Science and Engineering
Jadavpur University, Kolkata, West Bengal, India - 700032

Prof. Chandan Mazumdar

Dean, Faculty of Engineering and Technology
Jadavpur University, Kolkata, West Bengal, India - 700032

JADAVPUR UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate of Approval

This is to certify that the thesis entitled **"Security Analysis of Network Virtualization Implemented using Open Virtual Network"** is a bona-fide record of work carried out by **Soumya Banerjee (University Registration Number :- 154156 of 2020-21, Class Roll Number :- 002010502032, Examination Roll Number :- M4CSE22032)** in partial fulfillment of the requirements for the award of the degree of **Master of Engineering in Computer Science and Engineering** from the **Department of Computer Science & Engineering, Faculty of Engineering and Technology, Jadavpur University**, Kolkata - 700032, during the period of July 2021 to July 2022. It is understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion withdrawn therein, but approve the thesis only for the purpose for which it has been submitted.

_____

Signature of Examiner

Date :-

Name :-

_____

Signature of Supervisor

Date :-

Name :-

JADAVPUR UNIVERSITY

188, Raja S.C. Mallick Rd, Kolkata - 700032

Declaration of Authorship

I, hereby declare that the work described in this thesis titled **"Security Analysis of Network Virtualization Implemented using Open Virtual Network"** is entirely my own and in accordance to the requirements of my Master of Engineering in Computer Science and Engineering studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials that are not original to this work

_____

Name :- SOUMYA BANERJEE

Registration Number :- 154156 of 2020-21

Class Roll Number :- 002010502032

Examination Roll Number :- M4CSE22032

Acknowledgement

The satisfaction and euphoria that accompanies the successful completion of this task would be incomplete without the mention of the people who made it possible. Their constant guidance and encouragement crowned my effort with success.

I would like to extend my sincerest gratitude to my honorable thesis guide, Prof. Chandan Mazumdar, from the Department of Computer Science and Engineering, Jadavpur University, Kolkata, for his invaluable and meticulous guidance, constant encouragement, and motivating words during the entire period of my research work which culminated into my master's thesis.

I would like to thank all the professors of the Department of Computer Science and Engineering, Jadavpur University, Kolkata for the guidance they provided me throughout the duration of the Master of Computer Application course.

A special note of thanks goes to Prof. Anupam Sinha, Head, Department of Computer Science and Engineering, Jadavpur University.

Finally I would like to thank my parents and my family for being a constant source of motivation and guidance throughout this period.

Name :- SOUMYA BANERJEE

Registration Number :- 154156 of 2020-21

Class Roll Number :- 002010502032

Examination Roll Number :- M4CSE22032

Abstract

Network Virtualization (NV) refers to abstracting network resources that were traditionally delivered in hardware to software. NV can combine multiple physical networks into one virtual, software-based network, or it can divide one physical network into separate, independent virtual networks. Network virtualization allows the creation of multiple independent virtual network instances on top of a single physical substrate. This is made possible by instantiating one or more virtual routers on physical devices and establishing virtual links between these routers, forming topologies that are not limited by the structure of the physical network. In addition to the ability to create different topological structures, virtual networks are also not bound by other characteristics of the physical network, such as its protocol stack.

One of such Network Virtualization mechanism is known as Open Virtual Network (OVN) which is a series of daemons which can convert CMS originated virtual network configuration into physical flow rules. OVN uses Open vSwitch for its switching fabric and uses tunnels to provide the logical/physical separation and Geneve protocol for encapsulation mechanism and allows the binding of physical network elements to their logical counterparts. In the following we discuss about Open Virtual Network, its architecture, its working mechanism and try to map the security mechanisms, configuration methodologies and evidence collection mechanisms against our required security properties.

Contents

List of Figures and Tables

List of Figures:

List of Tables:

Chapter 1

Introduction

Applications have historically been deployed by both large and small organizations on "bare metal" servers with operating systems already installed. Multiple virtual machines (VMs) on the same physical server have lately gained popularity due to cost savings and flexibility. Due to their lighter weight than virtual machines (VMs), quicker start-up and shutdown times, and capacity to package application binaries and their dependencies/libraries into standalone units for seamless mobility, containers have also become more and more popular for application deployment in recent years. An image registry (such as Kubernetes Hub) and a code repository (such as GitHub) are both components of a typical container ecosystem where container images are built from source code and libraries and deployed as application containers. There are a number of platforms that use containers, but Docker, Kubernetes, and Amazon ECS, etc. However, the widespread use of containers has resulted in several security issues, including attackers hacking credentials, source codes, and sensitive data from image repositories and code repositories, launching DoS attacks on application containers, and gaining root access to misuse the underlying host resources, to name a few.

1.1. Virtualization

Virtualization typically refers to the creation of virtual machine that can virtualize all of the hardware resources, including processors, memory, storage, and network connectivity [1]. With the virtualization, physical hardware resources can be shared by one or more virtual machines. According to the requirements from Popek and Goldberg [1], there are three aspects to satisfy the virtualization. First, the virtualization should provide an equivalent environment to run a program compared to a native system. If the program shows a different behavior under the virtualization, it may not be eligible as a virtualized environment. The virtualization also needs to provide a secured control of virtualized resources. Having a full control of resources is important to protect data and resources on each virtual environment from any threats or performance interference in sharing physical resources. Virtualization often expects performance degradation due to the additional tasks for virtualization, but good performance should be achieved with a software or hardware support in handling privileged instructions. With these requirements, efficient virtualization is guaranteed. In the following section, different types of hypervisors are explained with the implementation level of virtualization. Virtualized resource is also presented in CPU, memory and I/O transactions.

Virtualization uses software to create an abstraction layer over computer hardware that allows the hardware elements of a single computer—processors, memory, storage and more to be divided into multiple virtual computers, commonly called virtual machines (VMs). Each VM runs its own operating system (OS) and behaves like an independent computer, even though it is running on just a portion of the actual underlying computer hardware.

Virtualization brings several benefits to data center operators and service providers:

Resource efficiency: Before virtualization, each application server required its own dedicated physical CPU—IT staff would purchase and configure a separate server for each application they wanted to run. In contrast, server virtualization lets you run several applications—each on its own VM with its own OS—on a single physical computer without sacrificing reliability. This enables maximum utilization of the physical hardware's computing capacity.

Easier management: Replacing physical computers with software-defined VMs makes it easier to use and manage policies written in software. This allows you to create automated IT service management workflows. For example, scheduled cron jobs and remote configuration changes. Policies can even increase resource efficiency by retiring unused virtual machines to save on space and computing power.

Minimal downtime: OS and application crashes can cause downtime and disrupt user productivity. Admins can run multiple redundant virtual machines alongside each other and failover between them when problems arise while running physical redundant servers are much more expensive.

Faster provisioning: Buying, installing, and configuring hardware for each application is time-consuming. Provided that the hardware is already in place, provisioning virtual machines to run all your applications is significantly faster.

There are different types of virtualization. Some of them are-

- Desktop virtualization
- Network virtualization
- Storage virtualization
- Data virtualization
- Application virtualization
- Datacenter virtualization
- CPU virtualization
- GPU virtualization
- Linux virtualization
- Cloud virtualization

1.2. Network Virtualization

Network virtualization represents the administration and monitoring of an entire computer network as a single administrative entity from a single software-based administrator's console.

Network virtualization can include storage virtualization, which contains managing all storage as an individual resource. Network virtualization is created to enable network optimization of data transfer rates, flexibility, scalability, reliability, and security. It automates many network management functions, which disguise a network's true complexity. All network servers and services are considered as one pool of resources, which can be used independently of the physical elements.

Advantages of Network Virtualization

The advantages of network virtualization are as follows –

- **Lower hardware costs** −With network virtualization, entire hardware costs are reduced, while providing a bandwidth that is more efficient.
- **Dynamic network control** − Network virtualization provides centralized control over network resources, and allows for dynamic provisions and reconfiguration. Also, computer resources and applications can connect with virtual network resources precisely. This also enables for optimization of application support and resource utilization.
- **Rapid scalability** − Network virtualization generated an ability to scale the network rapidly either up or down to handle and make new networks on-demand. This is a valuable device as enterprises transform their IT resources to the cloud and shift their model to an 'as a service'.

Types of Network Virtualization

The types of network virtualization are as follows −

- **Network Virtualization** − Network virtualization is a technique of combining the available resources in a network by splitting up the available bandwidth into different channels, each being separate and distinguished.
- **Server Virtualization** −This technique is the masking of server resources. It simulates physical servers by transforming their identity, numbers, processors, and operating frameworks. This spares the user from continuously managing complex server resources. It also makes a lot of resources available for sharing and utilizing, while maintaining the capacity to expand them when needed.
- **Data Virtualization** − This type of cloud computing virtualization technique is abstracting the technical details generally used in data management, including location, performance, or format, in favor of broader access and more resiliency that are directly related to business required.
- **Application Virtualization** − Software virtualization in cloud computing abstracts the application layer, separating it from the operating framework.

1.3. Containerization

Since the early days of mainframe machines in the late 1960s, the hypervisor, which is a component of the operating system, has been in charge of managing the life cycle of virtual machines. The majority of current virtualization systems, such Xen and KVM, have their roots in that time period. Around 2005, the need to properly manage and exploit the ever expanding clusters of computational resources was the primary driver behind the widespread adoption of these virtualization technologies. For those who were concerned about security, the added layer of protection between the host OS and the virtual machine was a good selling point, albeit there were security issues as with any other newly adopted technology.

Nevertheless, the adoption [9] of full virtualization and para-virtualization significantly improved the way servers are utilized and applications provisioned. In fact, virtualization such as KVM and Xen is still widely used today, especially in multitenant clouds and cloud technologies such as OpenStack.

Hypervisors provide the following benefits, in the context of the problems outlined earlier:

- Ability to run different operating systems on the same physical server
- More granular control over resource allocation
- Process isolation – a kernel panic on the virtual machine will not affect the host OS
- Separate network stack and the ability to control traffic per virtual machine
- Reduce capital and operating cost, by simplification of data center management and better utilization of available server resources

Arguably the main reason [9] against using any sort of virtualization technology today is the inherited overhead of using multiple kernels in the same OS. It would be much better, in terms of complexity, if the host OS can provide this level of isolation, without the need for hardware extensions in the CPU, or the use of emulation software such as QEMU, or even kernel modules such as KVM. Running an entire operating system on a virtual machine, just to achieve a level of confinement for a single web server, is not the most efficient allocation of resources.

Over the last decade, various improvements [9] to the Linux kernel were made to allow for similar functionality, but with less overhead – most notably the kernel namespaces and cgroups. One of the first notable technologies to leverage those changes was LXC, since kernel 2.6.24 and around the 2008 time frame.

LXD is a next generation system container and virtual machine manager.

1.4. Objectives

The objectives of this research are to study network virtualization and multiple methods of network virtualization and study how we can use Open Virtual Network for our Network Virtualization needs. We have conducted studies on the Security Mechanisms of Open Virtual Network and how we can adapt those security mechanisms for our security property needs.

Further we focused on the analysis of configuration methodologies and evidence collection methodologies of Open Virtual Network and how we can configure our network virtualization platform to counter modern day security challenges.

1.5. Structure of Thesis

The rest of the document is organized as follows: Chapter 2 describes Network Virtualization and its working mechanism. Chapter 3 goes into further detail about the components and the architecture of Network Virtualization. Chapter 4 details the implementation options of Network Virtualization. Chapter 5 describes our choice of network virtualization implementation i.e. Open Virtual Network, its architecture, information pathway and its working mechanism. Chapter 6 goes into the association of physical network with virtual network elements. Chapter 7 details the security issues of Network Virtualization (i.e. OVN). Chapter 8 describes the configuration methodology and evidence identification in network virtualization (i.e. OVN). Finally Chapter 9 concludes the work and discusses the future work.

Chapter 2

Network Virtualization

Network virtualization [6] provides logical devices in software on top of a physical network. The virtual devices support I/O sharing across multiple VMs with its serialized access path. As a part of I/O virtualization, networking in virtualized systems requires isolation between virtual machines and physical machines. Unlike a virtualizing storage device which is block based, a virtualizing network is packet based which expects intermixed packet streams. Virtual Machine Monitor (VMM) offers Network Interface Card (NIC) with an emulation or a para-virtualization to deliver logical network devices. This additional step compared to a native single o/s over a single physical machine generates performance degradation and increases network access latency.

In computing, network virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network. Network virtualization involves platform virtualization, often combined with resource virtualization. Figure 1 [3] depicts Network and Compute Virtualization.

2.1. Working Mechanism of Network Virtualization

A virtual network is a software construct that presents logical network services switching, routing, firewalling, load balancing, virtual private networks (VPNs), and more to connected workloads. These network and security services are delivered in software and require only Internet Protocol (IP) packet forwarding from the underlying physical network. The workloads themselves are connected via the logical network, implemented by overlay networking.

We separate the technology's architecture from the physical infrastructure, allowing us to test the deployment hardware or network directly. This gives us the flexibility to access large-scale modifications or automatically created and tested configurations quickly.

In order to efficiently supply a networking platform and the creation of dynamic virtual networks, network virtualization coordinates the virtual switches across a variety of settings (for example, hypervisors, clouds), as well as the network services (for example, firewalling, load balancing).

The ability to provision network resources and services through a variety of interfaces is another benefit of network virtualization. The native graphical user interface (GUI) and command-line interface are two choices that make use of the native user interface (CLI).

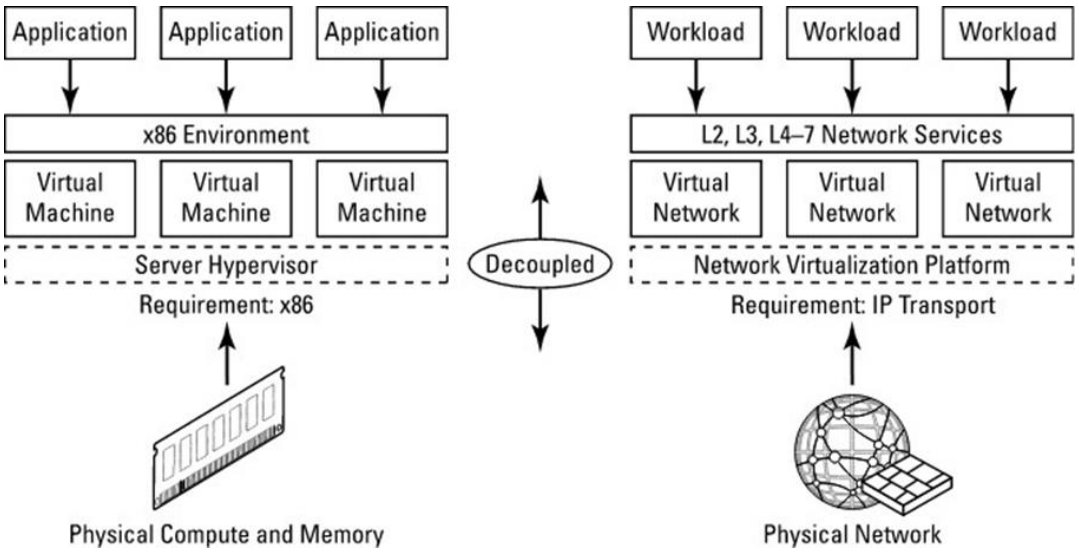

Figure 1. Network and Compute Virtualization

Another strategy involves scripting or baking in custom tools via the application programming interface (API). Network virtualization is integrated with new application frameworks like Kubernetes so that networking services are built when new apps, pods, and containers are spun up.

Cloud Management Systems (CMS) like OpenStack or VMware vRealize Automation can be used to provision virtual networks and add additional security services (such Firewalls, DDoS servers, and Monitoring) as the user deems necessary.

2.2. Traditional v/s SDN Switches in Network Virtualization

Many networking and security service providers have understood that in order to be more responsive and efficient, what was previously provided using a physical device needs to be closer to the application. The functionality of a single network function, such as a router, WAN accelerator, or network firewall, is typically delivered via virtual appliances in the form factor of a dedicated VM.

Traditional network switches make use of specialized hardware appliances that must be manually configured in accordance with their specific needs and settings. It should come as no surprise that deploying a network where each component is linked separately necessitates a lot of skill and maintenance, not to mention a high risk of human mistake.

The SDN switch enters the picture here. SDN offers centralized workload provisioning as opposed to traditional systems, enabling administrators to manage their whole network as a single entity with efficiency and flexibility. The SDN switch speeds up service delivery and improves the agility of device provisioning for both virtual and physical devices by abstracting the control and data planes.

An OpenFlow/SDN switch, when it receives a packet that it does not have a flow for (Match + exit port) will contact a SDN controller (Server) and ask what it must do with this packet. The controller can then download a flow to the switch, possibly including some packet manipulation. Once the flow is downloaded to the switch it will switch similar packets at wire-speed.

2.3. Architectures and Key Functionalities of Networking Virtualization

A virtual network is essentially a group of virtual nodes linked by a number of virtual connections to create a virtual topology, which is effectively a subset of the underlying physical topology. A virtual link spans over a path in the physical network and includes a section or a slice of the network resources along the path, which is sometimes referred to as a network slice. Each virtual node is hosted on a specific physical node.

Even though the underlying physical resources may be combined from various Infrastructure Providers, each VN is operated and controlled by a single service provider. As a result, separate virtual networks from different service providers can coexist. Figure 2 [3] represents a logical representation of Network Virtualization.

The architecture is shown in the following diagram:

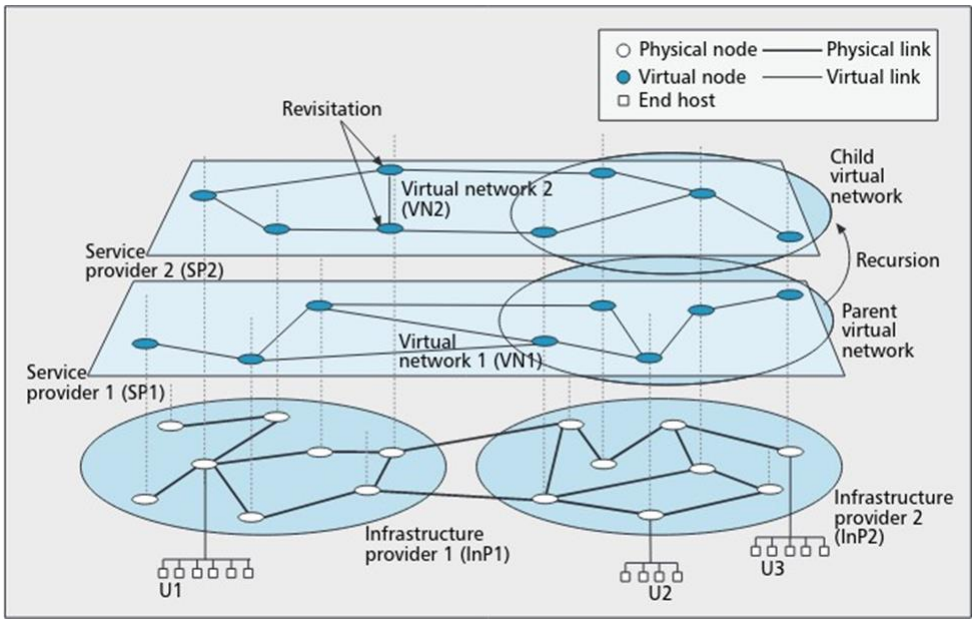

Figure 2. Logical Representation of Network Virtualization

Some **architectural principles** of network virtualization are discussed next:

- **Coexistence** - The distinguishing feature of a network virtualization environment is the coexistence of different VNs. It alludes to the possibility of coexisting VNs from several SPs that cover all or a portion of the underlying physical networks offered by one or more InPs. VN1 and VN2 are two coexisting VNs in the previous Figure.
- **Recursion** - Recursion and layering of VNs are terms used to describe when one or more VNs are created from another VN, resulting in a VN hierarchy with parent-child relationships. In the illustration above, service provider SP1 leased some of its assigned resources to SP2, who sees it as merely a virtual InP.
- **Inheritance** - In an NVE (Network Virtualization Environment), young VNs can inherit architectural characteristics from their parents, which also means that the restrictions placed on the parent VN immediately apply to its offspring as well. For instance, VN2 will immediately inherit the limitations and guidelines set forth by InP2 from VN1. Before reselling the produced offspring VNs to other SPs, inheritance enables an SP to increase their value.

- **Revisitation** - A physical node can host several virtual nodes of a single VN thanks to revisitation. An SP can logically change the structure of its network and make managing a VN easier by using numerous logical routers to handle various functionalities in a large, complicated network. Creating test bed networks can also benefit from revisitation.

When services like firewalling are used, communication on a traditional network might become inefficient. The physical firewall must be traversed before traffic is routed out of the virtual environment and returned to it. This action, which is also known as "**hairpinning"** or "**tromboning**", essentially leaves without ever getting to its target. It increases latency and complexity, which reduces performance, increases unpredictability, and makes it more difficult for application endpoints to relocate.

Some Key Functionalities of using Network Virtualization are listed below:

● Naming and Addressing

In the existing literature, mapping across several address contexts is a well-known issue. However, it gets considerably more challenging when there are various, frequently incompatible, addressing requirements in several VNs.

● Monitoring, Configuration, and Failure Handling

Decoupling the responsibilities of the traditional Internet service providers (ISPs) into two separate organizations—Infrastructure providers (InPs), who oversee the physical infrastructure, and Service providers (SPs), who build virtual networks (VNs) by pooling resources from various InPs and providing end-to-end services—defines network virtualization. The implementation of coexisting heterogeneous network architectures free from the built-in constraints of the current Internet will flourish in such an environment.

● Security and Privacy

Through the use of protected tunnels, encryptions, and other techniques, isolation between coexisting VNs may only offer a limited degree of security and privacy; yet, it cannot completely eliminate the dangers, incursions, and attacks that are frequently made against the physical layer and VNs. Additionally, network virtualization-specific security and privacy vulnerabilities must be found and investigated. For instance, if secure programming models and interfaces are not available, the network elements' programmability may make them more vulnerable. To develop a realistic NVE, each of these concerns needs to be carefully examined.

● Network Virtualization Economics

Virtual nodes and virtual links are equally significant in an NVE, in contrast to traditional networks where bandwidth is the most valuable resource. In this economy, SPs are the buyers and InPs are the sellers. Brokers that serve as middlemen between buyers and sellers are another possibility. End customers take part as service purchasers from several SPs.

2.4. Conceptual / Logical View

An overlay network is a virtual network that builds a virtual topology based on the physical topology of another network. Nodes are connected through tunnels that correspond to actual pathways in the underlying network, which is made possible by network hypervisors.

The only physical infrastructure we will ultimately need is a network of physical networks, which is what we can see at the most fundamental level. This acts as the parent virtual network. Figure 3 [2] shows the logical abstraction of a physical network into multiple parent virtual networks and consequently multiple children virtual network.

Smaller virtual networks, which are essentially parts of the bigger infrastructure, can now be connected to or created. These are completely configurable to behave anyway we desire, and can be used as such. End-to-end virtualization is offered. Only two end hosts and our virtual network, which appears as a black box from a higher perspective, are connected to create the flow.

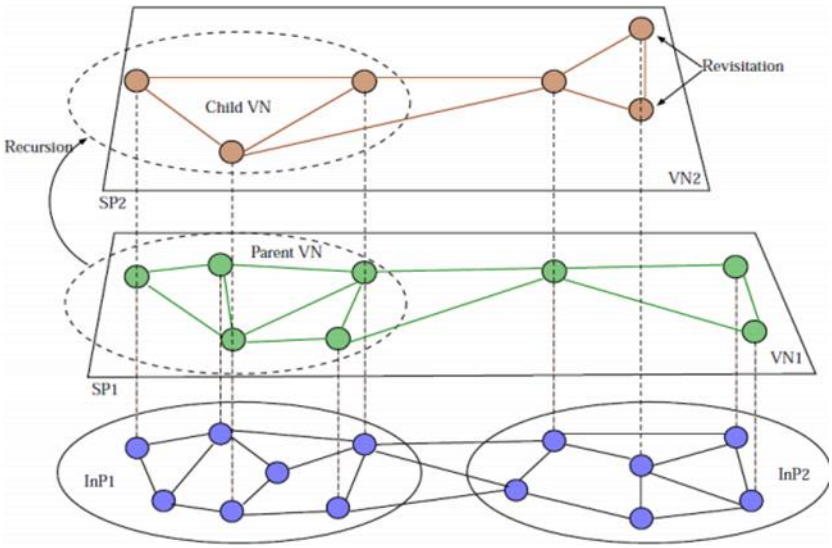

Figure 3. Logical Representation of the virtual network

Using and deployment are simple. Its "one-size-fits-all" design allows us to conceal hardware faults at any time as long as the network is up and running. It is located in between network operating systems and open flow networks.

A virtual network is a software construct that presents logical network services, such as switching, routing, firewalling, load balancing, Virtual Private Networks (VPNs), and more, to connected workloads. This is similar to how a virtual machine (VM) or a container is a software construct that presents logical services to an application. These network and security services are provided by software, and the underlying physical network is only required to forward Internet Protocol (IP) packets. The logical network, achieved using overlay networking, connects the workloads themselves.

Figure 4 [3] represents Virtual Network from a Logical Perspective. Figure 5 [3] represents Virtual Network from a physical perspective.

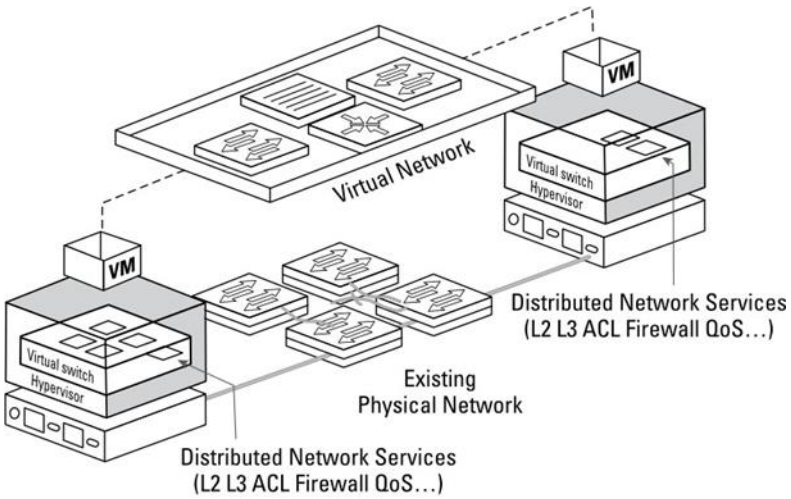

Figure 4. Virtual Network from a Logical Perspective

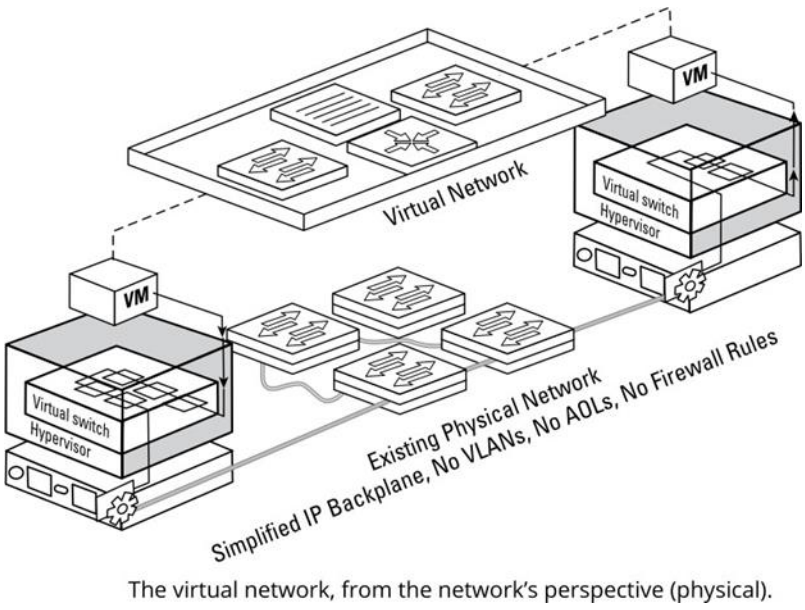

Figure 5. Virtual Network from a Physical Perspective

Network Slicing

Network slicing, which some people view as the distinguishing characteristic of 5G, enables service providers to designate virtual slices of their networks to certain functions. Data used for entertainment, communication, and the internet, for instance, will receive one slice of the network, and machine-to-machine data transfer (a key element of the Internet of Things or IoT) will receive a distinct slice. Critical data will have specialized 5G connectivity that cannot be shared with other services, such as that required for driverless cars, emergency services, and other essential infrastructure.

2.5. Service Models

Network virtualization abstracts away from proprietary hardware and pools all components of the IT infrastructure (compute, network, and storage). As needs and business requirements change, resources can be dynamically deployed from this pool where they are most needed. This is especially true in the telecommunications sector, where established providers face pressure to modernize their networks and business practices in order to keep up with advancements in technology. Figure 6 represents Service Model of Network Virtualization as a service.

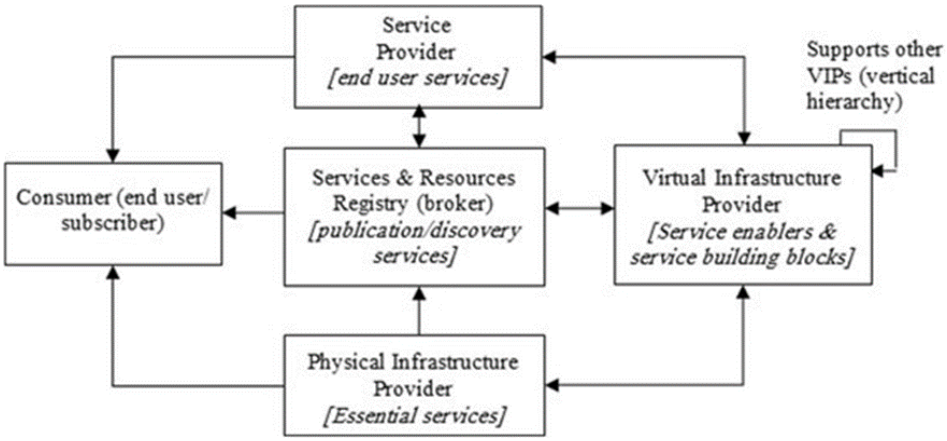

Figure 6. Service Model of Network Virtualization as a service

- **From Corporate Perspective:**

Large-scale network virtualization services are introduced by major corporations like Cisco, VMWare, and AT&T that might encourage industries to adopt this technology. This can be referred to as Network as a Service, but more from the perspective of management. In cases when the businesses maintain their own infrastructure, we can covertly deploy the management system or software. Otherwise, companies can choose to install their whole networks on third-party infrastructure spread out over the globe and not bother about it at all, saving money and labour. In essence, virtual networks have greater capacity and sophistication than traditional networks, which enables them to offer more services while also being faster and more flexible. Network virtualization can boost productivity while lowering operating expenses for enterprises.

- **Benefits of Network Virtualization for Business**

Flexibility - The capacity to adjust to the resources provided by the network is referred to as flexibility. The ability to quickly and effectively deploy new configurations or reconfigure network services is made possible by network virtualization.

Scalability - By offering an interface layer between applications and low-level networking hardware, network virtualization makes it possible to provide each "tenant" in the datacenter with its own address space, topology, and controller. This enables rapid deployment of new tenants and easy expansion of the physical networking equipment pool.

Reduced Costs - Network virtualization reduces hardware costs including product costs and maintenance costs based on the reduced requirement for physical servers, routers and switches amongst others. Centralized management and routines automation help to reduce maintenance costs.

Reduced energy consumption - Reduced powering devices means not only energy conservation but also reduced energy costs. The requirement for resource costs goes down as a less number of people are required for maintenance and monitoring purposes. This means employees are focused on high value-add tasks that improve productivity and business efficiency.

- **From End User Perspective:**

In this instance, the end user also gains and most likely does not even pay more for better service in terms of performance, dependability, and fewer downtime, which is likely the top worry for conventional home internet service providers. The typical end user has probably never experienced having the same network backbone as a large organization in their lifetime because infrastructure in centralized ISPs requires less labour to do routine maintenance and service upgrades.

2.6. Business Model

The fundamental difference between the network virtualization model's participants and those in the conventional model is the presence of two distinct roles, InPs and SPs, as opposed to the ISPs' exclusive role.

InP (**Infrastructure Providers**) - InPs deploy and manage the underlying physical network resources. They offer their resources through programmable interfaces to different SPs. InPs distinguish themselves through the quality of resources they provide, the freedom they delegate to their customers, and the tools they provide to exploit that freedom.

SP (Service Providers) - SPs lease resources from multiple InPs to create and deploy VNs by programming allocated network resources to offer end-to-end services to end-users. An SP can also provide network services to other SPs. It can also create child VNs by partitioning its resources and act as a virtual InP by leasing those child networks to other SPs.

End User - End users in the network virtualization model are similar to those of the existing internet, except that the existence of multiple VNs from competing SPs provides them with a wider range of choices. Any end user can connect to multiple VNs from different SPs for different services.

Level of virtualization - To enable network virtualization, one must virtualize the nodes, links, and every other resource in the network. The level of virtualization refers to the granularity at which each VN can administer itself. At one end of this spectrum, node virtualization creates VNs by connecting virtual nodes on different physical nodes.

Chapter 3

Components of Network Virtualization

The development of current datacenters, which have numerous commodity servers interacting with one another to complete computing tasks, is strongly related to network virtualization as it is known today. Large cloud providers (including AWS, Azure, and Google) and numerous enterprise organizations frequently use these datacenters. The VL2 article from Microsoft Research from back in 2009 outlined some of the difficulties in creating networks for such datacenters.

3.1. Architecture of Network Virtualization

The simplest possible network virtualization system is shown in Figure 7 [20]. Virtual switches reside at end hosts, and virtual machines connect to those virtual switches. The network virtualization controller exposes a northbound API that receives inputs describing the intended state of a virtual network. For example, an API request could specify "VM1 and VM2 should reside on the same virtual layer 2 subnet, network X". It is the responsibility of the controller to determine where those virtual machines are located, and then to send control commands to the appropriate virtual switches to create the virtual network abstraction that is required.

Since the VMs should be free to move around the datacenter, their IP addresses need to be independent of the physical network topology (indicated by the underlay network in the figure). In particular, we don't want a particular VM to be restricted in its location by the subnet addressing of the underlying physical network. For this reason, network virtualization systems invariably make use of an overlay encapsulation such as VXLAN or NVGRE. Encapsulation is a low-level mechanism that solves an important problem: decoupling the address space of the virtual network from that of the physical network.

One thing to notice about virtual network encapsulation, as illustrated in Figure 8 [20], is that there are a set of outer headers that are used by the physical network to deliver the packet to the appropriate end host, and there are a set of inner headers that are meaningful only in the context of a particular virtual network. This is how encapsulation decouples the virtual network addressing from that of the physical.

This simple example also shows one of the tasks that must be implemented by the network virtualization controller. When a VM wants to communicate with one of its peers in a virtual network, it needs to apply the appropriate outer header, which is a function of the

current server location of the VM. Providing the mapping from target VM to outer header is a natural task for the centralized controller. In VL2 this is referred to as a directory service.

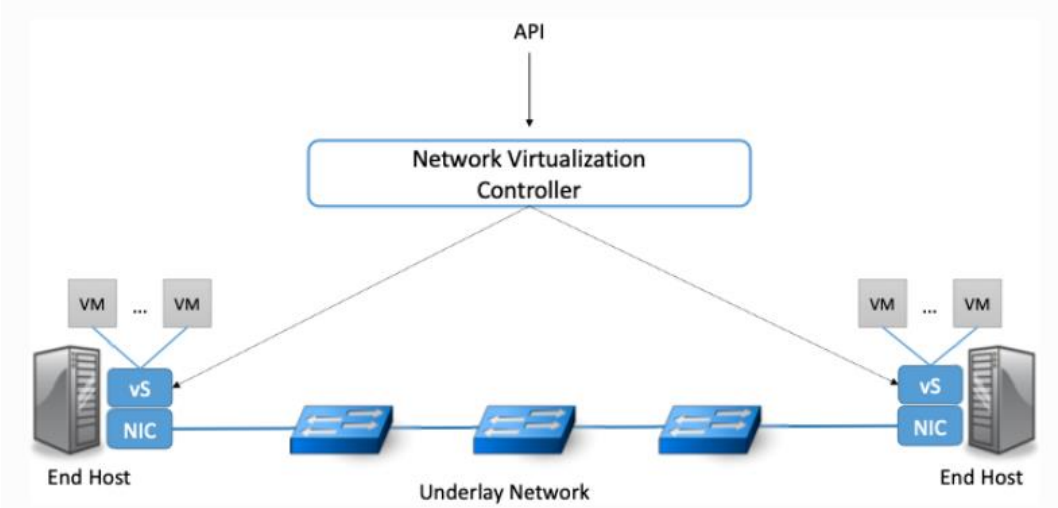

Figure 7. A Basic Network Virtualization System

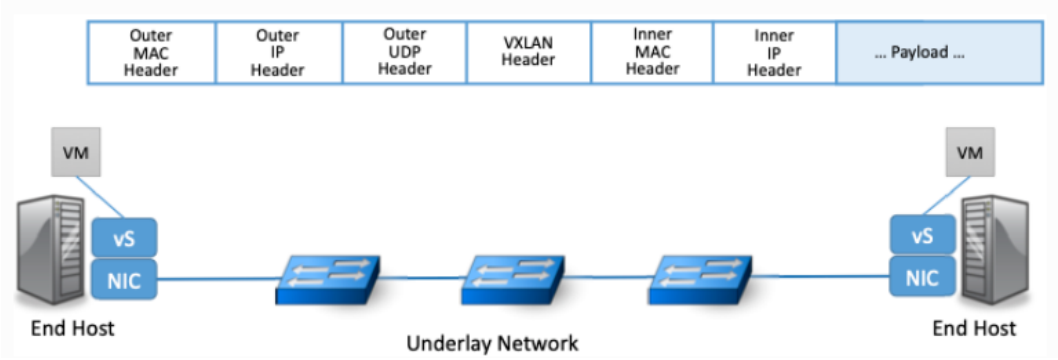

Figure 8. Encapsulation decouples virtual network addresses from physical network

3.2. Virtual Networks Defined

Virtual machines offer an accurate recreation of an actual server's features, including its processor, memory, peripherals, and other components. An unmodified operating system can run on the virtual machine exactly as if it were running on a physical computer since the reproduction is so accurate.

By analogy, virtual networks must also reproduce the full feature set of a physical network. This means that a virtual network includes routing, switching, addressing, and higher layer features such as NAT, firewalling, and load balancing. Just as an unmodified operating system can run on a VM exactly as it would on a physical machine, an unmodified distributed application should be able to run on a virtual network exactly as it would on a physical network. This is clearly a more elaborate proposition than a VLAN.

Importantly, a virtual network needs to keep operating correctly even as VMs move around. Thus, we can begin to see that the role for a network virtualization controller is to accept a specification of the desired virtual network and then ensure that this network is correctly implemented on the appropriate resources as conditions change and VMs move.

3.3. Management, Control and Data planes

Now that we have a better understanding of the fundamental structure of a network virtualization system. A current network virtualization system exposes a northbound API by which virtual networks are established and managed, in contrast to early types of virtual networks like VLANs and VPNs. The topology and services of a virtual network are specified through calls to this API, either by a human user or by another piece of software, such a cloud automation platform. Create a layer 2 subnet, attach VM A to subnet X, or apply firewall policy P to traffic entering VM B are examples of typical API queries. These API queries result in the construction of the desired state, or the condition in which the network ought to be, as seen in Figure 9 [20]. It is common to refer to the part of the system that receives API requests and stores them in a desired state database as the management plane.

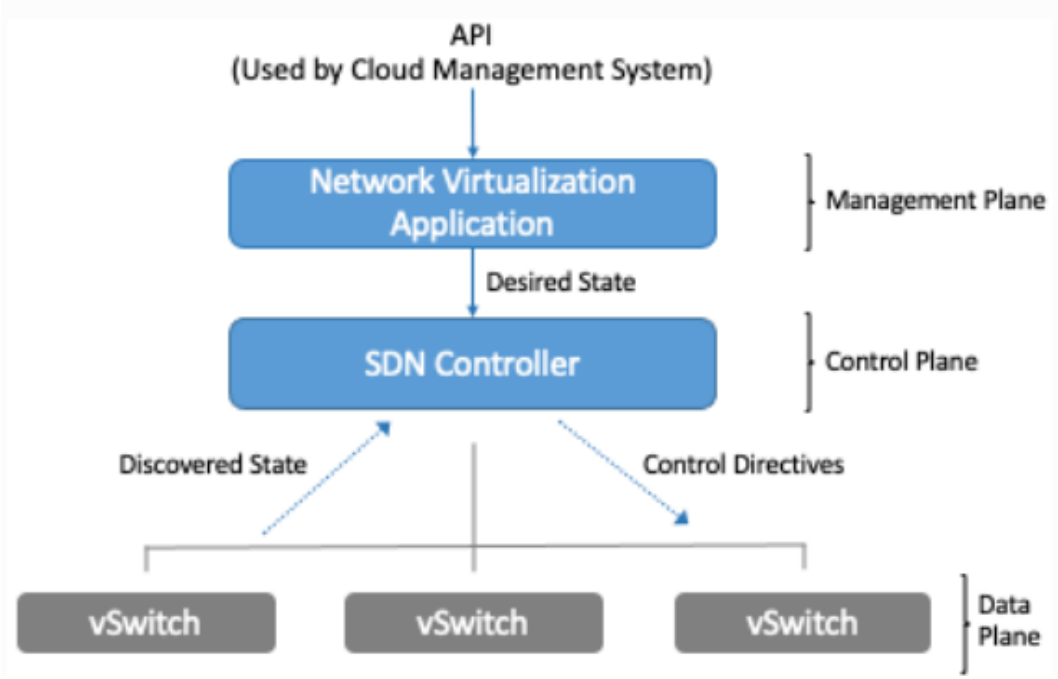

Figure 9. Three Planes of Network Virtualization

The data plane is shown at the bottom of Figure 9. Typically, this is a group of virtual switches (vSwitches) that run on hosts for containers or hypervisors. Virtual networks are implemented on the data plane. As we saw in the example, a virtual switch must apply the proper headers to the packets in order to forward them between VMs and the physical network. The higher levels of the network virtualization system must take into account the data plane's knowledge about the system's present state, including the locations of VMs.

The control plane is the brains of the system. It exists halfway between the ideal state and the system's current state. The control plane compares the found state information it receives from the data plane to the desired state. The control plane computes the necessary

changes and pushes them to the data plane if the desired state differs from the actual state, as shown by the arrow in the control directives. In distributed systems, this paradigm—constantly balancing desired and actual states—is prevalent.

A centralized controller gathers operational status and issues control instructions on top of a distributed data plane, which can be built from bare-metal switches or software switches. This controller is referred to as a Network OS when it is implemented in a broad, use-case independent manner. Onix, a very early network operating system created by the Nicira team, might be viewed as a forerunner to ONOS. A management layer at the top handles API requests and comprehends how a virtual network is abstracted. This administration layer can be compared to a network operating system (OS) application. In fact, there was at one time an ONOS-based virtual network application, called Virtual Tenant Network (VTN), that was integrated with OpenStack. VTN is no longer being maintained, due in part to the availability of other network virtualization subsystems that integrate with container management systems like Kubernetes.

3.4. Distributed Services

Software implementations of network functions such as firewalling, load balancing, and routing are essential aspects of network virtualization. However, it is not simply a matter of implementing a traditional network device in software. Consider the example of a firewall shown in Figure 10 [20]. A conventional firewall is implemented as a choke point: the network is set up in such a way that traffic must pass through the firewall to get from one part of the network to another.

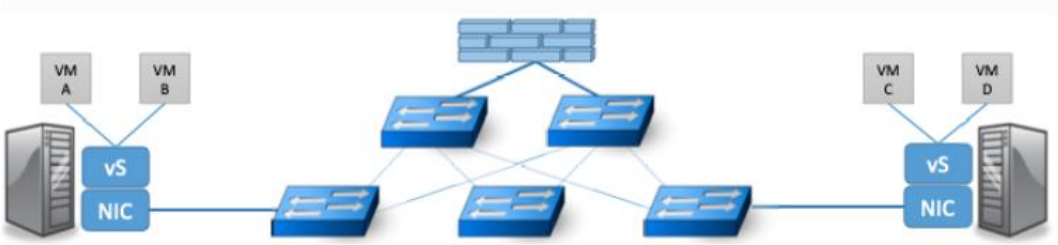

Figure 10. A conventional firewall

Consider the example in Figure 10. If traffic sent from VM A to VM C needs to be processed at a firewall in a conventional network, it needs to be routed over a path that traverses the firewall, not necessarily the shortest path from A to C. In the more extreme case of traffic from VM A to VM B, which sit on the same host, the traffic from A to B needs to be sent out of the host, across the network to the firewall, and then back to B. This is clearly not efficient, and consumes both network resources and, in the latter case, NIC bandwidth for the hairpinned traffic. Furthermore, the firewall itself has the potential to become a bottleneck, as all traffic requiring treatment must pass up to that centralized device.

Now consider Figure 11 [20], which illustrates a distributed firewall implementation. In this case, traffic sent from VM A to VM C can be processed by a firewall function at either (or both) of the virtual switches that it traverses, and still be sent over the shortest path through the network underlay between the two hosts, without hairpinning to an external firewall. Furthermore, traffic from VM A to VM B need never even leave the host on which

those two VMs reside, passing only through the virtual switch on that host to receive the necessary firewall treatment.

Figure 11. A distributed firewall

3.5. Virtual Network Encapsulation

Network virtualization requires some sort of encapsulation so that the addressing in the virtual network can be decoupled from that of the physical network. Inventing new ways to encapsulate packets seems to be a popular pastime for network architects and engineers, and there were a few potential candidates available already when network virtualization appeared on the scene. Figure 12 depicts the Geneve Header Format.

While VXLAN attracted considerable attention when it was first introduced in 2012, it was by no means the last word in network virtualization encapsulation. After many years of experimentation and collaboration among software and hardware vendors and other IETF participants, an encapsulation that combined most of the desired features was developed and standardized. A notable feature of GENEVE is its extensibility. This represented something of a compromise between those building software-based systems and those building hardware endpoints designed to support network virtualization (which we'll cover later in this chapter). Fixed headers make life easy for hardware, but limit flexibility for future expansion. In the end, GENEVE included an options scheme that could be efficiently processed (or ignored) by hardware while still giving the required extensibility.

Figure 12. GENEVE Header Format

GENEVE looks quite similar to VXLAN, the notable difference being the presence of a set of variable length options. The presence of options was a critical feature that built on the experience of earlier systems, where it was realized that the limited space in a VXLAN header was insufficient to pass metadata related to virtual networks from one end of a tunnel to another.

3.6. Virtual Switches

The Virtual Switch clearly plays a critical role in network virtualization. It is the main component of the data plane, and the richness of its feature set determines the ability of a network virtualization system to accurately reproduce the features of a physical network. The most widely deployed virtual switch is Open vSwitch (OVS).

Open vSwitch has been used in proprietary systems such as Nicira's Network Virtualization Platform and VMware NSX, as well as open source systems such as Open Virtual Network (OVN). It was designed to have the necessary flexibility to meet the requirements of network virtualization while also providing high performance.

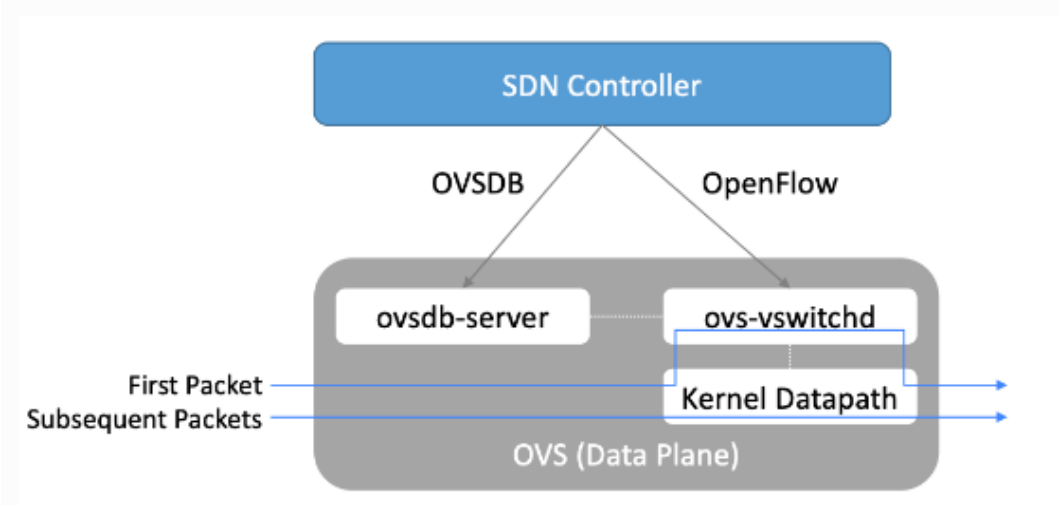

Figure 13. Open vSwitch Functional Blocks

As depicted in Figure 13, OVS is programmed by the control plane using OpenFlow, just like many hardware switches described in previous chapters. It also receives configuration information over a separate channel using the Open vSwitch Database (OVSDB) protocol, which is to say, OVSDB effectively serves the same purpose as gNMI/gNOI does for a hardware-based data plane. Again, the mapping between these building blocks and the components described in earlier chapters is straightforward, the differences in terminology and details largely being attributed to network virtualization evolving as a purpose-built solution.

OVSDB as depicted in the Figure refers to an RPC protocol used to access the database (called ovsdb-server), but in general, OVSDB can refer to either the protocol or the database. As a database, OVSDB has a schema, which you can think as OVS's counterpart to the OpenConfig schema. As a protocol, OVSDB uses a JSON-based message format.

As for the OVS data plane, performance has been achieved via a long series of optimizations described in the Pfaff paper, notably a fast-path in the kernel that uses a flow cache to forward all packets in a flow after the first. The first packet in a flow is passed to the userspace daemon ovs-vswitchd, which looks up the flow in a set of tables. This set of tables, being implemented in software, can be effectively unlimited in number, a distinct advantage over hardware implementations of OpenFlow switching. This enables the high degree of flexibility that is required in network virtualization. At the same time, there is also an effort to unify software- and hardware-based forwarding elements, using P4 as the lingua franca for writing packet forwarders. This also brings P4Runtime into the mix as the auto-generated interface for controlling the data plane.

3.7. Challenges of Network Virtualization

In such datacenters, there is a substantial amount of "east-west" traffic; that is, server-to-server traffic, as distinct from "north-south" traffic entering or leaving the datacenter. To efficiently support high volumes of traffic between any pair of servers in the datacenter, leaf-spine fabrics of the sort became popular due to their high cross-sectional bandwidth and scalable layer-3 forwarding.

At the same time, server virtualization became mainstream, which had several implications for datacenter operations. Provisioning virtual machines (VMs) can be carried out entirely in software, by contrast to installing and configuring physical servers, a time-consuming and manual process. As the ease of provisioning a VM shrank the time to obtain computational resources from days to minutes or seconds, it exposed the fact that network configuration was now the "long pole"—the slowest task to be completed before a user could put their infrastructure to work. Hence, there was a recognition that network configuration and provisioning needed to move towards a model more similar to virtual compute, setting the stage for network virtualization.

A second effect of server virtualization was to enable virtual machine mobility. This introduced some real challenges for datacenter networking. In the absence of network virtualization, the IP address of a VM is drawn from the physical network on which it resides, and must be specific to a subnet that connects to the server hosting the VM.1 So if a VM is to migrate to another server, either it needs to move to a server where that subnet is also present, or it needs a new IP address. The first choice limits where it can move within the datacenter, which affects the efficiency of resource usage. The second option is quite a disruptive thing: TCP connections are dropped, and applications may need to be restarted. Furthermore, some applications depend on layer-2 adjacency between communicating peers, and thus depend on some set of VMs staying in a given subnet even as they move around within the datacenter.

One proposed solution to this issue was to make layer-2 subnets ever larger in the physical network, but that is not really a scalable solution. Large datacenters invariably use layer-3 networking to connect racks of servers.

The approach proposed by Greenberg [3] can be considered a first step in network virtualization. They created a Virtual Layer 2 (VL2) network such that the addresses used by virtual machines are decoupled from the addresses used in the physical network, thus solving the mobility-related issues described above. A VM draws its IP address from a virtual layer 2 network, and VL2 constructs the appropriate overlay to extend that virtual network wherever it needs to go, even as VMs migrate across physical networks.

Solving the problem of network configuration is a bit more complex. Networks are not just simple subnets to connect servers; they have a host of features that need to be configured, including VLANs (or some equivalent construct to segment the network), firewall rules, network address translation (NAT) rules, and so on. It is the complexity of these tasks that made network configuration the barrier to agility in datacenter configuration.

Tackling these issues of configuration and provisioning ultimately led to the realization that SDN provided a means to simplify the creation and management of virtual networks, just as much as it simplifies the operation of physical networks. The key insight is that a central API to an SDN controller provides the ideal way to specify the desired behavior of the virtual network, with the central controller then taking responsibility for figuring out how to implement the network with the available resources, such as virtual switches in the hypervisors of the datacenter's servers. The core principles of SDN—separation of data plane from control plane, and centralization of the controller to manage a multitude of switching elements—provide the basis for this approach.

Chapter 4

Implementation Options of Network Virtualization

Network Virtualization has multiple methods in which it can be implemented and integrated with current existing technologies.

4.1. Flow Visor

FlowVisor [4] is a software platform for slicing an Open-Flow network into multiple resource pools or slices. Each slice has its own NOS (Network Operating System) and is associated with a subset of the network resources encompassing some or all of the switches and links. Figure 14 depicts the logical mechanism of Flow Visor.

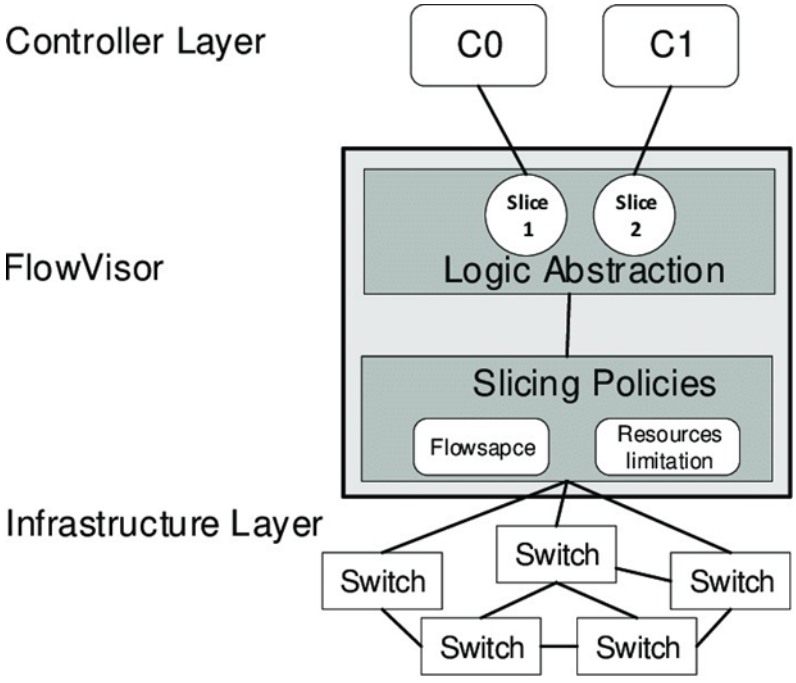

Figure 14. FlowVisor Visualization

There are some restrictions with FlowVisor's network slicing. A slice does not have a completely separate and independent address space because all slices effectively share the same flow or address space. Additionally, FlowVisor can only provide (a part of) the physical topology; it does not permit a slice to have an arbitrary (virtual) network topology.

Additionally, by designating non-overlapping portions of the packet header space to each slice's flow space, FlowVisor creates isolation between the slices under its control. Unfortunately, if the flow spaces overlap due to misconfiguration, FlowVisor is unable to isolate the affected slices.

4.2. OpenVirteX

OpenVirteX (OVX) [7] is a network hypervisor that takes virtualization in SDN a step further. OpenVirteX, a network virtualization platform that enables operators to create and manage virtual Software Defined Networks (vSDNs). Tenants are free to specify the topology and addressing scheme of their vSDN, and run their own Network Operating System (NOS) to control it. Similar to FlowVisor, it acts as a transparent proxy between OpenFlow switches and SDN controllers. Figure 15 gives the Logical Representation of Open VirteX.

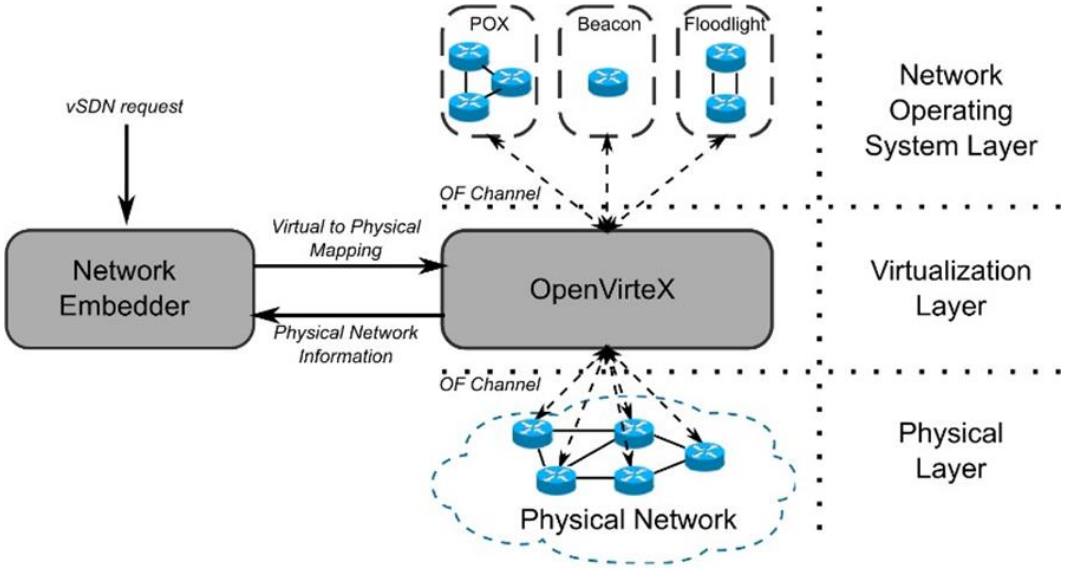

Figure 1: System architecture.

Figure 15. Logical Representation of OpenVirteX

The main distinction between OVX and FlowVisor is that OVX allows full flow space virtualization, or several slices with overlapping flow spaces, such as IP and MAC address ranges, VLAN IDs, etc. Each virtual network receives a globally unique identity from OVX, also known as a tenant ID, which is given to each tenant. The main drawback of FlowVisor is that it allocates packets based on flow space matching. OVX introduces a new functionality at SDN edge switches that directly rewrites the header fields of arriving packets into its own, distinct format. When packets are sent to the appropriate destination slices, such as hosts or controllers, this rewriting is undone. OVX virtual networks and OVX physical networks are the two logical levels that make up the OVX architecture. Tenants are given a totally isolated virtual network made up of virtual switches and links via the OVX virtual networks layer. Each tenant can thus design a different virtual network topology. Also it provides tenants with a completely isolated virtual network, consisting of virtual switches and links. As a result, each tenant can specify its own, unique virtual network topology.

Internally, OVX depends on a loose link between virtual and physical constituents. OVX maintains the mapping between all virtual and physical elements in order to accomplish this.

Below are some OVX highlights or features:

• **Topology customization**: vSDN topologies need not be isomorphic to the infrastructure, or be restricted to its subgraph. A virtual link may encompass multiple contiguous hops, and virtual switches may abstract away parts or all of a network.

• **Resiliency**: A virtual link or switch can be mapped onto multiple physical components to provide redundancy. A resilient virtual link is characterized by multiple physical paths between the points corresponding to the virtual link endpoints. A virtual switch that abstracts away portions of the physical network may take advantage of redundancies in the topology (e.g. multiple paths) to provide multiple paths between its ports.

• **Dynamic vSDN reconfiguration**: The reconfiguration of a tenant network reduces the manipulation of key-value pairs. This is a simple operation since the mapping is logically centralized in the global map, and key-value pairs employ references as opposed to storing the component representations. Since the mappings themselves do not store any network state, these can be changed at runtime.

4.3. Open Virtual Network

Open Virtual Network (OVN) is an Open vSwitch-based software-defined networking (SDN) solution for supplying network services to instances. OVN provides platform-neutral support for the full OpenStack Networking API. With OVN, its main goal is to provide Layers 2 and 3 networking, which distinguishes it from general-purpose, software-defined networking (SDN) protocols and controllers. OVN uses a standard approach to virtual networking that is capable of extending to other Red Hat platforms and solutions.

OVN provides a higher-layer of abstraction than Open vSwitch, working with logical routers and logical switches, rather than flows. OVN is intended to be used by cloud management software (CMS).

Some of the most innovative features of OVN are:

- Logical switches
- Flexible L2/L3/L4 security policies
- Distributed logical IPv4 and IPv6 routers
- Native support for NAT, load-balancing, and DHCP
- L2 and L3 gateways

OVN can also be used in OpenStack-based networks, where Open vSwitch is the most popular virtual-switch option. OVN developers are hoping their standard becomes accepted as the default network control plane for OpenStack, the open-sourcehybrid cloud operations platform. The network facilitator in OpenStack is called Neutron, and its default control plane uses OVS.

4.4. Libera

Libera and OpenVirteX are network hypervisors that can create multiple virtual and programmable networks on top of a single physical infrastructure. Each tenant can use the full addressing space, specify their own topology, and deploy the network OS of their

choice. Through the use of OpenVirteX, it appears as though the Cloud's physical infrastructure perfectly satisfies the client's needs in terms of isolation, topology, and addressing. Because OVX offers significantly lower latency with a growing number of TCP connections, which is essential for data centre virtualization, we picked it over FlowVisor. Figure 16 [7] represents the logical components of Libera.

Testbeds created using the OpenVirteX-based Libera platform, which provides benefits beyond those of standard network virtualization, such as scalability, mobility, and flexibility. Libera also gives us the freedom to programme our own custom topologies and install policies on vSwitches as needed. Additionally, it offers monitoring support, which makes load balancing and traffic engineering much easier.

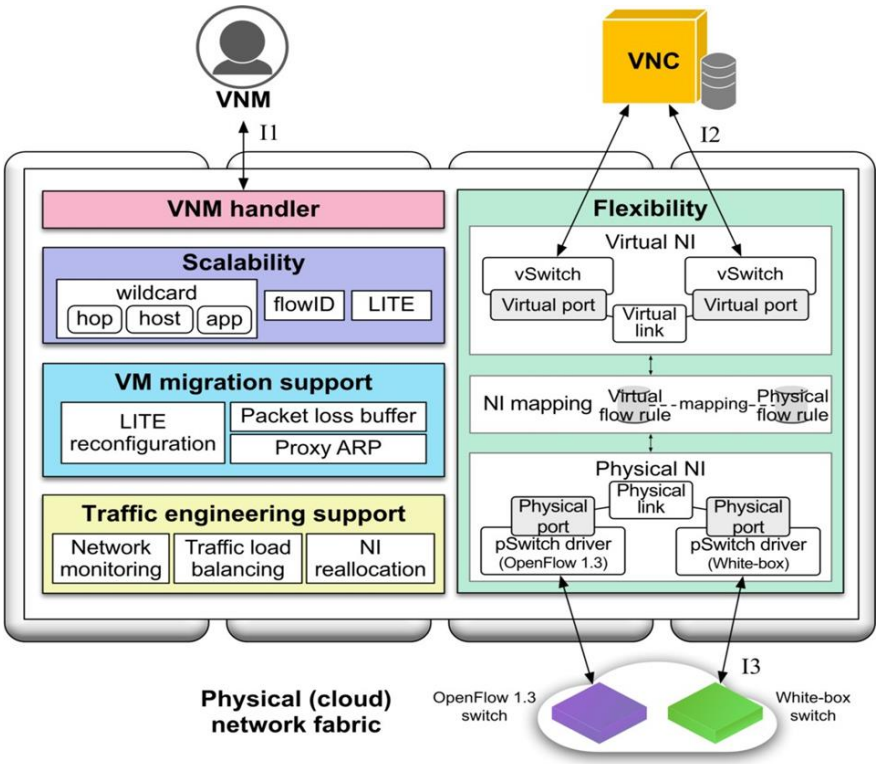

Figure 16. Logical Representation of Components of Libera

Components of Libera:

- **VNM Handler**: Libera's VNM handler delivers the VNM's requests to the proper Libera components. Topology provisioning requests are delivered to the flexibility component, and scalability level requests are given to the scalability component.
- **VN Controller**: Maintains the vSwitches and virtual vPorts and vLinks in the Virtual NI layer.
- **Virtual NI Layer**: This layer provisions VN topologies with virtual NIs.When the VNM creates a vSwitch, the vSwitch is designed to be compatible with existing SDN controllers. Therefore, the vSwitch emulates two generic operations of the SDN switches: the control channel and the message handler.
- **NI Mapping Layer**: This layer introduces the notion of virtual FRs and physical FRs. A detailed description of virtual and physical FRs and their mappings are provided here. For instance, when a host migrates, the host is connected to a new

pSwitch, and the forwarding path for the migrated host needs to be changed. Therefore, the vSwitches in the VN affected by the VM migration should be mapped with the new pSwitches.

- **Physical NI Layer**: This layer aims to support various physical Switches (e.g., OpenFlow 1.0, 1.3, and white-box switches). This is the large network of physical switches comprising the physical layer.
- **Wildcard**: Wildcard works on the match fields of each rule. When an FR matches the packet header field, its actions are applied to the matched packets. Therefore, the number of matched FRs can vary depending on the match fields.
- **LITE**: LITE consists of a tenant ID and an edge switch identifier to which a host of the tenant is attached. If a host is attached to a switch whose ID is 0a, and the host belongs to tenant 1, the LITE would be 0x010a.
- **Traffic Engineering Support**: It hosts the network monitoring interface which has the flow tables, and also load balancing and NI reallocation can be achieved whenever the requirement arises.

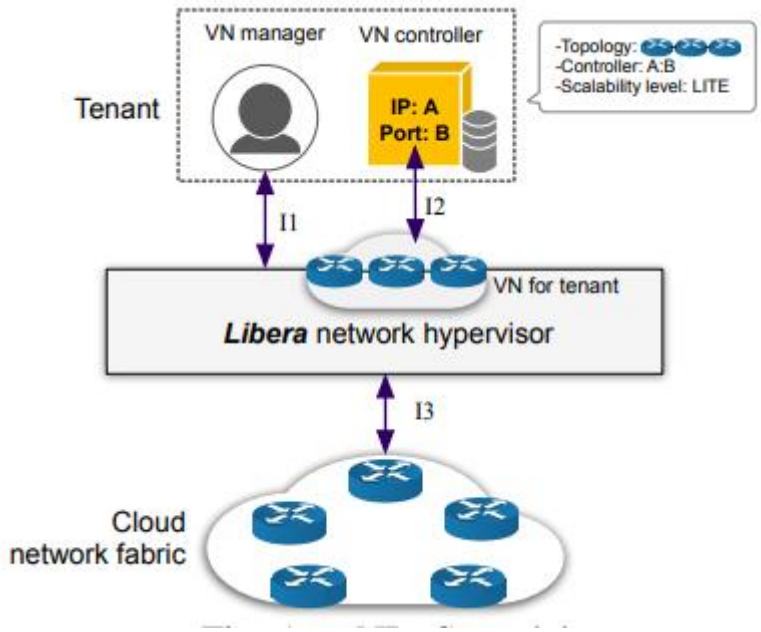

Figure 17. Basic Workflow Diagram of Libera

The p-NIaaS model's elements and their interactions are shown in this diagram. The interactions between tenants, the Libera hypervisor, and the physical network are shown in Figure by arrows I1, I2, and I3. The Libera is the foundation of the p-NIaaS model. Figure 17 represents the Basic Workflow Diagram of Libera.

With the help of the model, tenants will gain the previously unattainable capacity to programme virtual NIs in their virtual networks. We develop Libera and assess its performance and overheads to demonstrate the notion. Libera has the complete programmability of virtual NIs while operating with existing SDN controllers without requiring any modifications. Our findings demonstrate that Libera outperforms the current network hypervisors in terms of scalability by a factor of many. Libera reduces the virtualization overhead on the control plane, according to the results. Therefore, we think

that this effort narrows the gap between virtualized and non-virtualized SDN, which advances the network hypervisor's chances of being adopted by the industry.

4.5. Openstack Neutron

OpenStack Neutron is also widely used for Network Virtualization. Neutron controls the network connections between compute nodes in an OpenStack-based cloud system and allows the external SDN controller to be used for the control. **SONA** (based on **ONOS**) and **Neutron-ODL** (based on **OpenDayLight**) are examples.

SONA and **Neutron-ODL** automate the creation of overlay connections between edge switches, like NVP and VFP. Similar to NVP and VFP, policy installation on every vSwitch and custom VN topology provisioning is impossible because the physical network is controlled only by a cloud orchestrator. Network monitoring is possible for only edge switches. Only VM migration support is possible on Neutron-ODL.

The term “network as a service (NaaS)” takes on different meanings. NaaS from Cisco focuses on network management efficiency, which has a different scope than Libera. Hardware-based network programmability has been proposed in NaaS by Costa et al. in “NaaS: Network-as-a-Service in the Cloud”. It installs field-programmable gate array (FPGA)-based specialized hardware (called NaaS box) to switches and permits the direct control of tenants. The NaaS box provides a certain level of programmability to tenants but is quite different.

4.6. Cloud NaaS

A virtual network architecture called Cloud NaaS provides effective assistance for setting up and controlling business applications in clouds. The design offers a number of primitives, such as application-specific address spaces, middlebox traversal, network broadcasting, VM grouping, and bandwidth reservation that are suitable for the needs of typical enterprise applications. Although some of these difficulties (application-specific address spaces and bandwidth reservation) have already been addressed by numerous other data centre network virtualization approaches, they do not entirely handle all of the aforementioned issues. This insight led to the creation of CloudNaaS, which attempts to offer a unified, all-encompassing framework for operating business applications in clouds. To accomplish the aforementioned goals, including middlebox traversal, such as an Amazon Virtual Private Network or a Microsoft Azure Private Network, CloudNaaS relies on OpenFlow forwarding.

4.7. IP Networks: X-Bone

As a solution for the quick and automated deployment and maintenance of overlay networks employing encapsulation to support virtual infrastructure, X-Bone was initially presented. Later this idea was extended to the concept of **Virtual Internet** (**VI**), which is an IP network composed of tunneled links among a set of virtual routers and hosts, with dynamic resource discovery, deployment, and monitoring support.

The X-Bone architecture is based on a two-level multicast control protocol, combined with algorithms to deploy and configure overlays.

The architecture includes:

- Overlay manager - configures and manages overlays
- Resource daemons - manages resource use

● Multicast control protocol - enables efficient resource discovery and management communication

A VI virtualizes all the components of the Internet: hosts, routers and links between them. A single network node may participate as a virtual host (VH), a virtual router (VR), or multiple of them simultaneously in a VI. All components participating in the VI must support multi-homing since even a base host with a single VH is necessarily a member of at least two networks: the Internet and the VI overlay. Addresses within each VI are unique and can be reused in another overlay unless there is no shared common node in the underlying network between the two VIs.

VIs completely decouple the underlying physical network from the overlays and multiple VI can coexist together. VIs also support control recursion to allow divide-and-conquer network management and network recursion to stack one VI on another. Recently, P2P-XBone, a peer-to-peer based fusion of X-Bone, was proposed to enable dynamic join/leave of participating nodes from a VI. It also allows the creation and release of dynamic IP tunnels and customized routing table configuration.

4.8. ATM networks: Tempest

ATM stands for Asynchronous Transfer Mode. It is a switching technique that uses time division multiplexing (TDM) for data communications.

ATM networks are connection oriented networks for cell relay that support voice, video and data communications. It encodes data into small fixed-size cells so that they are suitable for TDM and transmits them over a physical medium.

Tempest [5] is a network control architecture that allows multiple heterogeneous control architectures to run simultaneously over a single physical ATM network. It is de- fined as a set of policies, algorithms, mechanisms, and protocols to control and manage various devices on the network.

Tempest is based on the concept of switchlets, which allows a single ATM switch to be controlled by multiple controllers by strictly partitioning the resources of that switch between those controllers. The set of switchlets that a controller or group of controllers possess forms its virtual network. Third parties can lease such virtual networks from the Tempest network operator to use them for any purpose as they see fit. Programmability in Tempest is supported at two levels of granularity: first, switchlets support the introduction of alternative control architectures in the network; and second, services can be refined by dynamically loading programs into the network that customize existing control architectures. This allows the users to have application-specific control.

4.9. Physical layer: UCLP

UCLP2 is a distributed network control and management system for CA*NET 4 network that allows end-users to treat network resources as software objects and lets them provision as well as dynamically reconfigure optical networks (at Layer 1). Users are able to join or divide lightpaths within a single domain, or across multiple independent management domains to create customized logical IP networks. UCLP takes a modular approach to resource management by introducing three distinct service layers.

Customers and administrators configure and use end-to-end UCLP resources through the user access layer. The service provisioning layer manages service logic and data regarding lightpaths. Finally, the resource management layer deals with actual physical resources.

Chapter 5

Network Virtualization using Open Virtual Network (OVN)

The practice of merging hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network, is known as network virtualization. Platform virtualization, which is frequently paired with resource virtualization, is a component of network virtualization. Network virtualization can be classified as either external virtualization, which combines many networks or portions of networks into a virtual unit, or internal virtualization, which gives software containers on a single network server access to network-like functions. Network virtualization is used in software testing to simulate the network settings in which the software is supposed to function while testing software that is still in development. As a component of application performance engineering, network virtualization enables developers to emulate connections between applications, services, dependencies, and end users in a test environment without having to physically test the software on all possible hardware or system software. The validity of the test depends on the accuracy of the network virtualization in emulating real hardware and operating systems.

5.1. What is Open Virtual Network?

OVN (Open Virtual Network) [10] is a series of daemons for the Open vSwitch that translate virtual network configurations into OpenFlow. OVN is licensed under the open source Apache 2 license. OVN is a network virtualization platform that separates the physical network topology from the logical one. Users are able to connect virtual and physical interfaces with logical switches and routers, regardless of the underlying physical topology. Users are also able to define security policies and load-balancing to these logical instances. OVN uses Open vSwitch for its switching fabric and uses tunnels to provide the logical/physical separation.

Open source bindings for OVN [10] are available for a number of platforms, such as OpenStack and Kubernetes. OVN is the SDN platform used in a number of Red Hat products, including Red Hat Virtualization, OpenStack and OpenShift. Some of the exclusive features of OVN are Logical switches, Flexible L2/L3/L4 security policies, Distributed logical IPv4 and IPv6 routers, Native support for NAT, load-balancing, and DHCP, L2 and L3 gateways.

OVN is built as a set of enhancements to OVS, leveraging OVS for the data plane and a set of databases (built on OVSDB) for its control and management planes. An important

aspect of OVN is its use of two databases (referred to as Northbound and Southbound) to store state. These databases are implemented using OVSDB, which was originally created to store configuration state for OVS.

Some of OVN's distinguishing features are as follows:

Manages overlays and physical network connectivity

● Flexible security policies (ACLs)

● Distributed L3 routing, IPv4 and IPv6

● Native support for NAT, load-balancing, DHCP

● Works with Linux, DPDK, and Hyper-V

● L2 and L3 gateways

● Designed to be integrated into another system (OpenStack, Kubernetes, Docker, Mesos, oVirt)

5.2. Working Mechanism of Open Virtual Network

OVN is assumed to operate in an environment where a **Cloud Management System** (CMS) is responsible for the creation of virtual networks. This is likely to be OpenStack, which was the first CMS to be supported by OVN. The OVN/CMS plugin is responsible for mapping abstractions that match those of the CMS into generic virtual network abstractions that can be stored in the Northbound Database. OVN uses an instance of ovsdb-server to implement this database. We can think of the plugin as the management plane and the Northbound DB is the desired state repository.

The control plane of OVN demonstrates a significant novel feature compared to the generic architecture of the Network Virtualization diagram. Importantly, it is divided into a centralized component, known as **ovn-northd**, and a distributed component that runs on every hypervisor, called the **OVN controller**. In OVN, a hybrid model is used. The decision to make this split followed the experience of the OVN team in working on earlier systems where the centralized controller had a full view of all flows, which presented a scaling challenge. OVN retains logically centralized control, so that a single API entry point can be used to create networks, query status, and so on, but distributes out to each hypervisor the control functions related to physical information such as the location of VMs in specific servers. This led to significant gains in the scalability of the system.

A centralized component, **ovn-northd**, translates the logical network configuration, expressed in terms of conventional network concepts like switching and routing, into logical datapath flows, which it stores in the OVN Southbound Database. We can see how logical flows work with an example shown in the below Figure 18 [20].

Logical data path flows provide an abstract representation of the forwarding rules that are populated in the data plane, specified in a way that is independent of the physical location of VMs. In this example, we have created a logical switch LS1, with two ports, LP1 and LP2. Each port connects to a VM and the MAC addresses of the VMs are AA and BB as shown in the Logical_Port table.

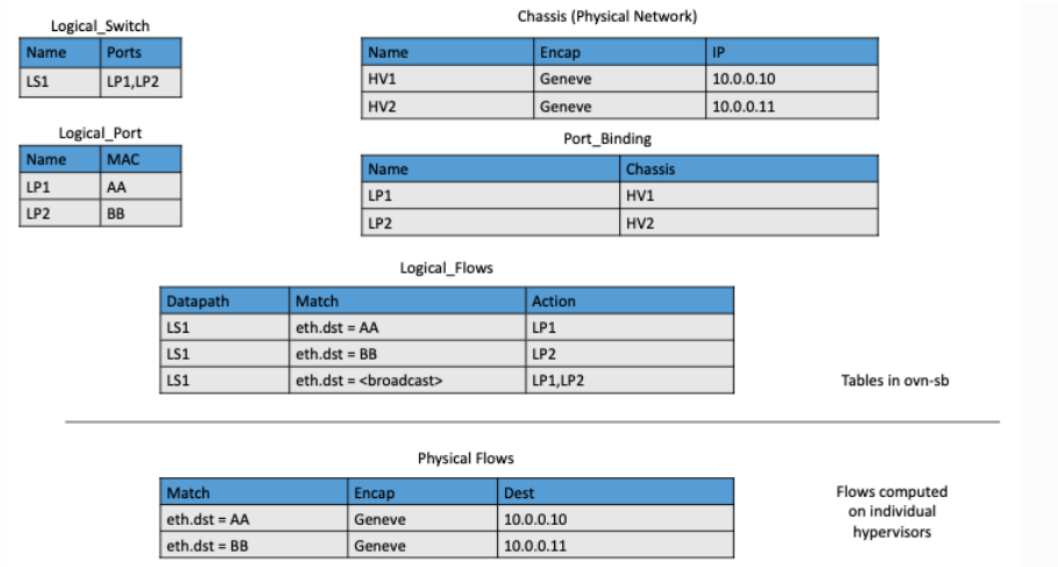

Figure 18. Example Logical and Physical Flows in OVN

The Logical_Flows table, built from the information in the Logical_Switch and Logical_Port tables, shows how packets are to be forwarded to implement this logical switch. For example, the first row indicates that packets destined for a MAC address of AA need to be sent to port LP1. But there is not enough information to actually forward packets in this flow, because that depends on which hypervisors currently host those VMs. Providing the binding of physical hypervisor nodes to VMs is a task performed by the OVN controller running on the appropriate hypervisor. This is an example of discovered state, in the sense that the hypervisors discover the location of VMs and report it up to the database. So we see that the controller on HV1 (hypervisor 1) has reported into the Chassis table that it can be reached using the Geneve encapsulation at IP address 10.0.0.10. And that same hypervisor has reported into the Port_Binding table that it is hosting the VM with LP1.

Based on the VMs that it currently hosts, the OVN controller for each hypervisor accesses the OVN Southbound DB to determine the logical flows that are pertinent to it in order to design the data plane. It is able to create the rules that must be encoded into the instance of OVS operating locally on the hypervisor in question. This information is combined with the knowledge provided by other hypervisors on the locations of other VMs.

Continuing with the example above, hypervisor 2 needs a flow rule in OVS to forward packets from LP2 to LP1. It is able to see this by looking at the Logical_Flows in OVN Southbound DB, and it is able to determine the details of how to encapsulate packets and forward them to the right destination server using information in the Port_Binding and Chassis tables. We can see the results for both hypervisors in the table at the bottom of the figure, which is not part of the Southbound DB but is a collation of the flows computed at the two hypervisors.

Since virtual machines (VMs) have been used as the endpoints of our virtual networks throughout this discussion, everything that applies to VMs also applies to containers (glossing over some implementation details). We can link a number of container hosts to the OVN Southbound DB so that they can build virtual networks for the containers they are hosting by setting up flow rules for their OVS instances. A container management system, such as Kubernetes, is most likely to be the "cloud management system" that OVN integrates with in this situation.

5.3. Architecture of Open Virtual Network

OVN [11], the Open Virtual Network, is a system to support logical network abstraction in virtual machine and container environments. OVN complements the existing capabilities of OVS to add native support for logical network abstractions, such as logical L2 and L3 overlays and security groups. Services such as DHCP are also desirable features. Just like OVS, OVN's design goal is to have a production-quality implementation that can operate at significant scale.

Physical lines, switches, and routers make up a physical network. A virtual network connects virtual machines (VMs) or containers to a physical network by extending a physical network into a hypervisor or container platform. A network built in software that is shielded from physical (and hence virtual) networks by tunnels or other encapsulations is known as an OVN logical network. This prevents conflicts between IP and other address spaces used in logical networks and those used on physical networks. The topologies of the physical networks on which they operate need not be taken into consideration when arranging logical network topologies. As a result, VMs connected to a logical network can move from one physical computer to another without affecting the network.

The encapsulation layer prevents VMs and containers connected to a logical network from communicating with nodes on physical networks. For clustering VMs and containers, this can be acceptable or even desirable, but in many cases VMs and containers do need connectivity to physical networks. OVN provides multiple forms of gateways for this purpose.

An OVN deployment consists of several components:

- The ultimate client of OVN is a Cloud Management System (CMS) (via its users and administrators). Installing a CMS-specific plugin and associated software is necessary for OVN integration (see below). Open Stack is the CMS that OVN first targets.
- An OVN Database physical or virtual node (or, eventually, cluster) installed in a central location.
- One or more (usually many) hypervisors. Hypervisors must run Open vSwitch and implement the interface described in Documentation/topics/integration.rst in the Open vSwitch source tree. Any hypervisor platform supported by Open vSwitch is acceptable.
- Zero or more gateways. A gateway extends a tunnel-based logical network into a physical network by bidirectionally forwarding packets between tunnels and a physical Ethernet port. This allows non-virtualized machines to participate in logical networks. A gateway may be a physical host, a virtual machine, or an ASIC-based hardware switch that supports the vtep

Figure 19 depicts the logical architecture of the Open Virtual Network.

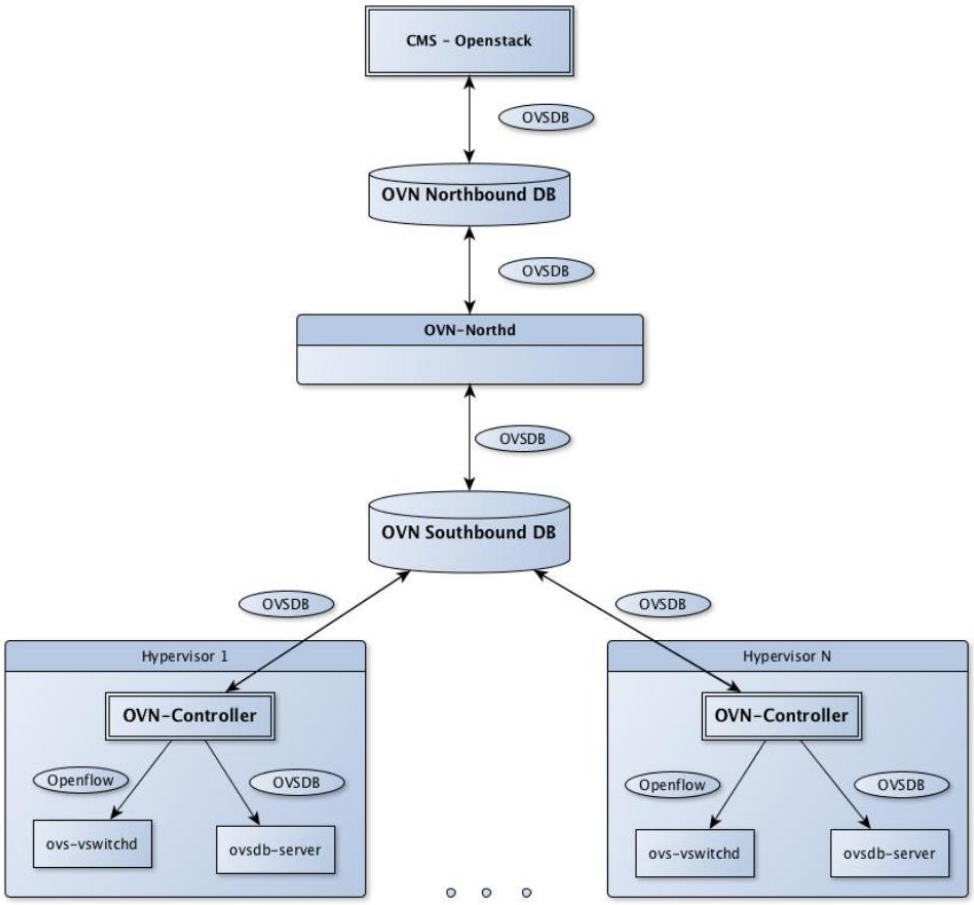

Figure 19. Logical Architecture of Open Virtual Network

The figure above shows how the major components of OVN and related software interact. Starting at the top of the diagram, we have:

- The Cloud Management System, as defined above.
- The **OVN/CMS Plugin** is the component of the CMS that interfaces to OVN. In OpenStack, this is a Neutron plug-in. The plugin's main purpose is to translate the CMS's notion of logical network configuration, stored in the CMS's configuration database in a CMS-specific format, into an intermediate representation understood by OVN.
  This component is necessarily CMS-specific, so a new plugin needs to be developed for each CMS that is integrated with OVN.
- The **OVN Northbound Database** receives the intermediate representation of logical network configuration passed down by the OVN/CMS Plugin. The database schema is meant to be ``impedance matched'' with the concepts used in a CMS, so that it directly supports notions of logical switches, routers, ACLs, and so on.

  The OVN Northbound Database has only two clients: the OVN/CMS Plugin above it and ovn-northd below it.

- **ovn-northd** connects to the OVN Northbound Database above it and the OVN Southbound Database below it. It translates the logical network configuration in terms of conventional network concepts, taken from the OVN Northbound Database, into logical datapath flows in the OVN Southbound Database below it.

- The **OVN Southbound Database** is the center of the system. Its clients are ovn-northd above it and ovn-controller on every transport node below it.

  The OVN Southbound Database contains three kinds of data: Physical Network (PN) tables that specify how to reach hypervisor and other nodes, Logical Network (LN) tables that describe the logical network in terms of ``logical datapath flows,'' and Binding tables that link logical network components' locations to the physical network.

  The hypervisors populate the PN and **Port_Binding** tables, whereas ovn-northd populates the LN tables.

The remaining components are replicated onto each hypervisor:

- **ovn-controller** is OVN's agent on each hypervisor and software gateway. Northbound, it connects to the OVN Southbound Database to learn about OVN configuration and status and to populate the PN table and the Chassis column in Binding table with the hypervisor's status. Southbound, it connects to ovs-vswitchd as an OpenFlow controller, for control over network traffic, and to the local ovsdb-server to allow it to monitor and control Open vSwitch configuration.
- ovs-vswitchd and ovsdb-server are conventional components of Open vSwitch.

5.4. Information Pathway for Open Virtual Network

In OVN [10], configuration information travels from north to south. The logical network configuration is passed to ovn-northd via the northbound database by the CMS through its OVN/CMS plugin. The configuration is then assembled into a more basic form by ovn-northd and distributed to every chassis via the southbound database.

In OVN, status information travels north to south. Only a few different types of status information are currently provided by OVN. The first thing that happens is that ovn-northd populates the up column in the northbound Logical Switch Port database. If the chassis column in the southbound Port Binding table is not empty, it sets up to true, otherwise to false. As a result, the CMS is able to recognize when a VM's networking has started.

Second, OVN informs the CMS of the realization of its configuration, or whether the configuration made available by the CMS has been implemented. This functionality necessitates CMS involvement in a sequence number protocol, which functions as follows:

1. When the CMS updates the configuration in the northbound database, as part of the same transaction, it increments the value of the nb_cfg column in the NB_Global table. (This is only necessary if the CMS wants to know when the configuration has been realized.)
2. When ovn-northd updates the southbound database based on a given snapshot of the northbound database, it copies nb_cfg from northbound NB_Global into the southbound database SB_Global table, as part of the same transaction. (Thus, an observer monitoring both databases can determine when the southbound database is caught up with the northbound.)
3. ovn-northd updates sb_cfg in the northbound NB Global table to the nb cfg version that was pushed down once it receives confirmation from the southbound database server that its modifications have been committed. (Therefore, without connecting to the southbound database, the CMS or another observer can identify when the southbound database has caught up.)

4. The ovn-controller process on each chassis receives the updated southbound database, with the updated nb_cfg. This process in turn updates the physical flows installed in the chassis's Open vSwitch instances. When it receives confirmation from Open vSwitch that the physical flows have been updated, it updates nb_cfg in its own Chassis record in the southbound database.
5. ovn-northd monitors the nb_cfg column in all of the Chassis records in the southbound database. It keeps track of the minimum value among all the records and copies it into the hv_cfg column in the northbound NB_Global table. (Thus, the CMS or another observer can determine when all of the hyper-visors have caught up to the northbound configuration.)

Logical Networks

Logical network concepts in OVN include logical switches and logical routers, the logical version of Ethernet switches and IP routers, respectively. Like their physical cousins, logical switches and routers can be connected into sophisticated topologies. Logical switches and routers are ordinarily purely logical entities, that is, they are not associated or bound to any physical location, and they are implemented in a distributed manner at each hypervisor that participates in OVN.

Logical switch ports (LSPs) are points of connectivity into and out of logical switches. There are many kinds of logical switch ports. The most ordinary kind represent VIFs, that is, attachment points for VMs or containers. A VIF logical port is associated with the physical location of its VM, which might change as the VM migrates. (A VIF logical port can be associated with a VM that is powered down or suspended. Such a logical port has no location and no connectivity.)

Logical router ports (LRPs) are points of connectivity into and out of logical routers. A LRP connects a logical router either to a logical switch or to another logical router. Logical routers only connect to VMs, containers, and other network indirectly, through logical switches.

Logical switches and logical routers have distinct kinds of logical ports, so properly speaking one should usually talk about logical switch ports or logical router ports. However, an unqualified ``logical port'' usually refers to a logical switch port. When a VM sends a packet to a VIF logical switch port, the Open vSwitch flow tables simulate the packet's journey through that logical switch and any other logical routers and logical switches that it might encounter. This happens without transmitting the packet across any physical medium: the flow tables implement all of the switching and routing decisions and behavior. If the flow tables ultimately decide to output the packet at a logical port attached to another hypervisor (or another kind of transport node), then that is the time at which the packet is encapsulated for physical network transmission and sent.

5.5. Architectural Physical Life Cycle of a Packet

This section explains the route an OVN packet takes as it moves from one virtual machine or container to another. This following section focuses on a packet's physical journey through an Open Virtual Network architecture.

A packet is initially sent on a port connected to the OVN integration bridge by a virtual machine or container running on the ingress hypervisor. Then:

1. OpenFlow table 0 performs physical-to-logical translation. It matches the packet's ingress port. Its actions annotate the packet with logical metadata, by setting the logical

datapath field to identify the logical datapath that the packet is traversing and the logical input port field to identify the ingress port. Then it resubmits to table 8 to enter the logical ingress pipeline. Packets that originate from a container nested within a VM are treated in a slightly different way. The originating container can be distinguished based on the VIF-specific VLAN ID, so the physical-to-logical translation flows additionally match on VLAN ID and the actions strip the VLAN header. Following this step, OVN treats packets from containers just like any other packets.

Table 0 also processes packets that arrive from other chassis. It distinguishes them from other packets by ingress port, which is a tunnel. As with packets just entering the OVN pipeline, the actions annotate these packets with logical datapath metadata. For tunnel types that support it, they are also annotated with logical ingress port metadata. In addition, the actions set the logical output port field, which is available because in OVN tunneling occurs after the logical output port is known. These pieces of information are obtained from the tunnel encapsulation metadata. Then the actions resubmit to table 33 to enter the logical egress pipeline.

2. OpenFlow tables 8 through 31 execute the logical ingress pipeline from the Logical_Flow table in the OVN Southbound database. These tables are expressed entirely in terms of logical concepts like logical ports and logical datapaths. A big part of ovn-controller's job is to translate them into equivalent OpenFlow (in particular it translates the table numbers: Logical_Flow tables 0 through 23 become OpenFlow tables 8 through 31).

Each logical flow maps to one or more OpenFlow flows. An actual packet ordinarily matches only one of these, although in some cases it can match more than one of these flows (which is not a problem because all of them have the same actions). ovn-controller uses the first 32 bits of the logical flow's UUID as the cookie for its OpenFlow flow or flows. (This is not necessarily unique, since the first 32 bits of a logical flow's UUID is not necessarily unique.)

Some logical flows can map to the Open vSwitch ``conjunctive match'' extension. Flows with a conjunction action use an OpenFlow cookie of 0, because they can correspond to multiple logical flows. The OpenFlow flow for a conjunctive match includes a match on conj_id.

Some logical flows may not be represented in the OpenFlow tables on a given hypervisor, if they could not be used on that hypervisor. For example, if no VIF in a logical switch resides on a given hypervisor, and the logical switch is not otherwise reachable on that hypervisor (e.g. over a series of hops through logical switches and routers starting from a VIF on the hypervisor), then the logical flow may not be represented there.

The majority of OVN actions have rather apparent OpenFlow implementations (with OVS extensions), such as next; which is implemented as resubmit, field = constant; and set field. There are a handful that merit further explanation:

output: Resubmitting the packet to table 37 allowed for implementation. Each output action that the pipeline executes is independently resubmitted to table 37 if there are multiple output actions. This can be used to transmit copies of the packet to various ports at once. (Utilizing a logical multicast output port might reduce bandwidth usage across hypervisors provided the packet was not modified between the output operations and part of the copies were headed for the same hypervisor.)

get_arp(P, A);

get_nd(P, A);

Implemented by storing arguments into OpenFlow fields, then resubmitting to table 66, which ovn-controller populates with flows generated from the MAC_Binding table in the OVN Southbound database. If there is a match in table 66, then its actions store the bound MAC in the Ethernet destination address field.

put_arp(P, A, E);

put_nd(P, A, E);

Implemented by storing the arguments into OpenFlow fields, then outputting a packet to ovn-controller, which updates the MAC_Binding table.

3. OpenFlow tables 37 through 39 implement the output action in the logical ingress pipeline. Specifically, table 37 handles packets to remote hypervisors, table 38 handles packets to the local hypervisor, and table 39 checks whether packets whose logical ingress and egress port are the same should be discarded.

Logical patch ports are a special case. Logical patch ports do not have a physical location and effectively reside on every hypervisor. Thus, flow table 38, for output to ports on the local hypervisor, naturally implements output to unicast logical patch ports too. However, applying the same logic to a logical patch port that is part of a logical multicast group yields packet duplication, because each hypervisor that contains a logical port in the multicast group will also output the packet to the logical patch port. Thus, multicast groups implement output to logical patch ports in table 37.

Each flow in table 37 matches on a logical output port for unicast or multicast logical ports that include a logical port on a remote hypervisor. Each flow's actions implement sending a packet to the port it matches. For unicast logical output ports on remote hypervisors, the actions set the tunnel key to the correct value, then send the packet on the tunnel port to the correct hypervisor. (When the remote hypervisor receives the packet, table 0 there will recognize it as a tunneled packet and pass it along to table 38.) For multicast logical output ports, the actions send one copy of the packet to each remote hypervisor, in the same way as for unicast destinations. If a multicast group includes a logical port or ports on the local hypervisor, then its actions also resubmit to table 38.

Flows in table 38 resemble those in table 37 but for logical ports that reside locally rather than remotely. For unicast logical output ports on the local hypervisor, the actions just resubmit to table 39. For multicast output ports that include one or more logical ports on the local hypervisor, for each such logical port P, the actions change the logical output port to P, then resubmit to table 39.

A special case is that when a localnet port exists on the datapath, remote port is connected by switching to the localnet port. In this case, instead of adding a flow in table 37 to reach the remote port, a flow is added in table 38 to switch the logical outport to the localnet port, and resubmit to table 38 as if it were unicasted to a logical port on the local hypervisor.

Table 39 matches and drops packets for which the logical input and output ports are the same and the MLF_ALLOW_LOOP-BACK flag is not set. It also drops MLF_LOCAL_ONLY packets directed to a localnet port. It resubmits other packets to table 40.

4. OpenFlow tables 40 through 63 execute the logical egress pipeline from the Logical_Flow table in the OVN Southbound database. The egress pipeline can perform a final stage of validation before packet delivery. Eventually, it may execute an output action, which ovn-controller implements by resubmitting to table 64. A packet for which the pipeline never executes output is effectively dropped (although it may have been transmitted through a tunnel across a physical network).

The egress pipeline cannot change the logical output port or cause further tunneling.

5. Table 64 bypasses OpenFlow loopback when MLF_ALLOW_LOOPBACK is set. Logical loopback was handled in table 39, but OpenFlow by default also prevents loopback to the OpenFlow ingress port. Thus, when MLF_ALLOW_LOOPBACK is set, OpenFlow table 64 saves the OpenFlow ingress port, sets it to zero, resubmits to table 65 for logical-to-physical transformation, and then restores the OpenFlow ingress port, effectively disabling OpenFlow loopback prevents. When MLF_ALLOW_LOOPBACK is unset, table 64 flow simply resubmits to table 65.

6. OpenFlow table 65 performs logical-to-physical translation, the opposite of table 0. It matches the packet's logical egress port. Its actions output the packet to the port attached to the OVN integration bridge that represents that logical port. If the logical egress port is a container nested with a VM, then before sending the packet the actions push on a VLAN header with an appropriate VLAN ID.

5.6. Implementing Open Virtual Network with LXD for High Availability for Clustering

High availability (HA) is a characteristic of a system which aims to ensure an agreed level of operational performance, usually uptime, for a higher than normal period.

OVN supports Layer 3 high availability (L3 HA) without any special configuration. OVN automatically schedules the router port to all available gateway nodes that can act as an L3 gateway on the specified external network. OVN L3 HA uses the gateway_chassis column in the OVN Logical_Router_Port table. Most functionality is managed by OpenFlow rules with bundled active_passive outputs. The ovn-controller handles the Address Resolution Protocol (ARP) responder and router enablement and disablement. Gratuitous ARPs for FIPs and router external addresses are also periodically sent by the ovn-controller.

Bidirectional Forwarding Detection (BFD) Monitoring

OVN uses the Bidirectional Forwarding Detection (BFD) protocol to monitor the availability of the gateway nodes. This protocol is encapsulated on top of the Geneve tunnels established from node to node.

Each gateway node monitors all the other gateway nodes in a star topology in the deployment. Gateway nodes also monitor the compute nodes to let the gateways enable and disable routing of packets and ARP responses and announcements.

Each compute node uses BFD to monitor each gateway node and automatically steers external traffic, such as source and destination Network Address Translation (SNAT and DNAT), through the active gateway node for a given router. Compute nodes do not need to monitor other compute nodes.

L3 HA for OVN supports the following failure modes:

- The gateway node becomes disconnected from the network (tunneling interface).

- ovs-vswitchd stops (ovs-switchd is responsible for BFD signaling)
- ovn-controller stops (ovn-controller removes itself as a registered node).

The OVN gateway is responsible for shuffling traffic between the tunneled overlay network (governed by ovn-northd), and the legacy physical network. In a naive implementation, the gateway is a single x86 server, or hardware VTEP. For most deployments, a single system has enough forwarding capacity to service the entire virtualized network, however, it introduces a single point of failure. If this system dies, the entire OVN deployment becomes unavailable. To mitigate this risk, an HA solution is critical – by spreading responsibility across multiple systems, no single server failure can take down the network. A High Availability solution is both critical to the manageability of the system, and extremely difficult to get right.

In order to achieve this goal, there are two broad approaches one can take. The HA cluster can appear to the network like a big Layer 2 Ethernet Switch, or like a big IP Router. These approaches are called L2HA, and L3HA respectively. L2HA allows Ethernet broadcast domains to extend into logical space, a significant advantage, but this comes at a cost. The need to avoid transient L2 loops during failover significantly complicates their design. On the other hand, L3HA works for most use cases, is simpler, and fails more gracefully. For these reasons, it is suggested that OVN supports an L3HA model, leaving L2 HA for future work (or third party VTEP providers). Figure 20 represents the complete High Availability cluster setup of OVN implemented in lab.

❖ **Pre-requisites:**

An Ubuntu (20.04 version) machine or server with LXC and LXD pre-installed with a minimum configuration of 16GB RAM and 4 CPU Cores.

❖ **Creating the OVN High Availability cluster system in a Lab Environment:**

Firstly we need to create the 4 virtual machines that we need to configure later where we use 3 virtual machines as OVN controller nodes and the final one as an OVN host node.

```
lxc init images:ubuntu/focal v5 --vm
lxc init images:ubuntu/focal v6 --vm
lxc init images:ubuntu/focal v7 --vm
lxc init images:ubuntu/focal v8 --vm
```

Next we want to ensure they have statically assigned IPs:

```
lxc config device override v5 eth0 ipv4.address=10.118.207.15
lxc config device override v6 eth0 ipv4.address=10.118.207.16
lxc config device override v7 eth0 ipv4.address=10.118.207.17
lxc config device override v8 eth0 ipv4.address=10.118.207.18
```

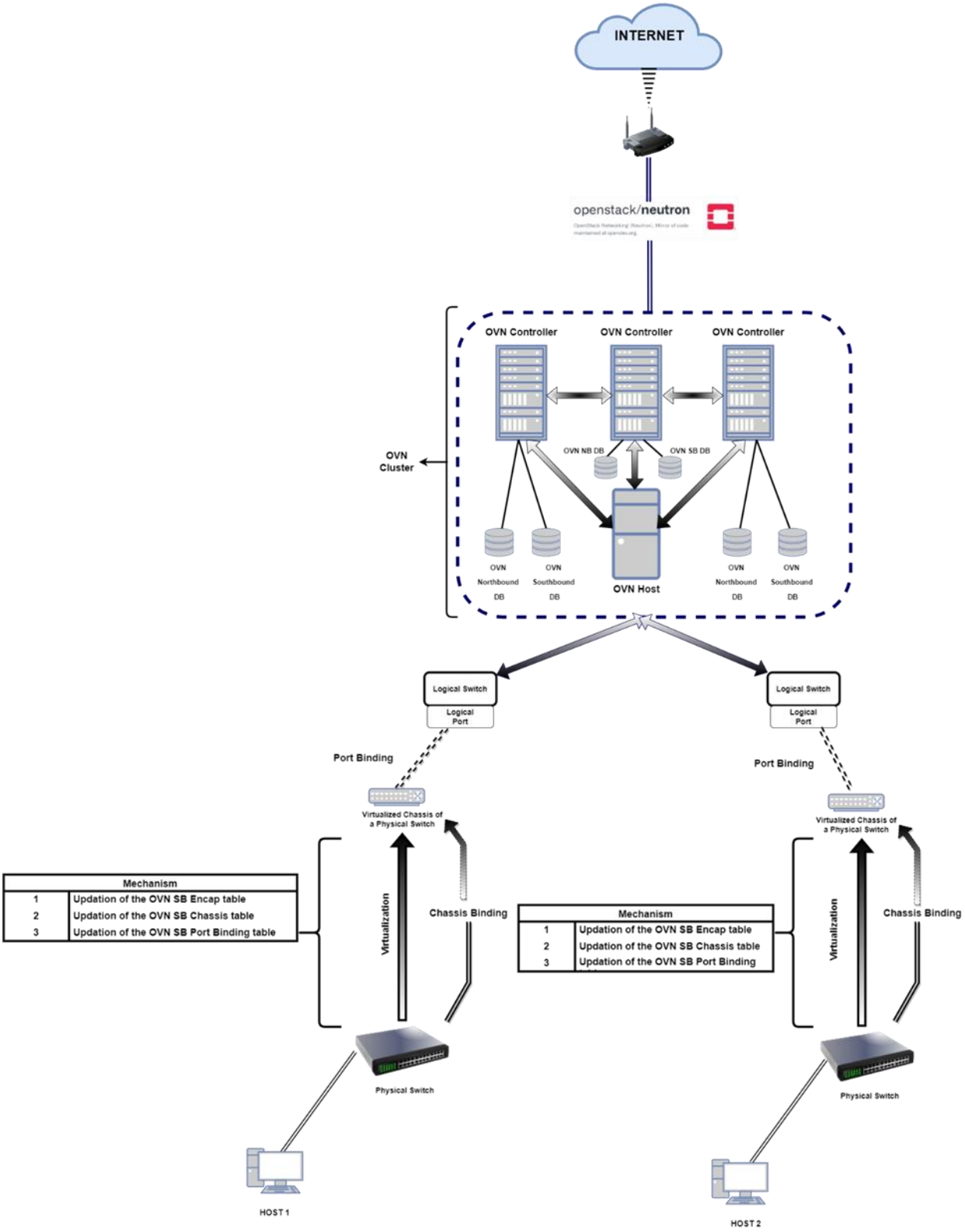

Figure 20. Complete Lab Setup of Open Virtual Network including Mapping of PNEs to VNEs

Now we want to start all the virtual machines to start configuring the setup:

```
lxc start v5 v6 v7 v8
```

NAME	STATE	IPV4	IPV6	TYPE	SNAPSHOTS
v5	RUNNING	10.118.207.15 (enp5s0)	fd42:6524:71c7:29ff:216:3eff:fe9e:da72 (enp5s0)	VIRTUAL-MACHINE	0
v6	RUNNING	10.118.207.16 (enp5s0)	fd42:6524:71c7:29ff:216:3eff:fe5a:2efb (enp5s0)	VIRTUAL-MACHINE	0
v7	RUNNING	10.118.207.17 (enp5s0)	fd42:6524:71c7:29ff:216:3eff:fea4:5cc4 (enp5s0)	VIRTUAL-MACHINE	0
v8	RUNNING	10.118.207.18 (enp5s0)	fd42:6524:71c7:29ff:216:3eff:fe19:5e4b (enp5s0)	VIRTUAL-MACHINE	0

Figure 21. Output of lxc-list

Now we try to setup our system with our focus on the first virtual machine v5 with the ip address **10.118.207.15** in order to enter into the linux virtual machine we need to use to shell command

```
lxc shell v5
```

To install lxd components we need the snap store up and running. Also we need to check ip address time to time for that we need “net-tools” package.

```
sudo apt install net-tools snapd –y
```

Now with the snapstore running we can easily install the latest stable lxd version.

```
sudo snap install lxd –y
```

We can consequently check whether our lxd version is the latest stable release or not. If not it will be updated.

```
snap refresh lxd --channel=latest
```

Next we need to install the open virtual network packages. The packages we need for the controller node are ovn-central and ovn-host.

```
sudo apt install ovn-central ovn-host –y
```

Now before we edit the configuration of the ovn-central file we need to make sure that the ovn-central is not actually running as then it will cause further issues and the configuration change will be corrupted.

```
systemctl stop ovn-central
```

and for configuration file editing we use the “nano” tool

```
sudo apt install nano
```

Now we need to add this section to /etc/default/ovn-central with the following settings configuration:

```
OVN_CTL_OPTS="

--db-nb-addr=SERVER1

--db-nb-create-insecure-remote=yes

--db-sb-addr=SERVER1

--db-sb-create-insecure-remote=yes

--db-nb-cluster-local-addr=LOCAL

--db-sb-cluster-local-addr=LOCAL

--ovn-northd-nb-
db=tcp:SERVER1:6641,tcp:SERVER2:6641,tcp:SERVER3:6641

--ovn-northd-sb-
db=tcp:SERVER1:6642,tcp:SERVER2:6642,tcp:SERVER3:6642"
```

Here in the places containing **SERVER1** we need to give the ip address of the first vm (i.e. v5) and in the places containing **SERVER2** we need to give the ip address of the second vm (i.e. v6), in the places containing **SERVER3** we need to give the ip address of the third vm (i.e. v7) and finally in the places containing **LOCAL** we need to give the ip address we are currently using for this terminal.

Now finally when the configuration is modified we need to restart the ovn-central module so that the ovn-central module restarts with the new configuration loadout.

```
systemctl start ovn-central
```

Now we need to check whether all the ovsdb server logs are properly generated or not. For this we use these below commands to monitor whether the ovn-central daemon is running with exactly the newly modified configurations without any errors or hiccups.

Checking the logs of OVN North Daemon.

```
tail –f /var/log/ovn/ovn-northd.log
```

Checking the logs of OVN Northbound database.

```
tail –f /var/log/ovn/ovsdb-server-nb.log
```

Checking the logs of OVN Southbound database.

```
tail –f /var/log/ovn/ovsdb-server-sb.log
```

```
exit
```

Now we need to replicate the same steps for our second and third controller nodes i.e. **v6** and **v7**

So we need to “lxc shell v6” and “lxc shell v7” and just do the same steps that were done for **v5.**

Now we need to configure our host node i.e. **v8**

```
lxc shell v8
```

We need to install the basic packages that we need for our configuration. We need "net-tools" to monitor the network status of our system and "snapd" to access the snap store.

```
sudo apt install net-tools snapd –y
```

We need to install the latest stable version of the LXD.

```
sudo snap install lxd
```

We need to verify whether the LXD installed version is the latest stable version or not.

```
snap refresh lxd --channel=latest
```

We need to install just the "ovn-host" package for our host node.

```
sudo apt install ovn-host
```

For host node v8 we don't need to change the configuration files.

On each VM check that the databases are working using (you should get empty output but no errors):

```
OVN_NB_DB=tcp:SERVER1:6641,tcp:SERVER2:6641,tcp:SERVER3:6641

ovn-nbctl show

OVN_SB_DB=tcp:SERVER1:6642,tcp:SERVER2:6642,tcp:SERVER3:6642

ovn-sbctl show
```

Here in the places containing **SERVER1** we need to give the ip address of the first vm (i.e. v5) and in the places containing **SERVER2** we need to give the ip address of the second vm (i.e. v6), in the places containing **SERVER3** we need to give the ip address of the third vm (i.e. v7).

Now we need to configure the Open vSwitch section of all the virtual machines where we decide that the ovn encapsulation type has been set to "Geneve" protocol.

```
ovs-vsctl set open_vswitch . external_ids:\

ovnremote=tcp:SERVER1:6642,tcp:SERVER2:6642,tcp:SERVER3:6642 \

external_ids:ovn-encap-type=geneve \

external_ids:ovn-encap-ip=LOCAL
```

Now we need to see what does the Open vSwitch control daemon shows on how the connectivity has been established.

```
sudo ovs-vsctl show
```

```
cdcju@cdcju-workstation:~$ lxc shell v5
root@v5:~# ovs-vsctl show
41819aca-455e-4711-9ff1-5471758bf92e
    Bridge br-int
        fail_mode: secure
        Port patch-br-int-to-lxd-net2-ls-ext-lsp-provider
            Interface patch-br-int-to-lxd-net2-ls-ext-lsp-provider
                type: patch
                options: {peer=patch-lxd-net2-ls-ext-lsp-provider-to-br-int}
        Port ovn-990dc8-0
            Interface ovn-990dc8-0
                type: geneve
                options: {csum="true", key=flow, remote_ip="10.118.207.17"}
                bfd_status: {diagnostic="No Diagnostic", flap_count="1", forwarding="true", remote_diagnostic="No Diagnostic", remote_state=up, state=up}
        Port br-int
            Interface br-int
                type: internal
        Port patch-lxd-net2-ls-ext-lsp-provider-to-br-int
            Interface patch-lxd-net2-ls-ext-lsp-provider-to-br-int
                type: patch
                options: {peer=patch-br-int-to-lxd-net2-ls-ext-lsp-provider}
        Port ovn-4ba2e3-0
            Interface ovn-4ba2e3-0
                type: geneve
                options: {csum="true", key=flow, remote_ip="10.118.207.16"}
                bfd_status: {diagnostic="No Diagnostic", flap_count="1", forwarding="true", remote_diagnostic="No Diagnostic", remote_state=up, state=up}
    ovs_version: "2.13.5"
```

Figure 22. Output of "sudo ovs-vsctl show"

Here in the places containing **SERVER1** we need to give the ip address of the first vm (i.e. **v5**) and in the places containing **SERVER2** we need to give the ip address of the second vm (i.e. **v6**), in the places containing **SERVER3** we need to give the ip address of the third vm (i.e. **v7**) and finally in the places containing **LOCAL** we need to give the ip address we are currently using for this terminal.

Now we do this same configuration in the terminal every virtual machine i.e. **v6**, **v7** and **v8**.

```
ovs-vsctl set open_vswitch . external_ids:\

ovnremote=tcp:SERVER1:6642,tcp:SERVER2:6642,tcp:SERVER3:6642 \

external_ids:ovn-encap-type=geneve \

external_ids:ovn-encap-ip=LOCAL
```

Hence at this point all the configuration that needed to be done is completed and the Open virtual network system is formed.

The next portion is that we need to create a High availability cluster using the 4 virtual machines.

Creating the high availability cluster should be very straight forward.

We need to shell inside our first virtual machine (i.e. v5)

```
lxc shell v5

lxd init
```

Now when configuring we need to make sure that we need to say "**Yes**" to "do you want to use clustering question?" and say "**No**" to the "Do you want to create a fan overlay network" question and for the rest we can choose the default option. Next we need to add the rest of the virtual machines to our clustering network.

So we need to add the virtual machines using

```
lxc cluster add v6

lxc cluster add v7

lxc cluster add v8
```

For each of these commands an lxc token will be generated and we need to keep a copy of those tokens for corresponding virtual machines.

Now for each of those virtual machines we need to shell inside and run the "**lxc init**" and we need answer "**Yes**" to the question that "Do you want to join an existing cluster?" question and we need to paste that token generated by the cluster add command when required and consecutively add all the virtual machines to the cluster.

Following these steps will create the high availability open virtual network cluster of 3 controller nodes and 1 compute node.

Now we can check the status of our OVN High Availability Cluster.

```
lxc-cluster list
```

```
root@v5:~# lxc cluster list
+------+---------------------------+-----------------+--------------+----------------+-------------+--------+------------------+
| NAME |           URL             |      ROLES      | ARCHITECTURE | FAILURE DOMAIN | DESCRIPTION | STATE  |      MESSAGE     |
+------+---------------------------+-----------------+--------------+----------------+-------------+--------+------------------+
| v5   | https://10.118.207.15:8443 | database-leader | x86_64       | default        |             | ONLINE | Fully operational |
|      |                           | database        |              |                |             |        |                  |
+------+---------------------------+-----------------+--------------+----------------+-------------+--------+------------------+
| v6   | https://10.118.207.16:8443 | database-standby | x86_64       | default        |             | ONLINE | Fully operational |
+------+---------------------------+-----------------+--------------+----------------+-------------+--------+------------------+
| v7   | https://10.118.207.17:8443 | database        | x86_64       | default        |             | ONLINE | Fully operational |
+------+---------------------------+-----------------+--------------+----------------+-------------+--------+------------------+
| v8   | https://10.118.207.18:8443 | database        | x86_64       | default        |             | ONLINE | Fully operational |
+------+---------------------------+-----------------+--------------+----------------+-------------+--------+------------------+
```

Figure 23. Output of lxc-cluster list

Now for the next step we need to create virtualized physical network and an open virtual network using the virtualized physical network.

For that we need to shell into our controller virtual machine i.e. **v5**

```
lxc shell v5
```

While creating the virtualized physical link we need to use a common parent port and for that we use the OVN integration bridge commonly abbreviated as "**br-int**".

```
lxc network create PHYSICAL-OVN-LINK --type=physical parent=br-int
--target v5

lxc network create PHYSICAL-OVN-LINK --type=physical parent=br-int
--target v6

lxc network create PHYSICAL-OVN-LINK --type=physical parent=br-int
--target v7

lxc network create PHYSICAL-OVN-LINK --type=physical parent=br-int
--target v8

lxc network create PHYSICAL-OVN-LINK --type=physical
```

This will actually create a virtualized version of a physical network with the name "PHYSICAL-OVN-LINK".

Now we can check whether our physical OVN link exists or not and how many nodes in the cluster are using the physical OVN link.

lxc network list

Now we need to configure the PHYSICAL-OVN-LINK and set the ipv4 address range and ipv6 address range.

lxc network set PHYSICAL-OVN-LINK ipv4.ovn.ranges=10.118.207.200-10.118.207.254

ipv6.ovn.ranges=2602:fc62:b:1016:1::200-2602:fc62:b:1016:1::254

Next we need to set the ipv4 and ipv6 gateways.

lxc network set PHYSICAL-OVN-LINK ipv4.gateway=10.118.207.1/24 ipv6.gateway=2602:fc62:b:1016::1/64 dns.nameservers=8.8.8.8

Then we need to set the tcp address for our OVN-Northbound connection using lxc config.

lxc config set network.ovn.northbound_connection tcp:SERVER1:6641

Finally we create the Open Virtual Network using the lxc network commands and name it as "ACTUAL-OVN".

lxc network create ACTUAL-OVN --type=ovn

To finally check the status of our created network and whether it is working and functioning or not.

lxc network list

```
root@v5:~# lxc network list
+------------------+----------+---------+---------------+---------------------------+-------------+---------+---------+
|       NAME       |   TYPE   | MANAGED |      IPV4     |           IPV6            | DESCRIPTION | USED BY |  STATE  |
+------------------+----------+---------+---------------+---------------------------+-------------+---------+---------+
| ACTUAL-OVN       | ovn      | YES     | 10.187.97.1/24 | fd42:d34c:42ce:b996::1/64 |             | 4       | CREATED |
+------------------+----------+---------+---------------+---------------------------+-------------+---------+---------+
| PHYSICAL-OVN-LINK | physical | YES     |               |                           |             | 1       | CREATED |
+------------------+----------+---------+---------------+---------------------------+-------------+---------+---------+
| br-int           | bridge   | NO      |               |                           |             | 1       |         |
+------------------+----------+---------+---------------+---------------------------+-------------+---------+---------+
| enp5s0           | physical | NO      |               |                           |             | 0       |         |
+------------------+----------+---------+---------------+---------------------------+-------------+---------+---------+
```

Figure 24. Output of lxc network list

Further we can check the configuration and components of the "ACTUAL-OVN" network.

lxc network show ACTUAL-OVN

Now we can see the Open Virtual Network is up and running and we can create VMs or VIFs which are given Open Virtual Network access.

Next we can check what are the components and configuration of each of the components of Open Virtual Network Northbound and Southbound Databases.

sudo ovn-nbctl show

```
root@v5:~# ovn-nbctl show
switch 5f0e1018-8b1f-48c6-8d1f-2b4bae24e3c2 (lxd-net2-ls-ext)
    port lxd-net2-ls-ext-lsp-router
        type: router
        router-port: lxd-net2-lr-lrp-ext
    port lxd-net2-ls-ext-lsp-provider
        type: localnet
        addresses: ["unknown"]
switch a89eea71-943e-4b39-a0b7-ef2d54aa0b4f (lxd-net2-ls-int)
    port lxd-net2-instance-b84c1403-c492-4b21-9609-e58c92fa1d6c-eth0
        addresses: ["00:16:3e:24:9c:d5 dynamic"]
    port lxd-net2-ls-int-lsp-router
        type: router
        router-port: lxd-net2-lr-lrp-int
router d53de686-d996-4b06-bb0a-1faddbea9d6d (lxd-net2-lr)
    port lxd-net2-lr-lrp-ext
        mac: "00:16:3e:bd:62:3e"
        networks: ["10.118.207.200/24", "2602:fc62:b:1016:1::200/64"]
    port lxd-net2-lr-lrp-int
        mac: "00:16:3e:bd:62:3e"
        networks: ["10.187.97.1/24", "fd42:d34c:42ce:b996::1/64"]
    nat 42542017-0b96-422f-bd62-5da2eaec9162
        external ip: "2602:fc62:b:1016:1::200"
        logical ip: "fd42:d34c:42ce:b996::/64"
        type: "snat"
    nat c4f1b3de-b29d-451b-8d3a-648246c7076a
        external ip: "10.118.207.200"
        logical ip: "10.187.97.0/24"
        type: "snat"
```

Figure 25. Output of OVN Northbound database

"**ovn-nbctl**" shows us the configurations of the OVN-Northbound database. The ovn−nbctl program configures the OVN_Northbound database by providing a high-level interface to its configuration database. ovn−nbctl connects to an ovsdb−server process that maintains an OVN_Northbound configuration database. Using this connection, it queries and possibly applies changes to the database, depending on the supplied commands. When it is invoked in the most ordinary way, ovn−nbctl connects to an OVSDB server that hosts the northbound database, retrieves a partial copy of the database that is complete enough to do its work, sends a transaction request to the server, and receives and processes the server's reply. In common interactive use, this is fine, but if the database is large, the step in which ovn−nbctl retrieves a partial copy of the database can take a long time, which yields poor performance overall.

```
sudo ovn-sbctl show
```

```
root@v5:~# ovn-sbctl show
Chassis "990dc862-39f0-4b01-ab4e-9704a39b7bb9"
    hostname: v7
    Encap geneve
        ip: "10.118.207.17"
        options: {csum="true"}
    Port_Binding lxd-net2-instance-b84c1403-c492-4b21-9609-e58c92fa1d6c-eth0
Chassis "4ba2e389-a4c7-4722-9255-47f48600a28d"
    hostname: v6
    Encap geneve
        ip: "10.118.207.16"
        options: {csum="true"}
    Port_Binding lxd-net2-instance-7963fb23-cdb3-4e74-8e88-035f72531cd6-eth0
Chassis "3f33f14d-ab62-42c8-8f84-73415ccd2118"
    hostname: v5
    Encap geneve
        ip: "10.118.207.15"
        options: {csum="true"}
    Port_Binding lxd-net2-instance-cc3cae38-8eae-4f33-aa98-e8c5c9b21fe6-eth0
```

Figure 26. Output of the OVN Southbound database

"**ovn-sbctl**" shows the configurations of the OVN-Southbound database. The ovn-sbctl program configures the OVN_Southbound database by providing a high-level interface to its configuration database. ovn-sbctl connects to an ovsdb−server process that maintains an OVN_Southbound configuration database. Using this connection, it queries and possibly applies changes to the database, depending on the supplied commands. When it is invoked in the most ordinary way, ovn−sbctl connects to an OVSDB server that hosts the southbound database, retrieves a partial copy of the database that is complete enough to do its work, sends a transaction request to the server, and receives and processes the server's reply. In common interactive use, this is fine, but if the database is large, the step in which ovn−sbctl retrieves a partial copy of the database can take a long time, which yields poor performance overall.

With this final command we can see all the configurations and connections for the Open Virtual Network has been created and subsequently deployed.

Chapter 6

Association of Physical Network with Virtual Network Elements

The most crucial aspect of network virtualization is the mapping of physical network elements to their virtual network elements. Network Virtualization (NV) refers to abstracting network resources that were traditionally delivered in hardware to software. NV can combine multiple physical networks into one virtual, software-based network, or it can divide one physical network into separate, independent virtual networks. In order to clarify about the mapping or integration of Physical Network Elements with Virtual Interfaces, we need to have conceptual clarity about VIFs (Virtual Interfaces)

6.1. Life Cycle of a VIF (Virtual Interfaces)

A VIF on a hypervisor is a virtual network interface attached either to a VM or a container running directly on that hypervisor (This is different from the interface of a container running inside a VM).

1. A VIF's life cycle begins when a CMS administrator creates a new VIF using the CMS user interface or API and adds it to a switch (one implemented by OVN as a logical switch). The CMS updates its own configuration. This includes associating unique, persistent identifier vif-id and Ethernet address mac with the VIF.

2. The CMS plugin updates the OVN Northbound database to include the new VIF, by adding a row to the Logical_Switch_Port table. In the new row, name is vif-id, mac is mac, switch points to the OVN logical switch's Logical_Switch record, and other columns are initialized appropriately.

3. ovn-northd receives the OVN Northbound database update. In turn, it makes the corresponding updates to the OVN Southbound database, by adding rows to the OVN Southbound database Logical_Flow table to reflect the new port, e.g. add a flow to recognize that packets destined to the new port's MAC address should be delivered to it, and update the flow that delivers broadcast and multicast packets to include the new port. It also creates a record in the Binding table and populates all its columns except the column that identifies the chassis.

4. On every hypervisor, ovn-controller receives the Logical_Flow table updates that ovn-northd made in the previous step. As long as the VM that owns the VIF is powered off, ovn-controller cannot do much; it cannot, for example, arrange to send packets to or receive packets from the VIF, because the VIF does not actually exist anywhere.

5. Eventually, a user powers on the VM that owns the VIF. On the hypervisor where the VM is powered on, the integration between the hypervisor and Open vSwitch (described in Documentation/topics/integration.rst in the Open vSwitch source tree) adds the VIF to the OVN integration bridge and stores vif-id in external_ids:iface-id to indicate that the interface is an instantiation of the new VIF.

6. On the hypervisor where the VM is powered on, ovn-controller notices external_ids:iface-id in the new Interface. In response, in the OVN Southbound DB, it updates the Binding table's chassis column for the row that links the logical port from external_ids: iface-id to the hypervisor. Afterward, ovn-controller updates the local hypervisor's OpenFlow tables so that packets to and from the VIF are properly handled.

7. Some CMS systems, including OpenStack, fully start a VM only when its networking is ready. To support this, ovn-northd notices the chassis column updated for the row in Binding table and pushes this upward by updating the up column in the OVN Northbound database's Logical_Switch_Port table to indicate that the VIF is now up. The CMS, if it uses this feature, can then react by allowing the VM's execution to proceed.

8. On every hypervisor but the one where the VIF resides, ovn-controller notices the completely populated row in the Binding table. This provides ovn-controller the physical location of the logical port, so each instance updates the OpenFlow tables of its switch (based on logical datapath flows in the OVN DB Logical_Flow table) so that packets to and from the VIF can be properly handled via tunnels.

9. Eventually, a user powers off the VM that owns the VIF. On the hypervisor where the VM was powered off, the VIF is deleted from the OVN integration bridge.

10. On the hypervisor where the VM was powered off, ovn-controller notices that the VIF was deleted. In response, it removes the Chassis column content in the Binding table for the logical port.

11. On every hypervisor, ovn-controller notices the empty Chassis column in the Binding table's row for the logical port. This means that ovn-controller no longer knows the physical location of the logical port, so each instance updates its OpenFlow table to reflect that.

12. Eventually, when the VIF (or its entire VM) is no longer needed by anyone, an administrator deletes the VIF using the CMS user interface or API. The CMS updates its own configuration.

13. The CMS plugin removes the VIF from the OVN Northbound database, by deleting its row in the Logical_Switch_Port table.

14. ovn-northd receives the OVN Northbound update and in turn updates the OVN Southbound database accordingly, by removing or updating the rows from the OVN Southbound database Logical_Flow table and Binding table that were related to the now-destroyed VIF.

15. On every hypervisor, ovn-controller receives the Logical_Flow table updates that ovn-northd made in the previous step. ovn-controller updates OpenFlow tables to reflect the update, although there may not be much to do, since the VIF had already become unreachable when it was removed from the Binding table in a previous step.

6.2. Mapping Physical Networking Elements with Virtual Networking Elements

Now as we have setup our OVN setup in a machine or server. We can now look forward towards integrating some physical network elements like switches with the Open Virtual Network and logical to physical mapping of physical network elements (PNEs).

Pre-requisites:

- ✓ A server or machine which has the Open Virtual Network setup on it.
- ✓ Ethernet Cables
- ✓ 2 OpenFlow enabled Raspberry Pi switches

1. In order to virtualize physical network elements using Open Virtual Network, we need to add the physical network elements using the virtual chassis mechanism to the OVN Southbound Database. This mechanism is called **Chassis Binding.**

Using this mechanism virtual chassis of physical network elements is added to the OVN Network setup. Physical network tables contain information about the chassis nodes in the system. This contains all the information necessary to wire the overlay, such as IP addresses, supported tunnel types, and security keys. The amount of physical network data is small (O(n) in the number of chassis) and it changes infrequently, so it can be replicated to every chassis.

The "**Chassis"** and "**Encap"** tables are the physical network tables.

2. In order to virtualize **PNE**'s in **OVN** they need to be updated to two tables ("**Chassis**" and "**Encap**" table) in the **OVN Southbound database**. Also for **logical to physical bindings**, The "**Port_Binding"** and "**Datapath_Binding"** tables contain binding data. These tables **link logical and physical components**. They show the current placement of logical components (such as VMs and VIFs) onto chassis, and map logical entities to the values that represent them in tunnel encapsulations.

The **Port_Binding** and **Datapath_Binding** tables contain binding data.

These tables link logical and physical components. They show the current placement of logical components (such as VMs and VIFs) onto chassis, and map logical entities to the values that represent them in tunnel encapsulations. These tables change frequently, at least every time a VM powers up or down or migrates, and especially quickly in a container environment. The amount of data per VM (or VIF) is small.

3. Now to integrate our physical network element to the OVN Soutbound database we need to use the following command "**sudo ovn-sbctl chassis-add chassis encap-type encap-ip**" where **chassis** is the name of the virtual chassis, **encap-type** is the mode of encapsulation of the network element and **encap-ip** is the encapsulation ip of the network element (physical switch).

Mechanism: **sudo ovn-sbctl chassis-add <chassis> <encap-type> <encap-ip>**

In our case, the code is **sudo ovn-sbctl chassis-add Physical_switch1 geneve 192.168.0.85** for virtualizing the first physical switch and the code is **sudo ovn-sbctl chassis-add Physical_switch2 geneve 192.168.0.86** for the second physical switch.

Now the physical switches have been virtually added to OVN Soutbound's "Chassis" and "Encap" tables.

In order to display the whole setup of the chassis binding

Code:

sudo ovn-sbctl list Chassis

sudo ovn-sbctl list Encap

4. The next step is **Port Binding**. In Port Binding table each row in this table binds a logical port to a realization. For most logical ports, this mean s binding to some physical location, for example by binding a logical port to a VIF that belongs to a VM running on a particular hypervisor. Other logical ports, such as logical patch ports, can be realized without a specific physical location, but their bindings are still expressed through rows in this table.

For every **Logical_Switch_Port** record in OVN Northbound database, **ovn-northd** creates a record in this table. ovn-northd populates and maintains every column except the chassis column, which it leaves empty in new records.

We can also create our own Logical Port and Logical Switches to do the Port Binding with.

Code:

```
ovn-nbctl ls-add sw0

ovn-nbctl lsp-add sw0 sw0-port1

ovn-nbctl lsp-set-addresses sw0-port1 00:00:00:00:00:01

ovn-nbctl lsp-set-port-security sw0-port1 00:00:00:00:00:01

ovn-nbctl ls-add sw0

ovn-nbctl lsp-add sw0 sw0-port2

ovn-nbctl lsp-set-addresses sw0-port2 00:00:00:00:00:02

ovn-nbctl lsp-set-port-security sw0-port2 00:00:00:00:00:02
```

Next step is the Port binding for that we use “sudo ovn-sbctl lsp-bind logical-port chassis” where “logical-port” is the logical port onto which we want to map our virtual chassis to.

Mechanism: **sudo ovn-sbctl lsp-bind <logical-port> <chassis>**

Now for our machines we port bind logical ports “**lxd-net2-ls-ext-lsp-router**” and “**lxd-net2-ls-ext-provider**” to the virtual chassis’s “**Physical_switch1”** and “**Physical_switch2”** respectively. Now the properties imposed on those ports are inherently imposed on the virtual chassis’s.

Code:

```
sudo ovn-sbctl lsp-bind sw0-port1 Physical_switch1

sudo ovn-sbctl lsp-bind sw0-port2 Physical_switch2
```

Next we can status check the whole Port_Binding status using

Code:

```
sudo ovn-sbctl list Port_Binding
```

5. Datapath Binding is the type of bindings which link logical and physical components. They show the current placement of logical components (such as VMs and VIFs) onto chassis, and map logical entities to the values that represent them in tunnel encapsulations

Each row in Datapath_Binding table represents a logical datapath, which implements a logical pipeline among the ports in the Port_Binding table associated with it. In practice, the pipeline in a given logical datapath implements either a logical switch or a logical router.

The main purpose of a row in this table is provide a physical binding for a logical datapath. A logical datapath does not have a physical location, so its physical binding information is limited: just tunnel_key. The rest of the data in this table does not affect packet forwarding.

6. Checking the logical flow of the virtual chassis is an important aspect of the virtualization.

Each row in Logical Flow table represents one logical flow. The **ovn-northd** populates this table with logical flows that implement the L2 and L3 topologies specified in the OVN_Northbound database. Each hypervisor, via ovn-controller, translates the logical flows into OpenFlow flows specific to its hypervisor and installs them into Open vSwitch.

Mechanism: **[--uuid] [--ovs[=remote]] [--stats] lflow-list [logical-datapath] [lflow...]**

Code: sudo ovn-sbctl <chassis-id> lflow-list

6.3. Monitoring Open Virtual Network

- **Monitoring Logical Flows**

OVN uses logical flows that are tables of flows with a priority, match, and actions. These logical flows are distributed to the ovn-controller running on each Compute node. We can use the **"ovn-sbctl lflow-list"** command on the Controller node to view the full set of logical flows, as shown in this example.

- **Monitoring packet transmission in OVN**

ovn-trace is an Open Virtual Network logical network tracing utility. This utility simulates packet forwarding within an OVN logical network. ovn−trace works by reading the Logical_Flow and other tables from the OVN southbound database. It simulates a packet's path through logical networks by repeatedly looking it up in the logical flow table, following the entire tree of possibilities.

ovn−trace simulates only the OVN logical network. It does not simulate the physical elements on which the logical network is layered. This means that, for example, it is unimportant how VMs are distributed among hypervisors, or whether their hypervisors are functioning and reachable, so ovn−trace will yield the same results regardless. There is one important exception: ovn−northd, the daemon that generates the logical flows that ovn−trace simulates, treats logical ports differently based on whether they are up or down.

Figure 27 shows the resultant output of an "ovn-trace" command.

The simplest way to use ovn−trace is to provide the microflow (and optional datapath) arguments on the command line. In this case, it simulates the behavior of a single packet and exits.

Syntax: ovn-trace [options] datapath microflow

The optional datapath argument specifies the name of a logical datapath. Acceptable names are the name from the northbound Logical_Switch or Logical_Router table, the UUID of a record from one of those tables, or the UUID of a record from the southbound Datapath_Binding table. (The datapath is optional because ovn−trace can figure it out from the inport that the microflow matches.)

The microflow argument describes the packet whose forwarding is to be simulated, in the syntax of an OVN logical expression.

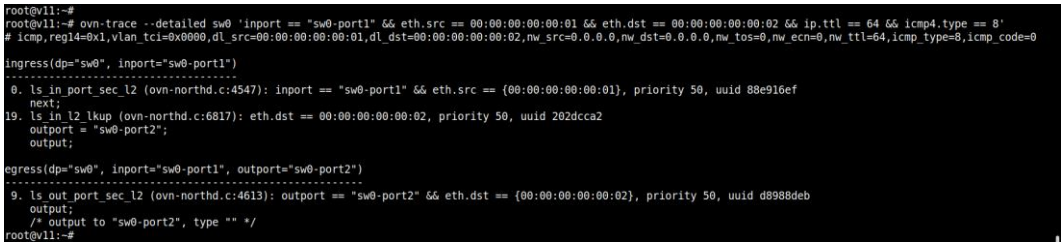

Figure 27. Running ovn-trace command in Open Virtual Network

- **Monitoring OpenFlows**

We use "**ovs-ofctl dump-flows"** command to monitor the OpenFlow flows on a logical switch in your network. Figure 28 shows an example of monitoring OpenFlows.

Syntax: **ovs-ofctl dump-flows <bridge name>**

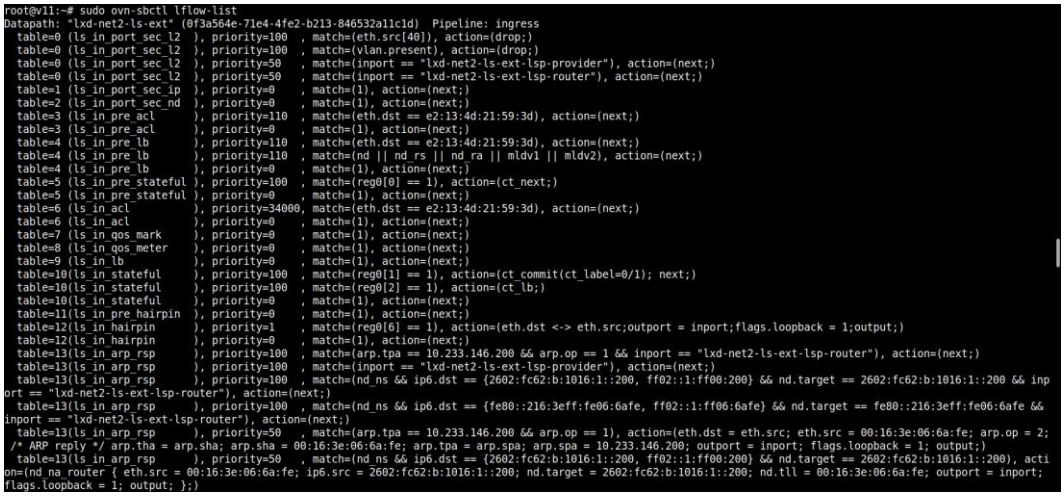

Figure 28. Running ovs-ofctl dump flows command on Open Virtual Network

Chapter 7

Security Issues of Network Virtualization

A security service refers to a subset of AAA (Authentication, Authorization, and Accounting) services. It is a processing or communication service given by a system to provide a specified level of protection to the resources, which may or may not reside with the system in question. An example scenario might help to establish the relationship between a Security Threat→ Security Policy→ Security Service→ Security Mechanism. A security threat is a possible means by which a security policy may be breached (e.g. man-in-the-middle attack undermines the confidentiality and integrity of a communication channel). So here maintenance of confidentiality of the ongoing communication is the security policy and man-in-the-middle attack is the corresponding threat. A security service is a measure which can be put in place to address the threat (e.g. Preserving the Confidentiality and Integrity of the communication channel to address man-in-the-middle attack). A security mechanism is a means to provide the security service (e.g. Use of VPN i.e. Virtual Private Network to guarantee the Confidentiality and Integrity of the communication channel in order to address man-in-the-middle attack). Therefore, it may be summarized that security services are used to implement various aspects of security policies utilizing different security mechanisms.

7.1. Security Services

ISO 7498-2 provides a general description of security services and related mechanisms specially meant for secure communications between systems. The five main categories of the available security services are as follows:

- Authentication
- Access control
- Data confidentiality
- Data integrity
- Non-repudiation

Authentication: Authentication is the process of determining whether someone or something is, in fact, who or what it declares itself to be. A user is usually identified with a Login ID and authentication is accomplished when the user provides a credential, for

example a password that matches with that user ID. Authentication is important because it enables organizations to keep their networks secure by permitting only authenticated users (or processes) to access its protected resources, which may include computer systems, networks, databases, websites and other network-based applications or services.

Access Control: Access control is a fundamental component of security that dictates, who is authorized to access and use company information and resources. Through authentication and authorization, access control policies make sure users are who they say they are and that they have appropriate access to company data. . Access control keeps confidential information, including customer data, personally identifiable information, and intellectual property, from falling into the wrong hands. Without a robust access control policy, organizations risk data leakage from both internal and external sources. The four access control models are:

- Discretionary access control (DAC)
- Mandatory access control (MAC)
- Role-based access control (RBAC)
- Attribute-based access control (ABAC)

Data Confidentiality: Data Confidentiality deals with protecting against the disclosure of information by ensuring that the data is limited to those authorized or by representing the data in such a way that its semantics remain accessible only to those who possess some critical information. Data confidentiality is a basic security service for data protection.

Data Integrity: The assurance that data received are exactly as sent by an authorized entity (i.e., contain no modification, insertion, deletion or replay).It refers to the reliability and trustworthiness of data throughout its lifecycle. It can describe the state of your data—e.g., valid or invalid—or the process of ensuring and preserving the validity and accuracy of data. Error checking and validation, for example, are common methods for ensuring data integrity as part of a process**.**

Non-Repudiation: Non-repudiation is the assurance that someone cannot deny the validity of something. Non-repudiation is a legal concept that is widely used in information security and refers to a service, which provides proof of the origin of data and the integrity of the data. In other words, non-repudiation makes it very difficult to successfully deny who/where a message came from as well as the authenticity and integrity of that message. Digital signatures (combined with other measures) can offer non-repudiation when it comes to online transactions.

7.2. Security Properties

Network Segregation Property: Network segregation refers to the separation of critical networks from the internet, as well as from other internal networks. It also enforces rules for communication between hosts and services. This strategy is capable of reducing the risk of ransomware attacks and can improve cybersecurity measures across an entire organization – regardless of their size. It also helps an IT personnel to perform better by enhancing auditing and alerting capabilities, both of which provide critical insight when it

comes to identifying a cyber-threat and deploying an appropriate response. Network segregation is important to limit network propagation or lateral movement after a first element of the system is compromised.

Mutual Authentication Property: Mutual authentication or two-way authentication refers to two parties authenticating each other at the same time in an authentication protocol. It is a default mode of authentication in some protocols (IKE, SSH) and optional in others (TLS). It is a desired characteristic in verification schemes that transmit sensitive data, in order to ensure that data security can be accomplished with two types of credentials: usernames: passwords and public key certificates. Mutual authentication is a crucial security step that can defend against many adversarial attacks, which otherwise can have large consequences if sensitive systems like IoT involving e-Healthcare servers are hacked. With mutual authentication, the server and the client authenticate each other. Mutual authentication is of two types: Certificate based and Username/Password based.

Data Protection Property: Data can exist in multiple states, and it can quickly change states based on an organization's needs. There are three states in which data can exist in an organization- Data at Rest, Data in Transit and Data in Use.

Some protection mechanism for Data In Transit are Network security tools like firewalls and authentication are simple but effective defenses against malicious attacks and attempted intrusions. Encrypting email ensures its contents are safe and that any attachments are encoded so they can't be read by prying eyes. Encryption can be applied to email delivery, directory sync and journaling, helping with both security and classification.

Some protection practices for Data In Use are Protections for data in use should be put in place before anyone can access the information. Once a sensitive document has been compromised, there is no way to control what a hacker does with the data they've obtained.

Some protection mechanisms for Data At Rest are Full disk encryption ensures malicious users cannot access the data on a lost drive without the necessary logins. Cloud access security brokers (CASBs) let companies apply DLP policies to information that they store and share in the cloud. This includes back-end systems and collaboration platforms like Slack or Microsoft 365. The mechanism of a CASB is similar to that of a DLP, with policies and functionality tailored to a cloud environment.

Source Identification Property: The source identification property is the property by virtue of which the sender of data or information in any form can be identified by the receiving end and also ensures that the sender cannot deny having sent the data at any point of time. This property is also ensured through the conjunction of or individual use of the mutual authentication and non-repudiation property.

Configuration Property: In the case of a software, configuration can also refer to the application's settings. These settings can either be set by default, or configured manually by the user. Information about configuration allows users to determine whether a particular application can be run or not. It could also aid in decisions on upgrading or purchasing a new system in order to execute certain applications. Configuration information can

motivate users in optimal usage of the system in order to prolong the performance and life of the system. Various configuration properties of a system are used to strengthen or weaken its overall security. These properties should be set in such a way so as to enhance and harden the security of the concerned system.

Resource Allocation Property: Resource allocation is the process of assigning and managing assets in a manner that supports an organization's strategic goals. Resource allocation includes managing tangible assets such as hardware to make the best use of softer assets such as human capital. Resource allocation is an important feature in a heterogeneous network meant to ensure its high efficiency as well as its maintenance as a cost-benefit network. Proper resource allocation improves the performances of both the associated system and the network, and also helps in avoiding the different kinds of transient bottlenecks encountered in a system.

Redundancy Property: Redundancy means having extra or duplicate resources available to support the main system. Network redundancy is the process of adding additional instances of network devices and lines of communication to help ensure network availability and decrease the risk of failure along the critical data path. It is a secondary or reserve system that can step in if the primary system fails. The reserve resources are redundant most of the time as they are not being used if everything is working properly. It is important in critical situations like hardware or software failure, when the redundant systems can step in.

User Management Property: User management describes the ability for administrators to manage user access to various IT resources like systems, devices, applications, storage systems, networks, SaaS services, and more. User management, in its simplest form, is the method by which you create, remove and maintain your user store. Any solution designed to serve multiple users must have some type of a user management system, be it a proprietary tool built into the product or a tie into an existing system such as Active Directory/LDAP, or another identity provider. User management not only establishes a user’s authorization to access secure resources, it also serves as a repository of identities and, if done efficiently, can be the source of all identities for an organization.

Password Management Property: Using proper passwords is important to ensure the protection and access control of a business. Hence password management is also an important activity. Following rigorous practices for password setting and selection should be followed.

Network Admission Control Property: Network Admission Control or NAC restricts access to the network based on identity or security posture. When a network device is configured for NAC, it can force user or machine authentication prior to granting access to the network. Network admission control systems allow noncompliant devices to be denied access or given restricted access to computing resources, thus keeping insecure nodes from infecting the network. When reducing cyber threats, NAC consists of two key steps: authentication and authorization. Authentication is when the system verifies based on credentials, while authorization is when the system accepts or denies access based on the policies in place. A user needs to pass both steps in order to achieve the access requested to the network.

7.3. Mapping of Security Properties and Security Mechanisms

Table I presents the mapping of the different Security Properties against the Security Mechanisms.

Table I. Security Properties vs. Security Mechanisms

Sl. No.	Security Property Type	Security Properties	Security Mechanism
1	**Network Segregation Property**	Two different network segments should be allowed / prohibited to communicate with each other.	Network Access Controls, firewalls, and intrusion prevention systems are used to achieve Network Segregation Property
2	**Mutual Authentication Property**	Controller and switch should be able to authenticate each other.	User Id: Password, Certificates and Cryptographic Protocols are used to achieve Authentication
3	**Data Protection Property**	The communication between the switch and the controller should be encrypted.	Encrypted IPSec Tunnel /SSH Tunnel can be used to achieve Traffic Flow Confidentiality (south bound)
		The communication between the controller and the switch should be over TLS.	TLS handshakes, and other encrypted packet transmission mechanisms are used to achieve authentication, confidentiality, and data integrity over the communication
		The logs generated at the switch and the controller should be stored securely.	When using secure syslog, log messages are encrypted and sent over the network using SSL/TLS.
		The communications between the applications and the controllers must be encrypted.	Secure Sockets Layer (SSL) and other encrypted packet transmission mechanisms are generally used to achieve Traffic Flow Confidentiality (north bound)
		Isolation among different applications must be ensured.	Enforcement of proper Access Control Policies, running the applications in separate Virtual Machines or Containers can be used to provide isolation.
4	**Source Identification Property**	The switch should be configured to be administered from a designated controller IP address.	SSL and execution of certain security scripts are generally used to achieve this property.
5	**Configuration Property**	The switches and the controller should be configured as per the policy.	Secure Configuration Mechanisms should be added in

Sl. No.	Security Property Type	Security Properties	Security Mechanism
			all necessary entities (firewall, files, etc.)
		The configurations of the switches and controller should be hardened.	Secure Configuration Mechanisms should be added in all necessary entities (firewall, files, etc.)
6	**Resource Allocation Property**	The network bandwidth on particular links should be according to the policy.	Specifying the required Access Control Rules and designing of the various functionalities of the Management/Orchestrator Modules are generally responsible for proper resource allocation.
7	**Redundancy Property**	In case of failure, another controller should take over to ensure continuation of the service.	Maintaining multiple machine clusters of network controllers or multiple snapshots of the service might help to achieve high availability.
8	**User Management Policy**	The controller and the switch should have provision for user creation and deletion.	Separate user management module (Inbuilt/Designed) is generally used for Identity Management which in turn facilitates the user management policy
		Appropriate logging should be there for user creation and deletion.	Compilation of detailed log records should be done by the user management module in order to achieve Integrity Monitoring
		Authorization to different users should be as per the policy.	Access Control Rules as per necessary security protocols are specified to achieve the required level of authorization
9	**Password Management Property**	The password complexity should be as per the policy.	Separate user management module (Inbuilt/Designed) is generally used for Identity Management which in turn facilitates password management. Password Creation and Management best practices should be followed according to the policy document if any.
		The password should be changed periodically for all the users (admin users, normal users etc.).	Separate user management module (Inbuilt/Designed) is generally used for Identity Management which in turn facilitates password management. Password Creation and Management best practices should be followed according to the policy document if any

Sl. No.	Security Property Type	Security Properties	Security Mechanism
10	**Network Admission Control Property**	All connections from conventional networks should be access controlled. Only whitelisted networks must be allowed to communicate.	Network Access Controls, firewalls, and intrusion prevention systems are used to achieve Network Admission Control Property.
		All devices (switches, virtual switches, end systems etc.) should be access controlled. Only whitelisted devices must be allowed.	Network Access Controls, firewalls, and intrusion prevention systems are used to achieve Network Admission Control Property.
11	**Backup Property**	The configuration backup of the controller and the switches should be taken periodically.	Full image backup and Periodic incremental backups should be taken to facilitate Backup Property and thereby encourages Integrity Monitoring

7.4. Mapping of Security Properties and Security Mechanisms with respect to Open Virtual Network (OVN)

In order to judge the security aspect of a Network Virtualization platform we need to map the functionalities of the Network Virtualization platform against our desired security mechanisms and their corresponding properties.

Table II maps some of the vital security properties against some specific security mechanisms used by Network Virtualization (OVN).

Table II. Security Properties v/S Security Mechanisms (NV)

Sl. No.	Security Property Type	Security Mechanism
1	Network Segregation Property	Major Network Virtualization platforms allow network segregation by allowing the creation of LINP (Logically Isolated Network Partition) from virtualized architectures. OVN is implemented using a tunnel-based overlay network with protocols such as VXLAN (**V**irtual **E**xtensible **LAN**), Geneve (**GE**neric **NE**twork **V**irtualization **E**ncapsulation), etc used for network segregation. OVN can be deployed with specified ipv4 and ipv6 ranges and thus on the same virtualized physical network multiple logically isolated network partitions can be created.
2	Mutual Authentication Property	OVN makes use of Public Key Infrastructure (PKI) to authenticate and authorize control plane communication. OVN Chassis (A chassis is nothing but a node where the ovn-controller service is running) authenticates each other by using certificates. The authentication succeeds if the other end in the tunnel presents a certificate signed by a trusted CA (Certificate Authority) and the common name (CN) matches the expected chassis name. The SSL certificates used in

		role-based access controls (RBAC) can be used in IPsec (default encryption protocol in OVN) for mutual authentication. OVS-PKI can also be used to create different certificates. The certificate is required to be x.509 version 3, and with CN field and subjectAltName field being set to the chassis name.
3	Data Protection Property	Data protection in Network Virtualization can be achieved via Cryptography (The use of cryptographic tunneling protocols prevents malicious entities from manipulating messages going through the network. OVN uses IPSec and Geneve as the tunneling protocols). For OVN the tunnel traffic is encrypted with IPsec. The CMS (Cloud management System) sets the IPsec column in the northbound NB_Global table to enable or disable IPsec encryption. If IPsec is true, all OVN tunnels will be encrypted otherwise if false, no OVN tunnels will be encrypted. The OVN-controller in each chassis monitors the southbound database and sets the options of the OVS tunnel interface accordingly. OVS tunnel interface options are monitored by the ovs-monitor-ipsec daemon which configures IKE daemon to set up IPsec connections. A few other practices for OVN data protection are, time stamping, batch processing and check pointing.
4	Source Identification Property	The Virtual Network can be associated with a particular controller of choice by using the *set-controller* command.
5	Configuration Property	Virtual Networks can be configured as per user designated security protocols by modifying files like "ml2_conf.ini" and "networking_ovn_metadata_agent.ini" in the case of Open Virtual Network.
6	Resource Allocation Property	To ensure fair allocation of resources on the control channel, the network that connects the controller to the switches (be it out-of-band or in-band) has to support mechanisms such as rate-limiting to prevent one VN from using all bandwidth on the control channel. QoS reservations for the control traffic can solve the problem and provide fairness in use of the control channel resources. In addition to this the Pacemaker module exists, which acts as the cluster resource manager for the OVN DB server. Pacemaker manages the resource with the help of the resource agents. One among the resource agents is OCF. OCF is nothing but a shell script which accepts a set of actions and returns an appropriate status code.
7	Redundancy Property	Multiple machine clusters of OVN following the OVN Gateway High Availability Plan can be used to achieve high availability.
8	User Management Policy	User Management property relies on the operating system in the OVN level whereas there is a different

		user management policy on the CMS (Cloud Management System) end.
9	Password Management Property	Password Management Property relies on the operating system in the OVN level whereas there is a password management policy on the CMS (Cloud Management System) end. Also LXD Clustering mechanism allows to set passwords to each VIF or VM.
10	Network Admission Control Property	Where SSL provides authentication when connecting to an OVS database, role based access control (RBAC) provides authorization to operations performed by clients connecting to an OVS database. RBAC allows administrators to restrict the database operations a client may perform and thus enhance the security already provided by SSL. In theory, any OVS database could define RBAC roles and permissions, but at present only the OVN southbound database has the appropriate tables defined to facilitate RBAC
11	Backup Property	Full image backup and Periodic incremental backup are generally used to achieve proper backup of the system. OVN databases contain one active and multiple backup servers in order to guarantee uninterrupted service.

Chapter 8

Configuration Methodology and Evidence Identification in Network Virtualization

It is of utmost importance that we should be able to configure and adapt our network virtualization platform to manage the security issues in the modern day world. The ability to configure a system to manage different problems according to the situation is a great example of flexibility and adaptability. OVN has multiple configuration files where it can be configured to handle with the situation.

The term "system configuration" typically refers to a collection of "parameter:value" pairs that can be used to specify, alter, and fine-tune the necessary system functionality. In other words, a configuration file, often known as a config file, specifies the settings, options, and preferences that have been applied to a system. Typically, configuration files are specified using the .CFG, .CONF, and .CONFIG extensions. In addition to this, there are a few formally defined configuration file formats, including .XML, .YAML, .JSON, .INI, .TOML, and .HCL. All of these files have a pattern in common because they all list the same key:value> pairs. Different configuration file formats may have certain structural or logical variations, such as how segments in a long configuration file are represented (often done using []), how key:value pairs are represented (may use: or =), and whether or not there are any special characters, nested configuration, strongly typed/loosely typed, allow/disallow comments etc. E.g. the firewall configuration in an UBUNTU system is generally stored in /etc/default/ufw and /etc/ufw/sysctl.conf. Generally packet forwarding should be enabled and to configure the same the following <key:value> pairs need to be added or modified.

DEFAULT_FORWARD_POLICY="ACCEPT"

net/ipv4/ip_forward=1

8.1. Mapping of Security Properties and Configuration Methodologies - with respect to Open Virtual Network

This section presents a detailed study on the Configuration methodologies that should be implemented to achieve the security properties of NV. Table III presents the different configuration files related to the Network Virtualization. Table IV. depicts the mapping of Security Properties against the Security Mechanisms implemented to achieve the concerned property. Finally the right most column specifies the configuration methodology that is required to implement the corresponding security mechanism.

Table III. Configuration files for Network Virtualization

Configuration files for Network Virtualization:

1.	Configuring the CMS (cloud management system side) for NV (in our case that's Openstack)	neutron.conf
2.	Configuring the NV components (in our case that's Open Virtual Network)	ovn.ini ml2_conf.ini networking_ovn_metadata_agent.ini

Table IV. Security Properties v/s Configuration Methodology (NV)

Sl. No.	Security Property Type	Security Mechanism	Configuration Methodology
1	Network Segregation Property	Major Network Virtualization platforms allow network segregation by allowing the creation of LINP (Logically Isolated Network Partition) from virtualized architectures. OVN is implemented using a tunnel-based overlay network with protocols such as VXLAN (**V**irtual **E**xtensible **LAN**), Geneve (**GE**neric **NE**twork **V**irtualization **E**ncapsulation), etc used for network segregation. OVN mostly uses the Geneve protocol and VXLAN for integration with TOR (Top Of Rack) switches that support the hardware_vtep (virtual tunnel endpoint) OVSDB schema and are used as L2 gateways between logical and physical networks.	Network segregation can be achieved by configuring the ML2 plug-in. Edit the /etc/neutron/plugins/ml2/ml2_conf.ini file: Configure the OVN mechanism driver, network type drivers, self-service (tenant) network types, and enable the port security extension *[ml2]* *...* *mechanism_drivers = ovn* *type_drivers = local,flat,vlan,geneve* *tenant_network_types = geneve* *extension_drivers = port_security* *overlay_ip_version = 4*
2	Mutual Authentication Property	OVN makes use of Public Key Infrastructure (PKI) to authenticate and authorize control plane communication. OVN Chassis (A chassis is nothing but a node where the ovn-controller service is running) authenticates each other by using certificates. The authentication succeeds if the other end in the tunnel presents a certificate signed by a trusted CA (Certificate Authority) and the common name (CN) matches the expected chassis name. The SSL certificates used in role-based access controls (RBAC) can be used in IPsec (default encryption protocol in OVN) for mutual authentication. OVS-PKI can also be used to create different certificates. The certificate is required to be x.509 version 3, and with CN field and subjectAltName field being set to the chassis name.	We can implement mutual authentication by configuring the "ovn.ini" file by editing the following fields: **ovn_nb_private_key**, **ovn_nb_certificate**, **ovn_nb_ca_cert**, **ovn_sb_private_key**, **ovn_sb_certificate**, **ovn_sb_ca_cert**.
3	Data Protection Property	Data protection in Network Virtualization can be achieved via Cryptography (The use of cryptographic tunneling protocols prevents	In OVN the tunnel traffic is encrypted with IPsec. IPSec can be configured as follows:

		malicious entities from manipulating messages going through the network. OVN uses IPSec and Geneve as the tunneling protocols). For OVN the tunnel traffic is encrypted with IPsec. The CMS (Cloud management System) sets the ipsec column in the northbound NB_Global table to enable or disable IPsec encryption. If ipsec is true, all OVN tunnels will be encrypted otherwise if false, no OVN tunnels will be encrypted. The OVN-controller in each chassis monitors the southbound database and sets the options of the OVS tunnel interface accordingly. OVS tunnel interface options are monitored by the ovs-monitor-ipsec daemon which configures IKE daemon to set up IPsec connections. A few other practices for OVN data protection are, timestamping, batch processing and checkpointing.	*$ ovs-vsctl set Open_vSwitch . * *other_config:certificate=/path/to/chassis-cert.pem * *other_config:private_key=/path/to/chassis-privkey.pem* *other_config:ca_cert=/path/to/cacert.pem* To enable OVN IPsec, set ipsec column in NB_Global table of the northbound database to true: *$ ovn-nbctl set nb_global . ipsec=true*
4	Source Identification Property	The Virtual Network can be associated with a particular controller of choice by using the *set-controller* command.	The Virtual Network can be associated with a particular controller of choice using the following configuration *ovs-vsctl set-controller br1 tcp: 192.168.0.54 <Controller-IP>: 6635 <openflow port>*
5	Configuration Property	Virtual Networks can be configured as per user designated security protocols by modifying files like “ml2_conf.ini” and “networking_ovn_metadata_agent.ini” in the case of Open Virtual Network.	OVN can be configured as per the desired security protocols by editing the following files: “neutron.conf”, “ovn.ini”, “ml2_conf.ini” and “networking_ovn_metadata_agent.ini”
6	Resource Allocation Property	To ensure fair allocation of resources on the control channel, the network that connects the controller to the switches (be it out-of-band or in-band) has to support mechanisms such as rate-limiting to prevent one VN from using all bandwidth on the control channel. QoS reservations for the control traffic can solve the problem and provide fairness in use of the control channel resources. In addition to this the Pacemaker module exists, which acts as the cluster resource manager for the OVN DB server. Pacemaker manages the resource with the help of the resource agents. One among the resource agents is OCF. OCF is nothing but a shell script which accepts a set of actions and returns an appropriate status code.	In OVN, Pacemaker manages the resource with the help of the resource agents. The YAML file for this purpose is the *tripleo-heat-templates/environments/services-docker/neutron-ovn-dvr-ha.yaml f*ile. When enabled, the OVN database servers are managed by Pacemaker, and a pacemaker OCF resource named ovn:ovndb-servers is created. Fig. 16 presents a sample snapshot of the ovndb-servers.ocf

7	Redundancy Property	Multiple machine clusters of OVN following the OVN Gateway High Availability Plan can be used to achieve high availability.	Pacemaker is a cluster resource manager which can manage a defined set of resources across a set of clustered nodes. Pacemaker manages the resource with the help of the resource agents. One among the resource agents is OCF (Open Cluster Framework). Whenever the active server dies, pacemaker is responsible to promote one of the backup servers to be active. Both ovn-controller and ovn-northd needs the ip-address at which the active server is listening. With pacemaker changing the node at which the active server is run, it is noet efficient to instruct all the ovn-controllers and the ovn-northd to listen to the latest active server's ip-address. This issue can be solved in two ways: 1. By using a native ocf resource agent ocf:heartbeat:IPaddr2 *$ pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=x.x.x.x op monitor interval=30s* *$ pcs constraint order promote ovndb_servers-master then VirtualIP* *$ pcs constraint colocation add VirtualIP with master ovndb_servers-master score=INFINITY* 2. Using load balancer vip ip as a master_ip. *$ pcs resource create ovndb_servers ocf:ovn:ovndb-servers* *master_ip="<load_balance_vip_ip>" * *listen_on_master_ip_only="no" * *ovn_ctl=<path of the ovn-ctl script> * *op monitor interval="10s" * *op monitor role=Master interval="15s"* *$ pcs resource master ovndb_servers-master ovndb_servers meta notify="true"*
8	User Management Policy	User Management property relies on the operating system in the OVN level whereas there is a different user management policy on the CMS (Cloud Management System) end.	User Management property relies on the operating system in the OVN level whereas there is a different user management policy on the CMS (Cloud Management System) end.
9	Password Management Property	Password Management Property relies on the operating system in the OVN level whereas there is a password management policy on the CMS (Cloud Management System) end.	Password Management Property relies on the operating system in the OVN level whereas there is a password management policy on the CMS (Cloud Management System) end.
10	Network Admission Control Property	Where SSL provides authentication when connecting to an OVS database, role based access control (RBAC) provides authorization to operations performed by clients connecting to an OVS database. RBAC allows administrators to restrict the database operations a client may perform and thus	For OVN RBAC is intended to supplement SSL. In order to enable RBAC, the connection to the database must use SSL. Some permissions in RBAC are granted based on the certificate common name (CN) of the connecting client.

		enhance the security already provided by SSL. In theory, any OVS database could define RBAC roles and permissions, but at present only the OVN southbound database has the appropriate tables defined to facilitate RBAC	RBAC is controlled with two database tables, *RBAC_Role* and *RBAC_Permission.* In order to enable RBAC, specify the role name as an argument to the set-connection command for the database. As an example, to enable the "ovn-controller" role on the OVN southbound database, use the following command: *$ ovn-sbctl set-connection role=ovn-controller ssl:192.168.0.1:6642*
11	Backup Property	Full image backup and Periodic incremental backup are generally used to achieve proper backup of the system. OVN databases contain one active and multiple backup servers in order to guarantee uninterrupted service.	With the help of the OCF resource agent ovn/utilities/vndb-servers.ocf, one can define a resource for the pacemaker such that pacemaker will always maintain one running active server at any time. After creating a pacemaker cluster, we can use the following configuration to create one active and multiple backup servers for OVN databases. *$ pcs resource create ovndb_servers ocf:ovn:ovndb-servers * *master_ip=x.x.x.x * *ovn_ctl=<path of the ovn-ctl script> * *op monitor interval="10s" * *op monitor role=Master interval="15s"* *$ pcs resource master ovndb_servers-master ovndb_servers * *meta notify="true"* The master_ip and ovn_ctl are the parameters that will be used by the OCF script. master_ip is the IP address on which the active database server is expected to be listening

```
[DEFAULT]


[ml2]

#
# From neutron.ml2
#

# List of network type driver entrypoints to be loaded from the
# neutron.ml2.type_drivers namespace. (list value)
#type_drivers = local,flat,vlan,gre,vxlan,geneve

# Ordered list of network_types to allocate as tenant networks. The default
# value 'local' is useful for single-box testing but provides no connectivity
# between hosts. (list value)
#tenant_network_types = local

# An ordered list of networking mechanism driver entrypoints to be loaded from
# the neutron.ml2.mechanism_drivers namespace. (list value)
#mechanism_drivers =

# An ordered list of extension driver entrypoints to be loaded from the
# neutron.ml2.extension_drivers namespace. For example: extension_drivers =
# port_security,qos (list value)
#extension_drivers =

# Maximum size of an IP packet (MTU) that can traverse the underlying physical
# network infrastructure without fragmentation when using an overlay/tunnel
# protocol. This option allows specifying a physical network MTU value that
# differs from the default global_physnet_mtu value. (integer value)
#path_mtu = 0
```

Figure 29. Initial section of the ml2_conf.ini file

```
[DEFAULT]

#
# From neutron
#

# Where to store Neutron state files. This directory must be writable by the
# agent. (string value)
#state_path = /var/lib/neutron

# The host IP to bind to (string value)
#bind_host = 0.0.0.0

# The port to bind to (port value)
# Minimum value: 0
# Maximum value: 65535
#bind_port = 9696

# The path for API extensions. Note that this can be a colon-separated list of
# paths. For example: api_extensions_path =
# extensions:/path/to/more/exts:/even/more/exts. The __path__ of
# neutron.extensions is appended to this, so if your extensions are in there
# you don't need to specify them here. (string value)
#api_extensions_path =

# The type of authentication to use (string value)
#auth_strategy = keystone

# The core plugin Neutron will use (string value)
#core_plugin = <None>

# The service plugins Neutron will use (list value)
#service_plugins =
```

Figure 30. Initial Section of neutron.conf

```
[DEFAULT]

#
# From networking_ovn.metadata.agent
#

# Location for Metadata Proxy UNIX domain socket. (string value)
#metadata_proxy_socket = $state_path/metadata_proxy

# User (uid or name) running metadata proxy after its initialization (if empty:
# agent effective user). (string value)
#metadata_proxy_user =

# Group (gid or name) running metadata proxy after its initialization (if
# empty: agent effective group). (string value)
#metadata_proxy_group =

# Certificate Authority public key (CA cert) file for ssl (string value)
#auth_ca_cert = <None>

# IP address or DNS name of Nova metadata server. (host address value)
# Deprecated group/name - [DEFAULT]/nova_metadata_ip
#nova_metadata_host = 127.0.0.1

# TCP Port used by Nova metadata server. (port value)
# Minimum value: 0
# Maximum value: 65535
#nova_metadata_port = 8775
```

Figure 31. Initial section of the networking_ovn_metadata_agent.ini

```
#!/bin/bash

: ${OCF_FUNCTIONS_DIR=${OCF_ROOT}/lib/heartbeat}
. ${OCF_FUNCTIONS_DIR}/ocf-shellfuncs
: ${OVN_CTL_DEFAULT="/usr/share/ovn/scripts/ovn-ctl"}
: ${NB_MASTER_PORT_DEFAULT="6641"}
: ${NB_MASTER_PROTO_DEFAULT="tcp"}
: ${SB_MASTER_PORT_DEFAULT="6642"}
: ${SB_MASTER_PROTO_DEFAULT="tcp"}
: ${MANAGE_NORTHD_DEFAULT="no"}
: ${INACTIVE_PROBE_DEFAULT="5000"}
: ${INACTIVE_PROBE_TO_MASTER_DEFAULT="60000"}
: ${LISTEN_ON_MASTER_IP_ONLY_DEFAULT="yes"}
: ${NB_SSL_KEY_DEFAULT="/etc/openvswitch/ovnnb-privkey.pem"}
: ${NB_SSL_CERT_DEFAULT="/etc/openvswitch/ovnnb-cert.pem"}
: ${NB_SSL_CACERT_DEFAULT="/etc/openvswitch/cacert.pem"}
: ${SB_SSL_KEY_DEFAULT="/etc/openvswitch/ovnsb-privkey.pem"}
: ${SB_SSL_CERT_DEFAULT="/etc/openvswitch/ovnsb-cert.pem"}
: ${SB_SSL_CACERT_DEFAULT="/etc/openvswitch/cacert.pem"}

CRM_MASTER="${HA_SBIN_DIR}/crm_master -l reboot"
CRM_ATTR_REPL_INFO="${HA_SBIN_DIR}/crm_attribute --type crm_config --name OVN_REPL_INFO -s ovn_ovsdb_master_server"
OVN_CTL=${OCF_RESKEY_ovn_ctl:-${OVN_CTL_DEFAULT}}
MASTER_IP=${OCF_RESKEY_master_ip}
NB_MASTER_PORT=${OCF_RESKEY_nb_master_port:-${NB_MASTER_PORT_DEFAULT}}
NB_MASTER_PROTO=${OCF_RESKEY_nb_master_protocol:-${NB_MASTER_PROTO_DEFAULT}}
SB_MASTER_PORT=${OCF_RESKEY_sb_master_port:-${SB_MASTER_PORT_DEFAULT}}
SB_MASTER_PROTO=${OCF_RESKEY_sb_master_protocol:-${SB_MASTER_PROTO_DEFAULT}}
MANAGE_NORTHD=${OCF_RESKEY_manage_northd:-${MANAGE_NORTHD_DEFAULT}}
INACTIVE_PROBE=${OCF_RESKEY_inactive_probe_interval:-${INACTIVE_PROBE_DEFAULT}}
INACTIVE_PROBE_TO_MASTER=${OCF_RESKEY_inactive_probe_interval_to_master:-${INACTIVE_PROBE_TO_MASTER_DEFAULT}}
NB_PRIVKEY=${OCF_RESKEY_ovn_nb_db_privkey:-${NB_SSL_KEY_DEFAULT}}
NB_CERT=${OCF_RESKEY_ovn_nb_db_cert:-${NB_SSL_CERT_DEFAULT}}
NB_CACERT=${OCF_RESKEY_ovn_nb_db_cacert:-${NB_SSL_CACERT_DEFAULT}}
SB_PRIVKEY=${OCF_RESKEY_ovn_sb_db_privkey:-${SB_SSL_KEY_DEFAULT}}
SB_CERT=${OCF_RESKEY_ovn_sb_db_cert:-${SB_SSL_CERT_DEFAULT}}
SB_CACERT=${OCF_RESKEY_ovn_sb_db_cacert:-${SB_SSL_CACERT_DEFAULT}}
```

Figure 32. ovndb-servers.ocf file

8.2. Mapping of Security Properties and Evidence Collection - with respect to Open Virtual Network

The importance of evidence collection is quite important as it provides validation for our network virtualization configuration when tested in different security scenarios.

For programmers and managers of computer systems, log files are a standard tool. They document the system's "what happened when by whom." This data can be used to track down problems and aid in their diagnosis. It can spot computer abuse and other security issues. It is applicable to auditing. It can be applied to accounting. Only when data is saved in an analytically-friendly format is it accessible for further analysis. For analysis, this data can be organised in a variety of ways. For instance, forcing the data into a query-able format by storing it in a relational database. As logging wouldn't be possible without the database, it would also be more challenging to recover if the machine crashed. A plain text format minimizes dependencies on other system processes, and assists logging at all phases of computer operation, including start up and shutdown, where such processes might be unavailable. As evidence the contents of a log file can be entered by a user/program at any point of time for security analysis.

Logging can produce technical information usable for the maintenance of applications or websites. It can be utilized to:

- Define whether a reported bug is actually a bug
- Help analyze, reproduce and solve bugs
- Help test new features in a development stage

Log management is an approach of dealing with large volumes of computer-generated log messages. It generally covers:

- Log collection
- Centralized log aggregation
- Long-term log storage and retention
- Log rotation
- Log analysis (in real-time and in bulk after storage)
- Log search and reporting.

The primary drivers for log management implementations are concerns about security, system and network operations (such as system or network administration) and regulatory compliance. Logs are generated by nearly every computing device, and can often be directed to different locations both on a local file system or remote system. Effectively analyzing large volumes of diverse logs can pose many challenges — such as:

Volume: Log data can reach hundreds of gigabytes of data per day for a large organization. Simply collecting, centralizing and storing data at this volume can be challenging.

Normalization: Logs are produced in multiple formats. The process of normalization is designed to provide a common output for analysis from diverse sources.

Velocity: The speed at which logs are produced from devices can make collection and aggregation difficult.

Veracity: Log events may not be accurate. This is especially problematic from systems that perform detection, such as intrusion detection systems.

This section presents a detailed study on the evidence collection methodologies that should be implemented in order to monitor the security properties of NV. Table V and VI depicts the mapping of the log files of the network virtualization stages to their respective networking agent. Table VII depicts the mapping of Security Properties against the Logging Mechanisms implemented to monitor the concerned property. Finally the right most column specifies the logging mechanism that is required to implement the corresponding security mechanism.

Table V. Corresponding Log files for the neutron service in Open Stack

Log file	**Service/Interface**
dhcp-agent.log	neutron-dhcp-agent
l3-agent.log	neutron-l3-agent
lbaas-agent.log	neutron-lbaas-agent
linuxbridge-agent.log	neutron-linuxbridge-agent
metadata-agent.log	neutron-metadata-agent
metering-agent.log	neutron-metering-agent
openvswitch-agent.log	neutron-openvswitch-agent
server.log	neutron-server

Table VI. Corresponding Log files for Open Virtual Network

1.	OVN Central	ovn-northd.log , ovsdb-server-nb.log , ovsdb-server-sb.log , ovsdb-server-ic-nb.log , ovsdb-server-ic-sb.log
2.	OVN Controller	ovn-controller.log

Table VII. Security Properties v/s Logging Mechanism (NV)

Sl. No.	**Security Property Type**	**Security Mechanism**	**Logging Methodology**
1	Network Segregation Property	Major Network Virtualization platforms allow network segregation by allowing the creation of LINP (Logically Isolated Network Partition) from virtualized architectures. OVN is implemented using a tunnel-based overlay network with protocols such as VXLAN (**V**irtual **E**xtensible **LAN**), Geneve (**GE**neric **NE**twork **V**irtualization **E**ncapsulation), etc used for network segregation. OVN mostly uses the Geneve protocol and VXLAN for integration with TOR (Top Of Rack) switches that support the hardware_vtep (virtual tunnel endpoint)	ML2/OVN supports network logging, based on security groups. Openstack Security Groups (SG) and their rules (SGR) map 1:1 into OVN's Port Groups (PG) and Access Control Lists (ACL). In Figure 5. The first command creates a networking-log for a given SG. The second shows an SGR from that SG. The third shell command is where we can see how the ACL with the meter information gets populated. These are the attributes pertinent to network logging. If the SGR is poked with packets that match its criteria, the ovn-controller local to where the ACLs

		OVSDB schema and are used as L2 gateways between logical and physical networks.	is enforced will log something that has been presented in Figure 33.
2	Mutual Authentication Property	OVN makes use of Public Key Infrastructure (PKI) to authenticate and authorize control plane communication. OVN Chassis (A chassis is nothing but a node where the ovn-controller service is running) authenticates each other by using certificates. The authentication succeeds if the other end in the tunnel presents a certificate signed by a trusted CA (Certificate Authority) and the common name (CN) matches the expected chassis name. The SSL certificates used in role-based access controls (RBAC) can be used in IPsec (default encryption protocol in OVN) for mutual authentication. OVS-PKI can also be used to create different certificates. The certificate is required to be x.509 version 3, and with CN field and subjectAltName field being set to the chassis name.	The logs of the ovs-monitor-ipsec daemon and the IKE daemon can be checked to locate issues related to IPsec (default encryption protocol in OVN) for mutual authentication. ovs-monitor-ipsec outputs log messages to /var/log/openvswitch/ovs-monitor-ipsec.log. A sample ovs-monitor-ipsec.log has been presented in Figure 35.
3	Data Protection Property	Data protection in Network Virtualization can be achieved via Cryptography (The use of cryptographic tunneling protocols prevents malicious entities from manipulating messages going through the network. OVN uses IPSec and Geneve as the tunneling protocols). For OVN the tunnel traffic is encrypted with IPsec. The CMS (Cloud management System) sets the ipsec column in the northbound NB_Global table to enable or disable IPsec encryption. If ipsec is true, all OVN tunnels will be encrypted otherwise if false, no OVN tunnels will be encrypted. The OVN-controller in each chassis monitors the southbound database and sets the options of the OVS tunnel interface accordingly. OVS tunnel interface options are monitored by the ovs-monitor-ipsec daemon which configures IKE daemon to set up IPsec connections. A few other practices for OVN data protection are, timestamping, batch processing and checkpointing.	In OVN the tunnel traffic is encrypted with IPsec. The logs of the ovs-monitor-ipsec daemon and the IKE daemon can be checked to locate issues related to IPsec (default encryption protocol in OVN) for mutual authentication. ovs-monitor-ipsec outputs log messages to /var/log/openvswitch/ovs-monitor-ipsec.log. A sample ovs-monitor-ipsec.log has been presented in Figure 35.
4	Source Identification Property	The Virtual Network can be associated with a particular controller of choice by using the *set-controller* command.	ovn-northd.log , ovsdb-server-nb.log and ovsdb-server-sb.log reflects any information related to the different connections between the networking devices and the OVN (Fig. 36 and Fig. 37)

5	Configuration Property	Virtual Networks can be configured as per user designated security protocols by modifying files like “ml2_conf.ini” and “networking_ovn_metadata_agent.ini” in the case of Open Virtual Network.	OVN can be configured as per the desired security protocols by editing the following files: “neutron.conf”, “ovn.ini”, “ml2_conf.ini” and “Networking_ovn_metadata_agent.ini”. The log files for the above config files are already depicted in Table V and VI.
6	Resource Allocation Property	To ensure fair allocation of resources on the control channel, the network that connects the controller to the switches (be it out-of-band or in-band) has to support mechanisms such as rate-limiting to prevent one VN from using all bandwidth on the control channel. QoS reservations for the control traffic can solve the problem and provide fairness in use of the control channel resources. In addition to this the Pacemaker module exists, which acts as the cluster resource manager for the OVN DB server. Pacemaker manages the resource with the help of the resource agents. One among the resource agents is OCF. OCF is nothing but a shell script which accepts a set of actions and returns an appropriate status code.	In OVN, Pacemaker manages the resource with the help of the resource agents. The corresponding log for the same is stored in ovn-northd.log and ovn-controller.log (Fig. 35 and Fig.36)
7	Redundancy Property	Multiple machine clusters of OVN following the OVN Gateway High Availability Plan can be used to achieve high availability.	In OVN, Pacemaker manages the resource with the help of the resource agents. The corresponding log for the same is stored in ovn-northd.log and ovn-controller.log (Fig. 34 and Fig. 36)
8	User Management Policy	User Management property relies on the operating system in the OVN level whereas there is a different user management policy on the CMS (Cloud Management System) end.	User Management property relies on the operating system in the OVN level whereas there is a different user management policy on the CMS (Cloud Management System) end. At OS level, the related logs are found in auth.log. At the CMS end the keystone log files of OpenStack are responsible for maintaining the logs for the same.
9	Password Management Property	Password Management Property relies on the operating system in the OVN level whereas there is a password management policy on the CMS (Cloud Management System) end.	Password Management Property relies on the operating system in the OVN level whereas there is a password management policy on the CMS (Cloud Management System) end. g. At the CMS end the keystone log files of OpenStack are responsible for maintaining the logs for the same.
10	Network Admission Control Property	Where SSL provides authentication when connecting to an OVS database, role based access control (RBAC) provides authorization to operations performed by clients connecting to an OVS database. RBAC allows administrators to restrict the database operations a client may perform and	For OVN, RBAC is intended to supplement SSL. In order to enable RBAC, the connection to the database must use SSL. The logs for the same are reflected in ovn-northd.log, ovsdb-server-nb.log and ovsdb-server-sb.log. (Fig. 36 and Fig. 37)

		thus enhance the security already provided by SSL. In theory, any OVS database could define RBAC roles and permissions, but at present only the OVN southbound database has the appropriate tables defined to facilitate RBAC	
11	Backup Property	Full image backup and Periodic incremental backup are generally used to achieve proper backup of the system. OVN databases contain one active and multiple backup servers in order to guarantee uninterrupted service.	With the help of the OCF resource agent ovn/utilities/vndb-servers.ocf, one can define a resource for the pacemaker such that pacemaker will always maintain one running active server at any time. The corresponding log for the same is stored in ovn-northd.log and ovn-controller.log

```
$ openstack network log create --resource-type security_group \
  --resource ${SG} --event ACCEPT logme -f value -c ID
2e456c7f-154e-40a8-bb10-f88ba51b90b5

$ openstack security group show ${SG} -f json -c rules | jq '.rules | .[2]' | grep -v 'null'
{
 "id": "de4ea1e4-c946-40ed-b5b6-53c59418dc0b",
 "tenant_id": "2600067ea3a446dba332d20a30ed44fa",
 "security_group_id": "c604e984-0789-4c9a-a297-3e7f62fa73fd",
 "ethertype": "IPv4",
 "direction": "egress",
 "standard_attr_id": 48,
 "tags": [],
 "created_at": "2021-02-06T22:17:44Z",
 "updated_at": "2021-02-06T22:17:44Z",
 "revision_number": 0,
 "project_id": "2600067ea3a446dba332d20a30ed44fa"
}

$ ovn-nbctl find acl \
  "external_ids:\"neutron:security_group_rule_id\""="de4ea1e4-c946-40ed-b5b6-53c59418dc0b"
_uuid               : 791679e9-237d-4732-a31e-aa634496e02b
action              : allow-related
direction           : from-lport
external_ids        : {"neutron:security_group_rule_id"="de4ea1e4-c946-40ed-b5b6-53c59418dc0b"}
log                 : true
match               : "inport == @pg_c604e984_0789_4c9a_a297_3e7f62fa73fd && ip4"
meter               : acl_log_meter
name                : neutron-2e456c7f-154e-40a8-bb10-f88ba51b90b5
priority            : 1002
severity            : info
```

Figure 33. Networking-log for a given SG (Security Group)

```
2021-02-16T11:59:00.640Z|00045|acl_log(ovn_pinctrl0)|INFO|
name="neutron-2e456c7f-154e-40a8-bb10-f88ba51b90b5",
verdict=allow, severity=info: icmp,vlan_tci=0x0000,dl_src=fa:16:3e:24:dc:88,
dl_dst=fa:16:3e:15:6d:e0,
nw_src=10.0.0.12,nw_dst=10.0.0.11,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,
icmp_code=0
```

Figure 34. Log generated by ovn-controller

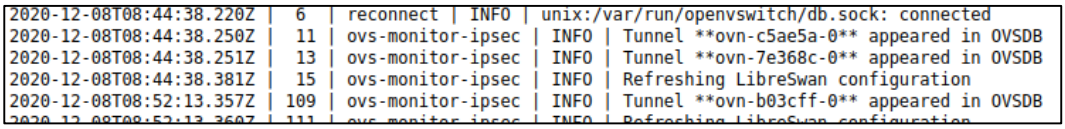

Figure 35. ovs-monitor-ipsec.log file

```
cdcju@cdcju-HP-Pro-3330-MT:~$ sudo cat /var/log/ovn/ovn-northd.log
2022-02-17T11:00:48.243Z|00001|vlog|INFO|opened log file /var/log/ovn/ovn-northd.log
2022-02-17T11:00:48.256Z|00002|reconnect|INFO|unix:/var/run/ovn/ovnnb_db.sock: connecting...
2022-02-17T11:00:48.256Z|00003|reconnect|INFO|unix:/var/run/ovn/ovnsb_db.sock: connecting...
2022-02-17T11:00:48.256Z|00004|reconnect|INFO|unix:/var/run/ovn/ovnnb_db.sock: connected
2022-02-17T11:00:48.256Z|00005|reconnect|INFO|unix:/var/run/ovn/ovnsb_db.sock: connected
2022-02-17T11:00:48.256Z|00006|ovn_northd|INFO|ovn-northd lock acquired. This ovn-northd instance is now active.
```

Figure 36. ovn-northd.log

```
cdcju@cdcju-HP-Pro-3330-MT:~$ sudo cat /var/log/ovn/ovsdb-server-nb.log
2022-02-17T11:00:48.000Z|00001|vlog|INFO|opened log file /var/log/ovn/ovsdb-server-nb.log
2022-02-17T11:00:48.005Z|00002|ovsdb_server|INFO|ovsdb-server (Open vSwitch) 2.13.3
2022-02-17T11:00:58.014Z|00003|memory|INFO|4696 kB peak resident set size after 10.0 seconds
2022-02-17T11:00:58.014Z|00004|memory|INFO|cells:156 monitors:2 sessions:1
cdcju@cdcju-HP-Pro-3330-MT:~$ sudo cat /var/log/ovn/ovsdb-server-sb.log
2022-02-17T11:00:48.111Z|00001|vlog|INFO|opened log file /var/log/ovn/ovsdb-server-sb.log
2022-02-17T11:00:48.127Z|00002|ovsdb_server|INFO|ovsdb-server (Open vSwitch) 2.13.3
2022-02-17T11:00:58.138Z|00003|memory|INFO|5076 kB peak resident set size after 10.0 seconds
2022-02-17T11:00:58.138Z|00004|memory|INFO|cells:2101 monitors:2 sessions:1
```

Figure 37. ovsdb-server-nb.log and ovsdb-server-sb.log

Chapter 9

Conclusion and Future Work

9.1. Conclusion

In the above document we discussed Network Virtualization, its architecture, its working mechanism, and various implementation techniques of Network Virtualization. Following that we chose Open Virtual Network (OVN) as our network virtualization mechanism and went on a detailed analysis about the architecture, working mechanism, security mechanism, configuration methodologies and evidence identification and collection mechanisms related to it. We find that OVN provides answers a lot of questions that a vulnerability analysis asks and most importantly OVN thoroughly improves the execution time when compared to OVS and ML2 Plugin. Some of OVN's integrated security features like the ability to create Logically Isolated Network Partitions, encapsulation using Geneve Protocol and RBAC policies thoroughly solve a lot of security questions in the threat landscape. Although OVN is a relatively nascent technology it has shown great promise that on further development it can and will provide complete network virtualization solution with integrated security mechanism and security properties.

With every year, the vulnerability landscape in the world of cyber security changing it is quite important for us to safeguard our systems against those unknown threats. Complete integration of a Cloud Management system and the network virtualization platform like OVN is necessary. Further studies and analysis should be made to integrate the following components e.g. Database clustering, ACL Logging, Securely handle a compromised hypervisor, scalability issues, Service function chaining, Encrypted tunnels, Native OpenStack LBaaS (Load Balancing as a Service) support and OpenStack support of multiple SNAT gateways on a network.

References

[1] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. Commun. ACM, 17(7):412–421, July 1974.

[2] O. Agesen, A. Garthwaite, J. Sheldon, and P. Subrahmanyam. The evolution of an x86 virtual machine monitor. ACM SIGOPS Operating Systems Review, 44(4):3–18, 2010.

[3] Albert Greenberg, James Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Malt, Parveen Patel, Sudeep Sengupta. ACM SIGCOMM Computer Communication Review Volume 39 Issue 4 October 2009

[4] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. "Can the production network be the testbed? In Operating Systems Design and Implementation", October 2010

[5] J.E. van der Merwe, S. Rooney, I. Leslie, S. Crosby. The Tempest—a practical framework for network programmability, IEEE Network Magazine 12 (3) (1998) 20–28

[6] Lee, Hyungro. "Virtualization Basics: Understanding Techniques and Fundamentals"

[7] A. Al-Shabibi et al., ''OpenVirteX: A network hypervisor,'' Open Networking Summit, Tech. Rep., 2014.

[8] Gyeongsik Yang, Member, Bong-yeol Yu, Heesang Jin, and Chuck Yoo. "Libera for Programmable Network Virtualization"

[9] Ivanov, Konstantin. "Containerization with LXC".

[10] "Open Virtual Network" [ONLINE] Available: https://www.ovn.org/en/

[11] "OVN Architecture" [ONLINE]

Available: https://www.ovn.org/support/dist-docs/ovn-architecture.7.html

[12] "OVN Controller". [ONLINE]

Available: https://www.ovn.org/support/dist-docs/ovn-controller.8.pdf

[13] "OVN Northbound Database". [ONLINE]

Available: https://www.ovn.org/support/dist-docs/ovn-nb.5.pdf

[14] "OVN Southbound Database" [ONLINE]

Available: https://www.ovn.org/support/dist-docs/ovn-sb.5.pdf

[15] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," Comput. Netw., vol. 54, pp. 862–876, April 2010.

[16] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," Apr. 2009, pp. 783–791.

[17] G. Schaffrath et al., "Network virtualization architecture: proposal an initial prototype," in VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures. New York, USA: ACM, 2009, pp. 63–72.

[18] Wang, Juan, Yu, Yang, Li, Yi, Fan, Chengyang, Hao, Shirong. “Design and Implementation of Virtual Security Function Based on Multiple Enclaves”

[19] Chen, XiaoJun, Zhang, Jing, Li, Junhuai, Li, Xiang. “Resource virtualization methodology for on-demand allocation in cloud computing systems” November 2011.

[20] Fernandes N, Moreira MD, Moraes I, Ferraz L, Couto R, Carvalho HT, Campista M, Costa LK, Duarte OB (2011) Virtual networks: isolation, performance, and trends. Ann Telecommun 66(5–6):339–355

[20] “Software-Defined Networks” [ONLINE]

Available: https://sdn.systemsapproach.org/netvirt.html#

[21] Fischer, Andreas; Botero, Juan Felipe; Beck, Michael Till; de Meer, Hermann; Hesselbach, Xavier (2013). "Virtual Network Embedding: A Survey". IEEE Communications Surveys & Tutorials. 15 (4): 1–19. doi:10.1109/SURV.2013.013013.00155. ISSN 1553-877X.

[22] D. Drutskoy, E. Keller and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," in IEEE Internet Computing, vol. 17, no. 2, pp. 20-27, March-April 2013, doi: 10.1109/MIC.2012.144.

[23] A. Khan, A. Zugenmaier, D. Jurca and W. Kellerer, "Network virtualization: a hypervisor for the Internet?," in IEEE Communications Magazine, vol. 50, no. 1, pp. 136-143, January 2012, doi: 10.1109/MCOM.2012.6122544.

[24] Alshaer, Hamada (2015), An overview of network virtualization and cloud network as a service, Int. J. Network Mgmt, 25, 1– 30, doi: 10.1002/nem.1882

[25] A. Blenk, A. Basta, M. Reisslein and W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 655-685, Firstquarter 2016, doi: 10.1109/COMST.2015.2489183.

[26] Mendiola, Alaitz & Astorga, Jasone & Jacob, Eduardo & Higuero, Marivi. (2017). A Survey on the Contributions of Software-Defined Networking to Traffic Engineering. IEEE Communications Surveys & Tutorials. 19. 918-953. 10.1109/COMST.2016.2633579.