Multi-tenant Isolation in Software Defined Networks

Sarah Irum(✉), Patrick Luedke, Klaus Warnke, and Gerrit Schulte

Acticom Gmbh, Am Borsigturm 48, 13507 Berlin, Germany
{sarah.irum,patrick.luedke,klaus.warnke,gerrit.schulte}@acticom.de
http://www.acticom.de

Abstract. Software Defined Networking (SDN) provides a flexible and programmable infrastructure for future networks. SDN supports multi-domain networks where customers, called tenants, can share network resources on the large data centers. In the multi-tenant environment, tenants can share the network elements while keeping them isolated from each other. In this paper, we describe an isolated multi-tenant solution where the tenants can have control over their assigned network resources. The described approach provides isolation through VxLAN and configuration of flow tables in the OpenFlow switch. VxLAN tunnels are used to isolate packets transmitted by different tenants. Virtual Network Identifiers (VNIs) are assigned to the flow table for identification of the tenant.

Keywords: Software defined network · VxLAN
Network function virtualization · OpenFlow

1 Introduction

Software Defined Networking (SDN) has emerged as an architecture to develop and deploy fast-growing applications over the past few years. SDN provides a dynamic, manageable, cost-effective approach that simplifies today's network capabilities and management. It helps to resolve the issues with the conventional network infrastructure such as large-scale integration of end systems and virtual networks. SDN gained considerable recognition from the researchers because of its benefits for the future Internet architectures such as information-centric networking [1]. SDN separates the control plane and forwarding paths to reduce network energy, provide security mechanisms [2] and data center network management [3]. A centralized network controller is used in SDNs to manage the entire network resulting in faster service provisioning, automation, and efficiency gains. The OpenFlow protocol is used for the communication of SDN controllers and switches.

Traditional network infrastructures manage servers, storage, and networking manually by highly skilled system administrators. This hardware-centric management approach is replaced by the multi-tenant services which offer virtualization and abstraction technologies. SDN technology allows the customers to have

V. Sucasas et al. (Eds.): BROADNETS 2018, LNICST 263, pp. 367–376, 2019.
https://doi.org/10.1007/978-3-030-05195-2_36

a higher level of control over their virtualized network resources in the multi-tenant environment. However, the isolation between tenant domains becomes important in the design of multi-tenant architectures as some of the network resources are shared between the multiple tenants. Virtual Network Function (VNF) architectures together with SDN virtualize the traditional network functions and replace the dedicated network hardware with software applications, which helps in accurate monitoring and manipulation of network traffic [4].

In this paper, we propose a testbed to isolate multi-tenant traffic in SDNs. In the testbed, multiple tenants are connected to multiple OpenFlow switches through one centralized SDN controller. The traffic is isolated using VxLAN overlay networks and flow tables. Virtual Extensible Lan (VxLAN) is the encapsulation protocol for the overlay network on existing Layer 3 infrastructure. It provides scalability to the system, while providing isolation to the tenants [5]. Flow tables are used in the OpenFlow protocol and they match incoming packets to specific flows and specify the functions that are to be performed on the frames.

The rest of the paper is organized as follows. Section 2 will present a brief description of Software Defined Networking and NFV. Section 3 will address the details of the testbed. Finally, Sect. 4 will present our conclusion and future work.

2 Concepts

In this section, we briefly explain the basic concepts and terminologies that are used in this paper.

2.1 Software Defined Networks

The traditional network paradigm focuses on hardware-centric networking, where switches have their own data and control planes, and adding new protocols or updating the existing protocols is a challenge. However, in SDN, switches have become simpler by taking out the control plane from the forwarding devices and managing the switches using the centralized controller devices. SDN introduces the softwarized programmable network, where data and control planes are separated and forwarding decisions are made on centralized SDN controllers. In the data plane, packets are transported through switches towards the destination. Whereas, in the control plane, the SDN controller decides about the packet flow through the network in the data plane. Control plane functionalities include system configuration, management, and exchange of routing table information. The SDN controller sends packet rules to the SDN switches and configures them with the information about the traffic they are handling using the OpenFlow protocol.

2.2 OpenFlow

The OpenFlow protocol is the communication protocol between a SDN controller and one or multiple SDN switches. The controller manipulates the flow entries

in the switch such as adding, updating or deleting the flow entries of a flowtable. The OpenFlow switch supports flow-based forwarding using the flow tables. Flow tables contain information about packet flows and their end points. Each packet flow entry contains information of packets such as Ethernet addresses and IP addresses of source and destination, output actions that are to be performed on the received packet, and the total number of transmitted bytes etc. When the switch receives a packet, it matches the flow entry with the flow table. In case a table miss entry, the switch forwards the packet back to the controller, and the controller gives the direction to the switch to manage the packet in the future or it just drops the packet [6].

2.3 SDN Controllers

The SDN controller manages the flows, application and business logic to enable intelligent networking. Tasks of the controller are monitoring of SDN network traffic, providing network statistics, and adding new rules throughout the network. Most popular SDN controllers are Ryu [7], Open Day Light (ODL) [8], POX [9], NOX [10], Floodlight [11] controller. Some of the controllers are configured using Python scripts such as Ryu, POX, NOX, while others are developed in C++ or Java.

2.4 VxLAN

There have been several approaches for the development of OpenFlow standards. Virtual Extensible LAN (VxLAN) was developed to address the scalability problem in the large computing environments. It is a virtual encapsulation technology for operating an overlay network on existing Layer 3 infrastructure [12]. The devices' which support VxLAN' are called virtual tunnel endpoints (VTEPs). The devices that can act as a VTEP are host endpoints, switches, and routers. VTEPs encapsulate VxLAN traffic by adding the number of fields such as the outer media access control (MAC) destination address, the outer MAC source address, the outer IP destination address, the outer IP source address, the outer UDP header, and the VxLAN network identifier (VNI), which is used to uniquely identify the VxLAN traffic. It also de-encapsulates the traffic leaving the VxLAN tunnel.

2.5 Network Function Virtualization - NFV

Network Virtualization (NV) [13] allows executing multiple instances of a network on a shared physical infrastructure. SDN enables NV by distinguishing and forwarding flows to different networks. In Network Function Virtualization (NFV) [14], specific in-network functions are virtualized such as firewalls, load balances, and VPN gateways etc. NFV focuses on data plane programmability and can be used to extend SDN because SDN mainly focuses on the control plane programmability.

3 Implementation

In our testbed, we developed our SDN environment using computer nodes as the hosts using Ubuntu Linux as an operating system. These hosts use the KVM hypervisor to spawn virtual machines. We used OVS switches on each host, which supports the OpenFlow protocol for communication inside SDN. The Ryu controller has been used as SDN controller which provides the control plane capability. In order to achieve isolation in the setup, we modified the flow tables in the OVS switch, and added the entries manually for the traffic control between VxLAN tunnels.

3.1 Creating Virtual Machines Using Hypervisors

There are several existing solutions for the isolation of multi-tenant network in SDNs. In our testbed, multiple tenants are isolated using VxLANs overlays and designing policies for flow tables. Isolation is achieved by assigning the end hosts and tagging the initial flows using tunnel ids. An OpenFlow switch is responsible to enforce policies and it inserts the label into the packet headers. We implemented our solution on the Ryu Controller to evaluate the traffic and isolation of multi-tenants setups.

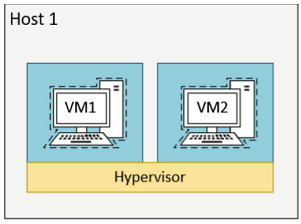

Fig. 1. Two Virtual vachines running on a hypervisor.

For the testbed we exclusively use hosts with KVM (Kernel Based Virtualization) as a hypervisor running on Ubuntu Server 16.04 LTS. For managing KVM, the virsh tool is used because it is scriptable, well documented and capable of communicating with KVM (see Fig. 1).

The host acts as a spawn point for the virtual machines that can be used at a later stage to serve as SDN. In Fig. 2, it is shown that the system consists of multiple hosts containing multiple guest virtual machines according to the requirements of the setup.

3.2 Connecting Virtual Machines with OVS Switch

In the setup, OVS is used as OpenFlow switch. OVS is the multilayer virtual switch that supports OpenFlow enabling network automation through programming. In Fig. 3, a simple topology is depicted in which two virtual machines are

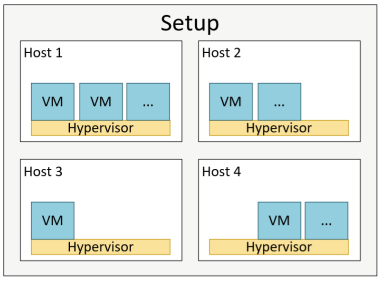

Fig. 2. Four hosts with hypervisors that act as spawnpoints for virtual machines.

connected to the OVS switch and the switch is controlled by the Ryu Controller. The OpenFlow table is configured in the OVS switch by the controller. In this topology, if VM1 transmits a packet to VM2, the packets are first received by the OVS switch connected to VM1, and depending on the flow table entries, the received packet would be forwarded to VM2. The controller is managing all the flow entries in the switch.

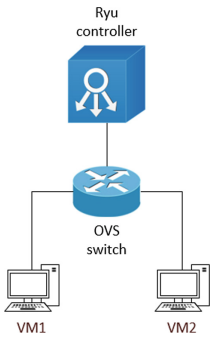

Fig. 3. VM1, VM2 and Ryu controller are connected through OVS switch.

3.3 Configuration of OVS Switch Using OpenFlow

An OVS switch has a flow table with flow entries that can be managed by the SDN controller. The flow entries can control the forwarding of the packets. The decision of forwarding or dropping the packets is made by the controller. Each flow entry specifies a matching pattern and defines the action that can be performed on the packet. When the switch receives the packet, it matches it with the forwarding matching pattern and if the flow entry is not found in the flow table, i.e. with a table miss entry, it forwards the packet to the controller. The controller then adds the flow entry to the switch or just drops it. A flow entry

contains information of the flow, such as Ethernet address and IP address of source and destination, output actions that are to be performed on the received packet, and the total number of transmitted bytes etc (see Fig. 4).

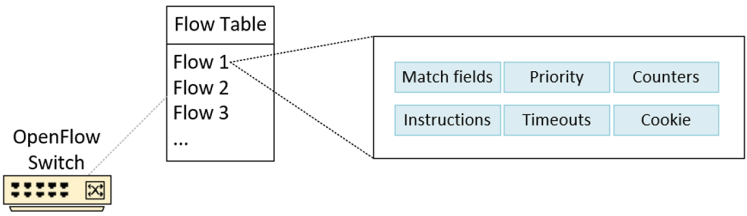

Fig. 4. Flow table and basic composition of a flow entry.

3.4 Connecting Switches Through VxLAN

A Virtual Extensible Local Area Network (VxLAN) supports a large number of tenants in comparison to previous solution VLAN. It encapsulates the network traffic in to the tunnel, and creates overlay networks that are isolated from the providers physical network. Virtual Tunnel endpoints (VTeps) are the devices to perform encapsulation and de-encapsulation on the VxLAN traffic. Each traffic segment is identified by a unique Virtual Network Identifier (VNI) (see Fig. 5).

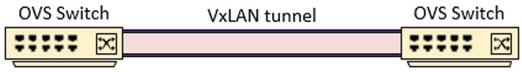

Fig. 5. OVS switches connected through a VxLAN tunnel.

In real world scenarios, multiple switches are connected to each other for the transmission of the network packets. The testbed is extended by adding more switches. As shown in Fig. 6, three virtual machines are added to the second switch and their virtual ports are defined as well. The Ryu controller manages the communication between the two switches. For the transmission of VxLAN encapsulated packets between the switches VxLAN tunnels are used in the setup.

The Fig. 7 explains the basic topology of a multi-tenant SDN network in which tenant A and tenant B have different virtual networks and their traffic is isolated using VxLAN tunnels. The tunnel is created on both VTEPs (switches) for the transmission of packets. VNIs are assigned on the VTEPs to separate the traffic of both tenants. The VTEP adds headers to the original packet sent by the VM such as VNI, and IP and MAC addresses of source and destination etc. and forwards the packet into the tunnel. Vtep2 receives the packet and removes the UDP headers assigned by the VTEP and sends the packet to the actual destination.

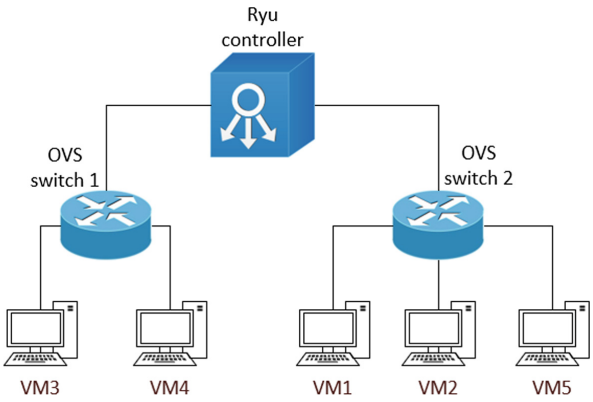

Fig. 6. Five Virtual Machines and two OVS switches that are controlled by a Ryu controller.

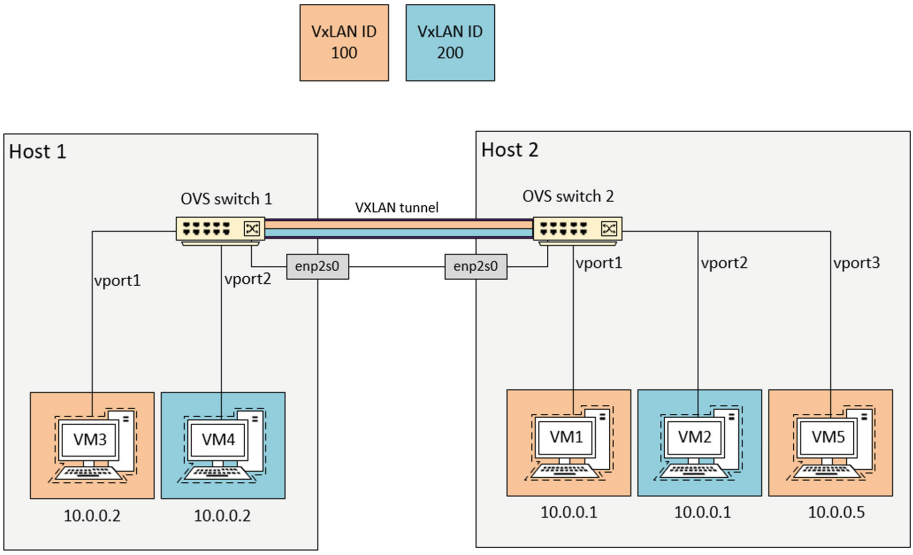

Fig. 7. Setup with five virtual machines distributed among two hosts. Both OVS switches are connected by a VxLAN tunnel with two isolated overlay networks (VNI 100- orange and VNI 200 - light blue). (Color figure online)

3.5 Isolation of Multi-tenants Using Flow Table

In the testbed scenario, two overlay networks are shown that are isolated from each other. The two networks are assigned different VNIs. The first network has VNI 100 and contains VM1, VM3, and VM5. The second network has VNI 200 and contains VM2 and VM4 as shown in Fig. 7. The virtual machines in a single host can have equal IP configuration and same MAC addresses as they

```
1  table=0,in_port=vport1,actions=set_field:100->tun_id,resubmit(,1)
2  table=0,in_port=vport2,actions=set_field:200->tun_id,resubmit(,1)
3  table=0,actions=resubmit(,1)
4
5  table=1,tun_id=100,dl_dst=52:54:00:73:9a:e9,actions=output:vport1
6  table=1,tun_id=200,dl_dst=52:54:00:bc:e0:1a,actions=output:vport2
7  table=1,tun_id=200,dl_dst=52:54:00:53:c2:0f,actions=output:vtunnel
8  table=1,tun_id=100,dl_dst=52:54:00:72:5c:8c,actions=output:vtunnel
9  table=1,tun_id=100,dl_dst=52:54:00:58:6f:45,actions=output:vtunnel
10
11 table=1,tun_id=200,nw_dst=10.0.0.2,actions=output:vport1
12 table=1,tun_id=100,nw_dst=10.0.0.2,actions=output:vport2
13 table=1,tun_id=100,nw_dst=10.0.0.1,actions=output:vtunnel
14 table=1,tun_id=100,nw_dst=10.0.0.5,actions=output:vtunnel
15 table=1,tun_id=200,nw_dst=10.0.0.1,actions=output:vtunnel
16
17 table=1,priority=100,actions=drop
```

Fig. 8. OVS switch 1 flow table.

are separated using the VNI. This prevents IP address collision and any tenant can create a network without having the information of other tenant networks.

In Fig. 8, the flow table of OVS switch 1 contains the information of VNIs assigned to the virtual machines and the forwarding path of the packets according to the assigned VNIs. In the first two lines of the flow table, VNIs are assigned using tun-id field to the respective ports of the VMs. For the incoming packet from host 2, tun-ids and the destination MAC address (dl-dst) are matched and the packets are forwarded to the specified ports (VMs) in the output field. Lines 7–9 shows the outgoing transmission to host 2 through the tunnel. In line 11–15, same process is followed for the incoming and outgoing flow for Layer 3 packets. All other packets are dropped by the switch.

```
1  table=0,in_port=vport1,actions=set_field:100->tun_id,resubmit(,1)
2  table=0,in_port=vport2,actions=set_field:200->tun_id,resubmit(,1)
3  table=0,in_port=vport3,actions=set_field:100->tun_id,resubmit(,1)
4  table=0,actions=resubmit(,1)
5
6  table=1,tun_id=100,dl_dst=52:54:00:72:5c:8c,actions=output:vport1
7  table=1,tun_id=200,dl_dst=52:54:00:53:c2:0f,actions=output:vport2
8  table=1,tun_id=100,dl_dst=52:54:00:73:9a:e9,actions=output:vtunnel
9  table=1,tun_id=200,dl_dst=52:54:00:bc:e0:1a,actions=output:vtunnel
10 table=1,tun_id=100,dl_dst=52:54:00:58:6f:45,actions=output:vport3
11
12 table=1,tun_id=100,nw_dst=10.0.0.5,actions=output:vport3
13 table=1,tun_id=100,nw_dst=10.0.0.1,actions=output:vport1
14 table=1,tun_id=200,nw_dst=10.0.0.1,actions=output:vport2
15 table=1,tun_id=100,nw_dst=10.0.0.2,actions=output:vtunnel
16 table=1,tun_id=200,nw_dst=10.0.0.2,actions=output:vtunnel
17
18 table=1,priority=100,actions=drop
```

Fig. 9. OVS switch 2 flow table.

Figure 9 shows the flow table of OVS switch 2 which contains two VMs of same network VM1 and VM5 respectively and one VM of different network. VM1, VM3, and VM5 are reachable, because they share the same network, while VM2 can't reach the VM1 and VM5 in host 2 because of network isolation. In the first three lines of the flow table VNIs are assigned to all the VMs in host 2. The matching of incoming and outgoing transmission in flow table in host 2 is similar to the matching done in flow table of host 1. However, in host 2 packets from VM2 can be forwarded to VM5 and vice versa as they share the same VNI as shown in line 12–13.

4 Conclusion and Future Work

In this paper, we discuss the main concepts of SDN and the techniques which are used to develop a testbed in a SDN environment. We also describe our testbed for isolation of multi-tenants in SDN. In the testbed, isolation is achieved using VxLAN and the configuration of flows in the flow table of the switch. Ryu controller is used as SDN controller to manage all the communication between switches and hosts. Two OVS switches are created in the testbed and multiple virtual machines are attached to the switches. VxLAN tunnels are created on the switches and VNIs are assigned for the identification of different networks in the tunnel. The flow table is configured for the isolated communication of packets between the switches. With the configured OpenFlow table in the OVS switches and specified VNIs for the VxLAN tunnels, we are now able to create multi-tenant SDNs in the testbed.

In the future, we will focus on the implementation of our testbed in the OpenStack environment [15]. OpenStack is a cloud operating system that allows us to create multi-tenant SDNs in a much faster and scalable way. This will improve the efficiency, and it will allow a more sophisticated SDN setup. We will also develop an outlook to integrate OpenFlow based authentication using IEEE standard mechanisms.

Acknowledgment. This project has received funding from the European Unions H2020 research and innovation program under grant agreement H2020-MCSA-ITN-2016-SECRET 722424

References

1. Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., Ohlman, B.: A survey of information-centric networking. IEEE Commun. Mag. **50**(7), 26–36 (2012)
2. S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, M. Tyson, Fresco: modular composable security services for software-defined networks. In: Proceedings of Network and Distributed Security Symposium (2013)
3. Yu, M., Jose, L., Miao, R.: Software defined traffic measurement with opensketch. USENIX NSDI **vol**, 31 (2013)

4. Fayazbakhsh, S.K., Chiang, L., Sekar, V., Yu, M., Mogul, J.C.: Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In: USENIX NSDI, Seattle, WA, USA, pp. 1–13, 533–546 (2014)
5. Kapadia, S., Subagio, P.H., Yang, Y., Shah, N., Jain, V., Agrawal, A.: Implementation of virtual extensible local area network (VXLAN) in top-of-rack switches in a network environment, Google Patents, US Patent 9,565,105 (2017)
6. OpenFlow Switch Specification. http://goo.gl/1DYxw6. Accessed 14 Oct 2013
7. Ryu: An Operating System for Software Defined Network. http://osrg.github.com/ryu/
8. OpenDayLight. https://www.opendaylight.org/
9. POX: A Python-Based OpenFlow Controller. http://www.noxrepo.org/pox/about-pox/
10. Gude, N., et al.: NOX: towards an operating system for networks. ACM SIGCOMM CCR **38**(3), 105–110 (2008)
11. Floodlight. http://floodlight.openflowhub.org/
12. Mahalingam, M., et al.: Virtual eXtensible Local Area Network (VXLAN): a framework for overlaying virtualized layer 2 networks over layer 3 networks. In: RFC7348 (2014). https://doi.org/10.17487/RFC7348
13. Chowdhury, N., Boutaba, R.: A survey of network virtualization. In: Elsevier Computer Networks (2010)
14. European Telecommunications Standards Institute, Network Functions Virtualisation (2012). http://portal.etsi.org/NFV/NFVWhitePaper.pdf
15. OpenStack. https://www.openstack.org/