Journal of Computer Networks, 2018, Vol. 5, No. 1, 1-7
Available online at http://pubs.sciepub.com/jcn/5/1/1

DOI:10.12691/jcn-5-1-1

Evaluation of Virtual Operating Systems and Switches under Ring, Star and Tree Topologies for Streaming Large Data Sets

Logan R. Peterson, Prakash Ranganathan*

University of North Dakota, United States
*Corresponding author: prakash.ranganathan@engr.und.edu

Abstract This paper discusses architectural performance of virtual operating switches, and systems under various topological structures to handle packet deliveries of large streaming and dynamic data sets. The topology structures studied include: tree, fully connected mesh, and star. A virtual network framework is developed that can generate any number of nodes with re-configurable parameters. An evaluation on memory requirements, disk-space, and CPU usage on three different virtual network switches (openvSwitch, openSwitch, openWRT) were also investigated. Two specific protocols–Routing Information Protocol (RIP) and Open Shortest Path First (OSPF) are tested against bandwidth requirements.

Keywords: *virtual networks, architectural performance, openSwitch*

Cite This Article: Logan R. Peterson, and Prakash Ranganathan, "Evaluation of Virtual Operating Systems and Switches under Ring, Star and Tree Topologies for Streaming Large Data Sets." *Journal of Computer Networks*, vol. 5, no. 1 (2018): 1-7. doi: 10.12691/jcn-5-1-1.

1. Introduction

When creating a large test network, a physical system would be the most ideal in that it would more closely mimic a real life structure. A physical system for a large network, though, would be very time consuming to configure as well as being exceedingly expensive. Thus, a virtual system was chosen as it allows the creation and configuring of the network to be automated. The software used to host a virtual machine is called a hypervisor. Two hypervisors were used in this paper, Oracle VM VirtualBox and VMware vSphere ESXi. Oracle VM VirtualBox, herein referred to as just VirtualBox, is an open source and free to use hypervisor that is supported on a number of different operating systems as well as containing support for the virtual machines to be tested. VirtualBox was the preferred for use due to its free to use policies and its support for all the major operating systems, therefore, making this paper largely cross platform [1]. Due to the large resource use of the paper, however, VMware vSphere ESXi was used on a high performance computer for which it was designed for [2].

Three virtual switch OS's were used: Open vSwitch, OpenSwitch and OpenWRT. These operating systems were chosen due to being open source, free to use, and their large market share. An open source allows a user to modify and compile the operating system to their desire. This can become useful for removing unneeded applications and other files to reduce space. Some virtual switch operating systems on the market require a paid subscription for updates to the system. By being free to use allows the use of the operating system without having to make a single purchase or a subscription for the OS. The strongest reason that these switch operating systems were chosen was because of their relatively large market use. Being one of the more popular operating systems leads to having updates that are more frequent and security patches as well as having a large community for help on the internet forums. The three operating systems are very similar in design. Each can be installed directly on a physical commercial router or network switch. Each runs their own modified version of Linux that is tailored with different applications to reach the same goals. Though the operating systems are all based on flavors of Linux, each requires a different hardware specification to run the operating system as intended. The largest contributors to the hardware specifications are the file size and memory requirements.

2. Virtual Operating System- OpenSwitch

OpenSwitch is one of the newest of all the virtual switch operating systems. It was released by Hewlett Packard in 2015 to rival the closed source operating systems in the market. OpenSwitch was chosen as one of the test operating systems due to its relatively new release as well as the frequent feature updates and security patches. Though OpenSwitch states not to be intended for virtual machines, OpenSwitch provides a Virtual Machine Appliance of all its releases that can be imported into virtual machine hypervisor [3].

Open vSwitch was trialed next. Perhaps the most popular open source virtual switch operating system, it was first released in 2009 and later acquired by VMware

[4]. Open vSwitch is unique because it is a suite of tools and not an actual operating system itself. Open vSwitch provides a release for installation on most Unix and Linux based operating systems. This makes Open vSwitch useful for the reason that it can be installed on most systems that are already configured. Open vSwitch, in addition to the tool suite, provides a Linux operating system Virtual Machine Appliance image that comes preinstalled with the latest version of the tools. This makes Open vSwitch unique in that it is not meant to be installed on physical switches like many of the other available operating systems. The virtual image was used for the following tests.

OpenWrt was the final virtual switch operating system chosen. It is an embedded operating system that is founded on Linux. First released in 2004, OpenWrt was intended to be an alternative firmware for Linksys routers. [5] By focusing on being an embedded operating system means that the firmware was optimized for low resource usage, including size, CPU, and memory usage. OpenWrt is the least flexible of the virtual switch operating systems chosen, due to being a bare minimum flavor of Linux. The update procedure for OpenWrt involves compiling the newest release and reflashing the firmware onto the device. For a virtual operating system perspective that means for every update the virtual machine has to be reimported into the hypervisor and the configuration changed to the preferred settings. This changing of the configuration is the most time consuming of the process and, therefore, the OpenWrt virtual machine was not kept updated with the current releases during this paper to save time.

For this paper, each virtual switch operating system was configured in similar ways. Many of the default options that would be changed for commercial use, such as the default usernames and lack of passwords, were left as default to make them easier for following usage guides and that they did not negatively impact the goal of this paper in creating a virtual network. Local management was permitted for each node by having HTTP and SSH management enabled. This allowed the configuration of each node to be independently changed by either a web interface through the HTTP or by command line using SSH protocol. Web interface management was important in the early stage of this paper in that it allows a visualization of which settings can be modified and enabled. The web interface allows changes to be made to the system using only the web browser. A SSH interface is far more important to the paper because it allows command line instructions to be sent to a node. Where a web interface can only make changes based via mouse clicks, the command line interface allows instructions to be sent by keyboard only. This makes modifying a large number of nodes much quicker in that many changes can be made with a few keyboard commands. To reach the goal of this paper by creating a network with many nodes, each node has to be independently configured and SSH management allows this task.

3. Automated Node Modules & Topological Structures

The creation of the network structure was explored in two ways. The first, creating many network nodes and then modifying them for each structure type that was chosen on run. The second, by using a single node and cloning multiple times to create the intended structure using node modules in Figure 1.

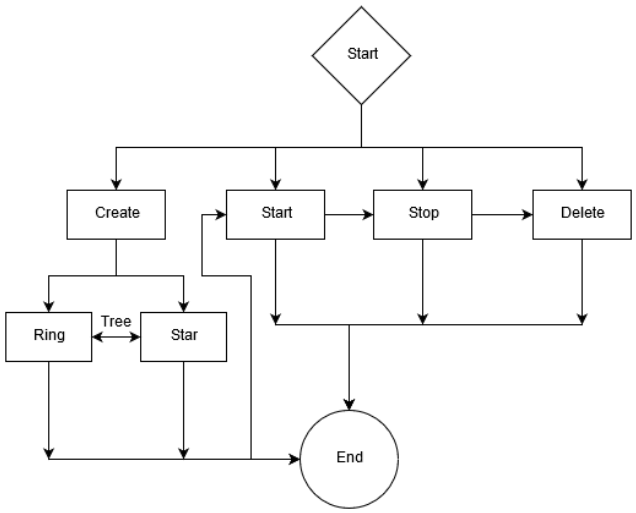

Figure 1. Node Modules in Architecture Hierarchy

Each method has its own pros and cons. The first method having an advantage in time-savings by only having to edit connections of each node to the desired network structure, and a disadvantage of each node would have to be updated or recreated should the operating system change. The second method of using one source node has the opposite pros and cons of the first method. A pro of having to only update or change a single node, but a con of having to re-modify every setting of node in the network is a challenge. The second method was ultimately chosen because it allowed for the operating system to be changed or updated by only having to change the source node. This ultimately saves considerable time when needing to make changes to the entire structure.

The process of creating a network was to start with a single node that will be cloned. This node will further be referenced as the "source node". The source node is a preconfigured and imported node in the hypervisor. Any settings made to the main node will be kept in all child nodes unless they are individually modified. By using a single node for the main structure of the network and then cloning it, allows the entire system to be created from the ground up for each chosen network structure. To set up the large number of nodes required for the system, a way to automate the development of nodes in VMs were developed. There are two ways that nodes can be cloned and set up in hypervisors. Like the configuration access to virtual network switches, hypervisors have a graphics user interface and a command line interface. The command line interface allows the creation and modifying of nodes that would take many clicks in the graphical user interface. The command line interface was chosen because it is the easiest method for automating the creation of the system.

The set up was written in Python language. The automation script was written to complete four main functions on the system: create, start, stop, and delete. The create function of the script is the most complex. Proper input into how the system is to be set up is first hard coded into the script. These settings include the number of nodes in the system, which node to clone, then type of network

structure to build, as well as other various options. The start, stop, and deletion functions are relatively straightforward. These functions take only the number of nodes to complete the task on as input. The start function launches a virtual machine one at a time, checking if the hypervisor has enough resources to start another node. Should the hypervisor run low on memory, the script will stop earlier preventing a lockup of the hypervisor that would otherwise require a full system restart. The stop and delete functions simply stop or delete the specified number of nodes respectively.

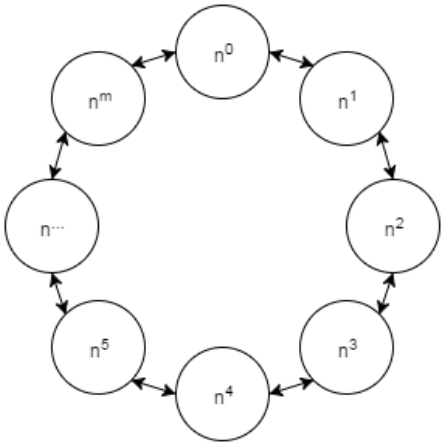

Figure 2. Ring Topology

For the node network connection process three network topologies were programmed to be automated. The chosen topology was first laid out in an array type data structure that was later transcribed to the creation process. The first is a ring topology. A ring topology consists of every node having two connections forming a single chain. Network packets that travel through a ring topology has to travel to each node in between two chosen nodes as seen in Figure 2. Using m max nodes ring topology was automated by having the n node connected to n^{n+1} and n^{n-1} node where the final node *m* was connected to the nodes n^1 and n^{m-1}.

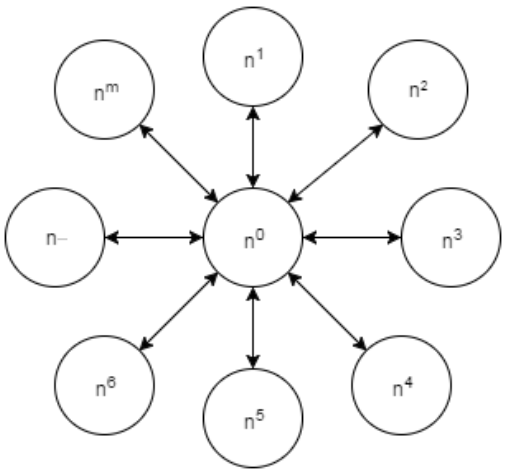

Figure 3. Star Topology

A star topology consists of every node connected to a single central node shown in Figure 3. For *m* number of nodes this topology will have the central node with *m*-1 connections and all child nodes containing one connection to the central node. The number of nodes in a star topology was limited in this paper due to the limited number of connections by the hypervisor and virtual operating systems.

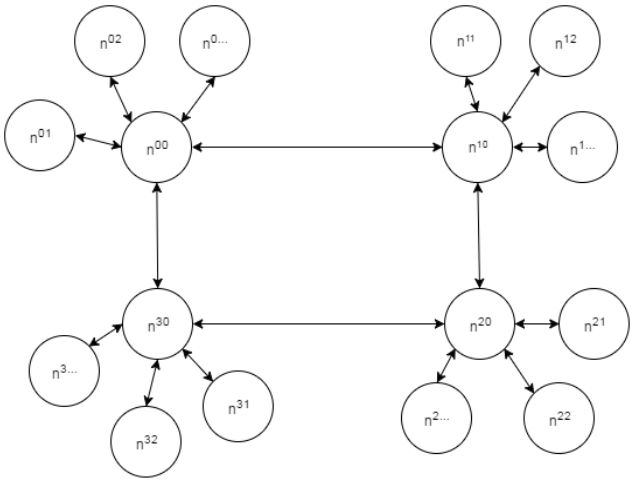

Figure 4. Tree Topology

The tree topology consists of a mixture of the ring and star topology. Automation was achieved by first creating a ring topology and then passing the node list into the star topology function to populate each node on the ring show in Figure 4. Though there is the same limit to the max number of nodes per star as the star automation, an infinite amount of star networks can be connected through the ring topology.

4. Network Routing-Types and Protocols

In computer networking, there are two main ways to route traffic packets, static and dynamic routing. Static routing is the manual method of creating routing tables. This allows a network administrator to specify exactly the destination node, where he wants the data packets to reach. If the network changes when using static routing, then the routing tables must be changed. Therefore, static routing is not ideal for networks that are complex, very large or change often [7].

To counter the issues with static routing, dynamic routing offers advantages. Dynamic routing is the process of automatically creating a routing table based on a routing protocol. The dynamic routing protocols can then monitor, and track for any changes within network. If need be, adjust the routing tables accordingly. Dynamic routing protocols use policy-based routing (PBR) when making routing decisions. The policies include size of the packet, protocol type, round trip time, bandwidth, and reliability.

Within dynamic routing protocols, there exist two main sub forms on how updating process of a routing table should be carried. The first being link-state routing protocols. The link-state protocol uses every node in the network to create a map. Then, every node on the network uses this map independently to calculate the best possible path for a packet destination. To accomplish this task, the entire network needs to be mapped out and sent to every

individual node. This can impact a severe performance cost, when the network contains a large amount of nodes [8].

The second form of dynamic routing protocols is the distance-vector routing protocol. In this method, a node does not know the path, where the data will be routed. Instead, nodes base their routing on the direction a packet, and the distance to the destination. This protocol has a better performance cost than link-state protocol, when updating the routing tables, because nodes only have to map a distance for one node hop away. However, the distance-vector routing protocol does have a performance cost, when sending packets because a node doesn't know the entire path of a network. A packet will be sent to multiple nodes causing increased congestion and usage within the network [7].

Furthermore, there are multiple other schemes for routing. These schemes contrast on how they send data. These routing schemes include unicast, broadcast, multicast, anycast, and geocast. Unicast will send data to one specified node; broadcast sends data to every node in the system; multicast sends node to a specified group of nodes; anycast will send data to a group of nodes based on a specified policy; and geocast sends data to a large geographic location.

The most common routing protocols include or Routing Information Protocol (RIP), Intermediate System to Intermediate System (IS-IS), Open Shortest Path First (OSPF), and Border Gateway Protocol (BGP). RIP is one of the oldest routing protocols. RIP is a distance-vector routing protocol that uses broadcast or multicast to update its nodes. RIP is one of the most common protocols used, as it do not require any configuration set-up phase. IS-IS use multicast to update nodes. IS-IS does not use IP for routing. OSPF is another link-state protocol that uses multicast like IS-IS. However, unlike IS-IS, OSPF was developed for use with the internet protocol (IP). OSPF is unique in that it does not use a transport protocol, but instead, uses its own IP number for transmitting data. BGP, or Border Gateway Protocol, is an exterior gateway protocol. This type of protocol is used for connecting multiple networks on the internet [8].

To test for bandwidth constraints, four network topologies were used: tree, ring, mesh and a hybrid structure that uses tree and ring. Two virtual test machines were used to communicate across the network topology. The operating system Ubuntu, a Linux-based operating system, was chosen to be installed on the virtual test machines for this work, because of its open source use, and its compatibility with applications that ran on the Linux-based OpenWrt nodes [9]. The Ubuntu flavor is a 32x Ubuntu Desktop version 16.0. A 32-bit operating system was chosen because of its smaller disk size requirements as well as its lower use of random-access memory compared to the 64-bit version. The version 16.04 is the latest stable release of the Ubuntu Desktop family and currently contains the most technical support for all prior versions.

The two Ubuntu virtual machines (VM) were created by installing the operating system to a single virtual machine, and then cloning the machine. One VM was chosen as the source machine, and the other the destination machine. The source VM begins the communication to the destination virtual machine through the network. The source VM is referenced as Ubuntu A, while the destination VM as Ubuntu B. Each virtual machine was virtually connected at a predetermined location in the node network topology, and its location can be varied at a later time.

Network bandwidth can be controlled individually through the network switch nodes by use of Quality of Service (QoS). Quality of Service settings allows a network administrator to limit the bandwidth of individual switches as well as give priority to certain packets or packet sources. By using Quality of Service, a connection to a switch can be modified to allow a specified limit of bandwidth through. This allows nodes to be given certain use priority by the different routing methods tested for this work.

To test for congestion, the VM connections, multiple programs were used to transmit data from Ubuntu A to Ubuntu B and back. Iperf was used as the primary testing application. Iperf is a tool that was written specifically for testing bandwidth [10]. The application was selected for this work, because it contains many parameters for the type of data being streamed. Some of these parameters include specifying packet size, and streaming random or inputted data. Iperf is run as a server or/and client configuration, where each machine can measure bandwidth directionally through a network topology. The default settings of an Iperf test were used for benchmarking this work. This consists of using the TCP port 5001 with a window size of 85.3 Kbytes [11]. Random streaming data was generated by the Iperf to be sent between the client and server. Three to four Iperf tests were run to determine the average bandwidth usage by a system.

Latency of the virtual network was measured using the command-line tool 'ping'. Ping works by sending a ICMP packet format to target machine and receiving an 'echo' packet back from the target containing a report of the connection. The ping application analyzes the multiple echoes and determines the packet loss of the network and the minimum, average, and maximum latency of a round trip connection. For this work, the default ping settings included using a 32-byte packet and a one-second packet timeout before it is considered as a 'loss' [12]. The tests were allowed to be run for twenty counts before finding the average round-trip latency of the system.

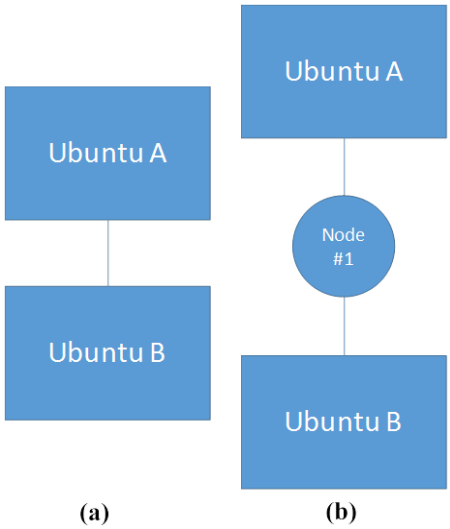

Figure 5. (a) Benchmark 1, (b) Benchmark 2

The first benchmark of the network was to directly connect the client to the server, Ubuntu A to Ubuntu B. This was done to determine the maximum throughput of the network before being applied to a network topology. When directly connecting two virtual machines, the network bandwidth is limited by the hypervisor and the CPU utilization. As seen in Figure 5a, the maximum throughput is roughly 2.5 Gbits/s. The average latency between the two virtual machines measured at 0.205 ms. From these results, it can be concluded that any network topology tested in this project will theoretically not be able to exceed this maximum bandwidth and latency measured from a direct connection.

The second benchmark for the system was to connect the two virtual machines through an intermediary node to measure the performance reduction by using a single switch. Using one network node (e.g., see Figure 5b-node #1), Ubuntu A was connected to the outside network interface while Ubuntu B was connected to the inside interface of the switch node. The switch node was configured to use a static routing method allowing all traffic to be passed through with limited decision-making. Iperf was then used again to measure the bandwidth from Ubuntu A to Ubuntu B. The maximum throughput of the system was measured to be roughly 2.5 Gbits/s, nearly the same as the first benchmark. A ping test was then conducted and found the average latency to be 0.321 ms. The results of this test show that using fewer multiple nodes in the system will not noticeably affect the average bandwidth of the network. However, the results from the latency test reveal that at least an average of 0.116 ms of latency will be acquired for each switch traversed in a network topology.

The results of two benchmarks indicate that the system is transferring data at an incredibly high rate. The average latency on a physical network will average anywhere from 1ms to 4ms through a single switch. These speeds, however, are not unexpected, when using high quality equipment and cabling. These benchmarks show that this virtual network is able to mimic a physical network.

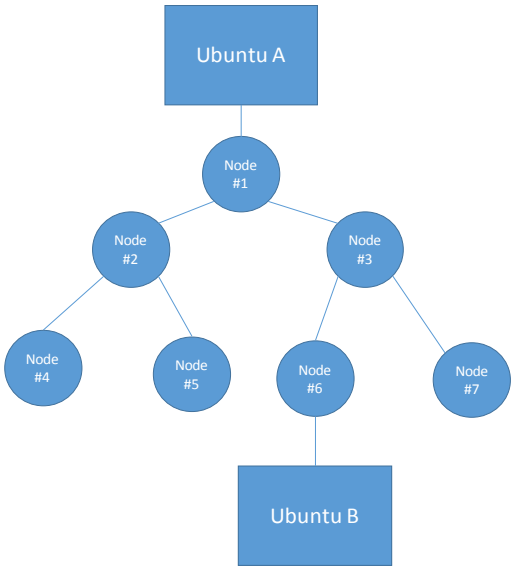

Figure 6. Ubuntu's in Tree Topology

The first network topology tested was a tree topology see in in Figure 6. Using seven virtual switch nodes, the tree topology was created by connecting two other nodes to the node above it. Each switch was set up to use the RIP protocol. This routing protocol will continually create a routing table by broadcasting to all routers within a certain number of hops. The server virtual machine, Ubuntu B, was connected under the sixth node on the third level of the tree. The client virtual machine, Ubuntu A, was connected above the first node on the first level of the tree. When communicating from Ubuntu A to Ubuntu B, a packet must traverse the nodes in a minimum of three hops.

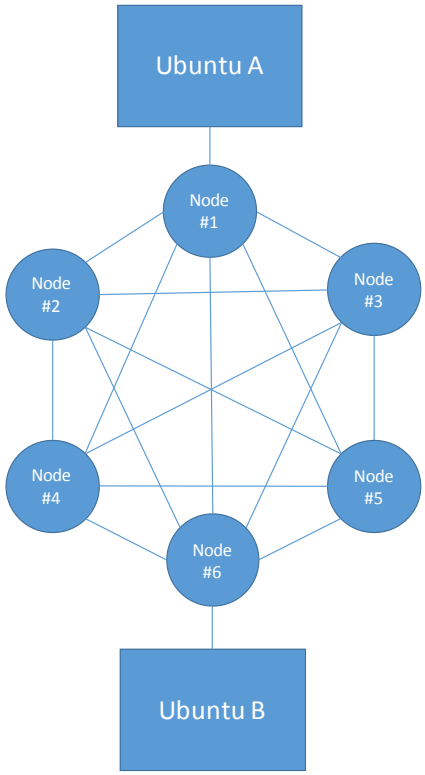

Figure 7. Ubuntu's in Fully connected Mesh Topology

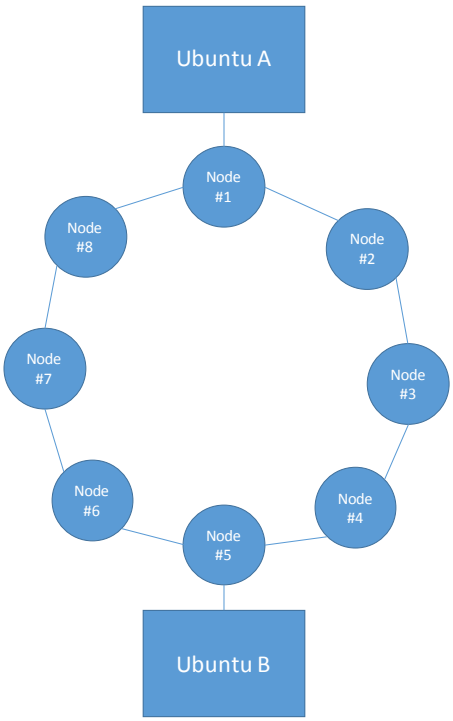

Figure 8. Ubuntus in Ring Topology

Figure 8 shows ring topology as the second topology tested. The ring topology was created by connecting eight virtual switch nodes with each node initially containing two connections to two other nodes. The nodes are labeled in a clockwise fashion. The client and server virtual machines were connected in an equal number of hops apart. The third topology tested was a fully-connected mesh topology. This topology was created by having each node in the system connected to every other node as seen in Figure 7. The client and server hosting nodes were again equidistant apart.

5. Results & Discussions

5.1. Open Switch versus Openwrt Performance with Memory, CPU, and Disk space.

For scalability, the VMs were tested using the University of North Dakota's High Performance Computing (HPC) cluster. Through testing, it was found out that a large amount of CPU usage is consumed during the initial start-up phase of a node creation. A node uses an average of 5% CPU power of the testing platform and can last for over five seconds. To counter this problem, the program was modified to allow the nodes to be given a five second startup time before initializing the next node. Though this delay time works well for a smaller node system, using the same delay time on a thousand-nodes system would theoretically take over 80 minutes to complete.

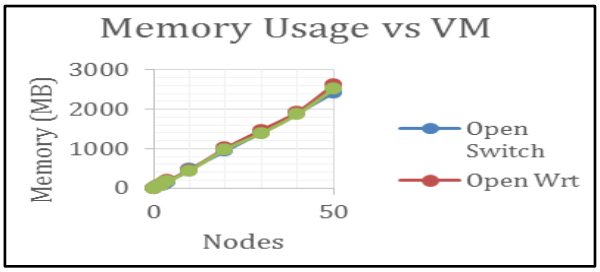

Figure 9. Memory versus VMs

Using a range of nodes and a combination of different topologies, the memory and CPU usage of system was recorded to find the optimal number nodes as seen in Figure 9. The graph in Figure 10 shows that CPU per operating system. As the number of nodes increased, Open Wrt and Open vSwitch had their CPU usage grow relatively slowly, while Open Switch used an abnormally high CPU usage that grew exponentially.

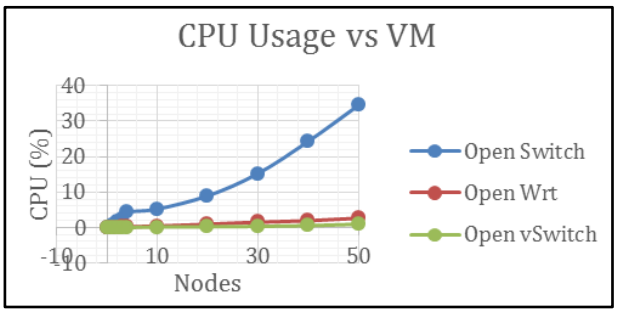

Figure 10. Memory versus VMs

The memory usage per node grew linearly for all operating memory was reached, the system slowed dramatically and nearly caused the system to freeze. Though memory usage was the limiting factor in creating a larger network, none of the tested operating systems will have an advantage. One of the largest discrepancies between the virtual operating systems is disk space usage as seen in Figure 11. For one node Open Wrt requires 11mb, Open Switch 500mb, and Open vSwitch 1,320 MB. Disk space usage will be the contributing factor in picking a virtual operating system. Due to the HPC account allotted to students has only 200 GB, a number that Open vSwitch would not be able to meet with its current configuration.

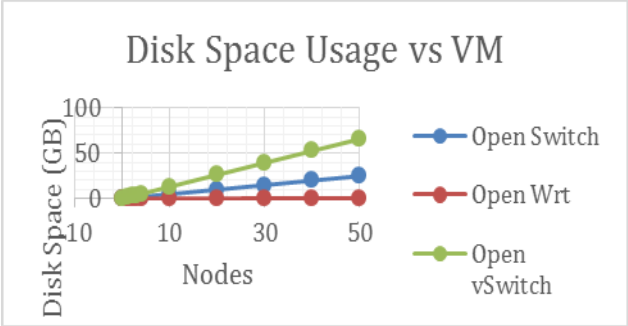

Figure 11. Disk Space Usage vs VM

5.2. Topological Performance with CPU, Memory, and Bandwidth

Figures 12-15 show a comparison of memory, bandwidth, CPU usage with topological changes.

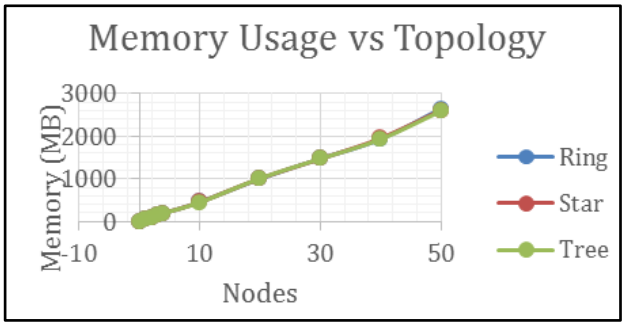

Figure 12. Memory Usage vs Topology

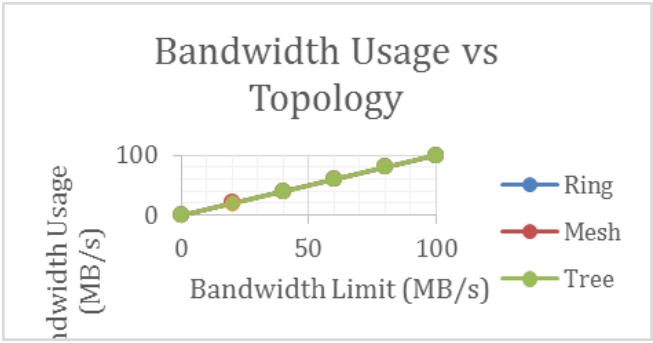

Figure 13. Bandwidth versus Topology

As per Figure 12, the network topology of the system has negligible results on the resource use of the system, with the number of nodes expectedly being the main

factor in resource usage. When the maximum amount of memory was reached, the system slowed dramatically and nearly caused the system to freeze. Though memory usage was the limiting factor in creating a larger network, none of the tested operating systems will have an advantage. Bandwidth usage was measured against a limited bandwidth connection on the three topologies. As seen from the results in Figure 13, all three topologies were able to match the limited bandwidth connection. The CPU usage of the hypervisor was measured during this bandwidth test and are shown in Figure 14.

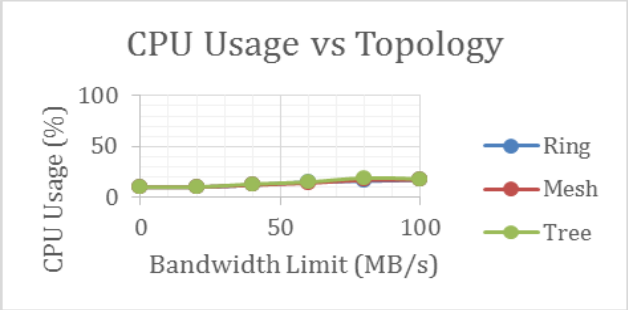

Figure 14. Bandwidth versus CPU

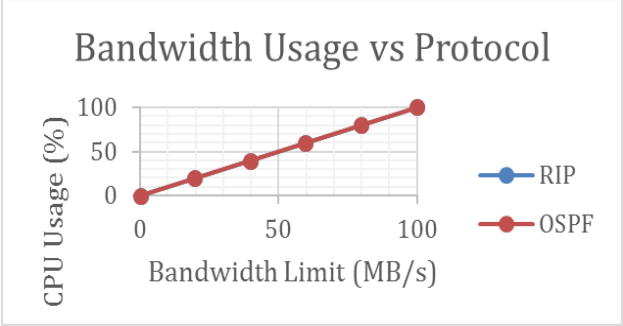

Figure 15. Bandwidth versus protocol

It can be concluded that CPU usage increases dramatically, while performing the bandwidth tests. Though the results show a near identical performance between each topology, further congestion of the CPU for larger number nodes may possibly cause a different result, and remain to be tested as part of future work.

Finally, all nodes of the in the tree topology had their routing protocol switched to OSPF and had the CPU usage vs bandwidth test performed. The results in Figure 15 show that OSPF and RIP performed almost identically with the limited bandwidth test. It can be concluded that impact on CPU performance, while using OSPF and RIP are negligible on systems with a 100MB/s or less connection.

6. Conclusion

An automated virtual architecture under tree, mesh and ring topology is investigated against CPU, Bandwidth, and memory requirements. For scalability, the VMs were tested using the University of North Dakota's High Performance Computing (HPC) cluster. The piloted program proved a success for streaming data sets, and was immensely helpful creating many nodes within a short period of time. Remote access through the wide area network interface could be disabled on each node to prevent an outside source from attempting to gain access and modify the configurations. Additional features and options however will consume more memory and need to be weighted for their usefulness through trade-off analysis.

References

[1] "Virtualization | Oracle", Oracle.com, 2016. [Online]. Available: https://www.oracle.com/virtualization/index.html. [Accessed: 17- Dec- 2016].

[2] J. Wickboldt, R. Esteves, M. de Carvalho and L. Granville, "Resource management in IaaS cloud platforms made flexible through programmability", Computer Networks, vol. 68, pp. 54-70, 2014. "vSphere ESXi Bare-Metal Hypervisor", Vmware.com, 2016. [Online]. Available: http://www.vmware.com/products/esxi-and-esx.html. [Accessed: 17- Dec- 2016].

[3] "OpenSwitch", Openswitch.net, 2016. [Online]. Available: http://www.openswitch.net/. [Accessed: 17- Dec- 2016].

[4] "Open vSwitch", Openvswitch.org, 2016. [Online]. Available: http://openvswitch.org/. [Accessed: 17- Dec- 2016].

[5] "OpenWrt Version History [OpenWrt Wiki]", Wiki.openwrt.org, 2016. [Online]. Available: http://wiki.openwrt.org/about/history. [Accessed: 17- Dec- 2016].

[6] J. Wickboldt, R. Esteves, M. de Carvalho and L. Granville, "Resource management in IaaS cloud platforms made flexible through programmability", Computer Networks, vol. 68, pp. 54-70, 2014.

[7] Hadjioannou, Vasos. "On the Performance Comparison of RIP, OSPF, IS-IS and EIGRP Routing Protocols." 2015.

[8] R. Graziani and A. Johnson, Routing protocols and concepts, 1st ed. Indianapolis, Ind.: Cisco, 2007.

[9] "About Ubuntu | Ubuntu". Ubuntu.com. N.p., 2017. Web. 9 May 2017.

[10] GUEANT, Vivien. "Iperf - Iperf3 And Iperf2 User Documentation". Iperf.fr. N.p., 2017. Web. 9 May 2017.

[11] GUEANT, Vivien. "Iperf - The TCP, UDP And SCTP Network Bandwidth Measurement Tool". Iperf.fr. N.p., 2017. Web. 9 May 2017.

[12] "Ping(8) - Linux Man Page". linux.die.net. N.p., 2017. Web. 9 May 2017.