Evaluating the Control and Management Traffic in OpenStack Cloud with SDN

Mu He, Alberto Martínez Alba, Ehab Mansour, Wolfgang Kellerer
Technical University of Munich
Email: {mu.he,alberto.martinez-alba,ehab.mansour,wolfgang.kellerer}@tum.de

***Abstract*—The performance of cloud computing depends heavily on the networking infrastructure, which carries the data traffic between the VMs, as well as the essential control and management traffic. Meanwhile, networks in data centers are becoming softwarized with the Software Defined Networking (SDN) paradigm to fuel their programmability and flexibility. However, since SDN switches need to contact the controller frequently for their configuration and flow rules setup, the overall data traffic forwarding performance can suffer, if the control and management plane gets congested. In order to proactively avoid such congestion, we need to study the involved traffic in detail. In this work, we perform an elaborate traffic evaluation on a real OpenStack cloud deployment and evaluate the types of exchanged messages in the control and management plane and their respective traffic volume. The evaluation reveals that the control and management traffic scales with the number of VMs and the VM-related events, and therefore provides guidelines for planning and operating the cloud networking infrastructure.**

***Index Terms*—Software Defined Networking, OpenStack, Network Measurement**

I. Introduction

In recent years, cloud computing is getting more popular due to its advantage in efficient infrastructure management and on-demand service provisioning. In fact, virtualizing computing, storage, and networking resources can drastically reduce the initial investment and operational maintenance costs [1]. Among several cloud computing solutions, OpenStack is an open-source project that aims to create a cloud with simplicity, scalability and rich functionality [2]. Until now, OpenStack has formed a strong ecosystem with abundant functional modules. Meanwhile, the rise of SDN tends to transform our networks into the ones that can be programmed. A centralized controller takes over the control of all forwarding entities and dynamically provisions the networking resources with its global knowledge in an efficient manner.

Integrating SDN into OpenStack benefits from the flexible management of forwarding entities and can therefore provide an optimized network virtualization service [3]. The controller configures the software switches in the servers (e.g., Open vSwitch) and controls the lifetime of the virtualized interfaces with management plane protocols. Furthermore, it installs flow rules in both software and hardware switches and guide their forwarding behaviors with the control plane protocol (e.g., OpenFlow). Typically, the traffic of control and management plane runs through the same networking infrastructure, which is separated from the one that carries the data plane traffic inside the cloud, i.e., out-band management and control. This separation is necessary, because the inter-VM communication can potentially occupy high bandwidth for distributed computing applications, leaving small room for the control and management traffic.

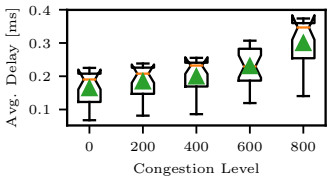


Fig. 1: Impact of the link congestion level on the average delay of control plane packets, i.e., Packet-Ins and Flow-Mods.

However, a cloud with a large number of VMs (more than 100 K [4]) can also suffer from high control and management traffic, which influences the responsiveness of the two planes and even the performance of data forwarding [5]. A motivational example demonstrates such impact. We connect an ONOS controller to a Cbench [6] instance, which emulates an SDN switch, with a 1 Gbps link. To add link congestion, we use iperf to generate cross traffic with bandwidth ranging from 0 to 800 Mbps. Fig. 1 shows that the average delay of control plane packets, i.e., Packet-Ins and Flow-Mods, experienced by the cBench instance, increases when the networking infrastructure has higher congestion level. In this case, a new flow needs to wait longer before its forwarding path can be configured in the forwarding entities, which is critical for time-sensitive communication and overall application performance.

Therefore, an elaborate measurement and analysis of the control and management plane traffic is needed to better understand its potential performance bottleneck. Previous works of networking performance evaluation and analysis in this regard focus on either OpenStack alone [3], [7], [8], or control plane of SDN itself [9]–[13]. An evaluation of OpenStack with SDN is still missing in the literature. Besides, most evaluations rely on software emulation such as Mininet, without a consideration of real hardware devices.

To fill this research gap, we face two major challenges in this work. First, there is no off-the-shelf solution of integrating OpenStack with hardware SDN switches available. Simplified Overlay Network Architecture (SONA) [14] suggests using ONOS to manage the servers; but for the data network it adopts legacy networking devices. Second, we need to design the evaluation procedure which can give us the most insights. The control and management plane traffic highly depends on the number of VMs and their networking interfaces, as

well as the actions we take for them such as VM creation and migration. We need to consider all possible scenarios. In summary, our contributions are listed as follows:

- we deploy a real OpenStack setup with an SDN controller running the control and management plane and controlling both servers and switches.
- we analyze the control and management plane messages, as well as the volume of each traffic type with different VM numbers and distributions.

The paper is organized as follows. Sec. II summaries the related-work of measurement regarding SDN control plane and OpenStack networking. Sec. III introduces the scenario of extending the networking component of OpenStack with SDN support. In Sec. IV, we present the measurement methodology and analyze the collected results. Sec. V concludes the paper.

II. Related Work

The analysis of SDN control plane traffic mainly focuses on (i) flow setup and (ii) inter-controller synchronization. Bianco et al. [9] analyzed the amount of OpenFlow messages for installing a new flow. For the synchronization, a single update to synchronize the NIB (Network Information Base) can generate 4π transactions, where π equals to the number of controllers [10]. Meanwhile, empirical models are developed to quantify the synchronous traffic of consistency protocols between the controllers [11], [12]. The impact of data plane topology on the control plane traffic is studied in [13].

There are several works that evaluate the performance of OpenStack Networking. Callegati et al. [7] focused on the network virtualization by measuring the packet throughput with two different networking providers: Linux Bridge and Open vSwitch. A comparison of networking performance between OpenStack and AWS EC2 demonstrated that OpenStack can provide high bandwidth and low latency between the VMs [8]. Malik et al. [3] evaluated OpenStack with different networking management entities such as OFAgentm, Ryu and OpenDaylight, and they concluded that OpenDaylight provides a significant advantage of efficiency over the other alternatives.

To the best of our knowledge, this is the first paper that evaluates the control and management plane traffic based on a real OpenStack deployment with SDN. Further, this work analyzes the exchanged messages, as well as the impact of the number of VM and their distributions on the induced traffic.

III. OpenStack with SDN

To integrate SDN into OpenStack, we need to understand how to deploy OpenStack manage the networking resources with Neutron. In the setup, both software (i.e., OVS) and hardware switches are controlled by a single SDN controller.

A. Setup Overview

OpenStack manages compute, storage and networking resources throughout the cloud computing infrastructure in a flexible manner. It keeps evolving to enable more features and higher stability. The operator can choose from an abundant set of components to create the deployment that best suits its own requirements [2]. The components run in heterogeneous infrastructure, and new hardware can be easily included.

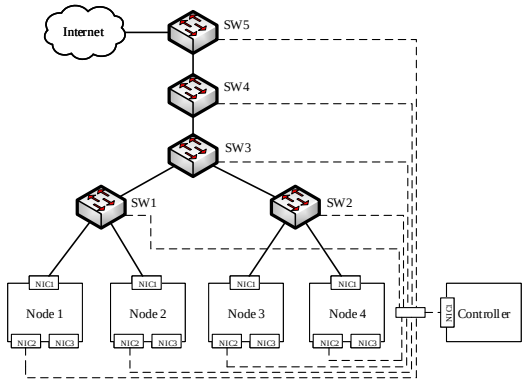


Fig. 2: Experiment OpenStack deployment with SDN integration. Node 1 to 4 and the controller are servers, and SW1 to SW5 are hardware switches.

We deploy five homogeneous servers: one as controller node and the other four as compute nodes (see Fig. 2). We enable the following components: Nova for compute service, Neutron for networking, Keystone for identity service, Glance for image service, Cinder for block storage and Horizon for dashboard.

Each compute node is connected to three networks. The data network which carries the traffic between the VMs is enabled via NIC1. We build the topology of the data network with a tree structure, which will be introduced in III-C. The management of software switches, the control of OpenFlow, and the internal communication between OpenStack components are realized by the management network via NIC2. We use a legacy Layer 2 switch to forward the management traffic. For security reasons, this network should be reached only within the data center. NIC3 provides the access to the storage network (by Cinder), which is omitted in the figure.

B. OpenStack Networking

Fig. 3 demonstrates the logical architecture of Neutron with several modules. The color represents the location of each module. *Neutron Server* exposes the networking API to the operator and listens to networking requests from other OpenStack components. After getting a request, e.g., creating a virtual network, *Neutron Server* calls the *Neutron Plugin* to handle it. *Neutron Plugin* maintains the logical networking states of OpenStack by fulfilling two tasks. First, it builds the logical connection between *Neutron Server* and *Neutron Database*. Second, it forwards the request from *Neutron Server* to *Neutron Agent*, which finally implements the request on *Network Provider*. *Neutron Database* stores all networking states persistently, including the created networks, subnets, ports, etc. *Network Provider* is the entity that implements the networking service, and it can be Linux Bridge, Open vSwitch (OVS) or physical switch that supports Neutron. The communication between the different modules is handled by the *Message Queue*. In OpenStack, various Advanced Message Queuing Protocol (AMQP) frameworks, e.g., RabbitMQ and ZeroMQ, are supported to provide peer-to-peer communication.

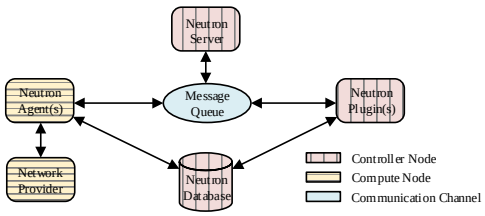


Fig. 3: Neutron architecture with components residing in different nodes.

With the Modular Layer 2 (ML2) plugin, Neutron endows the flexibility of realizing the same functionality with different *Network Provider*s in different compute nodes. For example, two VMs in two compute nodes of the same Layer 2 network can ping each other, when one node is equipped with Linux Bridge and the other with OVS. Furthermore, when we propose a new networking technology to Neutron, we only need to rewrite the interface to ML2 instead of the whole plugin.

C. Integrating SDN

The integration of SDN in our OpenStack deployment consists of two parts. First, we use SONA to provision virtualized networks with VXLAN-based L2 tunneling by OVS. Second, the data network between the compute nodes is connected via SDN hardware switches. Both software and hardware switches are controlled by a single ONOS controller through different applications. A legacy switch forwards the control and management traffic from the management port of each hardware switch and from NIC2 of each compute node to the controller node.

1) Virtual Network Providers: SONA in ONOS controller consists of *OpenStackNode*, *OpenStackNetworking* and a set of supporting applications. *OpenStackNode* bootstraps and manages the OVS in the compute nodes. When a compute node is added to the deployment, *OpenStackNode* connects to the OVSDB of the node and creates an integration and a tunneling bridge in the node. The interfaces of all VMs are connected to the integration bridge, which handles traffic filtering and VLAN header attachment. The tunneling bridge implements the VXLAN encapsulation and realizes the interconnection between VMs of a same internal network. The data network interface of the compute node (i.e., NIC1) is attached to the tunneling bridge.

2) Networking Fabric: Modern data center topologies are tree-like to ensure high bandwidth and connection reliability. To follow this structure, we connect each leaf switch (SW1 and SW2) with two compute nodes, and then connect the two leaf switches to the core switch (SW3). The data traffic between the VMs are transmitted in VXLAN tunnels. We enable the reactive forwarding application in ONOS to forward the VXLAN traffic. SW4 works as a hardware Virtual Tunnel End Point (VTEP) to facilitate the connection between the VMs and the external networks, e.g., the Internet. Notably, SW4 operates with its internal OVSDB to add and remove the VXLAN headers and does not need OpenFlow.

3) Hardware Gateway: SW5 acts as the gateway of the whole networking infrastructure. First, it filters out the part of traffic that is intended for the inside VMs from the external networks. Second, together with the controller, it handles the external ARPs for the VMs. The controller maintains a table of each VM's IP address and the MAC address of the compute node where the VM locates. When SW5 receives an ARP request, it forwards the request to the controller as a Packet-In message and afterwards the controller responds with an ARP reply as a Packet-Out message.

IV. Measurement Methodology and Results

In this section, we first present our measurement setup. Our intention is to evaluate all types of traffic exchanged in the control and management plane between the controller node and the devices. We expect that VM-related events and number of VMs will impact both periodic synchronous and asynchronous traffic.

A. Measurement Setup

We use the OpenStack Newton release to perform our measurement and analysis. For the SDN controller, we adopt ONOS 1.10 with SONA 2.1.0. The manipulation of the VM is performed via OpenStackClient CLI [15].

To analyze the periodic synchronous messages, we sample the traffic for 5 minutes and evaluate the intervals of different message types. To obtain the asynchronous messages, we create different numbers of VMs on compute node 1, migrate them to compute node 2, and terminate them in the end. The traffic traces are dumped during VM creation, migration and termination, which are used for bandwidth analysis. Following a similar approach [11], we measure the traffic every 1 second to get the consumed bandwidth samples, which we then average through a sliding window of 20 seconds.

Whereas bandwidth analysis indicates the potential high bandwidth consumption that can jeopardize real-time latency (shown in Fig. 1), traffic volume analysis, on the other hand, demonstrates the link usage on the long run. To this end, we create different numbers of VMs each with an internal interface and wait until the VMs are ready. We dump the traffic for 120 seconds and sum up the total number of Bytes of each message type. To gain statistical confidence, each scenario is repeated 20 times.

B. Traffic Type Analysis

The control and management traffic consists of the following components. (i) OpenFlow switch protocol runs between the controller and the software switches (OVS in all compute nodes), as well as the hardware switches (SW1, SW2, SW3 and SW5). The port of ONOS is 6653[1]. (ii) The database of the software switch, i.e., OVSDB, needs a connection to the controller (port 6640) with the OVSDB management protocol. (iii) AMQP messages implements the RPC (Remote Procedure Call) between different components of OpenStack. The port number in use is 5672. (iv) RESTful HTTP service by the controller node (port 9696) provides an up-to-date information of the networking state. All traffic types run with TCP.

[1]All port numbers are default ones in our OpenStack deployment.

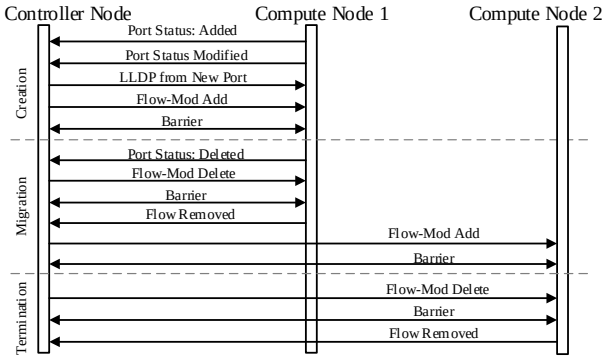


Fig. 4: Sequence of OpenFlow messages exchanged between the controller node and the compute nodes that host the VM during VM creation, migration and termination.

1) Periodic Synchronous Messages: Only traffic (i) and (ii) induce periodic synchronous messages. The controller node needs to maintain an updated view of the status of the software and hardware switches in the infrastructure. Synchronization messages of OpenFlow are observed for both OVS and hardware switches. First, LLDP packets are sent to each switch for topology discovery every 3 seconds. Second, the controller asks for the status of table, flow and port every 5 seconds and the status of group, group description and meter every 10 seconds. For the OVSDB in the four compute nodes and in SW4, the controller node first sends an echo request message to a compute node, which afterwards replies with an echo reply and a status report. This synchronization of type (ii) happens every 5 seconds. Periodic synchronization induces the least occupied bandwidth (i.e., zero bandwidth) during the lifetime of all involved devices in the cloud deployment.

2) Asynchronous Messages: The asynchronous messages take place whenever the status of the VM changes. There are three types of VM's status change (which we define as VM-related events) in OpenStack: creation, migration and termination. Note that for migration we only consider the live case, i.e., to migrate a VM without shutting it down, as it is widely adopted in modern clouds to enable runtime consolidation [1]. In the following, we analyze the exact message exchange sequence when a VM with one internal network interface is created, migrated and then terminated. We focus on OpenFlow because of its complexity; for the other message types, the involved packets are mainly status updates or process calls. Fig. 4 demonstrates the OpenFlow message sequence between the controller node and the two compute nodes that host the VM.

Whenever a VM is being created, the first generated OpenFlow message is a Port-Status from the compute node to the controller indicating a new detected port (port 9 in this case), after the OVS integration bridge creates that port. The second message is another Port-Status that reports the update of the newly detected port in the same direction. These two messages force ONOS to update its internal database of switch's ports and to start updating the database of topology. The controller issues the topology update by sending two Packet-Out with LLDP messages to the new port.

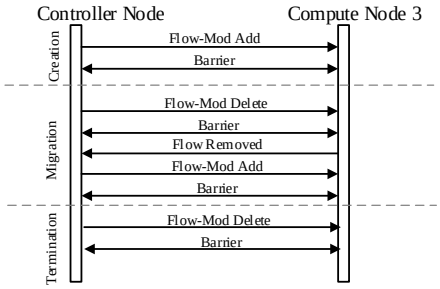


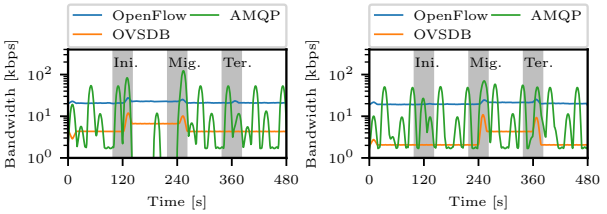
Fig. 5: Sequence of OpenFlow messages exchanged between the controller node and the other compute nodes (except node 1 and 2) during VM creation, migration and termination.

Now the *OpenstackNetworking* application of ONOS implements the respective flow rules in the compute node under the network configuration. We set the IP of the existing VM and the new VM as 10.0.0.1 and 10.0.0.10 respectively. A set of Flow-Mod(Add) messages from the controller install the following flow rules. (i) All traffic from port 9 is tagged with an ID (e.g., 100) and then passed to the next flow table. (ii) All traffic with IP destination 10.0.0.10 and tunnel ID 100 is forwarded to port 9 with the MAC destination of the 10.0.0.10 interface. (iii) All traffic with IP source 10.0.0.1 and destination 10.0.0.10 is passed to the next table. (iv) All traffic with IP source 10.0.0.10 and destination 10.0.0.1 is passed to the next table. (v) All traffic from IP source 10.0.0.10 are passed to the next table. All flow rules apply to IPv4 traffic and are with priority of 30000. SONA leverages a pipeline [14] of flow rule tables for packet processing. Flow rule (i) belongs to the VNI table and it associates traffic from a port with its corresponding network to ensure the logical isolation between different networks. The remaining flow rules belonging to the switching table facilitate the inter-connection between the two VMs. Since two VMs reside in the same node, we do not observe flow rules for VXLAN encapsulation. Furthermore, because of the default security group and no routing is configured, we do not see flow rules of the ACL and routing table. The Barrier messages make sure that all flow rules are successfully inserted in the respective tables.

When the new VM is migrated from node 1 to node 2, we first observe Port-Status indicating the corresponding port is deleted from the OVS bridge in node 1. ONOS then sends a set of Flow-Mod(Delete) together with Barrier to remove the obsolete flow rules. When a flow rule gets deleted, the OVS sends a Flow-Removed back to the controller and acknowledges the removal. Afterwards, a similar set of Flow-Mod(Add) are sent to compute node 2 with Barriers.

When terminating the VM, we observe Flow-Mod(Delete), Barrier and Flow-Removed messages in a row, which is similar to the first part of VM migration. The only difference is that there is no Port-Status indicating the delete of the port.

The message sequence between the controller node and the other compute nodes is demonstrated in Fig. 5. During VM creation, a new flow rule is added that forwards all traffic with IP destination 10.0.0.10 and VNI tag 100 to the port that is associated with the physical interface of the compute



(a) Compute Node 1 (b) Compute Node 2

Fig. 6: Traffic bandwidth of OpenFlow, AMQP and OVSDB messages between the controller node and the two compute nodes during VM creation, migration and termination.

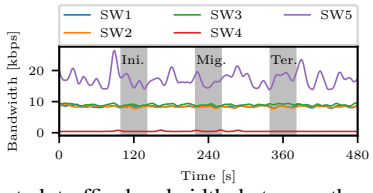


Fig. 7: Aggregated traffic bandwidth between the controller node and the five hardware switches during the creation, migration and termination of a VM. Note that the curve of SW3 is always higher than that of SW1 and SW2.

node. The flow rule will get refreshed when the VM migration happens: the same rule is deleted and afterwards added again. Finally, Flow-Mod(Delete) and Barrier erase the new flow rule.

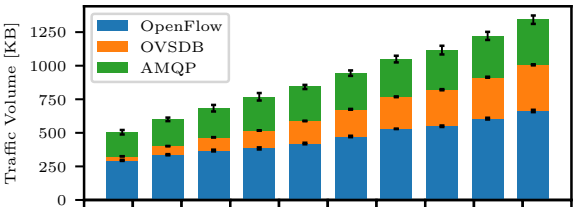
If the VM has another interface of the external network, i.e., it needs connectivity to outside the infrastructure, more flow rules need to be inserted to the OVS. In fact, we observe more Flow-Mod(Add) and Barrier messages during VM creation for the additional external interface. Following the same trend as before, these flow rules are also removed from one compute node and then reinstalled in another compute node during VM migration. Due to lack of space, we leave out the detailed analysis of this scenario.

As a summary, the integration of SDN induces different types of control and management traffic. Whereas only OpenFlow and OVSDB create synchronization messages periodically, all traffic types generate asynchronous messages when VM-related events happen.

C. Traffic Bandwidth/Volume Analysis

1) **Impact of the VM-related events on bandwidth:** Fig. 6 reveals the fluctuation of traffic from the controller node to the two compute nodes. Initially, one VM exists in node 1, whereas node 2 is empty. The bandwidth of OpenFlow messages is stable for both nodes. During VM creation at node 1, we only observe peaks of OpenFlow and OVSDB for node 1. VM migration again triggers peaks for both nodes, and during VM termination, peaks of all traffic types appear in node 2 and peaks of OpenFlow and AMQP appear in node 1.

In the following, we first evaluate the bandwidth of OpenFlow and OVSDB messages because of periodic synchronization (introduced in IV-B1), which are denoted as B_{OpenFlow} and B_{OVSDB} respectively. Each VM contributes the same amount of data that should be synchronized for both types of messages. Therefore, we can assume that the synchronization bandwidth



	0	1	2	3	4	5	6	7	8	9
OpenFlow	294.9	337.0	368.1	384.0	420.6	473.0	529.4	549.1	605.8	662.8
OVSDB	30.5	64.2	98.6	133.2	167.8	202.2	239.0	272.1	308.4	343.7
AMQP	179.0	199.2	216.7	251.3	253.3	269.4	281.0	295.0	307.7	336.0

Fig. 8: Total traffic volume of the synchronous messages (OpenFlow, OVSDB, and AMQP) for different number of VMs in the same compute node.

between the controller and the compute node is proportional to the number of VMs in the compute node, and we have

$$\begin{aligned} B_{\text{OpenFlow}} &= \Pi_{\text{vm}} \times b^{\text{vm}}_{\text{OpenFlow}} + b^{0}_{\text{OpenFlow}} \quad \text{[kbps]} \\ B_{\text{OVSDB}} &= \Pi_{\text{vm}} \times b^{\text{vm}}_{\text{OVSDB}} + b^{0}_{\text{OVSDB}} \quad \text{[kbps]}, \end{aligned} \tag{1}$$

where Π_{vm} represents the number of VMs, b^{vm}_{*} represents the average bandwidth increment per VM, and b^{0}_{*} denotes the zero bandwidth when no VM resides in the node. Note that $1\ \text{kbps} = 1\ \text{kB}/(8 * 1\ \text{s})$. After performing linear regression on the collected data, we can estimate that

$$\begin{aligned} b^{\text{vm}}_{\text{OpenFlow}} &= 2.68 \quad b^{0}_{\text{OpenFlow}} = 18.72 \quad \text{[kbps]} \\ b^{\text{vm}}_{\text{OVSDB}} &= 2.32 \quad b^{0}_{\text{OVSDB}} = 1.94 \quad \text{[kbps]}. \end{aligned} \tag{2}$$

For the bandwidth peaks, we focus on the maximal values because they indicate the worst case traffic increments when VM-related events should happen. We leave out the analysis of AMQP traffic, because the peaks appear rather irregularly. We add superscript "Max" on each bandwidth symbol B to represent the maximum. Similar to the synchronization traffic, the additional traffic grows linearly with the number of VMs:

$$\begin{aligned} B^{\text{Max}}_{\text{OpenFlow}} &= B_{\text{OpenFlow}} + \Pi_{\text{vm}} \times \delta^{\text{ini|mig|ter}}_{\text{OpenFlow}} \quad \text{[kbps]} \\ B^{\text{Max}}_{\text{OVSDB}} &= B_{\text{OVSDB}} + \Pi_{\text{vm}} \times \delta^{\text{ini|mig|ter}}_{\text{OVSDB}} \quad \text{[kbps]}, \end{aligned} \tag{3}$$

where $\delta^{\text{ini|mig|ter}}_{*}$ denotes the average bandwidth increment per VM because of VM-related events. Applying the same methodology, we can estimate that

$$\begin{aligned} \delta^{\text{ini}}_{\text{OpenFlow}} &= 6.19 \quad \delta^{\text{mig}}_{\text{OpenFlow}} = 3.57 \quad \delta^{\text{ter}}_{\text{OpenFlow}} = 4.04 \quad \text{[kbps]} \\ \delta^{\text{ini}}_{\text{OVSDB}} &= 7.62 \quad \delta^{\text{mig}}_{\text{OVSDB}} = 6.23 \quad \delta^{\text{ter}}_{\text{OVSDB}} = 5.04 \quad \text{[kbps]}. \end{aligned} \tag{4}$$

Fig. 7 depicts the traffic bandwidth between the controller node and the five hardware switches. The three tree switches, i.e., SW1 to SW3, create constantly similar bandwidths of OpenFlow synchronization. After zooming in, the traffic for SW3 (around 9.0 kbps) is slightly higher than that for SW1 and SW2 (both around 8.8 kbps). As the hardware VTEP, SW4 only synchronizes its OVSDB periodically with the controller node (no OpenFlow) and creates the least traffic. Since SW5 handles ARP requests from outside the cloud, it issues quite a lot of Packet-Ins on top of OpenFlow synchronization. We could not observe any regular pattern, and on average, SW5 has around 20 kbps traffic with the controller.

A study of the traffic bandwidth, particularly its peaks when VM-related events happens, can help with estimating the

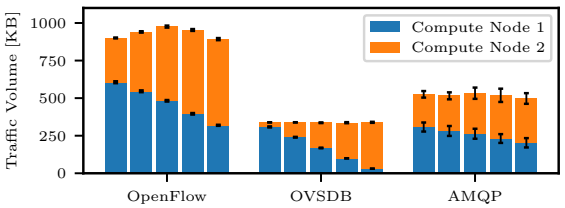


Fig. 9: Total traffic volume of the synchronous messages (i.e., OpenFlow, OVSDB, and AMQP) for different distribution of 8 VMs. The bars from left to right represent the number of VMs in node 1 as 8, 6, 4, 2, and 1.

worst-case control and management plane performance. For instance, when the VMs in the cloud need frequent migrations to ensure load balancing of computational resource [16], we would expect higher bandwidth of control and management traffic, which can potentially harm the flow setup performance.

2) **Impact of the number of VMs on traffic volume:** We split and categorize the traces according to the devices and traffic types. Fig. 8 illustrates the averaged traffic volume of OpenFlow, OVSDB and AMQP respectively, with 99.9% confidence interval. The exact numbers are listed under the figure. We leave out the traffic of RESTful service due to its tiny contribution to the overall traffic volume. With linear regression, we obtain the following on the collected data

$$\begin{aligned} V_{\text{OpenFlow}} &= 39.91 \times \Pi_{\text{vm}} + 282.87 \quad \text{[kB]} \\ V_{\text{OVSDB}} &= 34.84 \times \Pi_{\text{vm}} + \ \ 29.22 \quad \text{[kB]} \\ V_{\text{AMQP}} &= 16.17 \times \Pi_{\text{vm}} + 186.08 \quad \text{[kB]}, \end{aligned} \tag{5}$$

where V_* denotes the total traffic volume of one traffic type and Π_{vm} represents the number of VMs.

OpenFlow messages contributes the largest volume proportion (more than 50%). To maintain the global view, OpenFlow keeps a frequent synchronization of all details such as table, flow and port of each forwarding device. The volume of OVSDB synchronization messages also grows fast with more VMs. This is because the status of the connected VMs in the OVS is transmitted in the form of plain and sparse JSON string. Data compressing techniques can greatly reduce the size of the payload. Furthermore, the volumes of AMQP messages throughout different simulation runs differ, which results in large confidence interval.

3) **Impact of the VM distribution on traffic volume:** We next variate the distribution of VMs. In total 8 VMs distribute with different patterns in compute node 1 and 2, i.e., the number of VMs in node 1 ranges from 8 to 0. Fig. 9 shows the total traffic volume of OpenFlow, OVSDB and AMQP messages for node 1 and 2 from all scenarios.

We make the following observations. First, more VMs in a compute node induces more traffic for that node, which is consistent with the previous evaluation. There is no obvious difference for the total accumulated traffic on the two nodes for OVSDB and AMQP messages. On average, OVSDB messages add up to 340 kB, and AMQP messages add up to 520 kB. However, the accumulated OpenFlow traffic volume increases, when the VMs are more equally distributed. Furthermore, for the hardware switches, there is no obvious impact of the VM distribution on their respective control and management plane traffic (not shown in the figure).

To sum up, periodic synchronous control and management plane traffic scales with the number of VMs and their respective network interfaces. VM-related events induce asynchronous traffic, whose bandwidth also scales the number of involved VMs and interfaces. The distribution of VMs only impacts OpenFlow messages; the even distribution leads to the highest overall traffic volume.

V. Conclusion

We consider a real OpenStack deployment with an SDN controller managing all hardware switches and servers. Our investigation focuses on the traffic of the control and management plane, which consists of different message types. The evaluation provides insights for cloud operators to design and operate their deployment. The networking infrastructure should be able to host the potential control and management traffic volume that scales with the number of VMs and their interfaces. Furthermore, the VM-related events should be scheduled carefully to avoid sudden giant bandwidth increase.

Acknowledgment

This work has been funded by ERC FlexNets Project (grant No 647158), and the authors alone are responsible for the content of the paper.

References

[1] A. Corradi *et al.*, "VM consolidation: A real case based on OpenStack Cloud," *Futur. Gen. Computer Systems*, vol. 32, pp. 118–127, 2014.
[2] "OpenStack," https://www.openstack.org/, last accessed: 2019-04-10.
[3] A. Malik *et al.*, "A measurement study of open source SDN layers in OpenStack under network perturbation," *Computer Communications*, vol. 102, pp. 139–149, 2017.
[4] M. Dalton *et al.*, "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *Proceedings of NSDI*. USENIX, 2018, pp. 373–387.
[5] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM CCR*, vol. 43, no. 4. ACM, 2013, pp. 3–14.
[6] A. Tootoonchian *et al.*, "On Controller Performance in Software-Defined Networks," in *Proceedings of ICC*. ACM, 2012, pp. 1–6.
[7] F. Callegati *et al.*, "Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case," in *Proceedings of CloudNet*. IEEE, 2014, pp. 132–137.
[8] M. Kang *et al.*, "A Comparison of System Performance on a Private OpenStack Cloud and Amazon EC2," in *Proceedings of CLOUD*. IEEE, 2017, pp. 310–317.
[9] A. Bianco *et al.*, "Evaluating the SDN control traffic in large ISP networks," in *Proceedings of ICC*. IEEE, 2015, pp. 5248–5253.
[10] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of OSDI*, vol. 10. USENIX, 2010, pp. 1–6.
[11] A. S. Muqaddas *et al.*, "Inter-controller traffic in ONOS clusters for SDN networks," in *Proceedings of ICC*. IEEE, 2016, pp. 1–6.
[12] ——, "Inter-controller traffic to support consistency in ONOS clusters," *IEEE TNSM*, vol. 14, no. 4, pp. 1018–1031, 2017.
[13] M. Z. Naseer *et al.*, "The effect of network topology on the control traffic in distributed SDN," in *Proceedings of Networking*. IFIP, 2018.
[14] "SONA: DC Network Virtualization - ONOS - Wiki," https://wiki.onosproject.org/display/ONOS/SONA%3A+DC+Network+Virtualization, last accessed: 2019-04-10.
[15] "OpenStack Docs: OpenStackClient," https://docs.openstack.org/python-openstackclient/pike/, last accessed: 2019-04-10.
[16] C. Ghribi *et al.*, "Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms," in *Proceedings of CCGrid*. IEEE, pp. 671–678.