Regular Article

Performance Evaluation of OpenStack Cloud Integrated with SDN Controller

Hoang Thanh Loi, Pham Dinh Quy, Nguyen Nang Duc, Dinh Thi Thai Mai

Faculty of Electronics and Telecommunications, VNU University of Engineering and Technology, Hanoi, Vietnam

Correspondence: Dinh Thi Thai Mai, dttmai@vnu.edu.vn
Communication: received 07 February 2024, revised 23 May 2024, accepted 31 May 2024
Online publication: 01 June 2024, Digital Object Identifier: 10.21553/rev-jec.366

Abstract– **The Internet is undergoing a revolution thanks to the development of a very popular recent technology trend - "Cloud Computing". Various organizations as well as individuals increasingly prefer to use cloud computing services due to their advantages. Furthermore, the widespread implementation of cloud computing in large enterprises is an indicator of the technological advancement of information technology companies. It helps in the efficient allocation and utilization of infrastructure, power, and resources. In the meantime, centralized control provided by a Software-Defined Network (SDN) allows for flexibility in operations and management. For that reason, there has been an adequate amount of research and application of combining these two objects over the years. In this study, we will carry out the integration of the OpenStack platform with an SDN controller called Tungsten Fabric. Then we will evaluate network performance between virtual machines in terms of throughput, latency, and system utilization in terms of CPU and RAM in OpenStack with and without Tungsten Fabric.**

Keywords– **Software-defined network, cloud computing, openstack, tungsten fabric, system performance, network performance, monitoring.**

1 Introduction

Cloud Computing has transformed the landscape of Information Technology infrastructure management, with OpenStack emerging as a preferred open-source platform for deploying private and public clouds [1, 2]. OpenStack has different components that take on different roles; Neutron is the network component of OpenStack [3]. As cloud networks become increasingly complex, there is a growing need for efficient network management solutions that can optimize performance and enhance network agility.

Software-Defined Networking has emerged as a promising paradigm to address many challenges. By decoupling the network control plane from the data plane [4] SDN provides centralized control and programmability, allowing administrators to manage networks effectively, improve resource utilization, and enhance overall network performance. The SDN network architecture consists of three distinct layers: the application layer, the control layer, and the infrastructure layer [5]. The application layer encompasses the necessary applications and functions for the network. It connects to the control layer via the north API interface [6], which enables SDN to integrate with other technologies. Additionally, Tungsten Fabric (formerly known as OpenContrail) is a popular and powerful SDN controller used in network virtualization and network control in cloud environments [7].

Integrating an SDN controller with the OpenStack cloud platform can further enhance its capabilities, enabling control and automation of network resources [8, 9]. OpenStack Neutron provides a Neutron networking plugin which allows it to integrate with different SDN controllers [10]. In this context, the Tungsten Fabric SDN controller has gained popularity for its robust features and scalability. Tungsten Fabric provides a flexible and extensible platform for managing and orchestrating network services in OpenStack environments.

The idea behind this research is to evaluate the network efficiency and system utilization of the Tungsten Fabric SDN controller integrated with the OpenStack cloud platform and compare it with the standard OpenStack networking model. By conducting a comprehensive performance analysis, we seek to highlight the benefits and drawbacks of leveraging the Tungsten Fabric SDN controller in an OpenStack environment.

The remaining sections of this article are organized as follows: Section 2 provides an overview of the related works and our contribution. In section 3, we provide basic knowledge about OpenStack and Tungsten Fabric. In Section 4, we present the system model for integrating the Tungsten Fabric controller with OpenStack. Section 5 focuses on evaluating and analyzing the performance of the system, along with the presentation of implementation results. Finally, in Section 6, we draw conclusions based on our findings and discuss the future directions for further work.

2 Related Work

Multiple research studies have focused on enhancing network infrastructure by integrating OpenStack and SDN controllers. The authors of the article [11] focused on the GRE Tunnel model in Neutron, which is a key component of OpenStack. Measures of network performance between virtual machines include packet loss rate, packet transfer latency, and throughput. The experimental model includes three servers: one server acts as Controller, and the remaining two servers act as Compute nodes. The authors provide the following scenarios for evaluation using the IPERF tool: i) Virtual machines are located on the same network, same compute node, ii) different networks, same compute node, iii) same network, different compute node, and iv) different network, different compute node. The study's findings indicate that the location within the compute node and the established network range directly impact on the network performance between virtual machines.

With the same scenarios as the study mentioned above, the author in [12] performs a thorough evaluation of every aspect of network performance between virtual machines and the overall OpenStack system performance. Depending on the purpose of each type, the following tools are used in the article: tcpdump, Wireshark, top, atop, atopsar, PING, and IPERF. The model is tested on real environments (with physical servers) and virtual environments (using AWS EC2 services). The results provide us with a summary of OpenStack's functionality in various scenarios. When having direct impact to the Linux kernel, the obtained network performance results demonstrate that KVM/VHostnet is also a factor that significantly affects network performance. Furthermore, the enabling mode of DVR (Distributed Virtual Router) on each compute node enhances traffic routing capabilities.

The research team in [13] analyzed network performance through three separate models: Standard OpenStack (evaluated through OpenStack's Neutron), OpenStack when enabling the Distributed Virtual Router (DVR) function on Compute nodes, and OpenStack integrated with OpenDaylight SDN controller. Work is performed with packets of various sizes, such as TCP packets with 65,535 bits or UDP packets with 128, 1024, and 4096 bits. The authors' scenario involves assessing the capacity to deliver on Layer 2 (Layer 2 communication) and route on Layer 3 (Layer 3 routing) using both fixed and floating IP addresses with the help of the VMTP tool. Furthermore, these authors concentrate on virtual machines that are independent of one another and the same virtual network, whether they are on the same or separate nodes. The results obtained indicate that when using floating IP, DVR becomes the better option in terms of throughput on layer 3, while the OpenDaylight controller performs better with fixed IP.

In [14], the authors compare two SDN controllers, OpenDaylight and Open Kilda inside an OpenStack infrastructure. They evaluate several aspects of the system, such as throughput, latency, CPU utilization, round trip time, and topology change detection time while increasing the number of hosts and switches using Mininet. However, a limitation of this paper is that the SDN controllers were only deployed within the virtual machine of the OpenStack architecture instead of being integrated in parallel with the entire OpenStack system. As a result, the controllers could only monitor the virtual machine in which they were installed, rather than the entire OpenStack system. The paper also utilizes useful tools like OFNet and CBench for evaluating the system's performance.

The authors in [15] evaluate the network performance of two SDN controllers: ONOS and Floodlight based on TCP and UDP traffic. By using Mininet, authors create different SDN network topologies (single, linear, and tree) to measure transfer delay, throughput, and jitter. The IPERF tool is used to generate the TCP and UDP traffic. The result shows that ONOS controllers show better network performance than Floodlight in both TCP and UDP traffic. With the same method, the research team in [16] also presented the network efficiency of Ryu and Floodlight controllers in single, linear, tree, torus, and custom topology and pointed out that the Ryu controller has better performance than Floodlight. In addition, using the same scenario as before (and some additional test models), the author in [17] compared the performance of two Java language-based controllers, OpenDaylight and ONOS. The findings indicate ONOS controller performs better than the OpenDaylight controller in the testing environment they deployed. However, these researches evaluate the network performance of SDN controllers in the Mininet environment, not the OpenStack system which is integrated with the SDN controller.

In [18], the authors monitored the OpenStack Cloud system using Prometheus. They installed a node-exporter on each node to collect system data. The unique aspect of this article is the installation of an OpenStack-exporter on the controller node to collect data from the entire OpenStack cloud platform. The OpenStack-exporter parameter function includes monitoring RAM, vCPU, CPU statistics, disk, instances, and so on. It is far from node-exporter which takes data from the system on which OpenStack is installed. The collected data is transferred to Prometheus and displayed using Grafana. With the same standard OpenStack, the author in [19] evaluates the delay time to process the system operation when increasing the number of Compute nodes.

The authors in [20] evaluate network solutions performance for OpenStack, especially connecting the controller with Neutron via plug-ins to facilitate communication between them. OpenDaylight, Ryu, Floodlight, and Linux-bridge (a non-controller solution) are implemented. The scenarios provided include average packet transmission delay, TCP flows that may transmit for more than one second, and the maximum number of UDP packets shown while utilizing various parallel sections. Floodlight, Ryu, and the standard option (Linux bridge) outperform OpenDaylight in delay time and maximum data throughput. Nonetheless, Open-

Daylight demonstrated a high degree of self-healing, making it more reliable than Ryu and Floodlight.

In this research, at first, we build an integrated structure of OpenStack Cloud platform and Tungsten Fabric controller, in which, the controller is set up in a control node and able to manage other nodes in the OpenStack. Then, we evaluate the network and system performance of the OpenStack Cloud platform with and without the Tungsten Fabric controller. For networking, we focus on the ICMP, and TCP, UDP data streams generated between virtual machines in an OpenStack environment using the PING and IPERF tools, respectively. We assess several key performance metrics, including latency, and throughput, to gain insights into the impact of Tungsten Fabric on network performance. For system utilization, we use Prometheus and Grafana to monitor the system and evaluate the RAM and CPU of the system. By analyzing the collected metrics, we provide insights into the scalability and efficiency of Tungsten Fabric in managing network resources within an OpenStack environment. Our research will enable cloud administrators and network engineers to make informed decisions about adopting the Tungsten Fabric SDN controller in their OpenStack environments.

3 Basic Knowledge

OpenStack, a well-known open-source cloud computing platform, provides a comprehensive infrastructure-as-a-service (IaaS) solution. OpenStack, which was created to manage and orchestrate various cloud resources, gives enterprises the flexibility to build and manage private and public cloud environments. As shown in Figure 1, OpenStack architecture includes components such as Nova for compute resources, Swift for object storage, Neutron for networking, and so on. OpenStack's scalability, and flexibility have led to its widespread adoption in various sectors, enabling efficient cloud management and provisioning while promoting cooperative innovation within the current rapidly growing cloud computing landscape.

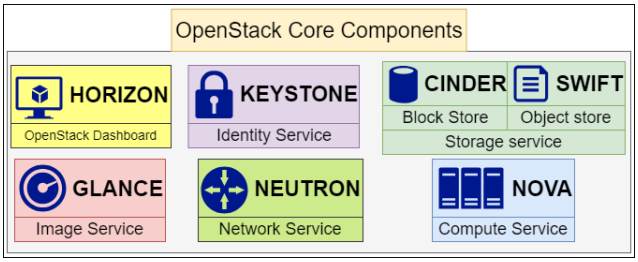

Figure 1. OpenStack components.

Tungsten Fabric is an open-source SDN solution that provides network virtualization and automation capabilities. Previously named OpenContrail, it is currently a component of the Linux Foundation's Networking project. As illustrated in Figure 2, Tungsten Fabric architecture can be divided into the control plane and data plane. Control plane components include Config, Control, and Analytics. Data plane components include vRouter and its agents. The Tungsten Fabric SDN controller uses various protocols and interfaces to communicate with network devices, such as switches, routers, and virtual network functions (VNFs). It leverages standard protocols like Border Gateway Protocol (BGP), OpenvSwitch Database (OVSDB), and Network Configuration Protocol (NETCONF) to exchange information and control the network elements.

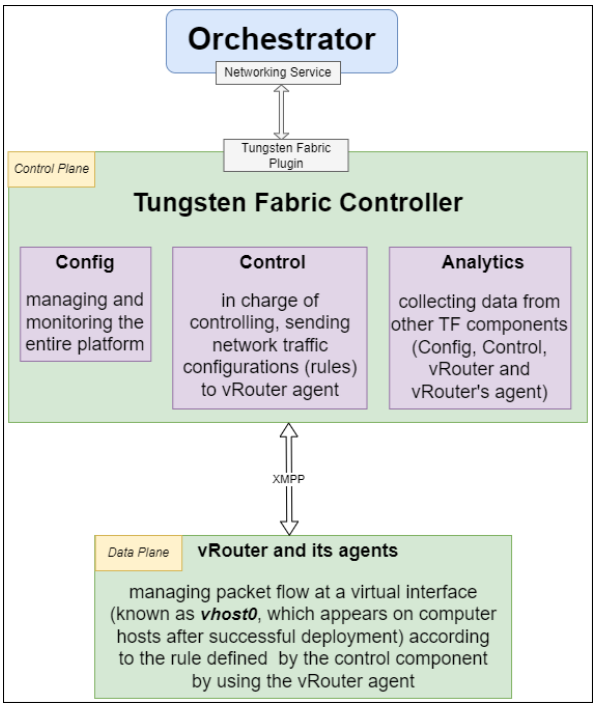

Figure 2. Tungsten Fabric controller architecture.

4 System Model

4.1 Networking in Standard OpenStack - Neutron

Figure 3 shows a network model in a standard OpenStack system. Neutron Server connects to OpenvSwitch utilizing the ML2 plugin (OpenvSwitch Mechanism Driver) and the Neutron OpenvSwitch Agent.

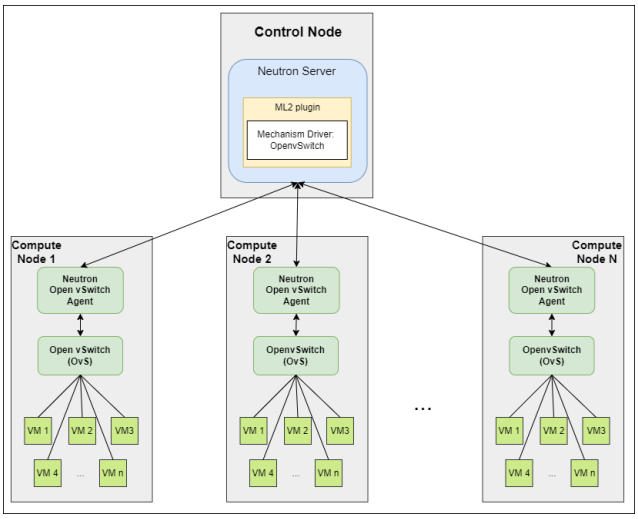

Figure 3. Networking in OpenStack system [21].

The OpenvSwitch driver creates a specific message and sends it to the Neutron OpenvSwitch agent on the specified Compute Node. The agent at the Compute Node interacts with the OpenvSwitch to program it according to the command received [21]. Last but not least, virtual machines are managed by OpenvSwitch. Figure 4 shows how OpenvSwitch (OvS) processes a packet between virtual machines in OpenStack.

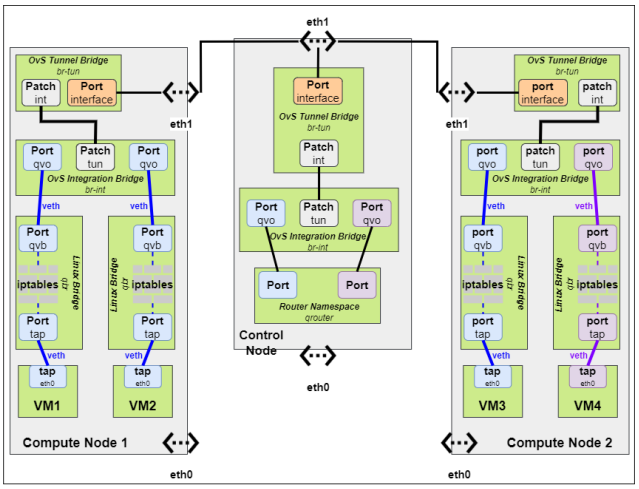

Figure 4. Packet transmitting between VMs using OpenvSwitch.

In the case of two VMs (virtual machines) sharing the same network range, if they are on the same node, the packet from VM1 to VM2 (both on Compute Node 1 and in the virtual network 1 range) proceeds as follows: The VM1 interface routes the packet to the Linux bridge port via *veth* pair. After that, the bridge handles the packet's firewalling and connection tracking before forwarding it (by port *qvb*) to the OvS integration bridge port *qvo* (on the left) via a *veth* pair. The OvS integration bridge port *qvo* (placed on the right) routes the packet to the Linux bridge port *qvb* via the *veth* pair. Finally, the bridge performs firewalling and connection tracking for the packet before forwarding it to the VM2 interface. On the other hand, if two VMs are on separate nodes, the packet from VM1 in Compute Node 1 to VM3 in Compute Node 2 (both with virtual network 1) will go as follows: Similarly, suppose two VMs are placed in the same node and network range, but when the OvS integration bridge gets the packet, it adds an internal VLAN tag to it and then swaps the internal VLAN tag for an internal tunnel ID. The OvS integration bridge *patch-tun* patch port routes the packet to the OvS tunnel bridge *patch-int* patch port, and the bridge then wraps the packet in VNI 1 (Virtual Network Infrastructure 1). The underlying interface for overlay networks routes the packet to the OvS tunnel bridge in the destination node (Compute Node 2) via the overlay network. The packet will be processed in reverse order by Compute Node 2 first and then sent to VM3 where it resides.

In the case of two VMs having different network ranges, if they are located in the same node, the packet from VM3 (which has virtual network 1) to VM4 (which has virtual network 2) (both on Compute Node 2) proceeds as follows: Because the packets must be carried over two separate network ranges, they must be routed to their intended destination. After VM3's packet has been processed by Compute Node 2, it will be transmitted to the Control Node. The packet is processed similarly by OvS components as in the Compute Nodes and routed to the network 1 interface in the router namespace. The router routes the packet to the next-hop IP address, which is usually the gateway IP address on network 2. The packet is routed to the OvS integration bridge port for network 2. The packet is then handled in the reverse process and sent to VM4. However, if these two VMs are on different nodes, the packet will be treated identically, but instead of being delivered to the same node, it will be routed to another node and processed in the same manner as VMs situated in the same node and in different network ranges.

4.2 Networking in OpenStack Integrated with Tungsten Fabric

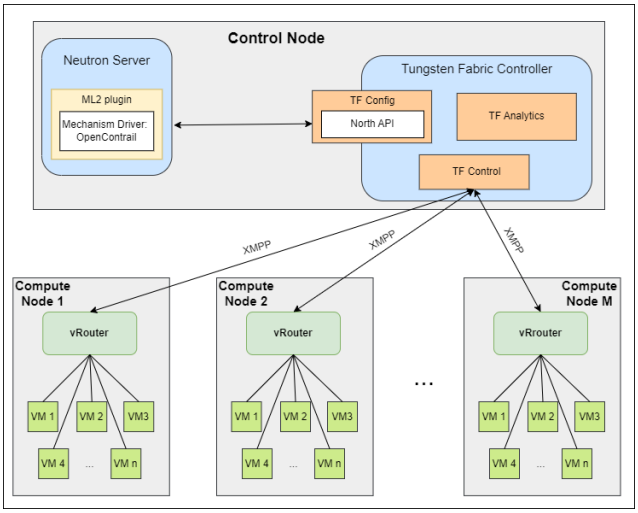

Figure 5. Networking in OpenStack combined Tungsten Fabric system.

In the integration system shown in Figure 5, two primary components connect the Neutron server to the Tungsten Fabric controller. The first is the ML2 plugin, which uses the OpenContrail Mechanism Driver. Secondly, North API, which not only is an agent of Tungsten Fabric namely Config, but also a crucial part that manages nearly all controller functions. Because it is used for communication in addition to the South API's existence (which is utilized by the Device Manager), the North API of the controller is extremely significant [22]. The controller manages all API calls sent to itself or sent to Neutron via this connection. Additionally, sending configuration or network rules to vRouters is the responsibility of the other Tungsten Fabric's component namely Control. They use XMPP, or Extended Messaging and Presence Protocol, to talk to one another [23]. Finally, virtual machines are managed by vRouters. Figure 6 shows how vRouter processes a packet between VMs in the integration system.

For an OpenStack system with the Tungsten Fabric controller, considering the case of two VMs (virtual

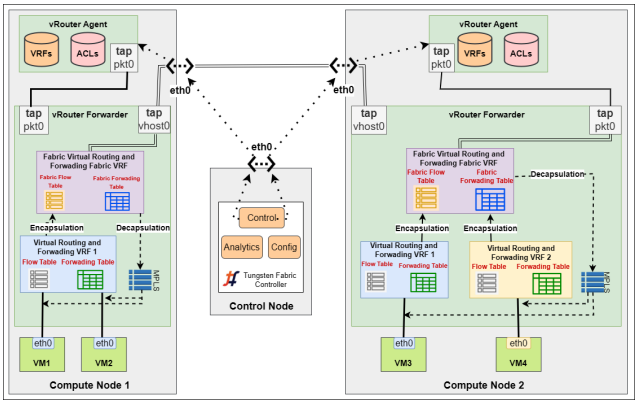

Figure 6. Packet transmitting between VMs using vRouter.

machines) placed in the same network and the same node here being Computing Node 1, we only need one VRF (Virtual Routing and Forwarding), which is VRF1 (VRF will be used when the virtual Network is created, so there will be as many virtual Networks as VRFs). When VM1 transmits a packet to VM2, if it is the first packet, it will look for VM2's IP address by making a DNS request to the DNS server received via DHCP and getting a response with the IP address. To get the MAC address, VM1 makes an ARP request to the vRouter Forwarder, which retrieves VM2's MAC address from its forwarding tables and encapsulates it in the Ethernet frame that will be sent to VM2. Once VM1 has VM2's IP and MAC addresses, further packets are routed continuously and seamlessly using the vRouter Forwarder methods. Following that, the forwarder decapsulates the packet and searches up the MPLS label to determine which virtual interface should deliver the original Ethernet frame. The Ethernet frame is sent across the interface and received by VM2.

If two VMs are on separate networks but on the same node (Compute Node 2), assume that VM3 uses VRF1 and VM2 uses VRF2. VM3 connects to the vRouter Forwarder, which serves as the default gateway. Initially, VM3 delivers messages enclosed in an Ethernet frame that includes the default gateway's MAC address. The vRouter Forwarder replies to an ARP request for the gateway IP with its own MAC address. VM3 then transmits packets with this gateway MAC, and the vRouter Forwarder uses the packet's destination IP to search its forwarding table in the VRF for the right route, which is usually via an encapsulation tunnel to the target host.

In both circumstances, whether sharing the same network or different networks, if the two VMs are on separate nodes, the vRouter Forwarder at the destination node will unwrap and handle the packet as described above. However, before being delivered to the target vRouter Forwarder, the packet must first pass via a virtual interface known as *vhost0*. It is noticeable that the vRouter agent maintains a connection with the controller and receives information about the VRFs, routes, and ACLs (Access Control Lists) it requires. The agent keeps the information in its database and configures the forwarder based on the data received from the Tungsten Fabric's element named Control.

5 Implementation and Results

5.1 Implementation

Following a successful deployment, we will test the network performance in this section and statistically examine how much RAM and CPU the server utilized to assess system performance. First and foremost, we subsequently generated numerous virtual machines on each Compute node, wherein these virtual machines operated on the Ubuntu 20.04 OS and were designated to particular network ranges. The deployment model of this study consists of three virtual servers rented from a cloud Service Provider, set up with one as a Control Node and two as Compute Nodes. These three virtual servers are connected to the virtual network, also called "Virtual Private Cloud" (provided by the Cloud Service Provider) via each virtual server's interface. One striking feature is that this virtual network limits the bandwidth between these virtual servers, specifically below 1000 Mbps.

5.1.1 Measuring network performance: Due to the fact that OpenStack's network and network services are resources and have been completely virtualized for ease of use and management; theoretically, they function exactly like actual networks, they have virtual ports, virtual routers, virtual local area networks, and other elements that are similar to real network systems. Consequently, we have to look at the network connectivity both within and between compute nodes in order to assess network performance between virtual machines.

In each scenario of network setup, two of the previously generated VMs (virtual machines) are selected at random, one entity functions in the role of the sender, while the other entity assumes the role of the receiver, aiming to assess the ability to transmit and receive data packets between them by considering factors like Throughput and Latency. In scenarios 1 and 2, the bandwidth is not limited, and the node will automatically handle the transmission and reception of packets between two VMs because they are located on the same node. But in scenarios 3 and 4, the bandwidth is limited to 1000 Mbps, leading to the bandwidth from node to router in these scenarios being limited to 1000 Mbps, moreover, only one node is responsible for sending the packet since it is the node where the VMs initiating the data transfer are located.

We utilize the IPERF tool to assess throughput across virtual machines for both TCP and UDP packets. IPERF's operational mechanism involves a single process without any parallel mode. It establishes a stream from the client to the server and assesses performance outcomes based on that stream. To get the best results, run the software for 5 minutes, rerun it 5 times, and average the results. In terms of latency, we utilize the PING tool to measure the transmission and reception timings of ICMP packets in 10 seconds, which is then repeated five times.

5.1.2 Measuring RAM and CPU utilization: As illustrated in Figure 7, our system consists of one Control Node and two Compute Nodes. The Control Node is equipped with 32 GB of RAM while each Compute

Node has 12 GB of RAM. Node-exporter is set up in Control Node and one Compute Node and they will export the system's data into Prometheus set up in a Ubuntu machine; additionally, Grafana is used to observe the data. During system deployment, we will assess the RAM and CPU utilization of the Control Node and one of the Compute Nodes (because the other Compute Node has the same specifications) when creating one virtual machine within the system and when conducting four different scenarios.

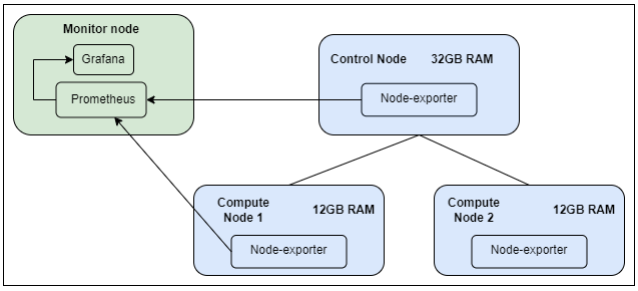

Figure 7. Monitoring system diagram.

5.1.3 Scenarios for evaluating: We propose four scenarios for testing network performance as follows:

Scenario 1: This test model is used to look into the latency and throughput between two virtual machines running on the same network node. Two virtual machines from Compute Node 1 that are situated within the address range 10.0.0.0/24 will be chosen for assessment.

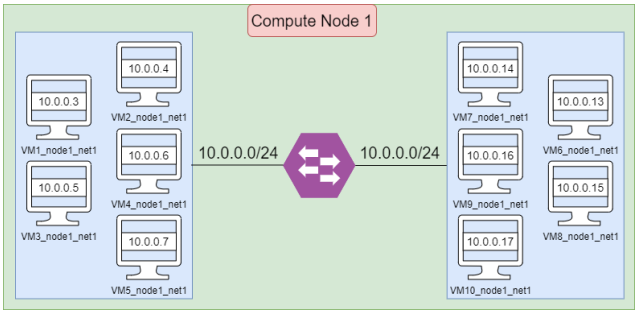

Figure 8. Scenario 1: Same node, same network.

Scenario 2: This test model is used to look into latency and throughput between two virtual machines that are situated on the same node but in two separate networks. Two virtual machines, 10.10.10.0/24 and 10.0.0.0/24, with different address ranges on the same Compute node 1 will be chosen for assessment.

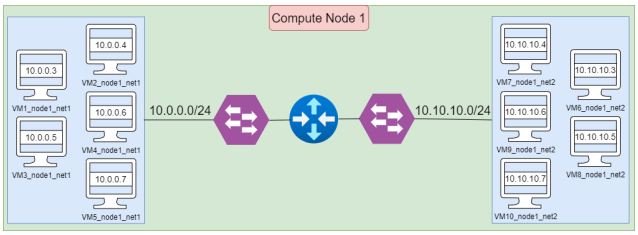

Figure 9. Scenario 2: Same node, different network.

Scenario 3: This test model is used to look into latency and throughput between two virtual machines that are situated at separate compute nodes but within the same network range. For evaluation, we will select any machine from Compute Node 1 and any machine from Compute Node 2. The network range of 10.0.0.0/24 is used by both machines.

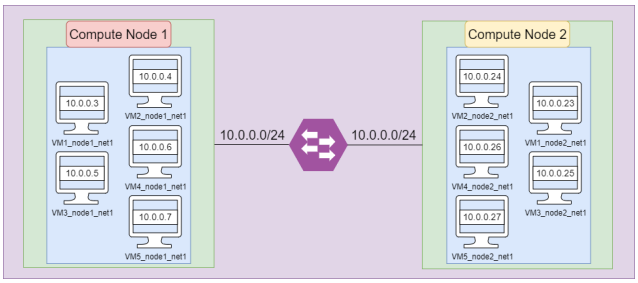

Figure 10. Scenario 3: Different node, same network.

Scenario 4: This test model is used to look into the latency and throughput between two virtual machines that exist across two distinct compute nodes and two distinct network ranges. For evaluation, we will choose any machine at Compute node 1, which is situated in the range 10.0.0.0/24, and any machine at Compute node 2, which is situated in the range 10.10.10.0/24.

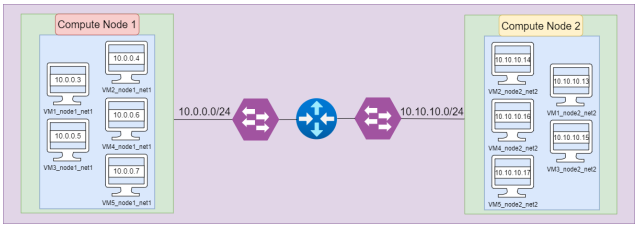

Figure 11. Scenario 4: Different node, different network.

5.2 Results

5.2.1 Network Performance: a) Throughput

TCP throughput: We can observe that when Tungsten Fabric is integrated with OpenStack, the TCP throughput is significantly higher than when it is not integrated (Figure 12) since the virtual machines are located on the same Compute Node.

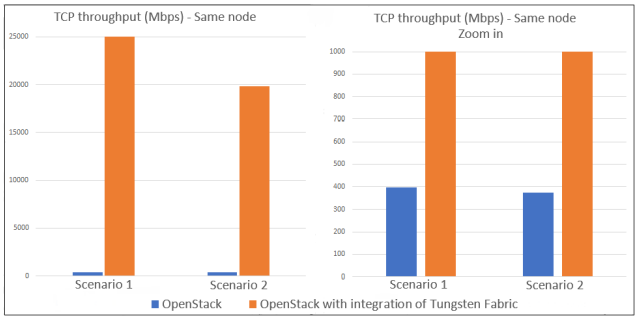

Figure 12. TCP throughput between virtual machines in the same node.

As previously indicated, we license three "virtual servers" for our implementation, and there is a 1000 Mbps bandwidth limitation between the servers. When

OpenStack and Tungsten Fabric are integrated, a very impressive 900 Mbps of TCP throughput is obtained; in contrast, when no integration occurs, results range from 200 Mbps to 300 Mbps (Figure 13). The Tungsten Fabric controller improves OpenStack's efficiency and network performance between virtual machines on different nodes.

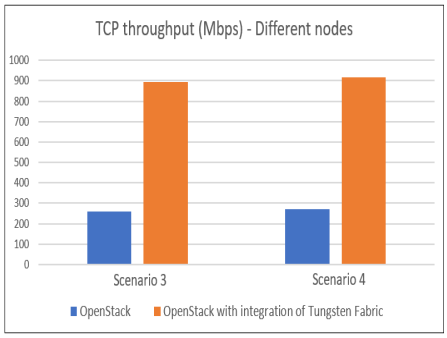

Figure 13. TCP throughput between virtual machines in different nodes.

Similar to TCP throughput, UDP throughput gained by integrated OpenStack with Tungsten Fabric is significantly higher than when not integrated (Figure 14 15). Although the UDP throughput of the integrated system is lower than that of TCP in all four scenarios, it still outperforms the standard OpenStack system significantly, even when the bandwidth is limited to 1000 Mbps in scenarios 3 and 4. These outstanding results highlight the significant potential of the Tungsten Fabric controller in addressing and mitigating the limitations imposed by bandwidth constraints. This controller has the capability to facilitate enhanced bandwidth utilization, enabling virtual machines to achieve the maximum level of throughput allowed by the underlying hardware. This holds true whether the virtual machines are located on the same node or separate nodes.

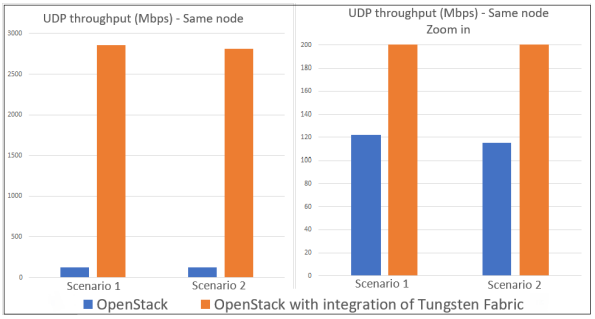

Figure 14. UDP throughput between VMs in the same node.

The obtained throughput results can be attributed to the limited bandwidth of the OpenStack version or its inability to utilize the maximum available bandwidth. We also conducted another test using a different version of OpenStack on the same operating system, CentOS 7.9. This OS version was previously used to deploy the OpenStack system for this research. The results showed that the throughput outcomes were almost identical. However, when deploying a newer OpenStack version

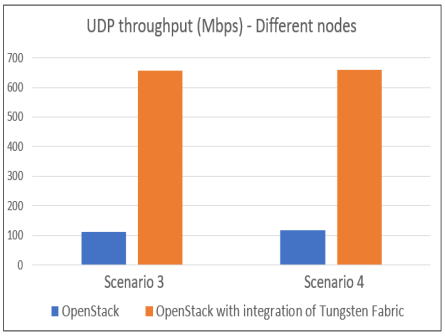

Figure 15. UDP throughput between VMs in different nodes.

in a more recent operating system such as Ubuntu 20.04, the obtained TCP throughput is significantly higher and can reach tens of Gbps, which is almost identical to the throughput when integrating Tungsten Fabric into OpenStack (running on CentOS7.9), but still slightly lower. However, Tungsten Fabric cannot be installed on Ubuntu 20.04 due to its inability to support Python 3 at the time of conducting this research. It is clear that users continue to use and favor the OpenStack version that is installed on the CentOS 7.9 operating system due to its continued effectiveness. Moreover, the throughput is influenced by the packet processing performed by OvS in conventional OpenStack and by the vRouter when Tungsten Fabric is integrated into OpenStack. The large amount of virtual interfaces that packets travel during processing by OvS has a substantial impact on the transmission and reception of packets, resulting in a significant decrease in network performance in term of throughput between virtual machines. As a consequence, the throughput achieved in the standard OpenStack system is lower than the throughput in the OpenStack integrated Tungsten Fabric system.

b) Latency

As shown in Figure 16, in all scenarios, the standard OpenStack system exhibits significantly higher latency compared to the integrated system. Moreover, when not combining OpenStack with Tungsten Fabric, the latency transmitting ICMP packets between virtual machines has a greater impact when they are situated in the same or different network ranges than when they are located in the same or different compute nodes (the latency of the standard system varies across the four cases).

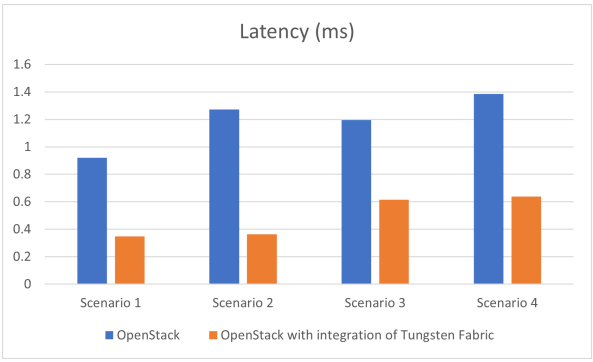

Figure 16. Latency.

However, the controller can help us resolve this issue. In the integrated system, the results obtained when comparing scenarios 1 with scenario 2 and scenarios 3 with scenario 4 are not significantly different. This means that regardless of whether the virtual machines are on the same virtual network or not, the latency remains unaffected, the delay only increases when the VMs are located on different compute nodes.

When a packet is processed by OvS to go between virtual machines, it must transit many virtual interfaces, such as virtual ports or taps. However, with vRouter, the packet does not need to pass through several virtual interfaces. There is just one virtual interface between the vRouter Forwarder and the server's primary network card. The performance was considerably affected due to excessive traffic traveling via several imperceptible interfaces. This explains why the integrated system has lower latency and more efficient utilization of network resources compared to standard OpenStack.

5.2.2 System Utilization: We will show the RAM and CPU utilization of the OpenStack Cloud system with and without the Tungsten Fabric controller when we deploy the system or create a VM (virtual machine), or conduct 4 scenarios. It is important to note that as we create more VMs, the system utilization increases exponentially. Additionally, during these scenarios, only the CPU usage of the Compute Node varies, therefore, we will focus on presenting the CPU utilization of the Compute Node in these cases. Finally, all conditions when comparing two systems are the same.

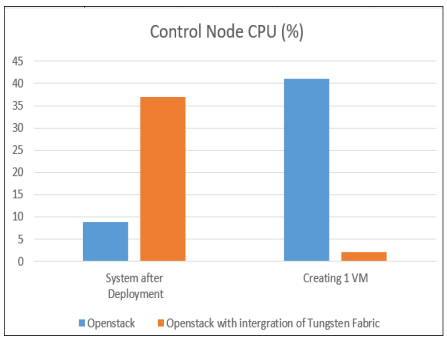

Figure 17. CPU utilization in Control Node.

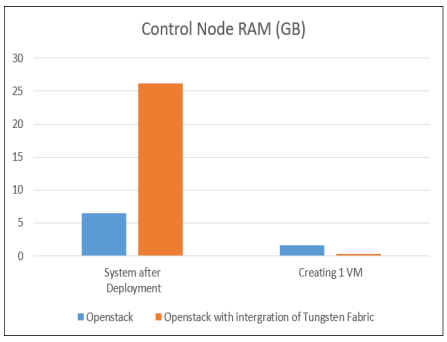

Figure 18. RAM utilization in Control Node.

As shown in Figure 17 and Figure 18, in the Control node, the Tungsten Fabric (TF) controller exhibits a significant impact on system performance upon installation, surpassing that of a typical OpenStack deployment. Before installation, the system CPU was at 0% but after installation, it was kept at a constant level of 9% for normal OpenStack and 37% for integration with Tungsten Fabric controller. This effect is especially evident in RAM usage as the system reaches 26 GB when integrated with TF compared to 6.5 GB for standard OpenStack. But when we created a VM, the integrated system consumed significantly fewer system resources in terms of RAM and CPU compared to the standard OpenStack system (2% CPU and 0.3 GB RAM compared with 41% CPU and 1.6 GB RAM).

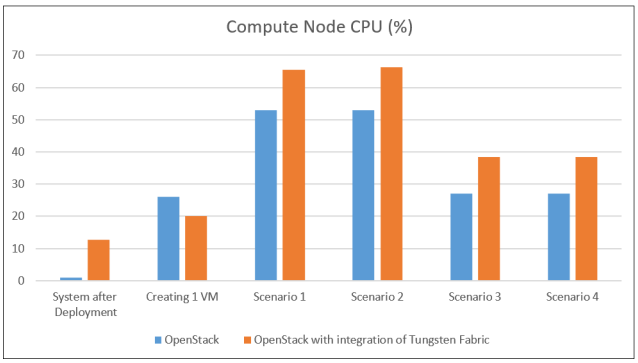

Figure 19. CPU utilization in Compute Node.

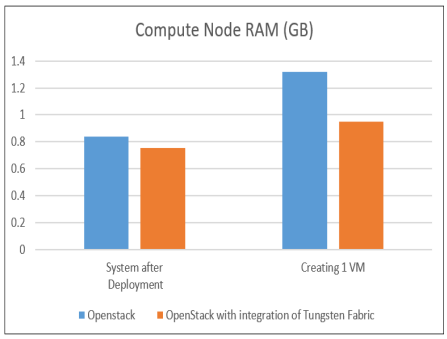

Figure 20. RAM utilization in Compute Node.

Figure 19 and 20 show the system performance in the Compute Node between the standard OpenStack system and the integrated system. After deployment, the conventional OpenStack system typically consumes minimal RAM performance, while the integrated system uses 13%. However, when creating a virtual machine, the regular system consumes more RAM than the OpenStack integrated with Tungsten Fabric system (26% compared to 20%). While conducting four scenarios, the Tungsten Fabric-integrated system's compute node CPU is about 10 to 15% higher than standard OpenStack. This can be explained by the outperforming network throughput of the integrated system; in scenarios 3 and 4, because the bandwidth is reduced to 1000 Mbps, this also causes the CPU usage performance of both systems to decrease compared to scenarios 1 and 2. In terms of RAM usage performance, after the successful system deployment, contrary to the control node, the regular OpenStack system uses slightly more RAM (around 20%) than the integrated system. When

creating a virtual machine, the regular system also consumes more RAM (around 37%) than TF-OpenStack (1.3 GB compared to 0.95 GB).

Following the system performance result, we can observe that the integration of OpenStack with the Tungsten Fabric controller requires more system resources in terms of RAM and CPU to set up compared to standard OpenStack. However, during normal operations such as VM creation, VM launch, and so on, the integrated system gives better speed but lower system resources than the standard OpenStack setup. Only in four scenarios, the system performance of Tungsten Fabric is higher, it can be attributed to the superior network performance of OpenStack integrated with Tungsten Fabric when compared to a standard OpenStack environment. It is essential to note that when we create VMs, the system's capacity grows exponentially with the addition of more VMs so Tungsten Fabric enables improved management of systems with a larger number of VMs.

6 Conclusion

In conclusion, our evaluation of Tungsten Fabric has demonstrated its superior network performance when integrating into OpenStack compared to conventional OpenStack deployments. It has shown remarkable network efficiency, delivering enhanced speed and reliability. However, it is worth noting that the deployment of the controller with OpenStack requires a lot of RAM consumption. Despite this drawback, Tungsten Fabric's outstanding network performance makes it a promising choice for organizations seeking improved networking capabilities in their OpenStack infrastructure. Further research and optimization are needed to address the RAM utilization issue during the integration and unlock the full potential of this powerful networking solution.

References

[1] D. Grzonka, "The Analysis of OpenStack Cloud Computing Platform: Features and Performance," *Journal of telecommunication and information technology*, pp. 52–57, 2015.

[2] I. Pavle and R. Harald, "OpenStack Cloud Tuning for High Performance Computing," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing and Big Data Analysis*, 2018.

[3] R. Tiago and B. Jorge, "An Overview of Openstack Architecture," in *Proceedings of the International Database Engineering and Applications Symposium - IDEAS'14*, 2014.

[4] N. Alexander, A. Joseph, I. Md Zahidul, and R. Pablo. (2023) Software defined networking: An overview. [Online]. Available: scribd.com/document/632304753/2302-00165

[5] M. Arienzo, M. Napolitano, and P. Romano, S, "Controller Placement Problem Resiliency Evaluation in SDN-based Architectures," *International Journal of Computer Networks and Communications (IJCNC)*, vol. 14, no. 5, 2022.

[6] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," in *Proceedings of the IEEE Communications Surveys and Tutorials*, vol. 17, 2015, pp. 27–51.

[7] Tungsten fabric architecture. [Online]. Available: https://tungstenfabric.github.io/website/Tungsten-Fabric-Architecture.html

[8] S. Rinki and R. Harshavardhan, "Effect of Load Balancer on Software-Defined Networking (SDN) based Cloud," in *Proceedings of the IEEE 16th India Council International Conference (INDICON)*, 2019.

[9] S. Parminder and M. Selvakumar, "Design and deployment of OpenStack-SDN based test-bed for EDoS," in *Proceedings of the 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)*, 2015.

[10] T. Olena, J. S. Mohammed, and R. Y. Abdulghafoor, "An analysis of SDN-OpenStack integration," in *Proceedings of the Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, 2015.

[11] N. V. Vo, M. C. Le, Q. L. Nguyen, and D. N. Le, "Network Virtualization Performance Analysis On OpenStack Cloud Computing Infrastructures," *Da Lat's Special Issue On Informantion Technology*, vol. 6, no. 2, 2016.

[12] T. A. Bui, "Cloud Network Performance Analysis: An OpenStack Case Study," in *Master Thesis, Université Catholique de Louvain*, 2016.

[13] S. Arsalan and M. Tahir, "Performance Evaluation of OpenStack Networking Technologies," in *Proceedings of International Conference on Engineering and Emerging Technologies (ICEET)*, 2019.

[14] G. Nithya, A. Shubham, and B. Thangaraju, "Performance Analysis of SDN controllers within an OpenStack infrastructure," in *Proceedings of IEEE India Council International Subsections Conference (INDISCON)*, 2022.

[15] H. E. Arwa, A. H. Hassan, and H. E. Hala, "Performance Analysis of ONOS and Floodlight SDN Controllers based on TCP and UDP Traffic," in *Proceedings of the International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2019.

[16] K. C. Ravindra, A. Mithilesh, and K. N. Naresh, "Performance Comparison of Ryu and Floodlight Controllers in Different SDN Topologies," in *Proceedings of the 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE)*, 2019.

[17] B. D. Maheshi, A. L. V. Kumari, and U. K. A. Udunuwara, "Performance Comparison Of Onos And Odl Controllers In Software Defined Networks Under Different Network Topologies," *Journal Of Research Technology & Engineering*, no. 3, pp. 94–105, 2021.

[18] C. Lei, X. Ming, and L. Jian, "Monitoring System of Openstack Cloud Platform Based on Prometheus," in *Proceedings of the International Conference on Computer Vision, Image and Deep learning (CVIDL)*, 2020.

[19] M. Priyatosh, J. Rekha, and S. Vaibhav, "Performance Analysis of OpenStack Controller," in *Proceedings of the International Conference on Smart Applications, Communications and Networking (SmartNets)*, 2019.

[20] T. Olena, J. S. Mohammed, and R. Y. Abdulghafoor, "An analysis of SDN-OpenStack integration," in *Proceedings of the Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, 2015.

[21] S. Sriram and V. Sreenivas, *Software-Defined Networking (SDN) with OpenStack (pp 75)*. Birmingham, UK: Packt Publishing, pp. 75, 2016.

[22] K. Szymon, M. Paweł, S. Piotr, Z. Paweł, and D. Łukasz. (2021) Tungsten fabric architecture. [Online]. Available: https://codilime.com/blog/tungsten-fabric-architecture-an-overview/

[23] Tungstenfabric. (2021) Tungstenfabric architecture — an overview. [Online]. Available: https://tungsten.io/tungsten-fabric-architecture-an-overview/

Hoang Thanh Loi is a Bachelor in Electronics and Telecommunications Engineering at VNU University of Engineering and Technology. He is a member of the Telecommunication Systems Laboratory at Faculty of Electronics and Telecommunications, VNU University of Engineering and Technology. His research interests include Software Defined Networks, System Monitoring, and Cloud Computing.

Pham Dinh Quy is a Bachelor in Electronics and Telecommunications Engineering at VNU University of Engineering and Technology. He is a member of the Telecommunication Systems Laboratory at Faculty of Electronics and Telecommunications, VNU University of Engineering and Technology. His research interests include Software Defined Networks, System Monitoring, and Cloud Computing.

Nguyen Nang Duc is a Bachelor in Electronics and Telecommunications Engineering at VNU University of Engineering and Technology. He is a member of the Telecommunication Systems Laboratory at Faculty of Electronics and Telecommunications, VNU University of Engineering and Technology. His research interests include Software Defined Networks, System Monitoring, and Cloud Computing.

Assoc. Prof. Dinh Thi Thai Mai received M.Sc. degree from University of Paris Sud 11, France, and PhD. degree from VNU University of Engineering and Technology, Hanoi, Vietnam in 2009 and 2017, respectively. She is Deputy Head of the Department of Telecommunications Systems, Faculty of Electronics and Telecommunications, VNU University of Engineering and Technology, Hanoi, Vietnam. Currently, her research interests include 5G/6G mobile networks, wireless communications, localization techniques, SDN/NFV and network security.