Optimization of the Migration of Virtual Machines Over a Bipartite Mesh Network Topology

by

Bethany Allison Louise McCollum

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master's of Applied Science
in
Computer Engineering

Waterloo, Ontario, Canada, 2015

Abstract

In today's society, the core network is becoming increasingly important to provide support for the ever growing number of end users as well as the applications that are required to run. While network technology continues to evolve, new topologies are formed to help optimize traffic and communication. One such topology is a bipartite mesh topology, a partial mesh which allows for a two hop distance for any source-destination pair with normal operation. Another trend that requires a good backend network is the act of virtualization, or creating virtual machines to run on configured hosts. One of the key aspects of the virtualization technology is the migration of virtual machines, moving them from one host to another via the network to increase performance or ease resource usage. Migration is a complicated procedure which has to be done quickly to avoid down time, so seeking ways to decrease this time of transfer is important. In today's environments, migration is only done by considering the hosts that it can move to and does not take the network into account. A way to help optimize the migration of virtual machines, especially over a bipartite mesh network, is to take the network state into account and to help minimize the congestion and the traffic on the network created by the migration.

This thesis explores the background and technical workings of virtual machines as of present day and debates the concept of 'cold' migration against the concept of 'live' migration, putting it into perspective of the network and how exactly these migrations are accomplished. This thesis also explores the bipartite mesh network and its operation, including how it should be operated efficiently. Every network is subject to link failures, however with this type of network, the number of failed links must be bounded to the number of spine switches in the topology, which also bounds the maximum number of hops from a source to a destination, though reaching the bound for failed links does not necessarily imply that the maximum number of hops will be reached. Utilizing these bounds and the information gleamed from the virtualization, the primary question of how to optimize the migration of virtual machines over this bipartite mesh topology is formed and examined. These solutions involve a 'network first' approach which examines the state of the network, finds the shortest path destination and only then looks at the resources on the host to determine whether the destination can accept the virtual machine being transferred, and a 'hypervisor first' approach which chooses a destination based on host resources and only then considers the network state and how far the destination is logically from the source. Both solutions have merits and drawbacks, and they are examined; the network first approach is more complicated from a development point of view and requires more back and forth traffic over the network but provides the best optimization in terms of transfer time for the migrating virtual machine, while the hypervisor first approach does not guarantee the best optimization, operating on a threshold of whether the destination is within acceptable parameters. This threshold can easily be seen as the number of spine nodes + 1 and as such, requires little to no computation or communication over the network, unlike the network first approach. These solutions can be fully realized utilizing the OpenStack cloud suite, which, as an open source alternative to virtual machine managers from Microsoft or VMWare, can be modified to do extensive testing on these solutions to determine what is more feasible.

Acknowledgements

There are many people I would like to thank in writing this thesis and the preparation along the way. First, I'd like to thank Professor Gordon B Agnew, who provided immeasurable guidance, support and critique along the way during the writing process and before when information was being gathered.

I'd also like to acknowledge Cisco Systems Inc and the National Science and Engineering Research Council for providing financial support.

Specifically, I'd like to thank Nader Lahouti from Cisco Systems who provided me with knowledge and information about the OpenStack suite and gave first hand explanations and demonstrations about the inner workings about the system, allowing me to conduct my personal research on it.

I'd also like to thank my parents who would ask me daily if I had gotten any work done on this thesis as that pushed me to complete it on time.

Dedication

I dedicate this to everybody who has supported me throughout my life, including friends who listened to me rant and rave about topics they didn't know about, the teachers who helped to push me to be where I am, and the family who supported me at both my highest and my lowest points. If I've ever bothered you with technological information at one time, this is for you to show that I'm not crazy, and that I probably owe some of this to you.

Table of Contents

List of Figures

List of Tables

List of Equations

Glossary

Storage Area Network (SAN): A networked group of storage devices accessible to any device that is located on the storage area network.

Network Area Storage (NAS): Similar to a NAS in theory, but is a single storage device that is accessible to anything on the local area network.

Virtual Machine (VM): A logical computing device that is run separately from a hardware host while using its resources to operate. The hardware host utilizes a hypervisor in order to run and manage virtual machines.

Time to Live (TTL): A measure in network packets determining how many more hops the packet can go over network devices before being considered lost for the purpose of retransmission.

Total Cost of Ownership (TCO): A measure in business environments to determine the total monetary value of a project, including non-traditional costs like labour and maintenance.

Open Shortest Path First (OSPF): A widely used routing protocol standard.

Virtual Network Interface Card (vNIC): A virtualized NIC that acts as hardware to a virtual machine and managed by the hypervisor. In actuality, the hypervisor utilizes the host's NIC and routes the information to the virtual machine.

Content Addressable Memory (CAM): A type of memory found in network devices useful for finding information quickly.

Virtual Desktop Infrastructure (VDI): Utilizing virtual machines hosted on a hypervisor to virtualize a user's desktop (instead of virtualizing servers).

Internet Small Computer System Interface (iSCSI): A transport layer protocol which allows for SCSI commands to be sent over Ethernet.

Virtual Hard Drive (VHD): A hard drive allocated to a virtual machine for it to run and install applications on; it is physically space on the hypervisor's hard drive.

Network Address Translation (NAT): A way of changing headers in IP packets to change one IP address into another; used when translating private IP addresses to public IP addresses (from LAN to WAN).

Virtual CPU (vCPU): The virtual CPU chip that is assigned to a virtual machine. For the virtual machine's purposes, behaves as a real CPU, but is actually utilizing the CPU on the host.

Dynamic Host Configuration Protocol (DHCP): A networking protocol that allows a machine to get all of its network configuration from a central server which manages all the addressing information for the network. An alternative to manually configuring all network information on every host.

Chapter 1

Introduction

The internet is constantly evolving and the user requirements for it increase daily. Companies use it to stay constantly connected and individual users use it for work, research and play. Over 22 PB of worldwide internet traffic a month is dedicated to internet video, and by 2018, it is forecasted that the number will increase to around 64 PB [1]. Other forms of traffic are also increasing to the point where roughly 23 EB of internet traffic will be generated and used monthly. It represents a shift in the thinking of culture to a more digital, virtual view of information. A similar shift is happening in the business market. Whether it is to meet user demand via web servers, creating e-applications to be used, storing user data on storage area networks (SANs) or network area storage (NAS), the back end infrastructure must be able to support the increasing user demand. Many companies are switching to use virtual machines (VMs) instead of purchasing multiple physical servers in order to decrease total cost of ownership, power consumption, and enhance reliability for their applications, used both internally and externally [2]. This trend is growing, outpacing the adoption of physical servers in an environment that requires the back end support. A key aspect of these virtual machines is reliability; if a physical host for these machines goes down, a virtual machine should be able to reliably fail over to a secondary host. To do this, a back end switching infrastructure is needed that can support this transfer in a timely fashion, while also supporting external traffic to the applications running on the virtual machines.

While many network configurations exist, each with their own specific drawbacks and benefits, a switch configuration that should be studied is that which takes the form of a bipartite mesh. Edge nodes are connected to all core nodes, and the core nodes are connected to each other, as demonstrated below in Figure 1. It emulates a bipartite graph, in that each level of the graph hierarchy is connected to each other, but adds the notion that the spine nodes are also connected to their neighbours. The higher level of the topology are considered 'spine' nodes, as they act as the backbone of the network, while the lower level are considered 'leaf' nodes, similar in functionality of the access edge; switches where all of the end user devices are held. This terminology will be used for the rest of the document. In a real world topology, this bipartite mesh idea provides enhanced reliability like a full mesh, where all nodes are connected to each other. With optimal behaviour, two hops are required for traffic to travel between any two leaf nodes, where physical servers holding virtual machines can lie. A hop decreases the Time To Live (TTL) of a datagram by one. If a physical server becomes overloaded, the virtual machine can be failed over manually to another physical server holding the same hypervisor over these two hops in a process called live migration. Maintaining optimality for switch node load is vital for both traffic and reliability purposes; overloading all the nodes will slow the infrastructure down, and having a fewer amount of virtual machines on each node will be an inefficient waste of resources and thus drive up the total cost of ownership (TCO).

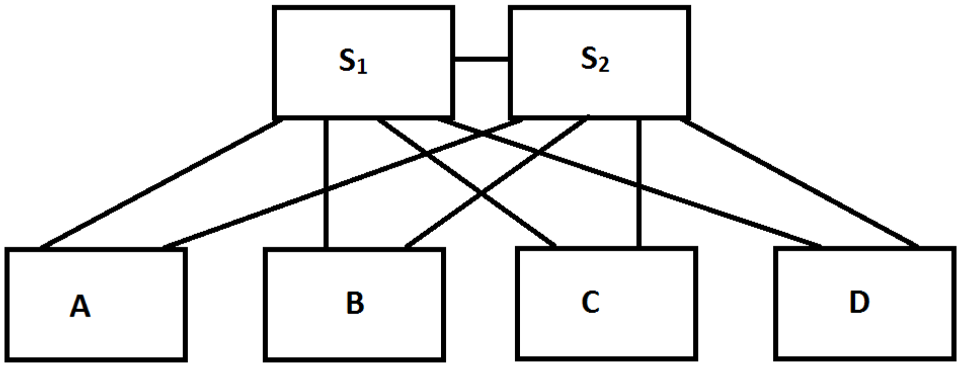

Figure 1. An example of a two spine, four leaf bipartite mesh network core topology.

Using this bipartite mesh, an optimal tradeoff between balancing load and latency can be found under the conditions where time to live is bounded by a maximum number of hops (or as such, a minimum acceptable TTL) and there can exist a maximum number of failed links in the topology, and this optimality is what is derived what is discussed below. While the minimum number of hops from edge node to edge node is two, because the core nodes are connected it is possible for traffic to travel between core nodes to reach the end destination, if it were so required, such as in the case where multiple links to the end node have failed; an example of the importance of such redundancy is obvious to see in this case. These bounds - an upper bound for failed links and a lower bound for TTL - are to emulate a real world scenario and to demonstrate worst possible cases. In an ideal situation, traffic would go over two hops and there would be zero failed links, providing the best possible latency. When not in an ideal situation, a model can be generated to figure out how to properly deal with the information and provide measures on how to properly balance the load between the servers automatically such that traffic is not delayed or impeded in a way that makes the end user's experience suffer.

1.1 Scope

This thesis aims to accurately look into how to optimize the load on servers attached to the leaf nodes within terms of specified bounds for the number of failed links and TTL. As such, these are the constraints that will be dealt with exclusively. The scope of this thesis will deal exclusively with the network topology in the context of hypervisors and virtual machines being attached to the leaf nodes. While other network applications such as NAS, web servers, etc. are used by companies in the real world on a day to day basis, virtualization is a growing trend and as such, will be the primary focus for this thesis.

Furthermore, this document will proceed with current day technology used for the modeling. As technology becomes more reliable, more sophisticated and more improved, there will obviously be changes to what has been written here. To keep these future changes into account is important, but near impossible to use to model, and as such, will be considered and noted but not used as a basis for information. In terms of context, it might be that the changes will be significant, but they will not be used as part of the analysis.

Due to the fact that the optimal load is being determined based on items such as links and TTL, the primary focus will be on the topology and not the individual devices themselves. A bipartite mesh topology can work with any number of network switches (layer 2 or layer 3) from many different companies. As such, we will be declaring the switches as generic 'nodes' and not take into account the internal workings that different switches might possess. While Cisco Catalyst switches were used for testing purposes, other companies such as HP, Juniper, Oracle, and Arista provide switches that could also be used to create this topology. However, due to the differences internally, there would be some slight differences in the way that packet queueing is performed, what type of forwarding method the switches use, processor speeds and switching fabric speeds. These will not be taken into account, as these differences with today's technology are slight and have no major bearing on the analysis.

1.1.1 Caveats and Assumptions

On top of the aforementioned focus of this thesis, there are some fundamental assumptions that have been made in order to make this topology be as generic and useful. The first major assumption is that nodes / switches are independent, as they are in any network topology. They can be in a small area, distributed over a large area, be administered by different groups, have heartbeat mechanisms in place to determine whether they are considered 'alive', but the switches are themselves independent, in that activity on one will not influence activity on the other in terms of operation or behaviour. On the same vein, links between nodes are also considered independent (for example, one link going down will not automatically bring another link down).

Another thing that is being assumed and addressed is that a link is a single link. There is technology available that can allow for multiple physical cables to be logically grouped together to present one link of increased throughput and reliability. Traffic is load balanced over these physical cables and if one of them goes down, information can still get through via the others. On HP technology, it's called Port Teaming. The general terminology is link aggregation. Link aggregation is becoming more popular in general use, but will not be considered in the analysis on this thesis. As such, any link mentioned in this document is a single cable without redundancies.

Bipartite meshes will only work with two or more spines. They can be generalized in terms of the number of spines and leaves, but the base case for analysis will always be two spines and four leaf nodes, as previously shown in Figure 1. If there is less than two spines, then the topology will fall apart and all the benefits of the system will be lost (notably, if the spine fails or the link from source leaf to spine fails, there will be no throughput).

1.2 Thesis Roadmap

This thesis aims to provide general information first and foremost, and then go to in depth analysis on the topics at hand. Background information in communication networks in general, a basic knowledge of the concept of server virtualization, as well as university level probability and statistics is useful while reading this thesis, though some of the conclusions can be drawn without the use of it.

The information this thesis provides includes the general problem, the background on the various components to be considered when looking at the problem as well as potential solutions. It is laid out in an effort to guide the reader through a logical path in order to provide context to conclusions made by the end of this thesis.

1.3 Chapter Descriptions

This thesis is divided into five main chapters, and they are defined as follows:

- Chapter two is detailed information about the topology itself, divided into four subsections: general information about bipartite meshes, calculating traffic hops in both ideal and non-ideal situations, taking into account what happens when a link fails, and how to take the information and model the topology as a graph.
- Chapter three is concerned with looking at the load of virtual machines and how they will behave on a network in general. This chapter then delves into looking specifically at the hypervisor and virtual machines for overhead and how it operates on a network.
- Chapter four looks at how to optimize the load based on the information presented in chapters two and three, using calculations in probability and other analysis to come up with a general solution for any bipartite mesh topology and a general hypervisor that is working in the

background to move virtual machines around. It is subdivided into sections to discuss the various aspects of the problem and demonstrate the way to the solution.

- Chapter five draws the conclusions of this thesis as well as examines future work that can be done, potential improvements that can be made and further areas of interest with this examination.

Chapter 2

Modeling a Topology as a Bipartite Mesh

In the networking world, many topologies exist and have evolved over time. Figure 2 demonstrates a few of these topologies; from basic topologies such as a ring, popularized by the Token Ring protocol prior to Ethernet, to the hub and spoke topology (also known as a star topology, or in some specific cases, an extended star), to general meshes in which everything is connected to one another, or, in some cases such as this topology, a partial mesh, which utilizes a few key aspects of a full mesh such as reliability, while retaining more scalability due to the fact that a full set of links are not required when adding a new device to the topology. Though two topologies might have the same physical layout with respect to connections, their logical characteristics such as protocol, link speed, throughput, physical location and device type may differ. As has been the case for the past 30 years, Ethernet (or variations of it, based on the IEEE 802.3 protocol [3]) are what are used in the network core [4]. Both are important when studying any impact that traffic may have on a topology. The physical network characteristics can define the necessary protocol; for example, the physical media will determine the length that traffic can go; typical Ethernet over copper based media (regardless of the speed of the Ethernet) is generally limited to roughly 100m, while fibre optic media can go upwards of kilometers [3]. The logical characteristics are often what is used internally by devices in order to figure out the shortest or the least expensive path for routing information from a source to a destination, based on the metrics of the characteristics themselves. Often, lower values for these metrics are desired (a low hop count, or low delay, for example) but there can be cases where higher metrics are considered, such as bandwidth and reliability. An example of the latter is Open Shortest Path First (OSPF), which uses a default cost measure which is inversely proportional to bandwidth (and thus, a higher bandwidth leads to a lower metric) [5].

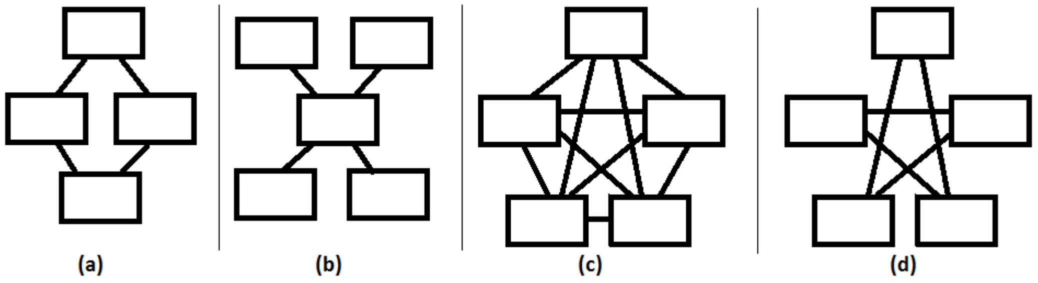

Figure 2. Examples of topologies throughout the ages; from left to right: a ring topology, a hub and spoke topology, an extended star topology, a full mesh topology and a partial mesh topology.

The topology that is being used and studied in this thesis is, in essence, a partial mesh similar to one as seen above. The exact topology has already been shown in Figure 1, above, though that is a specific example of a two-spine, four-leaf variation. Though the number of spine switches and the number of leaf switches can differ from the above figure, the same general rules will be followed: Spine nodes will always be connected to their neighbours and leaf nodes will have connections to all spine nodes. This allows for a large amount of redundancy without requiring a full mesh where all of the spines are connected to all other spines, and all leaves are directly connected to all other leaves. If the latter were the case, it would resemble a complete graph and the number of links would grow on the order of $O(n^2)$ according to the equation outlined below in Equation (1), where *n* is the total number of spine plus leaf nodes [6]. In our case, the growth of connections as new nodes are added can be shown to be linear, as demonstrated later in this chapter.

$$\frac{1}{2}\left(n^2 - n\right) \tag{1}$$

As previously mentioned, a logical topology exists alongside a physical topology, and this logical topology depends on the type of hardware that the switches are comprised of. This bipartite mesh topology can run with any switch hardware, and while the core functionality will not change, some of the numerical information might, depending on specific switch parameters such as the backplane fabric speed. As previously mentioned, Cisco Catalyst switches were what was used for testing situations and comprise the information used as a basis for extrapolation. However, these switches could have been replaced by comparable hardware with similar specifications from different vendors. The only requirement in terms of study that is necessary is for the switches to be able to understand traffic from a virtual NIC (vNIC), generated by the servers attached to the switch. Due to the adoption of virtualization, this traffic is converted into a readable format in the hypervisor and thus, any switch that supports modern protocols could work. Further information about how vNICs work is located in Chapter 3.

2.1 Hardware and Generalized Topology

With modern day networking equipment, it is possible for current networking to involve different protocols, hardware specifications and physical topologies while retaining the same amount of reliability and performance as in the past. Though hardware is not as vital in the grand scheme of things as in the past due to the fact that a lot of separate hardware can perform the same tasks, it is still important to consider the physical topology that was used for testing so further results have a physical context involved.

2.1.1 Hardware

The hardware involved for testing was a series of Cisco Nexus 4000 switches at the core, and a series of virtual Cisco Nexus 1000v switches for the leaves. A virtual switch is a program that is entirely emulated as a virtual machine in a standard hypervisor, and performs every function of a physical switch without needing to expend more of a cost for a physical switch [7]. The virtual switches need to run on physical servers and have obvious drawbacks in the amount of scalability that it can support (due to the fact it needs a server, and the processing power is split between standard server operations and the emulation of the switch), for a testing environment, the small size of the topology to test is easily handled by the Nexus 1000v.

The servers used were a series of Cisco Blade servers. Some blades were simple VMWare machines that ran only the Nexus 1000v VMs. Other blades were standard Red Hat installations that had components of the OpenStack suite installed on them. Due to the nature of the blade servers, each blade was independent, but shared a backplane that allowed for easy communication between each individual blade and the blade control node, which allowed for easy communication between machines on the blade and thus, easy communication between the blade servers that were connected to the virtual switches. Those servers with the OpenStack installations on it acted as the load for the leaves. The blades running the virtual switches were connected to the core Nexus 4000 series switches, as equivalent physical switches would in the logical topology.

Note that it should be said that the exact specifications of the blade servers are unknown, other than the fact that they were powerful enough to run OpenStack and a VMWare Hypervisor without any lag, slowdown or competition for system resources.

2.1.2 Generalized Topology

Though in Figure 1, the topology is shown to be 2 spines and 4 leaves, it can be expanded, as all topologies can. An infinite number of spines and leaves can be added to the topology in any order, making topologies such as 2 spine, 10 leaf or 3 spines, 3 leafs possible, as the convention is followed where all spines are connected to all leaves, and neighbouring spines are connected to each other. Figure 3 below demonstrates a generalized topology where there are *s* spines and *l* leaves. Furthermore, Equation (2) relates to Figure 3 by demonstrating the linear relationship for the total number of nodes in the generalized topology given *s* spine nodes and *l* leaf nodes.

$$n = s + l \tag{2}$$

When investigating the number of connections, they can be examined. In the case of Figure 1, there are 9 links - 8 between the leaves and the spines, and one between the spines themselves. This can be further extrapolated in the general case.

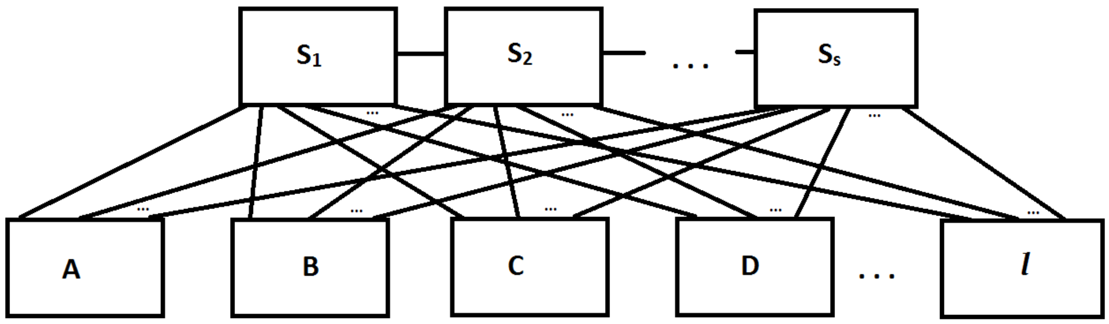

Figure 3. A generalized topology with *s* Spines and *l* Leaves

On the general topology, it is obvious to see that every leaf switch will have a number of links going out equal to the number of spine switches, as that is the definition of the topology. Furthermore, this will always be the case, no matter how many spines there are, or leaves that are added. Thus the number of connections from leaves to spines will always be *s(l)*. In a similar vein, all neighbouring spines will be connected together, meaning that between every three spines, there will be two connections. In a chain with *s* spine nodes, the number of connections between them will always be *(s - 1)*. This allows an easy expression to be generated for the total number of links in this generalized topology shown in Equation (3) below. Unlike the growth of a full mesh network on the order of n^2 (looking back at Equation (1)), this is a linear growth, which cuts back on a measure of complexity for the topology, making it easier to implement as well as maintain.

$$c=(s-l)+s(l) \tag{3}$$

2.2 Failure of a Link

A link failure will potentially inhibit the flow of traffic, creating congestion and lost packets in a real world scenario, if information is sent and is then unable to reach its final destination. These failures can be caused by hardware being unplugged, cables being damaged or cut, electrical outages, devices being maintained or upgraded, et cetera. All of these events, barring power outages, allow the concept of a link failure to be modeled as independent events - one link failure does not automatically depend on others, or more specifically, one link failure does not automatically cause other links to fail. Also, these failures are independent of time. Modeling these link failures as independent events follows tradition of most networking related analysis being able to be modeled as a set of independent events, such as network queues being modeled as independent Markovian processes [8].

Due to the fact that failed links can be modeled independently, the probability for a link from source to destination to be failed can be modeled independently. Using Figure 1's topology, if we want to send traffic from Leaf A to Leaf B, the default path will be A-S_1-B. Thanks to the topology's redundancy, A-S_2-B will also get the traffic to the destination, if the link between A and S_1 has failed. The probability of that specific A-S_1 link failing is modeled by Equation (4). This is due to the independent nature of the connections, as all links have an equal chance of failing (and thus, the other 8/9 links are still operational).

$$P(\text{failed link being } A - S_1 \mid \text{one failed link}) = \frac{1}{9} \tag{4}$$

As can be seen, through this 'backup route' of A-S_2-B, even though one link has failed, there is no increase in the number of required hops; traffic can still get from A to B over two hops. In a scenario where the link from A to S_1 is down, A will know about this link being down and thus send it on its secondary device in its Content Addressable Memory (CAM) Table, which would be S_2, automatically. This would not affect the speed or latency of transmission. In this case, the minimum number of hops is still two, and this demonstrates the fact that a link failure along a traffic's path does not necessarily change the minimum number of hops, even though it can, like in the following scenario.

If traffic is still going from Leaf A to Leaf B, but a failed link exists between S_1-B. A would have no knowledge of this failure and would transmit the data to S_1, due to the defaults in its system. However, S_1 would have no way to transmit the information to B, and thus would have to transmit the information to S_2, who would then move it to B. This new path of A-S_1-S_2-B now has three hops, even though the probability of the link from S_1 to B being down is also 1/9, like in the above scenario. It's obvious to see that with one failed link, increasing the hop count has a probability of 1/9, while there is an 8/9 probability that the hop count will stay the same.

In this topology, if a second link failure is added, it will either have no effect on traffic from A to B since it is not on the route, a minimal effect, or it will have the effect of completely preventing the traffic from reaching the destination. For example, if the link from S_1-B has already failed, making the traffic go via A-S_1-S_2-B, there are different options as to where the second failed link can be and its effect on the traffic. These are outlined below in Table 1. Failed links to Leaves C and D are disregarded because traffic is not flowing to them in this example. There is a case where if the link between S_1-A is also failed, then the 'backup route' using S_2 alone will be forced for use on a retransmission of the information; this diminishes the number of hops back to the minimum of two. The interesting case is that, with the failed link from S_1-B, if the link from S_1-S_2 or the link from S_2-B fails, traffic cannot get to the destination. That is, there is a 2/8 probability that traffic cannot get through, if the first failed link is S_1-B.

Table 1. Adding a second failed link to Leaves A or B in a 2 Spine, 4 Leaf topology and the effect it will have on traffic

Failed Link	Traffic Route	Number of Hops	Probability
A-S_1	A-S_2-B	2	1/8
S_1-S_2	N/A	N/A	1/8
A-S_2	A-S_1-S_2-B	3	1/8
S_2-B	N/A	N/A	1/8

The above demonstrates the fact that with too many failed links along a path, traffic can not reach the destination. Though the probability of this occurring is fairly small - in this example $\left(\frac{1}{9}\right)\left(\frac{2}{8}\right) = 0.0278$, it is still a non-zero probability. This will, in effect, require the maximum number of failed links on a topology to be bounded in order to ensure that the traffic will reach the destination with a reasonable probability. This is necessary, no matter the topology size or the number of devices.

In the general case outlined in Figure 3 in section 2.1, where we have *s* spines and *l* leaves and a total number of links demonstrated in Equation (3), it can be similarly shown that for one failed link, there is a probability of $\frac{1}{n}$ that it will increase the hop count and a probability of $\frac{(n-1)}{n}$ that the hop count will stay the same. Furthermore, increasing the number of failed links, *f*, like in the example above, will increase the probability for the number of hops to exceed the minimum. Though in the previous example, a second failed link in a strategic place was enough to render the topology 'broken', with more spines, a second failed link would, at worst, increase the number of hops by one, as shown in Figure 4. While it is possible for traffic in this figure to go straight from A-S_3-B, following the same convention where A-S_1 is the default traffic path, the path there would be 4 hops (outlined in blue in the figure), and there are more opportunities for combinations of failed links to lead to these 4 hops; this will lead to the probability for the average number of hops increasing, while the minimum stays at two. Similar to the previous example, if there are three failed links and they are in strategic spots (such as S_1-B, S_2-B, S_2-S_3), traffic will once more be unable to get through to the destination; any fewer number of failed links is not capable of such a feat. As such, it can be said that if the number of failed links $f \geqslant s$, there is the chance of the topology failing.

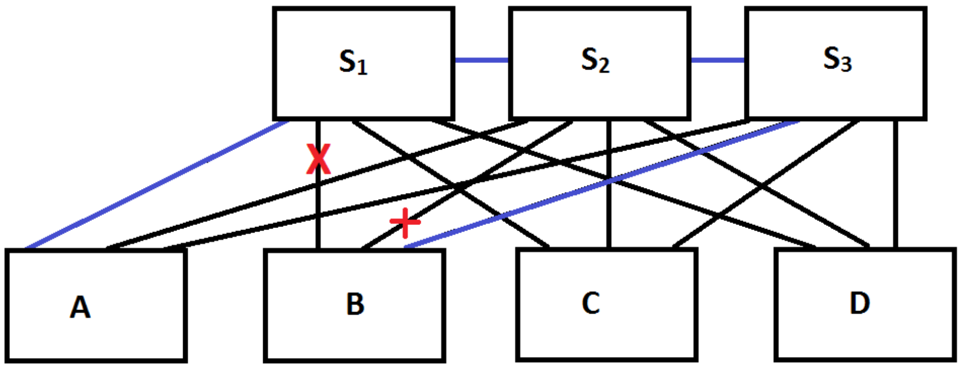

Figure 4. A 3 Spine, 4 Leaf topology with 2 failed links, demonstrating how the number of hops could increase without dropping traffic like smaller topologies

Due to this knowledge that $f \geqslant s$ can cause traffic to fail for a given source-destination pair (and thus impair the network's integrity), for a given topology, the number of failed links must be bounded in order to ensure that the system will not fail in any scenario. The logical boundary to ensure zero failures would be an f_{max} such that $f_{max} < s$.

2.3 Traffic Hops

Going in depth into analyzing traffic hops in this bipartite topology can be complicated, but is necessary due to the connection that exists with the number of failed links. In fact, the number of traffic hops can be considered the most important metric, and often is in terms of network traffic; finding the smallest number of hops is usually tantamount to finding the quickest path through a topology from source to destination [9]. Though this is not true in all cases, (a link's bandwidth might be extremely low, for example, or a problem with the cable or interim node might yield long throughput times) it is still vital in a topology such at this, which is built on having a small hop count between source and destination.

In the best case scenario, as previously mentioned, due to the mesh infrastructure, the number of hops is two. Source leaf, spine, destination leaf. Without any failed links, this is guaranteed to be the number of hops taken. With some failed links, it is not guaranteed, but as shown above in section 2.2, there is a probability that the position of failed links will increase the hop count. The question becomes: how does the hop count increase? By blocking a path with a failed link, it's known that the hop count will have to go up due to the fact that the destination can't be reached on the original path, but how it increases needs to be taken into consideration.

The fact is that the increase per failed link is, at worst case scenario, one hop. Note that when 'worst case' is used, this is barring the situation where the entire topology fails and traffic cannot reach the destination, thus bringing an 'infinite' hop count. If a failed link is the link that goes to the destination from the spine the information is currently sitting at, the traffic will have to hop to another spine and then to the destination. There are obviously scenarios where adding a failed link will not change the hop count (the failed link is not on a vital path, as demonstrated above) or in some cases, even decrease a hop count. For example, Table 1 above has already shown that taking out the path from A-S_1 will automatically force the source to send the information to S_2 and thus decrease the hop count back to two from three. Without that A-S_1 path blocked, eventually the topology would recalculate and find that A-S_2 would be the better route and start sending traffic there until the original link comes back up. This recalculation is often done on a fixed timer in network topologies, depending on the routing algorithm used. But this is far from the worst case. The worst case, already demonstrated above, will always increase the hop count by one.

This linear increase for the worst possible scenario can thus be modeled as the following, where the maximum potential hops for any number of failed links is equated as the following shown in Equation (5), where f is the number of failed links.

$$h_{max} = 2 + f \tag{5}$$

The two will always be there due to the fact that it's the minimum hop count. As previously mentioned, it is known that this h_{max} will not be reached in every scenario. In fact, as the name suggests, it bounds the maximum number of hops in the topology and it also relates it to the number of failed links in a linear manner. This relationship is useful, since in fact, if one parameter is bounded at a maximum, then the other will similarly be bounded at a maximum, in an easy linear relationship.

The effect of this maximum bound is taken into account for 'thresholding', or providing a maximum boundary for situations that the topology can find itself in. This maximum hop count will not necessarily be reached even if f reaches $f_{max,}$ due to the nature of probability; the maximum f_{max} can be reached, but if all the failures are not on the same path, then the hop count will not increase to the point where h_{max} will be reached. However, going in depth into the probability of this bound occurring for varying scenarios is outside the scope of this document.

Chapter 3

Virtual Machines, Hypervisors, And the Importance of Networks

The trend in the modern consumer and industrial market is steadily moving away from pure hardware deployments [10]. It is often cost effective as well as eco-friendly to virtualize the environment by way of installing hypervisors on a small number of powerful bare metal servers and from there to create smaller, virtual machines that provide the same functionality as physical servers. There are also new pushes to create virtual machines that ask as desktops assigned to specific users, allowing their machine to be accessible wherever they choose to work; this is referred to as a Virtual Desktop Interface (VDI) and is a part of the new consumer trend to the cloud [10]. Whether the situation is for the VM to act as a server or a desktop, many things are needed behind the scenes beyond the physical hardware for the VM hosts, such as a good storage infrastructure as well as a good network infrastructure. The latter is redundant due to the way that storage area networks work; in 2002, network based SANs utilizing protocols such as iSCSI and Fibre Channel were starting to come of age [11] and modern day SANs force these protocols as minimum requirements, requiring a stable, reliable and high speed network infrastructure to support the data requirements [12]. These are in turn to provide the ability for virtual machines to not only properly exist and be utilized, but also to move from one host to another through the network so that in case of physical failure, system instability or overloading. This chapter aims to give a general overview as to exactly how this is accomplished and the network traffic involved in using virtual machines.

3.1 Hypervisor

While many hypervisors exist in the modern market, made by many vendors - Microsoft, VMWare, open source movements - they all perform the same basic function, to create the underlying platform that interfaces between the virtual machines it creates and the server's operating system and physical hardware. This allows things like iSCSI (or fibre channel) traffic and general IP traffic to be sent to the physical server's Ethernet ports and then be relayed into the correct VM. Figure 5 below is typically the system where a hypervisor exists, no matter the vendor supporting it. For the purposes of testing out load on the network topology in question, an open source networking suite and monitoring named OpenStack was used to generate, test and capture statistics for VMs. While it itself is not a hypervisor, merely utilizing what hypervisor is on the systems it is installed on (or alternatively, installing a default hypervisor when the suite is installed), it is compatible for use with many of the major market hypervisors such as Microsoft's Hyper-V, VMWare's ESX and Linux KVM [13]

OpenStack itself is a combination of efforts from many companies and corporations to give a free and open source management tool for the new and growing cloud platform, including managing virtual machines for deployment through whatever underlying hypervisor is installed [14]. The suite is made of many different products, each having a core community of contributors who ensure that the code is bug free and stable. The OpenStack suite is made of numerous components that can be installed on numerous *NIX servers for reliability and redundancy, including - most importantly - the hypervisor [14]. There is a central controller which coordinates all the various pieces of the OpenStack suite, no matter their location, and allows for an overview of all aspects of it [15]. The constant addition of features allows for the suite to be updated frequently, and is done in a manner by a conglomerate of programmers from the supporting companies dedicated full time to working on the project. The various components can be utilized or ignored depending on a specific setup and what the end consumer needs; the extensibility was important to consider when the suite was used to perform tests and study the underlying topology.

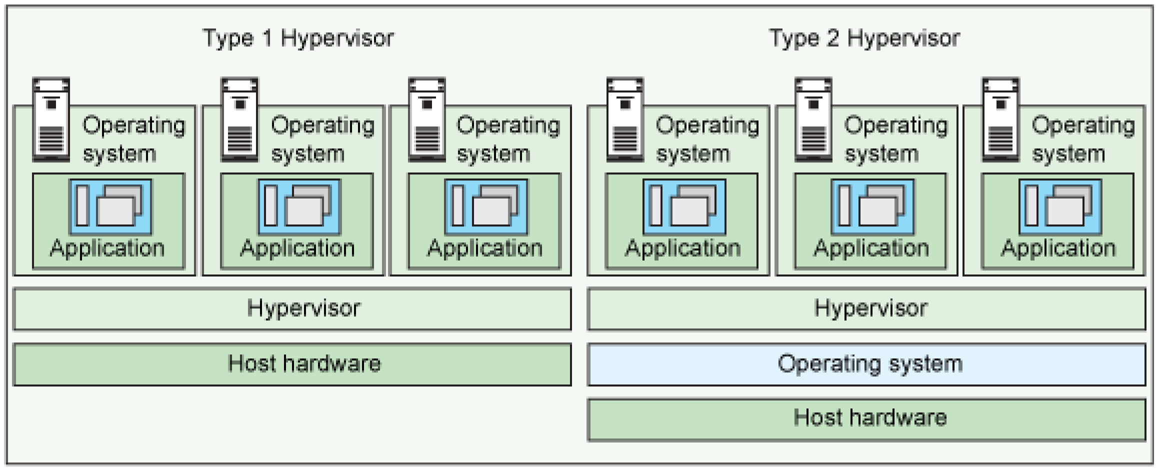

Figure 5. The general behaviour of a hypervisor with respect to Virtual Machines and the underlying physical hardware. Type 2 Hypervisors are most common in standard Enterprise markets for virtual machine deployment [16].

The importance of the network in a virtualized environment is not to be dismissed. The SAN is often a standalone hardware appliance that utilizes network connections to transmit the data to where it needs to go, often via iSCSI or Fibre Channel. While there are SAN appliances that are connected directly to a specific system that acts as a file server and the information is transmitted directly over standard Ethernet, the amount of traffic is comparable; payload information of the files being accessed stays the same and the only difference is in the wrapping of the datagram due to the differences in protocol, and is thus negligible in terms of the overall information that is to be transferred. In a virtualized environment, a redundant SAN is useful to store virtual hard drives (VHDs), which are what virtual machines use as their logical drives. While they can be stored locally on the physical server, these VHD sizes can be the same size as a physical HD (upwards of 500 GB, depending on the size the administrator sets for it), and thus often need a larger repository for storage.

This, of course, implies that the contents of the VHD need to be transferred via the network infrastructure, and if the underlying network infrastructure fails or is prone to failure, then the VM will start failing. This is why SANs or NASs are used; the VHD is accessible to any machine that can connect to the device without needing a given hypervisor to transfer the data when necessary.

As demonstrated above in Figure 5, the vNIC talks to the physical NIC on the server, and the physical NIC segregates traffic based on a concept similar to Network Address Translation (NAT) [16]. Information through the vNIC can be monitored. But in general, the hypervisor doesn't use any network load itself, unless it's communicating with the controller (another assumption being that the controller is on another server in the network); that type of communication is not unique to the hypervisor alone and thus should be considered 'background traffic', which is not important to study for this thesis. Further information about the in depth workings of the hypervisor are also not covered in scope, but can be readily found in other locations already referred to.

3.1.1 Migrating a Virtual Machine

A relatively modern advancement to virtualization is the concept of live migration. While virtualization as a whole started not long before the advent of live migration, there was a period where it could not be performed and thus the only opportunity was to perform cold migration. Live migration allows a virtual machine to be moved from one host to another host without requiring the virtual machine be powered off. This allows an end user to keep using it even while the back end infrastructure is being changed behind the scenes.

3.1.1.1 Live Migration v. Cold Migration

The concept of live migration, as mentioned above, is the concept of moving a virtual machine between hosts 'live'; the virtual machine does not go down and any user accessing it or an application that lives on it will not be disconnected. This involves the careful transfer of the configuration file of the virtual machine from its current host to its future host, and then at the very least, copying the contents of the VM's RAM over to the new host repeatedly so that any small changes that are made are continuous when the user's traffic is then redirected to the new host [17]. This is done through changing the network information in the configuration file on the new host and creating a new, corresponding vNIC on the new host to accept the traffic.

The process is easier if the virtual machine is turned off before transfer, since it is not necessary to preserve an end user experience. This process is called 'pure stop and copy' [17], or colloquially termed in this thesis as cold migration. It is also less load to transfer, as there is no contents of any virtual memory to transfer from one point to the other to preserve the current operating information of the system; a cold migration can be transferred faster than a live migration, but requires shut down and boot up sequences as well as downtime for any applications resting on the virtual machine [18]. This leads to live migration being done more frequently in infrastructures that can support it; the benefits severely outweigh the negatives. Often, a server can take minutes to become operational once it is booted up and any changes to the system could result in instability upon reboot; live migration does not deal with any of that and is the choice used commonly [17].

3.1.1.2 Network Communication During Migration

As determined by testing, with OpenStack, the main load on the network that is transferred from one device to another when a live machine is being live migrated are four simple messages: vNIC DOWN, VM DOWN, VM UP and vNIC UP. vNIC DOWN and VM DOWN are sent by the source host, to inform devices on the network that the vNIC has been torn down and no longer exists, and that the virtual machine also no longer exists, and therefore any switching information for that vNIC/device should be removed from switching tables. Conversely, the VM UP and vNIC UP messages are sent by the destination host to let the nodes on the network know that a new destination is available, and its network information is now ready for reading and receiving data. This process takes a matter of microseconds, but changes in the switching tables can take longer, depending on the protocol used. Thus it is generally expected that switching table updates are triggered immediately upon receiving a VM DOWN or VM UP message before the message is passed on to all neighbouring devices.

If a VM is turned off when it is being transferred, the VM DOWN message would have already been given prior to it being moved, as VM DOWN messages occur when the VM is shut off (and thus, why it occurs during live migration is that the virtual machine on the source host is flipped 'off' at the instant the VM is flipped 'on' on the destination host). Furthermore, after the transfer there is no VM UP. That would only occur when the VM is manually turned on. However, the vNIC DOWN and vNIC UP messages would be performed as usual, because the virtual network cards still disappear from the hypervisor and reappear on the new host, and the nodes in the network need to know this information.

In general, adding to the traffic load on the network is the aforementioned RAM transfer, as well as a potential VHD transfer. However, in this topology, it is assumed that the VHDs are located on centrally accessible storage and thus, the actual contents of the drive do not need to be transferred rather than the new host simply accessing the central storage and making a storage connection to it. In this case, only the configuration file for the virtual machine needs to be copied over to the destination host. While it is possible to do live migration if the VHD are not on central storage, through a process called 'block migration', studies through the University of Zurich show that with OpenStack, block migration is significantly slower than standard live migration using central storage [18].

These can be set as variables C and R for configuration and RAM respectively, and sized in whatever format is wanted (bits, KB, Kb, GB, Gb, etc.) Equation (6) below demonstrates the amount of information to transfer in both the cases of live migration, and can later be used to determine the effect of a VM moving from one node to another, which is necessary to be taken into consideration if the load is to be optimized.

$$T_{LM}=C+i(R) \tag{6}$$

i in this case represents the number of passes over the VM's RAM that are needed to ensure that a constant state is kept for any end user. This can vary depending on the Hypervisor by a few passes (more granular versus the risk of losing a small amount of information). It is true that after the RAM is transferred, in later passes only the delta would be required (information in RAM that had changed between when the original transfer started and the current pass), but the worst possible scenario is that all of the information will change between passes, and thus the equation is taking that into consideration. In the case of cold migration, *R* doesn't need to be transferred at all (i.e., *i* is zero), and thus the equation simplifies to simply *C*, documented in Equation (7).

$$T_{CM}=C+i(R)=C+0(R)=C \tag{7}$$

The size of the configuration file is usually small (a few hundred bytes) that contains a list of the path to the VHD, the physical specifications of the VM such as RAM, number of vNICs and other information needed by the hypervisor to bring the virtual machine up and let it run. Thus transferring *C* is a small amount of overhead, compared to the contents of gigabytes of RAM in a live migration or a VHD in a block migration. It is almost negligible in the grand scheme of a live migration. Thus, Equation (6) can be modified into Equation (8). While in effect, live migration should be slower than cold migration due to the fact that $T_{LM} \gg T_{CM}$, it is more commonly used in production environments due to its benefits mentioned above.

$$T_{LM} = i(R) \tag{8}$$

3.2 Virtual Machine Average Load

When considering the virtual machines, measuring and considering their load can be done in different ways such as user load, which will consider how many concurrent users are accessing physical sessions on virtual machines located on a hardware server (such as remote desktops, application requirements), measuring the number of requests (for a file server or a database server) or examining VM performance metrics such as the virtual CPU (vCPU) and usage of the RAM assigned to the VM. In terms of a backend sense, one of the most useful metrics for load is examining network usage and the bandwidth that is required. Because the logical characteristics of a VM are allowed to be changed via the hypervisor dynamically (whether or not it requires turning the VM off is not necessary to be considered, though from a user perspective it is important), the more physical backend measurements can often be considered more vital, as they require more investment to change. Thus, creating a series of VMs that have their network needs optimized can arguably be considered important.

When the OpenStack suite is installed, it leverages an existing hypervisor on the system, such as KVM on Linux or Xen from VMWare. As such, images need to be readable by OpenStack so that the system can deploy virtual machines based on that image. There are many different images that work with OpenStack, including a variation of Linux called CirrOS, which is specifically made for cloud test environments such as OpenStack [19]. For testing purposes, this is the VM image that was used to create virtual machines on the deployment. Each virtual machine can be configured with a disk size and RAM allocation, as well as dictating the number of CPUs it can use on the host's chip. Though this is set up through OpenStack, it uses the underlying Hypervisor to set up and be allowed to control the virtual machines.

3.2.1 Average Load

3.2.1.1 Network Load

Since CirrOS is a custom built Linux kernel and extremely pared down from a fully fledged operating system to use, there is very little in the way of proper documentation for it, though its code is open source and easily locatable on the internet [20]. Thus it took some trial and error in order to determine exactly the process for how networking with respect to virtual machine boot up and the load that it would generate on the network.

On boot, CirrOS will send a DHCPDISCOVER packet to its locally attached switch intended to find the DHCP server on the network. This discovery will wait for a response for 60 seconds before considering the discovery a failure. In the case of failure, packet retransmission is done. The discovery will be attempted a maximum of 3 times. The DHCP failure case was not examined in any detail, because once the DHCP failure occurs, no more network information is generated with respect to the VM upon boot. The CirrOS instance will have no IP address, no gateway and no other networking information. The minimal traffic afterwards is thus from ARP requests, or IP traffic if networking information is manually entered into the VM, which was not done.

On a successful case, however, when DHCP information has been relayed from the switch to the host and has been accepted, the handshaking takes a total number of three more steps post discovery (OFFER, REQUEST and ACKNOWLEDGE), as done in the typical handshaking process. In general, traffic then depends on the usage of the VM. Idly, like in the false case, it will only reply to ARP requests and broadcasts, and handle IP inquiries from other devices sent over the networks. When it is in use, however, the traffic will vary depending on the reason the server is set up for; if it is a file server, the files transferred to other servers is network traffic. If it acts as a web server, it will generate the content in response to each request it would receive. This traffic is dynamic, and for a small deployment, will be very small compared to the notion of migrating virtual machines from server to server.

3.2.1.2 Host Load

In general, the host load is the load that the virtual machine puts on the host, in terms of resource use. When the number of virtual machines on a host gets too large for the system to handle, that is when the system automatically will initiate a failover in order to balance out the load between other virtual machine hosts that the server is aware about, and in the course of this failover, will migrate a number of virtual machines. This process is done strictly by the Hypervisor, but given OpenStack's integration with the hypervisor, it will monitor the location of the virtual machines and display it on the monitoring dashboard in real time.

Chapter 4

Optimization of Load on Leaf Nodes

Knowing the background of the network topology and virtual machines with their respective hypervisors, the question, as it has always been, is optimizing the virtual machine load on the servers such that the performance of the virtual machines is optimal and network traffic is not impacted in a negative way with unnecessary overhead. Being able to optimize this will allow for a balance of hypervisor and network performance which is necessary in today's business environment. In the past, networks were vital, but as mentioned, today, their need has been increased to a point where most services require some form of Ethernet to work, so ensuring that the network is free of unnecessary data flow is important. This includes extraneous communication from virtual machines during VM transfer.

4.1 Migration in a Live Setting

In the past, the primary way of optimizing virtual machine performance was managing the host load and ensuring the hypervisor was not completely utilized or overcommitted (where the requirements for the virtual machines on the hypervisor are more than the underlying physical hardware of the hypervisor is able to give). Live migration is used in modern day systems [21], with hypervisors, or in some cases the virtual machine managing applications on top of the hypervisor (such as OpenStack or Microsoft Virtual Machine Manager) automatically migrating VMs on overloaded systems to less loaded systems (in terms of resources). This has become possible recently due to the advent of live migration, even though this automatic transfer happens regardless of the fact of whether the VM is turned on or off. In the past, it would require manual administrator intervention to transfer the machines due to the need for them to be manually shut down before transfer could happen.

When dealing with live v. cold migration, the question could be asked of which is better. As previously shown, cold migration takes significantly less time for the transfer process to be completed (since the live migration transfer time is proportional to RAM [17], which is shown in equation (8) in section 3.1) but requires more time for shut down and boot up sequences, as well as the immeasurable statistic of user inconvenience. Live migration, according to studies, also can slow down VM performance during the migration and can have minimal downtime of seconds for situations where there are high concurrent users [22]. However, all main hypervisors (regardless of the company of origin), when put into a clustered environment, automatically live migrate in order to load balance appropriately, based on thresholds set by the administrator. In non-clustered environments, this does not happen. However, if the assumption is made that each of the servers attached to leaf nodes in the above topology are clustered together (a valid assumption, given that clustering has become common place in IT environments and has been since 2000 [23]), then automatic live migration will be triggered. When this live migration is triggered, the only information that the hypervisors take into account is the 'health' and available resources of the other machines in its cluster, to ascertain the most appropriate host to transfer a VM to. It does not take into account the network state, which now becomes the crux of the problem.

4.2 Taking Network State into Account

As previously shown in chapter 2 of this thesis, the bipartite mesh topology that has been examined has a minimum of two traffic hops, as well as a theoretical best of two traffic hopes for traffic to get from the source to the destination for any source and any destination located on a leaf node. When dealing with natural traffic flow to outside of the network, traffic from the source to the gateway is also of two hops. Virtual Machine migration, no matter the type, must follow the network topology to get from the source hypervisor to the destination hypervisor. Though the virtual machine managers on the hypervisor take into account the state of the destination, they do not take the network state into account. To be truly optimal, network state should be taken into account such that the number of hops is minimized. As shown above in equation (8), the virtual machine for a live migration needs to transmit *iR* bytes worth of data; minimizing the number of hops will help minimize the delay of the transfer, because *iR* bytes of data needs to be transmitted over every hop. The time to transmit that live migrated VM data over a link between hops can be ideally modelled as done in Equation (9), below, which takes the data from Equation (8) and divides it by the link's bandwidth. This is not taking into account retransmission due to flow control or errors. In general, this equation can differ between links due to the fact that a link's maximum bandwidth (BW) can be different. However, for the sake of simplicity, we will take all links in the mesh to be of the same bandwidth.

$$\frac{iR}{BW} \tag{9}$$

Ideally, taking network state into account when moving virtual machines from one location to another would be an easy procedure but as it stands today, it is difficult to balance the leaf hypervisor load and the network load. For example, you cannot start transferring the virtual machine to the closest leaf (in terms of hops) and hope the hypervisor is able to accept the virtual machine because if it cannot, time and resources have been wasted and it also increases overhead on the network. Furthermore, the virtual machine cannot be transferred as currently done to the least loaded hypervisor, since that hypervisor could be only reachable by a large number of hops on a less than optimal path. Note that if the hypervisor was completely unreachable, the source would not receive any heartbeat messages from it and thus consider the server 'dead' until is it once more reachable. Today's optimization occurs solely in finding the least loaded hypervisor, but true optimization for the network would involve finding a balance between the load on the network and on the hypervisors; to find an acceptable place for the migration of the virtual machine which does not waste network resources or cause undue delay.

The virtual machine manager on the hypervisor should want to transmit the virtual machine information to its leaf and the leaf, having information about its neighbours and the state of the topology, should query the leaf with the shortest distance and the hypervisors located on that leaf should be examined to determine whether they can support the virtual machine being migrated. If the hypervisors on that leaf cannot support the new virtual machine or are close to the thresholds set by administrators in terms of load, then the next closest leaf should be queried. When an acceptable hypervisor is found, the virtual machine is migrated. This ideal situation would be difficult to implement, especially for industry which already has current systems in play, but it would properly balance the load on the network and on the hypervisors simultaneously. Other options exist and allow for many variations on this theme.

4.3 Shortest Path and Options to Transfer Information

Though there are many routing and switching protocols that are meant to find the shortest path through a network via metrics, as mentioned, to explicitly define the time it takes for a virtual machine to be transmitted from the source hypervisor to the destination, Equation (10) is generated, which is simply the sum of the time to live migrate the VM over a hop for all hops over the shortest path from source to destination. If the assumption can be made that all links in the network are the same bandwidth, this equation simplifies to the number of hops multiplied by the time taken to transmit the VM over a single hop. However, this assumption can not be taken as fact due to multiple configurations of the bipartite mesh possible, depending on the way it is implemented and the resources put into it.

$$\sum_{x\ hops} \left(\frac{iR}{BW_x}\right) \tag{10}$$

In order to obtain optimal behaviour on the network load, i.e. to minimize the overall delay and traffic required, equation (10) needs to be minimized. By minimizing it, the overall delay will be minimized. This isn't necessarily a correlation of the smallest number of hops due to the fact that bandwidth can vary widely and links with larger bandwidths take precedence in metric systems [5, 24], however, if the number of failed links (or maximum hop) limit is taken into account, then the overall minimization can be considered acceptable to whoever defined that limit. One such way to do that is by keeping track of the smallest number of hops between devices in table form in the switch, as devices do for routing information. With this information, the design could be such that when a hypervisor wishes to transfer a virtual machine automatically (without administrator intervention, as they can manually specify a destination without question), a 'transfer request' is sent to the attached leaf node, but instead of forwarding that transfer request to the destination hypervisor as chosen by the virtual machine manager on the hypervisor on the source, as would be done in present systems, it would be sent to any hypervisor on the shortest path. If one of the hypervisors have the resources to accept it, that information is returned back in an acceptance message and the transmission begins. This is a 'network first' approach to the minimization of the problem, where the network state is taken into account before the hypervisor, and is a complete reversal of the modern day paradigm where all that is considered is the hypervisor. There are obviously many tradeoffs with this type of design. As leaf numbers grow, the table holding the shortest path pairs between sources and destinations will grow and take up more operating resources on the switch. At worst, if the table includes information for all sources and all possible destinations, it will grow exponentially with any new leaf. If this table only includes information about its leaf as the source, then with new leaves added it will grow linearly. Furthermore, the hypervisor might become overcommitted if there are other hypervisors on the same leaf attempting to utilize it for live migration of their own (which would be the first attempt, given that if there were hypervisors on the same leaf, that would be the first attempt, with a hop count of one, or zero depending on the definition).

A second possible solution would be to modify the current paradigm of looking at the least loaded hypervisor in the cluster first, and then having the source leaf determine whether the TTL would fall within acceptable parameters for the limit of maximum number of hops. If it does not fall within acceptable parameters, then the next least loaded hypervisor would be chosen and the same determination performed; this would continue until a hypervisor that is both within appropriate range as well as with a sufficient amount of resources is located. There is no way to determine whether the live migration would actually be the lowest possible transfer time but it will always be transferred and accepted by the destination, barring emergencies like another hypervisor going completely offline and multiple VMs flooding the network in an instant. This is a 'hypervisor first' scenario, where the hypervisor is taken into account first and the network state is then taken into account. Arguably this could be considered easier to implement, given that it's an addition to the current operational behaviour of the hypervisor, while the 'network first' scenario would be a complete change to how things work. Furthermore, these parameters for maximum acceptable TTL would need to be implemented beforehand so that the solution could work, and all devices would need to know these acceptable parameters. Querying the hypervisor isn't necessarily a problem since that is simply an ARP request, and as previously mentioned, the switch can easily find a best path.

The odds that the second solution will be optimized in terms of the shortest path subject to link failures is not necessarily the ideal assumption of $\frac{(l-1)}{l}$, as that assumption takes into accounts that all failed links congregate on one path instead of randomly being spread over the entire network - for example, a path could have many paths of cost three but a single path of cost two. As the number of spine leaves increases and thus more increase the alternate ways to get from leaf to leaf, this is especially true, though in cases with a smaller number of spines, the ideal assumption has a higher probability of being held true (a smaller number of paths means that even though failed links would be placed randomly, they have a higher probability of being located on the same path, and as the number of paths increase, this probability decreases).

4.4 Dealing with Overhead

Both of the aforementioned solutions - network first and hypervisor first - both have overhead if initial placement fails. For the scenario taking the network into primary account, if the destination hypervisor(s) have no room on the selected leaf, then a new leaf has to be selected and those hypervisors checked, while the 'hypervisor first' scenario has overhead if the destination hypervisor is logically too far away for the administrator defined thresholds for maximum hops. The question posed is to determine which type of overhead causes more problems. It should be noted that in both of the cases, the virtual machine is not migrated before an acceptable destination is found, and as such, the bulk of the virtual machine as outlined in equation (8) is not transferred, as was originally worried about (and is currently worried about) in present day situations. This alone would cut down on a significant amount of overhead, compared to certain situations in present day environments.

In terms of information that is transferred, there is the request to find an acceptable hypervisor to place the virtual machine. In the network first situation, the source leaf node would have to choose the shortest path and obtain MAC addresses (or, in a layer three environment, IP addresses) of the host(s) at the destination of that path so that it could report back to the source hypervisor; the hypervisor would then check its information about those hosts and locally determine whether one is acceptable to migrate the virtual machine to.

In contrast, the hypervisor first scenario would perform that determination first (as is done in modern environments), then the source leaf would need to determine the leaf the destination is located on and then determine whether it falls within defined parameters. In this case, defined parameters could easily be set to a maximum hop count based on the worst case *f*. From previous examination in section 2, we know that f_{max} < s for system stability, and the moment $f_{max} = s$, the system has a non-zero probability of failure. Defining a threshold shown below in equation (11), where f_{max} is considered to be (*s - 1)*, we obtain the maximum hop threshold.

$$h_{\max} = f_{\max} + 2 = (s-1) + 2 = s + 1 \tag{11}$$

This approach could easily be done by the source leaf looking at its table and, having the defined threshold, then either allowing the transfer or communicating back to the source host that the considered hypervisor is not a potential candidate. This involves less communication over the network, versus local communication which wouldn't count to overall network traffic. However, it would put calculation burden on the source node, however minimal.

As previously mentioned, the time to transfer the virtual machine as outlined in equation (10), would vary slightly, depending on the number of hops. The network first solution would ensure that the number of hops x in equation (10) is as minimized as possible, while the hypervisor first scenario could not guarantee that x would be optimal. The communication between the source leaf and the various potential destinations is minimal in terms of data transferred (and thus the time it takes), but is also considered overhead. Though with the scale of the transmission of the RAM of the virtual machine over the network, this 'set up time' as it may be called, is negligible as it would be on the scale of bytes transferred (and for bandwidths of 1 Gbps or above, it would be a matter of nanoseconds, compared to the order of seconds required to transfer the contents of a virtual machine even a single hop). This optimality is important, however the design of the system would be difficult to incorporate into modern architecture, so any backwards compatibility for hypervisors and their virtual machine managers would be difficult. Of course, given OpenStack as an emerging technology, and one that is open source on top of it, it allows for changing this system in subsequent version of the applications by the community. While this would not work on closed systems such as Microsoft's Hyper-V, VMWare's ESX or Citrix, it would be an initial starting point to be able to test, and if acceptable, would be a feature in new versions. Incorporating these changes would be an undertaking, but the hypervisor first solution requires minimal changes to the hypervisor itself, since it is building off of the current paradigm and not changing it entirely, compared to the network first scenario, due to the back and forth communication that the latter has. While this document does not go into depth into these costs of labour, production and development time, they are real world factors which must also be taken into account. As such, the development process for the network first scenario would theoretically be more complicated, involving a redesign of more parts to work together while simultaneously keeping their current functionality.

Chapter 5

Conclusions And Drawn Information

Virtual machines are the future trend for multiple reasons such as stability, cost effectiveness, reduced power consumption and a lowered green footprint. Both desktop virtualization and server virtualization will continue to grow and as a result, a way to improve the virtualization of today is vital to future growth and optimization. The idea of virtual machine migration in a clustered environment may be relatively new in terms of technology due to its advent in the mid 2000s, however it is quickly becoming the standard for how to ensure uptime and stability for end users. This technology itself has evolved, providing benefits and drawbacks of the various types of migration, but further work to ensure that it can keep up with the increased demand and the situations that will arise in the future.

Furthermore, in the ways that virtual machines need to continually improve to meet the future demand, back end networks are also vital. Arguably, they have been vital since they were implemented, but with technology such as iSCSI and the importance of NASs and SANs, they are becoming even more vital, requiring a need for greater service that they can provide. This involves keeping the networks free of obstruction, making networks reliable, and ensuring that speed can meet the requirements of the system and the end users who are using it. Networks, especially ones with solid foundations like the concept of the average two hop bipartite mesh, can be optimized to deliver the level of service required in a way that will meet the needs of evolving requirements.

This thesis has attempted to examine potential ways in which both the virtual machines and the network can be optimized in order to meet these requirements going forth. The idea of the technology might be new, but moving forward can only be beneficial.

5.1 Conclusions

It was concluded that in terms of the virtual machine migration, there are advantages to both the live migration and the cold migration; namely that live migration can be done while the VM is powered up and in a state of use, but cold migration is faster. In terms of an 'always on' environment, the former is more important and the latter loses time in the required shut down and start up of the server that would have to happen before and after the transfer, respectively. Furthermore, live migration is a common offering from all major hypervisors, so there is no adoption issues to worry about in the future.

Virtual machines cause load on both their hosts as well as the network, but the network load is analogous to every day usage for any other server - that is to say, fully dependent on the applications that it runs. There is additional traffic to and from the NAS or SAN where the VHDs are stored so that changes can be made to them, and the network needs to take that into account, however that type of traffic is not limited to virtual machines due to the every day usage of networked storage.

Furthermore, from a network point of view, it was concluded that any number of failed links need to be bounded in order to ensure that the topology remains stable and does not fail; this boundary of failed links must be less than the number of spine switches, or else the probability of transmission failure between a source-destination pair becomes non zero and if that number of failed links were to increase, the probability would grow. This upper bound on failed links is correlated to an upper bound on the number of hops due to the fact that there is a linear relationship between them ($h_{max} = 2 + f$).

This h_{max} will only be reached in specific cases where the upper bound for f is hit and even then, due to the fact that there are multiple paths between source and destination, and other paths that are completely unrelated to the source and destination, not every scenario will result in a maximum hop count.

It was concluded that a network first approach to the problem of optimizing the migration of virtual machines over this type of topology would fully optimize the network load, minimizing the number of hops to the smallest number possible while still finding an acceptable hypervisor to migrate the virtual machine to. This optimization would allow for the smallest possible transfer time of the virtual machine from the source to the destination. However, this process is one that is more difficult to consider in terms of working with legacy systems, due to the fact that the paradigm and idea of the design isn't inherently compatible with what is currently in environments.

Conversely, a hypervisor first approach to the problem would not necessarily guarantee the shortest transfer time of the virtual machine, due to the fact that the least loaded hypervisor would be considered priority and then the network state would be checked. It would, however, be easier to implement as it would be built on the current environment. This tradeoff for a little less network optimization for backwards compatibility and less computation on the leaf nodes should be considered.

5.2 Future Work

While this work has postulated potential solutions for the optimization of virtual machine migration over a bipartite mesh, there is still far more work to be done. Both solutions need to be developed and tried in an environment such as OpenStack which would allow for the community to add features and perform stress testing. This open source community is more inviting than closed source hypervisors and virtual machine managers like Hyper-V, ESX and others. It also allows for greater minds to have input into this problem and the solutions in order to determine the best one to be implemented, as well as give numerical context to the comparisons. This allows for the expansion of ideas and convergence of multiple minds working together to make the optimization better. It also allows for the idealizations to be put to the test and to determine how accurate models are compared to real world scenarios with imperfect systems.

Further extrapolation could also be done on the topology itself, with rigorous proofs to be done to find general formulas for the average probability of failure as f increases beyond its maximum bound, the probability of hop increase to the maximum h_{max} and other factors. This work could also be extended to the optimization of virtual machine migration taking into account any network topology, as many areas would not necessarily adopt this topology versus others which are still popular to this day.

Bibliography

[1] Cisco Systems Inc. "Cisco Virtual Networking Index: Forecast and Methodology, 2013-2018," June 14, 2014, pp8-12. [Online] Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf. Accessed: December 28, 2014.

[2] Mark F. Mergen, et al. "Virtualization for high-performance computing." *ACM SIGOPS Operating Systems Review* vol. 40, issue 2, pp. 8-11, 2006.

[3] *IEEE Standard for Ethernet,* IEEE 802.3-2012, 2012.

[4] Intel Pro Network Connections, "10 Gigabyte Ethernet Technology Overview," 2003. [Online]. Available: http://www.larryblakeley.com/Articles/networks/pro10gbe_lr_sa_wp.pdf. Accessed: October 29, 2014.

[5] Cisco Systems Inc., "Section 3.1: OSPF Cost." *OSPF Design* Guide, August 10, 2005. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html. Accessed: January 14, 2015.

[6] Narsingh Deo, "Chapter 2: Paths and Circuits," in *Graph Theory with Applications to Engineering and Computer Science.* India: Prentice-Hall of India Pvt.Ltd, October 15, 2004.

[7] Cisco Systems Inc., "Cisco Nexus 1000V Series Switches for VMWare vSphere," 2014. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-1000v-switch-vmware-vsphere/data_sheet_c78-492971.pdf. Accessed: January 14, 2015.

[8] Gunter Bolch, et al. "Chapter 6: Single Station Queueing Systems." *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Hoboken, NJ: John Wiley & Sons, 2006.

[9] Roch Guérin and Ariel Orda. "Computing shortest paths for any number of hops." *IEEE/ACM Transactions on Networking (TON)* vol. 10, issue 5, pp. 613-620, 2002.

[10] - Tomislav Petrović and Fertalj Krešimir. "Demystifying desktop virtualization." *Proceedings of the 9th WSEAS international conference on Applied computer science*. 2009.

[11] Tom Clark, "Chapter 1: Introduction," in *IP SANs: a guide to iSCSI, iFCP, and FCIP protocols for storage area networks*. USA: Addison-Wesley Professional, 2002.

[12] Jon Tate, et al. "Chapter 6: Storage Area Network as a Service for Cloud Computing," in *Introduction to Storage Area Networks and System Networking, 5th edition*. USA: IBM Redbooks, 2012.

[13] - Tom Fifield, et al. "Chapter 4: Compute Nodes," in *OpenStack Operations Guide.* Sebastopol, CA: O'Reilly Media, Inc, April 21, 2014.

[14] Tom Fifield, et al. *OpenStack Operations Guide*. Sebastopol, CA: O'Reilly Media, Inc., April 21, 2014.

[15] Kenneth Hui, et al. *OpenStack Architecture Guide*. pp25, 2014. Available: http://docs.openstack.org/arch-design/arch-design.pdf. Accessed: September 3, 2014.

[16] Bhanu P. Tholeti. " Hypervisors, virtualization, and the cloud: Learn about hypervisors, system virtualization, and how it works in a cloud environment," September 23, 2011. [Online]. Available: http://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/cl-hypervisorcompare-pdf.pdf. Accessed: October 29, 2014.

[17] Christopher Clark, et al. "Live migration of virtual machines." *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005.

[18] Zurich University of Applied Sciences. "An analysis of the performance of live migration in OpenStack," September 18, 2014. [Online]. Available: http://blog.zhaw.ch/icclab/an-analysis-of-the-performance-of-live-migration-in-openstack/. Accessed: October 29, 2014.

[19] OpenStack Foundation. "Chapter 2: Get Images," in *OpenStack Image Guide*. May 28, 2013. Available: http://docs.openstack.org/image-guide/image-guide.pdf. Accessed: September 3, 2014.

[20] Scott Moser. "CirrOS a tiny cloud guest." [Computer program]. Available: https://launchpad.net/cirros. Accessed: January 4, 2015.

[21] Haikun Liu, et al. "Live migration of virtual machine based on full system trace and replay." *Proceedings of the 18th ACM international symposium on High performance distributed computing*. ACM, 2009.

[22] William Voorsluys, et al. "Cost of virtual machine live migration in clouds: A performance evaluation." *Cloud Computing*. Berlin, Germany: Springer Berlin Heidelberg, pp. 254-265, 2009.

[23] Trevor Schroeder, Steve Goddard, and Byrov Ramamurthy. "Scalable web server clustering technologies." *Network, IEEE* vol. 14, issue 3, pp. 38-45, 2000.

[24] Jeff Doyle, Jennifer Carroll. "Chapter 4: Dynamic Routing Protocols," in *Routing TCP/IP, Volume 1, 2nd Edition*. Indianapolis, IN, USA: Cisco Press, 2006.