2021 IEEE 2nd International Conference on Signal, Control and Communication (IEEE-SCC 2021)
December 20 – 22, 2021, Hammamet - Tunisia

Vendor-Independent Software-Defined Networking

Santiago Pagola Moledo
Dept. of Computer Science
IDA, Linköping University
Linköping, Sweden
santipagola@gmail.com

Abhimanyu Rawat
Dept. of Computer Science
IDA, Linköping University
Linköping, Sweden
abhimanyur2010@gmail.com

Andrei Gurtov
Dept. of Computer Science
IDA, Linköping University
Linköping, Sweden
andrei.gurtov@liu.se

2021 IEEE 2nd International Conference on Signal, Control and Communication (SCC) | 978-1-6654-0270-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SCC53769.2021.9768368

***Abstract*—Software-Defined Networking (SDN) is an emerging trend in networking that offers several advantages such as smoother network management over traditional networks. By decoupling the control and data planes from network elements, a huge amount of new opportunities arise, especially in network virtualization. In cloud datacenters, where virtualization plays a fundamental role, SDN presents itself as the perfect candidate to ease infrastructure management and to ensure correct operation. Even if the original SDN ideology advocates openness of source and interfaces, multiple networking vendors offer their proprietary solutions. In this work, an open-source SDN solution, named Tungsten Fabric, is evaluated in a virtualized datacenter and several SDN-related industry use-cases are examined. The main goal of this work is to determine whether Tungsten Fabric can deliver the same set of use-cases as proprietary solutions.**

***Index Terms*—SDN, datacenters, Tungsten Fabric, network virtualization**

I. Introduction

Software-Defined Networking (SDN) has long been a promising technology with potential to change both the economics of networking and the way we design and manage our network infrastructure [1]. Unlike traditional IP networks, where configuration and maintenance is carried out on every network element (NE) individually in a time-consuming manner [2], SDN is believed to provide new ways to automate parts of today's network configuration, particularly within the context of cloud environments [3].

SDN is a networking paradigm that aims to separate network control and data planes. In addition, it proposes a centralized network control that has a global overview of the underlying forwarding plane, thus making network management more effective, scalable and agile [4]. By transitioning from a distributed to a centralized network control, the network becomes programmable, thus yielding a smoother configuration and maintenance.

Originally, SDN advocates the open-source ecosystem. The primary organization behind the promotion of SDN, the Open Networking Foundation (ONF), believes that SDN should have open interfaces and well-defined APIs. Despite this thought, many vendors have chosen to implement their own proprietary solutions. An example of that is Juniper, which develops and maintains a well-known SDN system named *Contrail*.

The goal of this paper is to explore alternate open-source SDN solutions, such as Tungsten Fabric. Formerly named OpenContrail[1], Tungsten Fabric is a fully-featured multi-cloud, multi-stack network virtualization solution widely adopted by many networking companies such as Juniper and Cisco.

This paper makes the following contributions: (1) it examines a number of practical use-cases that are of key importance in cloud datacenters; and (2) it presents Tungsten Fabric as a viable open-source SDN solution to be used in datacenters.

The rest of the paper is organized as follows: Section II presents some related work on SDN in datacenters. Sections III and IV describe the proposed architecture and evaluation metrics to use, Section V goes through novel results, and finally Section VI provides some guidelines and thoughts on Tungsten Fabric's eligibility in cloud datacenters.

II. Related work

SDN has been used within the research community for many years since it enables rapid network prototyping and deployment. Examples of the different fields where SDN has proved to be an excellent candidate are 5G [7]–[9], [23], industrial control systems [10], IoT [11] and automotive sector [12] and more.

Needless to say, SDN's potential has also been tested in cloud datacenters ([13]–[15]). In particular, SDN-enhanced VM migration has been studied in [16], [17]. Authors in [18] show how SDN can also help establish VPLS tunnels in IP/MPLS underlay networks.

Solutions using different open-source tools have been proposed in [19], [22], where an IT resource manager using OpenStack, OpenROADM, OpenConfig, T-API, ODTN et. al. has been developed. It enables the SDN/NFV use cases for efficient utilization of resources, network maintainability, multi-tenant slicing and management of multiple OpenStack based datacenters. The flexibility of the solutions are limited and the framework has multiple pieces working independently making them a complex structure to operate, which eventually would require some manual work to get them all function in cohesion.

In the paper by TianZhang et. al. [20] different types of live VM migration aspects have been put forward. Using OpenStack, several systems and networking aspects have been

[1]http://www.opencontrail.org/opencontrail-is-now-tungsten-fabric/, last visited on: November 17, 2021

Authorized licensed use limited to: UNIVERSITAT POMPEU FABRA. Downloaded on May 14,2022 at 14:43:27 UTC from IEEE Xplore. Restrictions apply.

discussed which directly affects the performance of the VM migration. SDN plays a key role in making sure that the migration process sees a minimum downtime. Performance comparison between different migration approaches such as parallel and sequential has been done. Different tools that make up a framework have been proposed but they are loosely integrated, which makes it difficult to standardize into a generic use-case tool.

Toosi et. al. [21] proposed a low cost SDN solution using the Raspberry-Pi and low cost embedded systems for conducting research for SDN-enabled cloud computing. The proposed solution paves a way to integrate the Open vSwitch, low-cost embedded computers, to build up a network of OpenFlow switches. In data center environments we use commodity servers but low cost and energy devices are still not very prevalent, however it could lead to a testing playground before the real world deployments.

To the best of our knowledge, this is the first paper that studies Tungsten Fabric. This is most likely due to the fact that it was renamed from OpenContrail and adopted by the Linux Foundation in December 2017. On the other hand, authors such as Harrabi et al. [5] and Yu et al. [6] use OpenContrail in their work. It is not unlikely that in the upcoming years Tungsten Fabric will gain traction both in industry and in the research community.

III. Proposed architecture

Figure 1 shows the proposed deployment of Tungsten Fabric and OpenStack. There are a total of 4 compute nodes, where in each of these the Tungsten Fabric *vRouter* instance runs, one Tungsten Fabric controller (leftmost orange box) and one OpenStack controller (rightmost orange box). The leftmost gray VM is an Ubuntu Server 16.04 which acts as the DC-GW, performing NAT so traffic originated from inside the datacenter is able to reach the Internet through the provider network. NAT, and other network-related configuration such as traffic forwarding between the DC-internal networks and the provider network (green), was implemented using *iptables*.

The red network in figure 1 represents the management and API network, where instances are accessed and the different API servers listen on, whereas the blue network is the DC-internal network on top of which the overlays are created. Note that, in terms of virtualization, both networks are created as internal bridges, created prior to the deployment.

The deployment is fully automated with Ansible, using a set of playbooks. Note that since the infrastructure is purely virtual, all instances (i.e., controllers, compute nodes and the DC-GW) run under the host machine's KVM hypervisor. The host bare-metal machine is a Dell PowerEdge R730xd server with 377GB of RAM memory and 3.3 TB of hard disk. The KVM-based compute nodes, in turn, spawn VMs using QEMU, achieving nested virtualization.

Two PoDs are shown in figure 1. This is designed this way to test OpenStack's Availability Zone (AZ) feature, where two such zones are defined: *left*, containing compute nodes 1 and 3, and *right*, containing compute nodes 2 and 4.

IV. Tungsten Fabric evaluation

The deployment has been made with Ansible and the virtual infrastructure is created, a set of use-cases are performed in order to observe Tungsten Fabric's potential. These use-cases have have also been studied in parallel using Juniper's Contrail Cloud SDN solution. Table I lists such use-cases. Table II lists the different ways these use-cases being tested, per OSI layer. A more detailed description of these follows in the upcoming subsections.

TABLE I
Proposed use-cases to evaluate Tungsten Fabric

ID	Description
UC0	Achieve external connectivity from any VM
UC1	Create an IP network between VMs in the same AZ
UC2	Create an IP network between VMs in different AZ's
UC3	Create a routed L3 connection between two IP networks
UC4	"Stitch" IP networks to existing MPLS L3VPN's
UC5	VM migration across AZ's

TABLE II
Evaluation metrics for the use-cases to be executed

OSI Layer	Protocol	Command	Applicable Use-Cases
L2	ARP	arp	1, 2
L3	ICMP	ping	0, 1, 2, 3, 4, 5
L4	TCP	nc	0
L7	HTTP	wget	0, 3
	SSH	ssh	4

A. Achieve external connectivity from any VM

The main goal of this use-case is for any VM in the datacenter to be able to achieve external connectivity. Tungsten Fabric offers a number of ways for achieving external connectivity. For this use-case, a *distributed NAT* is used. This means that instead of achieving NAT functionality by using a logical router to which virtual networks are connected, individual virtual networks can reach the IP fabric underlay using existing forwarding infrastructure on the compute node. As figure 2 shows, two NAT stages: the first one on the compute node to which the VM belongs, and the second one on the DC-GW, as mentioned in section III. This, however, only ensures outbound L3 connectivity. For L4 and upper layers, port forwarding has also been configured in the Tungsten Fabric controller.

As shown in table II, correct operation of this use-case is checked by (1) pinging a remote host (ICMP, on L3), (2) establishing a TCP (L4) connection to a remote server, and (3) downloading a text file from a remote server using HTTP (L7).

B. Create an IP network between VMs in the same AZ

This is the most basic form of L2 reachability between two VMs. This use-case spawns two VMs belonging to the same AZ, i.e., VMs running on either compute nodes 1 or 3. Since this use-case is about verifying whether two VMs belong to the same L2 domain, both VMs' ARP tables are checked. We

Authorized licensed use limited to: UNIVERSITAT POMPEU FABRA. Downloaded on May 14,2022 at 14:43:27 UTC from IEEE Xplore. Restrictions apply.

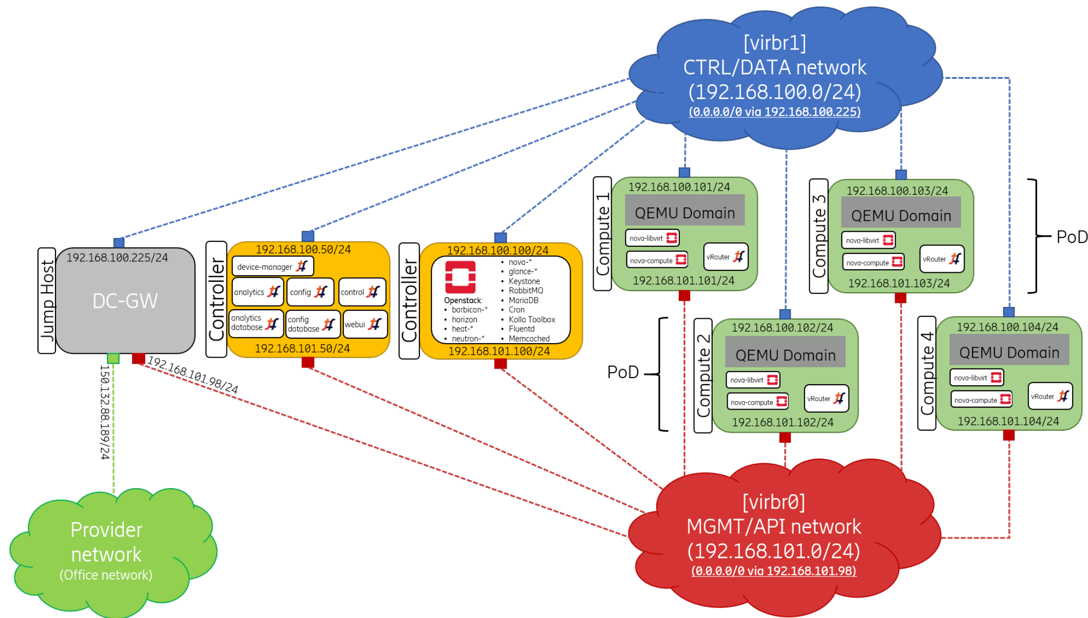

Fig. 1. Proposed virtual datacenter architecture

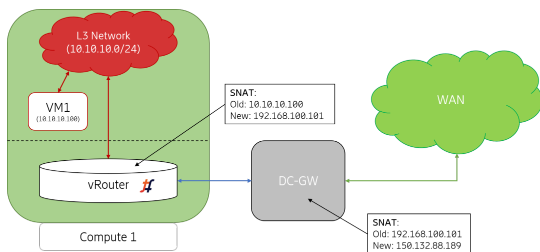

Fig. 2. Use-case 0

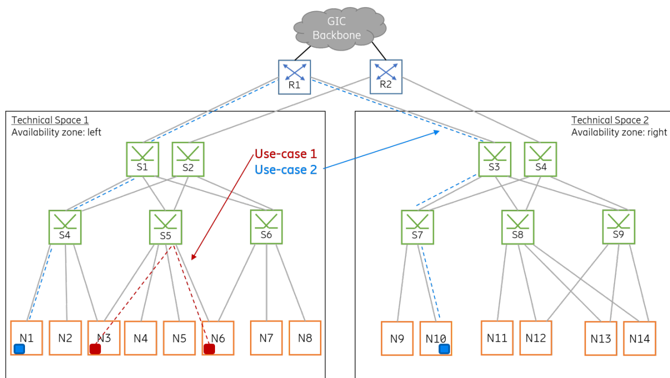

Fig. 3. Use-cases 1 and 2

expect the Tungsten Fabric vRouter to act as an ARP proxy to avoid ARP flood messages through the whole datacenter, thus reducing traffic. In addition, as shown in table II, one VM pings the other VM in the same virtual network to test basic L3 connectivity. Figure 3 illustrates this use-case.

C. Create an IP network between VMs in different AZ's

This use-case is an extension of use-case 1, defined in section IV-B above. The difference is that both VMs, still belonging to the same virtual network, now be running in different AZ's: VM1 is hosted on compute nodes 1 or 3, and VM2 is running on compute nodes 2 or 4. As for the previous use-case, both VMs' ARP tables are checked to verify that the can reach each other within the same broadcast domain, and one VM pings the other one to verify L3 functionality. Figure 3 depicts this use-case.

D. Create a routed L3 connection between two IP networks

Figure 4 shows a combination of the previously defined use-cases. This use-case is useful when a routed IP connection is desired between virtual networks. There are two main ways to

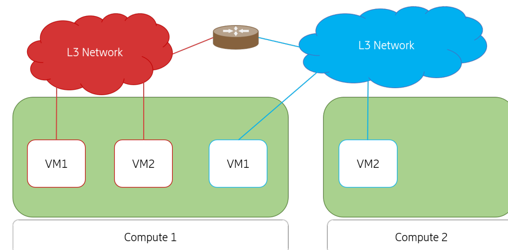

Fig. 4. Use-case 3

achieve this use-case: (1) using a logical router and attaching one interface of each virtual network to it; and (2) using network policies defined in Tungsten Fabric, which is one of its main networking features. Although figure 4 shows a logical router interconnecting both L3 networks, the second alternative has been chosen to implement this use-case.

Authorized licensed use limited to: UNIVERSITAT POMPEU FABRA. Downloaded on May 14,2022 at 14:43:27 UTC from IEEE Xplore. Restrictions apply.

As table II shows, correct operation is proved by having one VM in one virtual network ping another VM on the other virtual network, as well as performing a file download between these VMs using *wget*.

E. "Stitch" IP networks to existing MPLS L3VPN's

This use-case is about importing cloud-internal virtual networks into existing MPLS L3VPNs which the DC-GW has knowledge about. Using BGP, the Tungsten Fabric controller and the DC-GW exchange routes so any VM having a port on the exported virtual network is accessible from the target L3VPN. Note that for this use-case, a Juniper vMX router acting as an extra DC-GW is deployed in the datacenter layout proposed in figure III. As shown in table II, correct operation

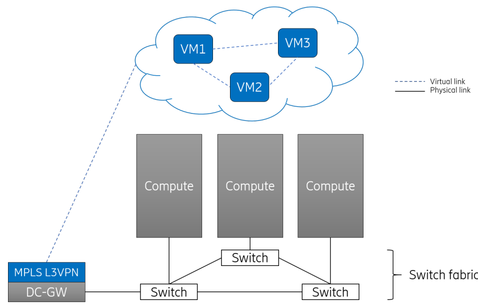

Fig. 5. Use-case 4

is shown by pinging a cloud-internal VM, with private IP address 10.10.10.100, from the vMX router (DC-GW2), and establishing an SSH session with it.

F. VM migration across AZ's

This use-case is about migrating workloads between the different AZ's defined: *left* and *right*. Two variants are tested: *live* migrations, where the migrated VM is to experience zero-downtime while being migrated, and *cold* migration, where the migrated VM will first be shut off, migrated to the other AZ and booted up to resume operation. Note that in the first case, the *live* migration will not have any shared storage, so it will be a *live block* migration[2]. Figure 6 illustrates this use-case. The

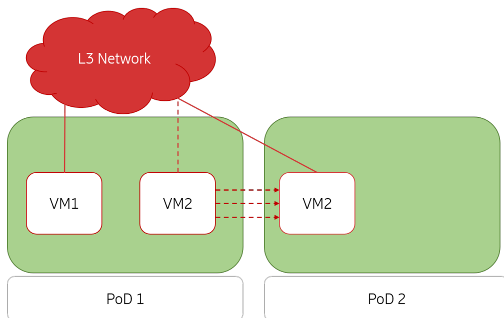

Fig. 6. Use-case 5

way of verifying both migration types is the following: one VM will ping a second VM while the latter is being migrated. The migrated VM is an Arch Linux cloud image of 1.79GB of disk size. As mentioned, in the *live block* migration we expect no service disruption, but for the *cold* migration case we expect an approximate downtime of:

$$t_{downtime} = t_{shutdown} + t_{migrate} + t_{grub} + t_{boot} + t_{control} \quad (1)$$

Equation 1 breaks down the total expected downtime. $t_{shutdown}$ represents the time it takes for the VM to fully shut down (2s), $t_{migrate}$ represents the time to copy the disk contents of the VM, t_{grub} the default timeout for the GRUB bootloader (default of 5s), t_{boot} represents the time to boot the actual VM OS (in technical terms, since it's a *systemd*-controlled Arch distribution, to reach the *network* target: 6s) and $t_{control}$ the remaining time for other operations such as API calls, etc. done by the OpenStack engine (usually around 3-4 seconds).

To estimate $t_{migrate}$, the upload bandwidth between the source and destination compute nodes is needed. This is done by the *iperf3* tool, which prior to the migration, reports a bandwidth of 2.34 GB/s. Hence, the total disk transfer time (VM migration) can be estimated by:

$$t_{migrate} = \frac{1.79GB}{2.34GB/s} = 0.765s \quad (2)$$

With this in mind, the total expected downtime is $t_{downtime} = 2 + 0.765 + 5 + 6 + 4 \approx 18s$.

V. Results

Most code listings in this section show a pattern, '[...]', representing truncated output in order to fit a given packet within a single line.

A. Achieve external connectivity from any VM

Figures 7a and 7b show a VM, with IP address **10.10.10.103**, pinging a remote host, *ida.liu.se*. From both figures it can be seen that both SNAT (red boxes) and DNAT (green boxes) are correctly working. In the first NAT stage, shown in figure 7a, the source IP address is replaced by the compute node's fabric underlay IP address, **192.168.100.104**. In the second NAT stage, done at the DC-GW (shown in figure 7b), this source IP address if further replaced by the DC-GW's IP address on the external network, i.e., **150.132.88.189**. The opposite process (DNAT) happens when packets are destined for the VM with IP address **10.10.01.103**.

Figure 7c shows the initial TCP three-way-handshake between a VM, with IP address **10.10.20.100** and a remote server (**81.228.138.88**), to whom it sends a "Hello world" message (not shown). As for the previous test, NAT is correctly being applied at both points. In addition, port translation is also being performed by the vRouter, i.e., source port **40524** is being replaced by **51500**, which corresponds to the first available port on the allocated port pool in the vRouter for outgoing L4 connections.

Finally, figure 7d shows a successful file download hosted on the same remote server, **81.228.138.88**, from a VM, with

[2] According to OpenStack terminology

Authorized licensed use limited to: UNIVERSITAT POMPEU FABRA. Downloaded on May 14,2022 at 14:43:27 UTC from IEEE Xplore. Restrictions apply.

IP address **10.10.20.100**. The red boxes highlight the HTTP GET and its response code.

```
[root@contrail-compute-04 -]# tcpdump -i any -n "host ida.liu.se"
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
15:53:57.031812 IP 10.10.10.103 > 130.236.180.77: ICMP echo request, id 1748, seq 1, length 64
15:53:57.032743 IP 192.168.100.104 > 130.236.180.77: ICMP echo request, id 1748, seq 1, length 64
15:53:57.040956 IP 130.236.180.77 > 192.168.100.104: ICMP echo reply, id 1748, seq 1, length 64
15:53:57.041037 IP 130.236.180.77 > 10.10.10.103: ICMP echo reply, id 1748, seq 1, length 64
```

(a) Pinging from a VM to a remote host: *ida.liu.se*. NAT stage 1, capture at compute node

```
[root@jump-host -]# tcpdump -i any -n "host ida.liu.se"
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
17:53:59.607737 IP 192.168.100.104 > 130.236.180.77: ICMP echo request, id 1748, seq 1, length 64
17:53:59.607797 IP 150.132.88.189 > 130.236.180.77: ICMP echo request, id 1748, seq 1, length 64
17:53:59.615641 IP 130.236.180.77 > 150.132.88.189: ICMP echo reply, id 1748, seq 1, length 64
17:53:59.615676 IP 130.236.180.77 > 192.168.100.104: ICMP echo reply, id 1748, seq 1, length 64
```

(b) Pinging from a VM to a remote host: *ida.liu.se*. NAT stage 2, capture at DC-GW

```
root@contrail-compute-04 - $ tshark -i any -nn -Y 'tcp.port == 5555'
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
301 8.878596817 10.10.20.100 -> 81.228.138.88 TCP 76 40524 > 5555 [SYN] Seq=0 Win=64240 [...]
302 8.879575397 192.168.100.104 -> 81.228.138.88 TCP 76 51500 > 5555 [SYN] Seq=0 Win=64240 [...]
303 8.890824348 81.228.138.88 -> 192.168.100.104 TCP 76 5555 > 51500 [SYN, ACK] Seq=0 Ack=1 Win=65160 [...]
304 8.890885184 81.228.138.88 -> 10.10.20.100 TCP 76 5555 > 40524 [SYN, ACK] Seq=0 Ack=1 Win=65160 [...]
305 8.891091724 10.10.20.100 -> 81.228.138.88 TCP 68 40524 > 5555 [ACK] Seq=1 Ack=1 Win=64256 [...]
306 8.891108988 192.168.100.104 -> 81.228.138.88 TCP 68 51500 > 5555 [ACK] Seq=1 Ack=1 Win=64256 [...]
```

(c) Extract of the TCP initial three-way-handshake with the remote server: *81.228.138.88* . Capture at compute node

```
[root@contrail-compute-04 -]# tshark -i tap47a974de-3f -nn -Y 'tcp.port == 8080'
Running as user "root" and group "root". This could be dangerous.
Capturing on 'tap47a974de-3f'
3 0.575665866 10.10.20.100 -> 81.228.138.88 TCP 74 38934 > 8080 [SYN] Seq=0 Win=64240 [...]
4 0.587638602 81.228.138.88 -> 10.10.20.100 TCP 74 8080 > 38934 [SYN, ACK] Seq=0 Ack=1 Win=65160 [...]
5 0.588395058 10.10.20.100 -> 81.228.138.88 TCP 66 38934 > 8080 [ACK] Seq=1 Ack=1 Win=64256 [...]
6 0.644960567 10.10.20.100 -> 81.228.138.88 HTTP 227 GET /remote-file2.txt HTTP/1.1
7 0.654272194 81.228.138.88 -> 10.10.20.100 TCP 66 8080 > 38934 [ACK] Seq=1 Ack=162 Win=65024 [...]
8 0.655389090 81.228.138.88 -> 10.10.20.100 TCP 251 [TCP segment of a reassembled PDU]
9 0.655430446 81.228.138.88 -> 10.10.20.100 HTTP 89 HTTP/1.0 200 OK (text/plain)
10 0.656056398 10.10.20.100 -> 81.228.138.88 TCP 66 38934 > 8080 [ACK] Seq=162 Ack=186 Win=64128 [...]
11 0.666158200 10.10.20.100 -> 81.228.138.88 TCP 66 38934 > 8080 [FIN, ACK] Seq=162 Ack=210 Win=64128 [...]
12 0.675851723 81.228.138.88 -> 10.10.20.100 TCP 66 8080 > 38934 [ACK] Seq=210 Ack=163 Win=65024 [...]
^C10 packets captured
```

(d) File download from a remote server: *81.228.138.88*. Capture at compute node

Fig. 7. Verification of correct operation for use-case 0

B. Create an IP network between VMs in the same AZ

Figure 8a shows the ARP tables from VM1. The red boxes indicate VM1's IP (**10.10.40.100**) and MAC (**02:b3:fd:83:54:28**) addresses, while the green box shows VM2's IP (**10.10.40.101**) and MAC (**02:40:7f:fc:0e:e0**) addresses. L2 reachability is confirmed by taking a look at figure 8b where the opposite happens: VM2, with IP address **10.10.40.101** and MAC address **02:40:7f:fc:0e:e0** is able to ”see” VM1 in its ARP tables (shown in the red box). This proves that both VMs belong to the same broadcast domain. Interestingly, the vRouter acts as an ARP proxy in order not to overload the underlay topology with ARP flood requests.

Figure 8c show a ping trace from VM1 to VM2. As opposed to the first use-case, no NAT is done this time, since traffic remains inside of the datacenter. Hence, this use-case is concluded.

C. Create an IP network between VMs in different AZ's

The results obtained through the execution of connectivity tests for this use case have no practical differences with respect to the previous use-case, where, as can be seen from figures 9a and 9b, both VMs belong to the same L2 domain and are able to ping each other.

```
[root@vm1 -]# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:b3:fd:83:54:28 brd ff:ff:ff:ff:ff:ff
    inet 10.10.40.100/24 brd 10.10.40.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::b3:fdff:fe83:5428/64 scope link
       valid_lft forever preferred_lft forever
[root@vm1 -]#
[root@vm1 -]# arp -nav
? (10.10.40.101) at 02:40:7f:fc:0e:e0 [ether] on eth0
? (10.10.40.1) at 00:00:5e:00:01:00 [ether] on eth0
? (10.10.40.2) at 00:00:5e:00:01:00 [ether] on eth0
Entries: 3      Skipped: 0      Found: 3
```

(a) Checking VM1's ARP tables

```
[root@vm2 -]# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:40:7f:fc:0e:e0 brd ff:ff:ff:ff:ff:ff
    inet 10.10.40.101/24 brd 10.10.40.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::40:7fff:fefc:ee0/64 scope link
       valid_lft forever preferred_lft forever
[root@vm2 -]#
[root@vm2 -]# arp -nav
? (10.10.40.2) at 00:00:5e:00:01:00 [ether] on eth0
? (10.10.40.1) at 00:00:5e:00:01:00 [ether] on eth0
? (10.10.40.100) at 02:b3:fd:83:54:28 [ether] on eth0
Entries: 3      Skipped: 0      Found: 3
```

(b) Checking VM2's ARP tables

```
[root@contrail-compute-01 -]# tshark -i any -Y "icmp and not ip.src == 127.0.0.1"
165 5.568024677 10.10.40.100 -> 10.10.40.101 ICMP 100 Echo (ping) request  id=0x0b02, seq=0/0, ttl=64
168 5.569458204 10.10.40.101 -> 10.10.40.100 ICMP 100 Echo (ping) reply    id=0x0b02, seq=0/0, ttl=64
202 6.569023961 10.10.40.100 -> 10.10.40.101 ICMP 100 Echo (ping) request  id=0x0b02, seq=1/256, ttl=64
205 6.569526743 10.10.40.101 -> 10.10.40.100 ICMP 100 Echo (ping) reply    id=0x0b02, seq=1/256, ttl=64
```

(c) Pinging VM2 from VM1, capture from compute node hosting VM1

Fig. 8. Verification of correct operation for use-case 1

```
$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 02:b3:fd:83:54:28 brd ff:ff:ff:ff:ff:ff
    inet 10.10.40.100/24 brd 10.10.10.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::de:2eff:fe6f:d784/64 scope link
       valid_lft forever preferred_lft forever
$
$ arp -nav
? (10.10.40.1) at 00:00:5e:00:01:00 [ether] on eth0
? (10.10.40.102) at 02:ab:08:48:84:d0 [ether] on eth0
? (10.10.40.2) at 00:00:5e:00:01:00 [ether] on eth0
```

(a) Checking VM1's ARP tables

```
[root@vm2 -]# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:ab:08:48:84:d0 brd ff:ff:ff:ff:ff:ff
    inet 10.10.40.102/24 brd 10.10.10.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::ab:8ff:fe48:84d0/64 scope link
       valid_lft forever preferred_lft forever
[root@vm2 -]#
[root@vm2 -]# arp -nav
? (10.10.40.100) at 02:b3:fd:83:54:28 [ether] on eth0
? (10.10.40.2) at 00:00:5e:00:01:00 [ether] on eth0
? (10.10.40.1) at 00:00:5e:00:01:00 [ether] on eth0
Entries: 3      Skipped: 0      Found: 3
```

(b) Checking VM2's ARP tables

```
[root@contrail-compute-01 -]# tshark -i any -Y "icmp and not ip.src == 127.0.0.1"
6956 217.686273673 10.10.40.100 -> 10.10.40.102 ICMP 100 Echo (ping) request  id=0x0c02, seq=0/0, ttl=64
6961 217.689227985 10.10.40.102 -> 10.10.40.100 ICMP 100 Echo (ping) reply    id=0x0c02, seq=0/0, ttl=64
6991 218.686823857 10.10.40.100 -> 10.10.40.102 ICMP 100 Echo (ping) request  id=0x0c02, seq=1/256, ttl=64
6996 218.688503123 10.10.40.102 -> 10.10.40.100 ICMP 100 Echo (ping) reply    id=0x0c02, seq=1/256, ttl=64
```

(c) Pinging VM2 from VM1, capture from compute node hosting VM1

Fig. 9. Verification of correct operation for use-case 2

D. Create a routed L3 connection between two IP networks

Figure 10a shows a packet capture of VM1, with IP address and subnet **10.10.10.100/24**, while it pings VM2, with IP address and subnet **10.10.20.101/24**. Even if both VMs belong to two different IP networks, both belong to the same AZ. On the other hand, figure 10b shows the same connectivity test between VM1 (**10.10.10.100/24**) and another VM (named VM3, with IP address **10.10.20.100/24**), hosted on a different AZ. ICMP requests get their corresponding response, meaning that traffic between both virtual networks is enabled thanks to the network policy defined in Tungsten Fabric.

Figure 10c shows a file download originated from VM1. The

Authorized licensed use limited to: UNIVERSITAT POMPEU FABRA. Downloaded on May 14,2022 at 14:43:27 UTC from IEEE Xplore. Restrictions apply.

file is hosted on VM3. The red boxes highlight the HTTP GET request and its corresponding response message. This proves L7 connectivity across initially isolated virtual networks.

```
[root@contrail—compute—03 ~]# tshark —i any —Y "icmp and not ip.src == 127.0.0.1"
27378 688.411199553 10.10.10.100 —> 10.10.20.101 ICMP 100 Echo (ping) request  id=0xae01, seq=0/0, ttl=64
27381 688.412276481 10.10.20.101 —> 10.10.10.100 ICMP 100 Echo (ping) reply    id=0xae01, seq=0/0, ttl=63
27409 689.411983593 10.10.10.100 —> 10.10.20.101 ICMP 100 Echo (ping) request  id=0xae01, seq=1/256, ttl=64
27412 689.412521124 10.10.20.101 —> 10.10.10.100 ICMP 100 Echo (ping) reply    id=0xae01, seq=1/256, ttl=63
```

(a) Pinging VM2 from VM1, capture from compute node hosting VM1

```
[root@contrail—compute—02 ~]# tshark —i any —Y "icmp and not ip.src == 127.0.0.1"
1256 33.344502983 10.10.10.100 —> 10.10.20.100 ICMP 100 Echo (ping) request  id=0xac01, seq=0/0, ttl=64
1261 33.348153624 10.10.20.100 —> 10.10.10.100 ICMP 100 Echo (ping) reply    id=0xac01, seq=0/0, ttl=63
1286 34.345116314 10.10.10.100 —> 10.10.20.100 ICMP 100 Echo (ping) request  id=0xac01, seq=1/256, ttl=64
1291 34.346414838 10.10.20.100 —> 10.10.10.100 ICMP 100 Echo (ping) reply    id=0xac01, seq=1/256, ttl=63
```

(b) Pinging VM3 from VM1, capture from compute node hosting VM1

```
[root@contrail—compute—03 ~]# tshark —n —i tap20d2a6e7—62 —Y 'http or tcp'
Running as user "root" and group "root". This could be dangerous.
Capturing on 'tap713d2b92—66'
7 2.933926105 10.10.10.100 —> 10.10.20.100 TCP 74 44630 > 80 [SYN] Seq=0 Win=29200 [...]
8 2.936935710 10.10.20.100 —> 10.10.10.100 TCP 74 80 > 44630 [SYN, ACK] Seq=0 Ack=1 Win=65160 [...]
9 2.937418524 10.10.10.100 —> 10.10.20.100 TCP 66 44630 > 80 [ACK] Seq=1 Ack=1 Win=29200 [...]
10 2.938895445 10.10.10.100 —> 10.10.20.100 HTTP 156 GET /remote—file.txt HTTP/1.1
11 2.940913456 10.10.20.100 —> 10.10.10.100 TCP 66 80 > 44630 [ACK] Seq=1 Ack=91 Win=65152 [...]
12 2.944832938 10.10.20.100 —> 10.10.10.100 TCP 252 [TCP segment of a reassembled PDU]
13 2.945214632 10.10.20.100 —> 10.10.10.100 HTTP 189 HTTP/1.0 200 OK (text/plain)
14 2.945558605 10.10.10.100 —> 10.10.20.100 TCP 66 44630 > 80 [ACK] Seq=91 Ack=187 Win=29200 [...]
15 2.983038863 10.10.10.100 —> 10.10.20.100 TCP 66 44630 > 80 [ACK] Seq=91 Ack=311 Win=29200 [...]
16 3.055261382 10.10.10.100 —> 10.10.20.100 TCP 66 44630 > 80 [FIN, ACK] Seq=91 Ack=311 Win=29200 [...]
17 3.056420842 10.10.20.100 —> 10.10.10.100 TCP 66 80 > 44630 [ACK] Seq=311 Ack=92 Win=65152 [...]
^C11 packets captured
```

(c) Downloading a text file hosted on VM3 from VM1

Fig. 10. Verification of correct operation for use-case 3

E. "Stitch" IP networks to existing MPLS L3VPN's

Figure 11a shows the routing information to reach the VM (**10.10.10.100**). From the blue box one can see that this route is learned from the Tungsten Fabric controller at **192.168.100.50**, and the green box indicates what the controller tells the DC-GW about this route: it is reachable via the compute node at **192.168.100.103**. Finally, figures 11b and 11c show that the ping and the SSH connection are functional when using the specific vMX VRF of the MPLS L3VPN, named **TF-VPN**, thus concluding this use-case.

F. VM migration across AZ's

1) Block live migration: Figure 12 shows the ping Round-Time Trip (RTT) measured from a VM which pings another VM while being live migrated. The ping interval is set to 100ms instead of the default 1s. Recall that the live migration has no shared storage, thus called *block live* migration. As seen in figure 12, the ICMP pings are uninterrupted throughout the whole migration process, but a significant peak can be noticed. In the live migration process, the step usually referred to as *stop and copy* "suspends" the VM and copies the remaining memory pages from the source to the destination compute node. Since the migrated VM is an Arch Linux image with 44 MB of active RAM usage at the time of the migration (checked with the UNIX *free* command), having a measured upload bandwidth of 2.05GB/s (using *iperf3*), this step took at most $\frac{44MB}{2.05GB/s} = 26ms$. Thus, the observed peak is observed due to the fact that the last memory pages are being copied and control is being transferred to the new compute node, thus resulting in a slower ICMP reply reception.

2) Cold migration: Figure 13 shows the obtained ping RTT times when a VM pings another VM while being "coldly" migrated. As expected, the pings get interrupted while the VM is being migrated. In particular, from figure 13, a downtime of approximately 17 seconds can be observed, which corresponds to the expected downtime calculation according to equation 1. Observe that in both cases, the measured bandwidths are highly elevated, thus achieving small migration times. This is due to the fact that the deployment made in this PoC is purely virtual, as explained in section III, so all running instances are hosted within the same physical server. Hence, DC-internal traffic never leaves the physical server.

```
root@vCP—vMX1> show route 10.10.10.100/32

TF—VPN.inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, — = Last Active, * = Both

10.10.10.100/32     *[BGP/170] 03:59:37, MED 100, localpref 200, from 192.168.100.50
AS path: ?, validation—state: unverified
> via gr—0/0/0.32770, Push 25

bgp.l3vpn.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, — = Last Active, * = Both

192.168.100.103:2:10.10.10.100/32
*[BGP/170] 03:59:37, MED 100, localpref 200, from 192.168.100.50
AS path: ?, validation—state: unverified
> via gr—0/0/0.32770, Push 25
```

(a) Route information about the cloud-internal VM

```
root@vCP—vMX1> ping 10.10.10.100 routing—instance TF—VPN
PING 10.10.10.100 (10.10.10.100): 56 data bytes
64 bytes from 10.10.10.100: icmp_seq=0 ttl=63 time=2.153 ms
64 bytes from 10.10.10.100: icmp_seq=1 ttl=63 time=2.000 ms
64 bytes from 10.10.10.100: icmp_seq=2 ttl=63 time=2.547 ms
64 bytes from 10.10.10.100: icmp_seq=3 ttl=63 time=2.138 ms
^C
—— 10.10.10.100 ping statistics ——
4 packets transmitted, 4 packets received, 0% packet loss
round—trip min/avg/max/stddev = 2.000/2.210/2.547/0.204 ms
```

(b) Pinging the cloud-internal VM from the new DC-GW

```
root@vCP—vMX1> ssh root@10.10.10.100 routing—instance TF—VPN
Last login: Fri May 17 09:49:13 2019 from 10.20.30.1
[root@VM1: ~]# ip address show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 02:03:fb:72:dc:06 brd ff:ff:ff:ff:ff:ff
inet 10.10.10.100/24 brd 10.10.10.255 scope global eth0
valid_lft forever preferred_lft forever
inet6 fe80::3:fbff:fe72:dc06/64 scope link
valid_lft forever preferred_lft forever
[root@VM1: ~]# exit
logout
Connection to 10.10.10.100 closed.

root@vCP—vMX1>
```

(c) Establishing an SSH connection to the cloud-internal VM

Fig. 11. Verification of correct operation for use-case 3

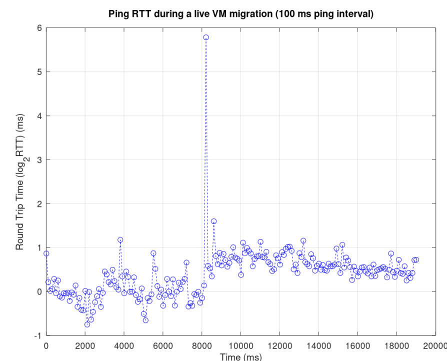

Fig. 12. Ping RTT obtained when VM1 pings VM2 while being migrated (block live)

Authorized licensed use limited to: UNIVERSITAT POMPEU FABRA. Downloaded on May 14,2022 at 14:43:27 UTC from IEEE Xplore. Restrictions apply.

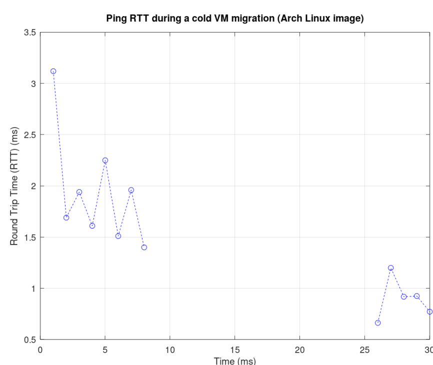

Fig. 13. Ping RTT obtained when VM1 pings VM2 during migration (cold)

VI. Conclusion & Future work

In this paper, a fully-featured open-source SDN system, Tungsten Fabric, along with a production OpenStack platform, was deployed on a virtualized datacenter hosted within a bare-metal server. The approach for testing the efficacy of Tungsten Fabric was to examine a set of use-cases that had also been carried out in parallel with a proprietary SDN solution, Contrail Cloud, developed and maintained by Juniper.

The obtained results show that Tungsten Fabric can deliver the exact use-cases that Contrail Cloud has to offer, but with the attractive advantage of being open-source. Deploying such a solution has potential to replace the expensive propriety solutions. This eases research and promotes innovation, which is a core ideology of SDN. The main limitation that this study had purely virtual nature. In real-life deployments, VM migration times (possibly even across datacenters on different continents) are likely to be longer.

References

[1] Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. ACM SIGCOMM Computer Communication Review, 44(2), 87-98.

[2] Benson, T., Akella, A., & Maltz, D. A. (2009, April). Unraveling the complexity of network management. In NSDI (pp. 335-348).

[3] Son, J., & Buyya, R. (2018). A taxonomy of software-defined networking (sdn)-enabled cloud computing. ACM Computing Surveys (CSUR), 51(3), 59.

[4] Kreutz, D., Ramos, F. M., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1), 14-76.

[5] Harrabi, M. A., Jeridi, M., Amri, N., Jerbi, M. R., Jhine, A., & Khamassi, H. (2015, June). Implementing NFV routers and SDN controllers in MPLS architecture. In 2015 World Congress on Information Technology and Computer Applications (WCITCA) (pp. 1-6). IEEE.

[6] Yu, Y., Lin, Y., Zhang, J., Zhao, Y., Han, J., Zheng, H., ... & Yang, H. (2013, November). First field demonstration of network function virtualization via dynamic optical networks with OpenContrail and enhanced NOX orchestration. In Asia Communications and Photonics Conference (pp. AF2C-4). Optical Society of America.

[7] Guerzoni, R., Trivisonno, R., & Soldani, D. (2014, November). SDN-based architecture and procedures for 5G networks. In 1st International Conference on 5G for Ubiquitous Connectivity (pp. 209-214). IEEE.

[8] Sun, S., Gong, L., Rong, B., & Lu, K. (2015). An intelligent SDN framework for 5G heterogeneous networks. IEEE Communications Magazine, 53(11), 142-147.

[9] Cho, H. H., Lai, C. F., Shih, T. K., & Chao, H. C. (2014). Integration of SDR and SDN for 5G. IEEE Access, 2, 1196-1204.

[10] Piedrahita, A. F. M., Gaur, V., Giraldo, J., Cardenas, A. A., & Rueda, S. J. (2018). Leveraging software-defined networking for incident response in industrial control systems. IEEE Software, 35(1), 44-50.

[11] Kalkan, K., & Zeadally, S. (2017). Securing internet of things (IoT) with software defined networking (SDN). IEEE Communications Magazine, (99), 1-7.

[12] Azizian, M., Cherkaoui, S., & Hafid, A. S. (2017). Vehicle software updates distribution with SDN and cloud computing. IEEE Communications Magazine, 55(8), 74-79.

[13] Cziva, R., Jouët, S., Stapleton, D., Tso, F. P., & Pezaros, D. P. (2016). SDN-based virtual machine management for cloud data centers. IEEE Transactions on Network and Service Management, 13(2), 212-225.

[14] Martini, B., Adami, D., Sgambelluri, A., Gharbaoui, M., Donatini, L., Giordano, S., & Castoldi, P. (2014, June). An SDN orchestrator for resources chaining in cloud data centers. In 2014 European Conference on Networks and Communications (EuCNC) (pp. 1-5). IEEE.

[15] Muñoz, R., Vilalta, R., Casellas, R., Martinez, R., Szyrkowiec, T., Autenrieth, A., ... & López, D. (2015). Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks. Journal of Optical Communications and Networking, 7(11), B62-B70.

[16] Mayoral, A., Vilalta, R., Muñoz, R., Casellas, R., & Martinez, R. (2015, March). Experimental seamless virtual machine migration using an integrated SDN IT and network orchestrator. In 2015 Optical Fiber Communications Conference and Exhibition (OFC) (pp. 1-3). IEEE.

[17] Liu, J., Li, Y., & Jin, D. (2014, August). SDN-based live VM migration across datacenters. In ACM SIGCOMM Computer Communication Review (Vol. 44, No. 4, pp. 583-584). ACM.

[18] Liyanage, M., Ylianttila, M., & Gurtov, A. (2016, January). Improving the tunnel management performance of secure VPLS architectures with SDN. In 2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC) (pp. 530-536). IEEE.

[19] Garrich, M and Hernández-Bastida, M and San-Nicolás-Martínez, C and Moreno-Muro, FJ and Pavon-Marino, P (2019, March). The Net2Plan-OpenStack Project: IT Resource Manager for Metropolitan SDN/NFV Ecosystems. In Optical Fiber Communications Conference and Exhibition (OFC). IEEE.

[20] He, TianZhang and Toosi, Adel Nadjaran and Buyya, Rajkumar (2019). Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers. IJournal of Parallel and Distributed Computing (pp. 55-68). Elsevier.

[21] Toosi, Adel Nadjaran and Son, Jungmin and Buyya, Rajkumar (2018, April). Clouds-PI: a low-cost Raspberry-PI based testbed for software-defined-networking in cloud data centers. In ACM SIGCOMM Comput Commun Rev 7 (pp. 1-11). ACM.

[22] Garrich, Miquel and Moreno-Muro, Francisco-Javier and Delgado, María-Victoria Bueno and Mariño, Pablo Pavón (2019, January). Open-source network optimization software in the open SDN/NFV transport ecosystem. In 2019 Journal of Lightwave Technology, Vol. 37, No. 1, (pp. 75-88). IEEE.

[23] Q. Schueller and K. Basu and M. Younas and M. Patel and F. Ball (2018, November). A Hierarchical Intrusion Detection System using Support Vector Machine for SDN Network in Cloud Data Center. In 2018 28th International Telecommunication Networks and Applications Conference (ITNAC) (pp. 1-6). IEEE.

Authorized licensed use limited to: UNIVERSITAT POMPEU FABRA. Downloaded on May 14,2022 at 14:43:27 UTC from IEEE Xplore. Restrictions apply.