Network-in-a-box: An excursion into a virtualized world

M a r t i n J o h a n s s o n

Master of Science Thesis

Stockholm, Sweden 2010

TRITA-ICT-EX-2010:67

Network-in-a-box: An excursion into a virtualized world

Martin Johansson
Master's Thesis in Information Technology
School of Information and Communication Technology
Royal Institute of Technology
September 2009 – February 2010

Supervisors
Bob Melander, Ericsson
Jan-Erik Mångs, Ericsson

Examiner
Dr. Johan Montelius, KTH

Abstract

This thesis goes deeper into how to create a big virtualized network in a distributed cluster of physical nodes using Xen hypervisor. It should not matter on what Xen node the VMs are located; the virtual network should work in the same way anyhow. By still keeping a simple user-friendly interface while having the advanced technical mechanisms running in the background is a key goal in this project. This thesis guides the reader through how to create a distributed virtual network using Xen hypervisor and it also goes through all the problems that we stumbled on during our work. This report also goes into how to make the virtual network more usable in a simulation environment by using OpenFlow on all the switches. OpenFlow is a new technology which gives switches more control of the packets passing through.

Setting up hundreds of VMs is a time consuming job when they need to be configured individually. Having pre-configured roles will reduce this configuration time drastically. When a VM is created it is set to a specific role. The roles specify operating system, software and configuration on the VM. Our complete solution and implemented software includes all this parts to decrease the work of deploying a big virtual network.

Acknowledgements

This master thesis report is the end of my studies at KTH and at the same time a start of my working career. It's been a great four and a half years studying at KTH. I want to thank friends and family who have supported me all these years.

Special thank goes to those who have been working close to me in my thesis work;

My Co-Worker in this thesis

Daniel Öman

Examiner at KTH

Dr. Johan Montelius

Supervisors at Ericsson

Bob Melander

Jan-Erik Mångs

Contents

1 INTRODUCTION 1

1.1 Problems description and requirements 1
1.2 Background 4
1.3 Xen 4
1.4 Management Software's for Xen 5
1.5 Thesis Outline 6

2 IMPLEMENTATION AND DESIGN 7

2.1 Get familiar with Xen terminology 7
2.2 Using several Xen nodes 8
2.3 Making communication links into tunnels 10
2.4 Distributed switch problem 11
2.5 Adding OpenFlow into a virtualized world using Open vSwitch 14
2.6 Network topology used in our test environment 16
2.7 Backend 18
2.8 Shared Storage between virtual machines 20
2.9 The NFS Image Storage Solution 21
2.10 Communication protocol frontend to backend 22
2.11 Tunneling software used 22
2.12 Boot order of VMs 23
2.13 GUI 23

3 TEST AND VERIFICATION 25

3.1 Verify traffic leakage using one Xen machine 25
3.2 Verify traffic leakage using two Xen machines 25
3.3 Test setup with two Open vSwitches and three Xen nodes 25
3.4 IP conflicts between TAP and Virtual Network 27

4 CONCLUSIONS 28

4.1 Distribution 28
4.2 OpenFlow 28
4.3 Shared storage 29

5 IMPROVEMENTS AND FUTURE WORK 30

5.1 Test network 30
5.2 Communication protocol 30
5.3 Roles 30
5.4 Shared disk 30
5.5 Mount over network 30
5.6 Graphical User Interface 31

6 REFERENCES 32

List of figures

Figure 1 - Problem with sending packets between Xen nodes 2
Figure 2 – Network topology supported by OpenNebula 6
Figure 3 – Xen bridge with physical interface attached 7
Figure 4 – VM with two NICs attached to bridge 7
Figure 5 – Test network topology with three VMs and two switches. 8
Figure 6 – One Xen node with three VMs 8
Figure 7 – Setup of three Xen machines with one VM running on each of them 8
Figure 8 – Packets not being sent correctly 9
Figure 9 – Packet sent to wrong destination 9
Figure 10 – Tunnelling between Xen nodes 10
Figure 11 - Using tunnels to ensure correct packet delivery 10
Figure 12 – Virtal network setup with three VMs connected to one switch 11
Figure 13 – Describing the solution to having distributed switches to look as one 11
Figure 14 – Removed link between two switches for broadcast messages only 12
Figure 15 – Complete solution to our distributed virtualized network problem. 13
Figure 16 – OpenFlow controlled switches 14
Figure 17 – Network topology 16
Figure 18 – Example of Open vSwitch test setup 17
Figure 19 – Flowchart showing how the process of setting up a VM is made on the backend 18
Figure 20 – Flowchart describing the algorithm to setup VTUN. 19
Figure 21 – Describing Algorithm for setting up VTUN using an example with three Xen nodes. 19
Figure 22 – Directory structure on NBD file server 21
Figure 23 – Virtual network topology in GUI 24
Figure 24 – Physical network topology in GUI 24
Figure 25 – Topology used to verify packets were sent correctly 25
Figure 26 – Two Xen machines used to verify no traffic leakage 25
Figure 27 – Test network used to verify our implemented software 26
Figure 28 – Change OpenFlow paths in the GUI. 27

List of abbreviations

VM	Virtual Machine
BR	Bridge
STP	Spanning Tree Protocol
NFS	Network File Storage
NBD	Network Block Device
GFS	Global File System
IMG	Image
Dom	Domain
FTP	File Transfer Protocol
LAN	Local Area Network
NIC	Network Interface Controller
MAC	Media Access Control
CLI	Command Line Interface
GUI	Graphical User Interface

1 Introduction

More and more enterprises and network operators are deploying their systems using virtualization platforms, meaning that operating systems and other applications are no longer running directly on top of the hardware. The reasons are manifold, to reduce operational and equipment costs and make more efficient utilization of underlying hardware. Many of these systems are getting increasingly complex, spanning hundreds of nodes and a multitude of networks. It is therefore not a trivial task to design, deploy and operate them.

However, existing tools for management are mostly focused on operation of virtualized server installations and provide little help in designing and deploying them. Specifically, more advanced aspects of network connectivity (VLANS, routers etc) are not covered.

There are many different virtualization machine monitors (VMM or hypervisor) both proprietary and open source. We wanted an open one in case we would need to alter the virtualization software somehow. In this thesis we did not have the opportunity to choose one. The choice had already been made before the thesis started, so there will be no comparison of virtualization software. The hypervisor we use in this thesis is Xen. It is an open source project and is originated from University of Cambridge and the first public release was made in 2003 [1].

OpenFlow is an "add-on" for commercial switches that enables researchers to run experimental protocols on the networks without exposing it. It also supports the traffic being re-routed when a device moves between two network-access points to get the lowest possible latency. OpenFlow is a newly started concept and it's only a few physical switches that support this and they are all in the research stage. Open vSwitch [14] is a multilayer virtual switch. The important thing with this switch is that it is supporting OpenFlow [15]. OpenFlow is a protocol that can be used by switches to control traffic in the network. Since switches are talking on layer two, the OpenFlow rules can be specified by using MAC addresses. Each OpenFlow switch can have its own flow rules, but it can also let the controller decide what to do with the packet [16].

This thesis goal is to research and implement in the area of how to make it easier to deploy large scale virtual networks over a cluster of Xen nodes. The focus in this thesis is going to be about how we can create a virtualized network that is distributed and spread out over a cluster of physical nodes and all the VMs inside this virtual network should have a shared memory which they can use to configure or updating their roles in the network. Our aim is to make it easier for the administrators to setup new simulations of networks.

1.1 Problems description and requirements

1.1.1 Distribution

We want to have a private virtualized network. These networks can contain a lot of VM's and to make this network run on only one server is not feasible. We need therefore to have several servers that can host VM's. We want to have a solution where it doesn't matter on what server the VM is hosted. The virtual network topology should not be requiring a certain or several VM's to be hosted on a specific server. Since our chosen technology to host VM's supports migration between the servers it will be a good load balancing solution if all the VM's can function on all the different servers.

We realize that when having a virtualized network that is spread out over several servers we could get some problems when sending traffic from the virtualized network on the physical network. A switch in the virtualized network could be placed on several servers depending on where the VM's are hosted which is attached to this switch. Therefore traffic is needed to be sent over the physical network when the virtual switch is sending packets from a VM to another.

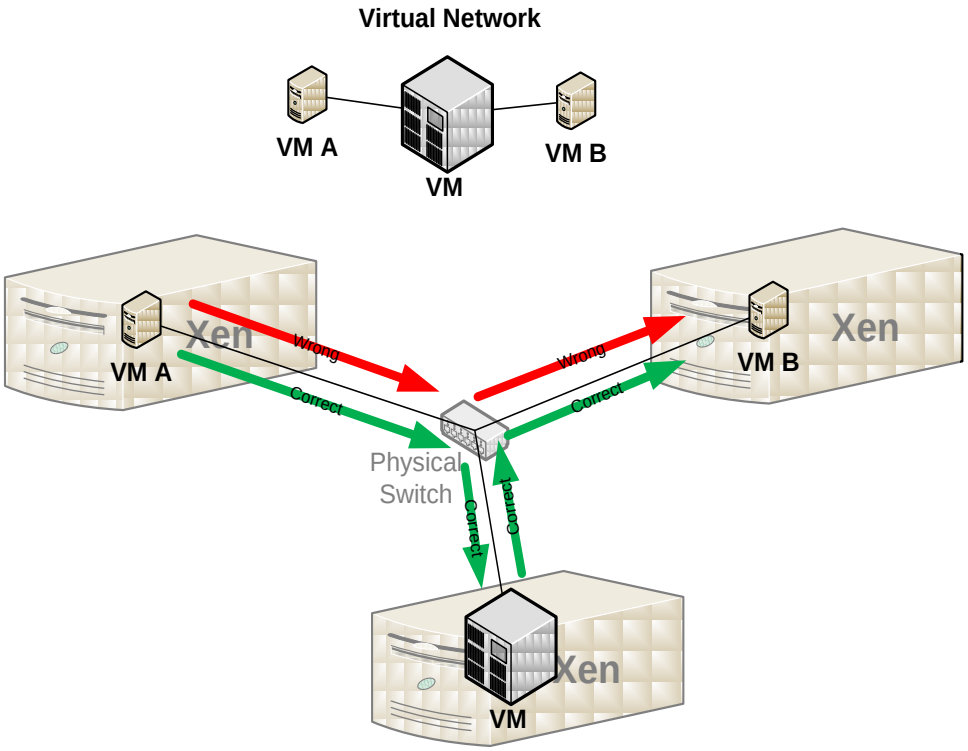
When a package is sent, we want it to take a certain route through our private virtual network. Because the package must be sent through the *physical* network, we must make sure the package does

not get routed directly to its destination; we need to come up with a solution. The reason the package would get switched directly to the destination is that the physical switches have MAC learning enabled which will make them know where all connected nodes are in the network.

The following figure (Figure 1), shows what could happen if a network is setup as following; VM A wants to communicate with B. There is a VM between these two machines that will act as a router and route all allow all traffic to go through it. In the physical world all these three VM's was placed each on its own Xen host. This is a very possible scenario if you want to have some kind of load balancing.

With this setup the following problem will occur: When A sends a package to B, the physical switch in between have learned where VM B is and will send it directly to B. So the package will then follow the *(Wrong)* path. Though we have in our virtual world created the network such as it would take the *(Correct)* path, which will never occur.

Figure 1 - Problem with sending packets between Xen nodes



Requirements

During the first period of this thesis project pre-study and experimental work was made. During this time we defined requirements of what our project should contain.

GUI specific requirements

- **Give the user an easy way of creating and viewing the network topology of both the virtual network and the test network**
  Build a GUI in Java to draw a virtual network topology.

- **Send requests to the backend**
  To deploy the VMs there should be a way to deploy the drawn network topology.

Backend specific requirements

- **Accept requests sent by GUI**
  It is the GUI that decides what should be executed on the backend, therefore it needs to listen to the requests made by it.

- **Software should be multithreaded and support handling of multiple requests in parallel**
  To speedup the heavy process of starting VMs.

- **Communicating with multiple Xen machines**
  We want a distributed solution by using multiple Xen machines.

- **Use Libvirt API**
  The Libvirt API is a powerful and very basic way of managing VMs on a hypervisor. It supports plenty of different hypervisors.

- **Support VM migration between Xen nodes**
  To get a better load balance of the servers its needed to move the running VMs and also doing so without shutting it down.

- **Create tunnels between the virtual switches (bridges) of a Xen machine to all other Xen machines**
  So that the packets get safely to its destination.

- **Add OpenFlow enabled switches into the test network and control the traffic flows with OpenFlow rules**
  To simulate traffic speed and latency.

- **Exchange the Linux Bridge with Open vSwitch which is OpenFlow enabled**
  So that we can use OpenFlow to control the VMs traffic.

1.1.2 Shared Storage

A configuration of a VM can be done before its setup and started on Xen. But often you want to change something after it's started. You could login to the machine and make your changes and then you're done. But when you have over one hundred VM's this process is going to take way too long time. So what we then want is to have a shared storage that the VM's can look at and see if they should be updated. Then all that has to be done is change at one place and after that all VM's having this shared storage will get updated with the new software and configuration.

Requirements

- **Storage is reachable by all VMs**
  Make sure that all the VMs will have the disk attached and be able to read data from it.

- **Directory structure should be expandable**
  The structure should be made so that new functionalities could be added without changing any existing features.

- **Changes to the storage should be seen by the VMs**
  The VMs should be able to update new software and configuration while they are running.

1.1.3 Roles

There can be different kinds of VM's in a network like, FTP Server, DHCP Server, Routers etc. To get a VM to be one of these we want to define different roles to them. The roles can include OS, Amount of RAM/CPU/Ethernet cards, Software, IP and so on. Applying these roles to the shared storage we can also get an easier configuration process and a much lower setup time of a VM. This would enable that large-scale virtual networks could be deployed much faster.

Requirements

- **Specify OS, Software, Configuration, CPU, RAM, IP, NIC**
  The chosen specification will lead to a role such as a FTP server.
- **Use the shared storage to save roles**
  Because the shared storage is shared between all VMs it is perfect to use it for the roles.

1.2 Background

The hardware for servers is becoming more and more powerful. Having one server and only one running operating system will not utilize that hardware at is max. The servers may only get utilized five to fifteen percent [8]. To increase that percentage there could be several operating systems running on the same hardware. Then the hardware can be used more efficiently. When the hardware is used more efficiently it is not needed to have that many servers as before. Having less hardware means less space in the server rooms and most important of all it makes it cheaper. This cost effective solution to use virtualized OS's on powerful servers have become of big importance in today's modern industry.

It's now not impossible to run several servers on one single server machine and of course the number of virtualized servers that you can run on one machine depends on the hardware specification of that machine. The advantages from running these much virtual servers can be taken within the simulation. Simulations can be done using real operating systems and real software installed to achieve the biggest possible real-like environment.

Processors with multiple cores are also having a big importance in virtualization. To run a lot of VMs in one server it's required to have the necessary processing power. The number of processors can be few because you can have a more than one core in each processor which makes the processors to fit into smaller servers. Since the hardware part has been taking such a sudden turn to stop increasing the frequency and instead increasing the number of cores, the software is lacking a bit behind and it cannot take the fully advantage of all cores.

1.2.1 Virtualization

For a long time it was not really possible to run more than one operating system at once in one computer. Though it has been possible for a long time to do virtualization (since the 1960th) [5] it has not really been so popular that it is starting to be right now. Before it was more used in research work to isolate the VM and also getting the advantage of better monitoring from outside than a physical machine. The hardware is getting better and better, especially the processors with its multiple cores which is extra beneficial for running multiple OS's on one machine.

1.2.2 Clouds

Nowadays we talk about "clouds" which means that we take a lot of physical machines that got lots of VM's running and connect this together over the Internet, all over the world. All together this creates huge processing power and can serve a lot of costumers everywhere in the world. Amazon EC2 is an example of this [7][18]. They provide a web service to run virtual operating systems in their cloud. A customer rents a space by hour and used resources. Google is another example of providing services in a cloud [6]. Their Appengine is a service where their customer can develop whatever service they want and run it in their cloud. The Appengine service is free of charge up to a limit of resources used per month, if you want to use more then you have to pay.

1.3 Xen

1.3.1 Xen Description

Xen is virtualization software, also called a hypervisor. On top of Xen you can run several guest operating systems that are executed concurrently on the same hardware. The performance is kept close with almost native performance. Xen places itself between the hardware and the OS. The first guest operating system is called in "Xen Terminology" domain0 or short dom0. Dom0 have access to the physical hardware and is booted automatically when the Xen hypervisor boots. From dom0 guest operating systems is started called domU's and they are being controlled and managed by dom0.[1] [2] [3]

Xen is not meant to be used by anyone that is looking for something which is fast to learn and user-friendly. For that there is a numerous of software's like Virtual Box and VMware. To use Xen you need to have a special reason. It's more suited as an automation system that is handling a lot of VMs. For us, the choice of Xen was made for us by Ericsson and it serves all our requirements well.

1.3.2 Xen Switching

To really get a fully functioning virtualized network you need to have all kind of network hardware virtualized, including network switches. Introducing switches into the virtualized cloud gives a more real like environment. When Xen is configured to be in "bridged mode" it's using the Linux Bridge as a switch for the VM's. Attaching interfaces to the bridge is the same as plugging in a cable if comparing to a physical switch. All the VM's on a Xen machine uses the bridge to communicate with other VM's. If VM's are attached to different bridges they can by default not communicate between the bridges. To enable communication between bridges, they need an interface attached that can reach another bridge through a network.

1.3.3 Live migration between physical servers

Xen supports migration and also live migration [3], which is very useful when having a distributed system. The load of the servers can be monitored and balanced by migrating VMs between Xen hypervisors.

Hardware on the Xen machines can be important for the migration to work. We discovered after testing it, that it can cause problems using different hardware. But it's supporting a migration between different hardware's up till some degree. For example migration between an Intel Xeon processor with fore cores to an Intel Core2Duo processor with two cores is working. But of course the best would be to have all the machines using same type of hardware, to not risk getting compatibility problems.

1.4 Management Software's for Xen

There are a numerous of different management software's out on the Internet that supports management for Xen. There was a small challenge to find the one that fitted our purposes most, we also had in mind that if we couldn't find any, we had to develop our own software. To control a big system of virtualized machines it's needed to have a reliable management system. You need it to control start/stop/deletion etc on VMs and setup of the topology. The topology can be rather big if using a lot of machines and because of that it is important to have a good management tool. The management software also needs to be able to control the system while it's running by doing load balancing and configurations on existing VMs.

Most of the management software's are basically telling what type of servers are running and some basic information of those such as OS, CPU, memory etc. This is not really what we were looking for to use in this thesis. What we wanted was a reliable and active project containing all the important parts that can help us as much as possible to setup our virtual network. We ended up with the three following software's described below. The one we finally choose to use was Libvirt, though we started to use OpenNebula in the beginning, but that one didn't fulfill all our requirements, so we left OpenNebula to build our own software using the Libvirt API.

1.4.1 OpenNebula

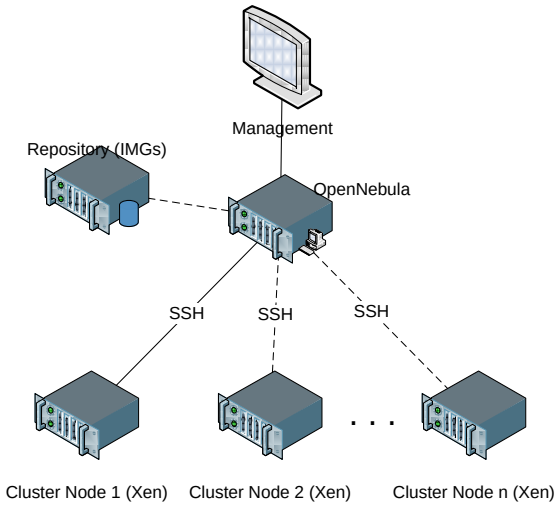
OpenNebula is an active open source project with a good reputation. It supports Xen and KVM hypervisors among many others and it also got an interface with Amazon's EC2[10]. It's mainly aimed for a cloud with at least two physical cluster nodes with a Xen hypervisor running on each of them. It's using CLI to control the life-cycle of VMs. OpenNebula we saw as good management software to start out with. It was fairly easy to get started with, we were setting up VMs after only a few days of testing and we used it for about one month until we then realized we needed something with more functionalities and our only option was then to start developing our own management software.

1.4.2 Network Topologi supported by OpenNebula

The topology that they suggest requires having another machine running OpenNebula as a front-end server. The cluster node machines (Xen servers) run a SSHd that the OpenNebula uses to connect and communicate with.

The Following figure (Figure 2) describes the topology that OpenNebula supports. In this figure you can see that it support several cluster nodes and also a repository containing IMGs that needs to be accessible from OpenNebula. It is also possible to save all the IMGs on the node(s).

Figure 2 – Network topology supported by OpenNebula



1.4.3 Libvirt

Libvirt is open source software that supports multiple hypervisors including Xen which is the one that is interesting for us. It provides an API for remote management of virtual machines. The communication to the hypervisor can be made using encryption. A goal with Libvirt is to be an API which makes it possible for others to implement their own software on top of Libvirt, which can be seen in some projects such as Eucalyptus. Libvirt don't want to have advanced features like automatic load-balancing but instead provide a long-term stable API that is written in C [11][12].

1.4.4 Eucalyptus

Eucalyptus is a program built on top of Libvirt and therefore supports everything that Libvirt supports. It is however released under two different licenses "Eucalyptus Enterprise Edition" (EEE) and Open Eucalyptus [13]. Since we won't spend any money in this thesis EEE will be ignored. Eucalyptus was tested together with OpenNebula but was discarded and instead we choose to use OpenNebula as our first management software.

1.5 Thesis Outline

This thesis report will give the readers knowledge on how to solve the problem of having a virtual network distributed over several physical machines. It will go through all the technologies used including OpenFlow. The report contains plenty of illustrative figures to help the reader to better understand.

In chapter 1.1 there are problems and requirements described. Then the implementation and design in chapter 2 will go through all about what we've done in this thesis. The implementation chapter of the report is written in a chronologic order of how we worked and solved our thesis problems. This is made for the reader to better understand how and why we did as we did.

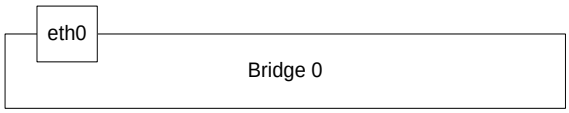
This report is aimed to be read by readers with general knowledge about Information technology, communication systems and distributed systems. It's also good but not required if the reader know what is virtualization.

2 Implementation and design

2.1 Get familiar with Xen terminology

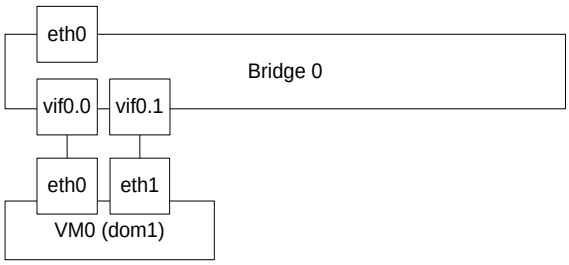
A Xen hypervisor is called dom0 and is the one controlling the VMs hosted on that machine. The VMs hosted on dom0 is called domU or dom1,2,3…n. The domU's can have network interfaces eth0, eth1, etc. and each interface on the VM will also have a virtual interface in the dom0. When Xen is configured in Bridge mode, it means that the Linux Bridge will take care of the communication between the machines and out on the physical network. [4] When Xen is started, a bridge is created and a physical interface is attached to it (Figure 3). With the physical interface attached, anything sent to the bridge can use that interface. The bridge can be seen as a normal physical switch and all interfaces attached to it can be seen as if you plugged in a cable to a switch. For VMs attached to this bridge the switch will be very fast because it only requires some memory operations, though sending packets out on the network it depends on that interface specification.

Figure 3 – Xen bridge with physical interface attached



When a VM is started on dom0 each of that VMs interfaces will get a virtual interface (vif) attached to the bridge. If a VM have two interfaces, the bridge will get two vif's. Each vif is numbered uniquely with vif<VM ID>.<NIC ID>. An example of this is shown in (Figure 4).

Figure 4 – VM with two NICs attached to bridge

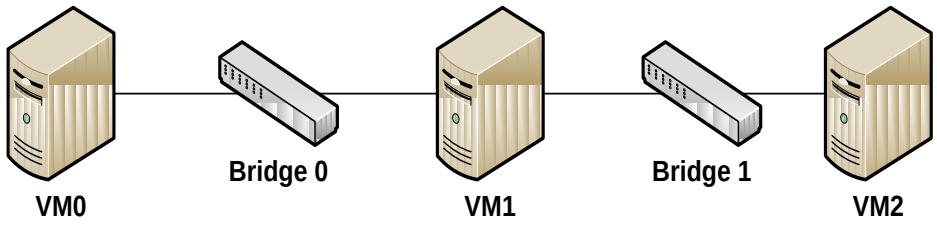


2.1.1 An example using one Xen machine and three VM's

As we started out with only having one single Xen machine, the complexity was not a problem and we could concentrate to get to know all the components and get a feeling of how Xen and virtualization works.

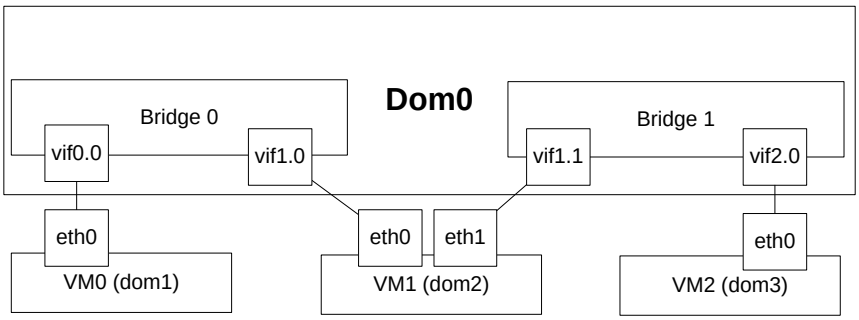
We will be using the following example setup of a virtual network shown in Figure 5. All the VMs are placed on one Xen machine. Each of them have a vif that is correalated to their NIC(s). And these vifs are in turn attached to a bridge wich can actually be seen as a virtual switch. VM1 is supposed to route traffic between its two NICs. When we tested this we used the Linux "IPTables" to route the traffic between the two interfaces. The traffic sent from VM0 to VM2 had to go through VM1 in order to get there and vice versa.

Figure 5 – Test network topology with three VMs and two switches.



The figure below (Figure 6) describes how the Xen node (or dom0) is looking with the setup from Figure 5. Two bridges is needed for the two switches and VM1 is connected to both of these so it can route the traffic. Its important to have two bridges for the VM0 and VM2 not being able to reach each other directly without passing VM1, because then we're not following the topology from Figure 5.

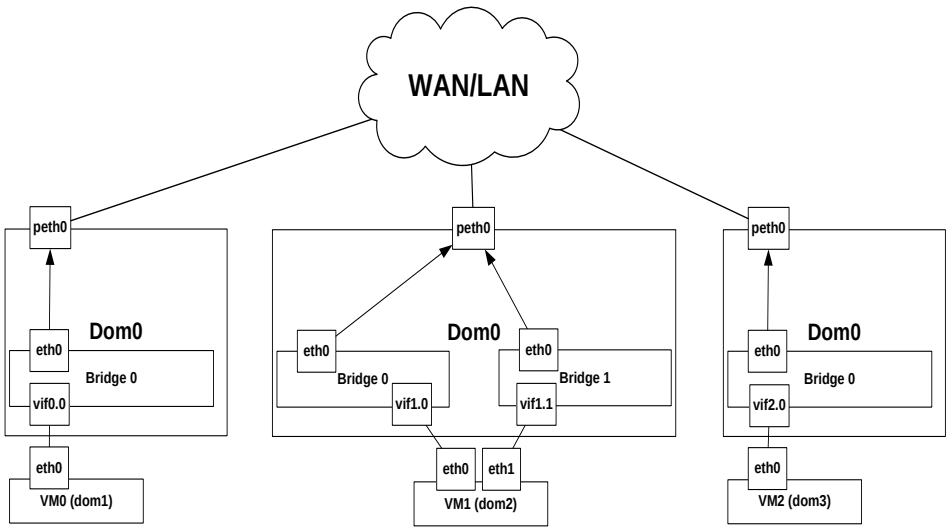
Figure 6 – One Xen node with three VMs



2.2 Using several Xen nodes

When using two Xen servers we don't get the all the problems like when we use three and more. The reason for that is explained latter. In the following figure (Figure 7) it is described how it can look like when using three Xen machines and one VM running on each of them using the setup from Figure 5.

Figure 7 – Setup of three Xen machines with one VM running on each of them

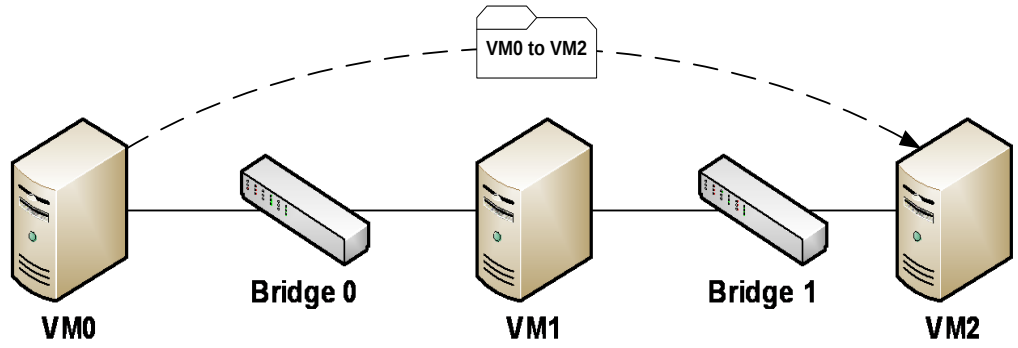


After setting this virtual network up and performing tests on it we discovered a problem. The packets from the virtual network were sent out on the WAN/LAN area. This would absolutely not be working if it would be on the Internet as the virtual network can contain conflicting IP address ranges.

The packets from the virtual world are being sent out because we have a bridge (switch) in the Xen machine that have the VM's connected to it and then also the physical interface attached. As this is a switch and nothing else it will not do anything to the packets from the virtual world and therefore send them out unprotected which will lead to that the WAN/LAN will have two IP subnets coming from the same interface.

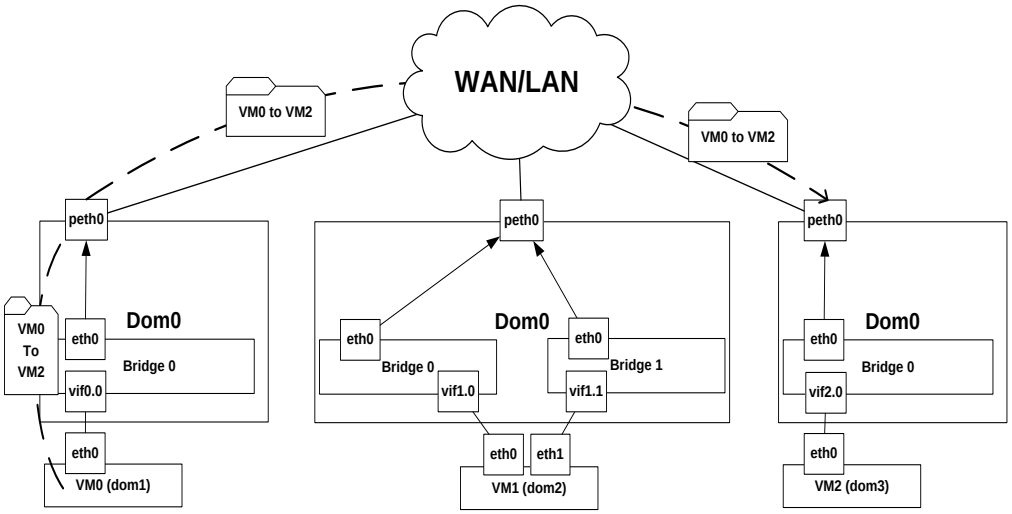
Further study of the problem proved that packets were affected by the Network between the Xen nodes. Looking at the Figure 8, you see that the packet is not sent to the correct VM. The problem lies in that we are using two IP subnets on the same Port of the Xen machine. One IP subnet is the VMs used to communicate between the VMs. The other is the IP on the physical interface that is used by all of the Xen nodes. The physical switch we had between the Xen nodes had learned where VM0 and VM2 were and it started to send packets directly to the Xen node they were hosted on.

Figure 8 – Packets not being sent correctly



The packets from the virtual network could not be sent out on some network we didn't know the topology of. It was essential to find another way of doing this. We started to think about modifying or route packets in the WAN/LAN cloud area (Figure 9) to ensure the destination. This could be done using OpenFlow enabled switches, but then we were locked to having a known network topology. That would obviously not be a good solution. We wanted to have something that would not rely on the network infrastructure between the Xen nodes. The functionality should be on the Xen nodes for it to be an acceptable solution.

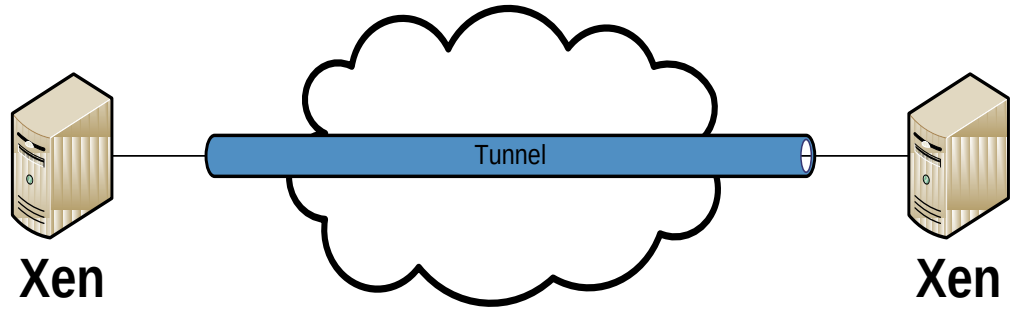
Figure 9 – Packet sent to wrong destination



2.3 Making communication links into tunnels

When distributing Xen nodes there will be some kind of network infrastructure between these nodes. The problem then rises that all the data packages we send between these can be modified or re-routed to a wrong place. The package we send out from one node should get in to correct destination node unmodified. Therefore we have created a tunnel between all Xen nodes so they can safely communicate with each other without the in-between network can disturb the traffic. The tunneling can also be used to encrypt the traffic if the data were to be sent out over the Internet. It can also be seen as a way to make the Xen nodes being located on the same LAN even though they don't have to be physically.

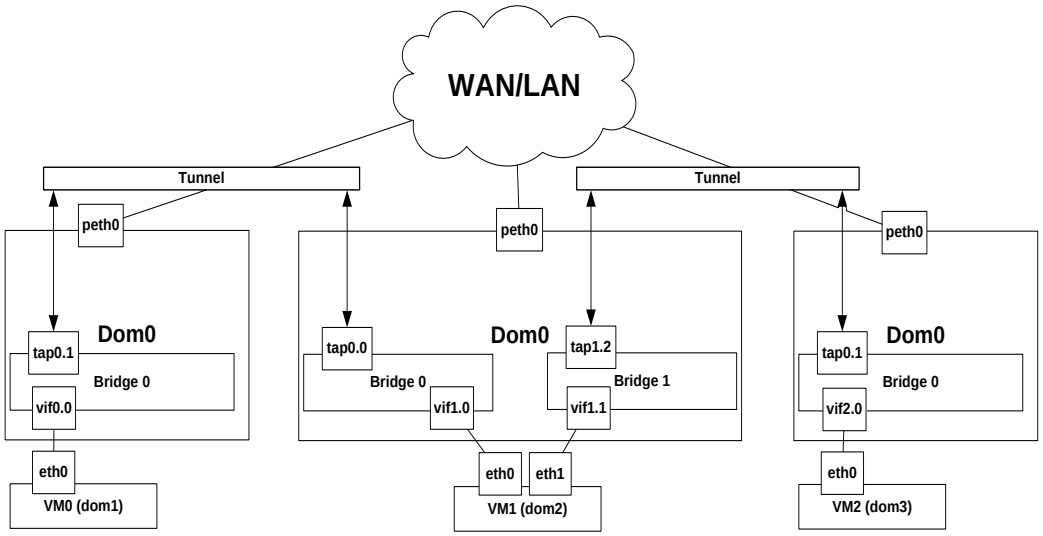
Figure 10 – Tunnelling between Xen nodes



The tunnelling concept is simply to package the packet in a new packet and send it out on the network and in this way hiding the original packet for all kinds of network infrastructure in between. The packaged packets data could also be encrypted to ensure privacy, but in our case encryption would only slow down since we already are in a private LAN and not out on the Internet.

Shown in Figure 11 we are encapsulating the packet by using a tunnel. What we have done is to add a tunnel between Xen 1 and Xen 2, and also between Xen 2 and Xen 3. Now when VM0 wants to send a packet it gets encapsulated in a new packet and sent away in the tunnel. The packet will now end up at Xen 2 and VM1, as it's supposed to do.

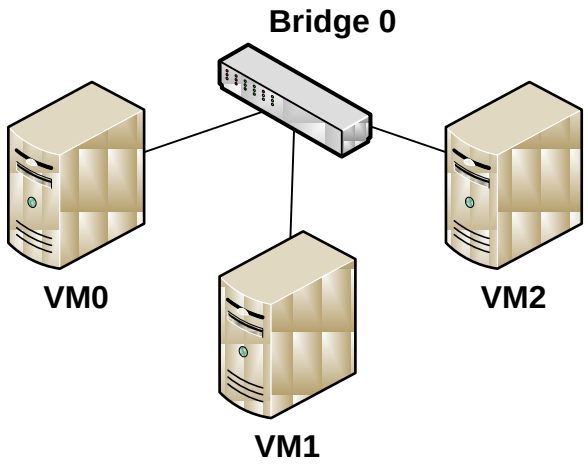
Figure 11 - Using tunnels to ensure correct packet delivery



2.4 Distributed switch problem

Now we start look into another problem with a new setup of VMs (Figure 12). We now have three VMs and three Xen nodes. To evenly distribute them we put each one of the VMs on their own Xen node. Between each Xen node there is a tunnel (Figure 15). The virtual switch (or bridge) will now be distributed over three Xen nodes. Each Xen node will have a switch with one port connected to the VM and two other ports which are the tunnels who are going to the other two Xen nodes.

Figure 12 – Virtal network setup with three VMs connected to one switch

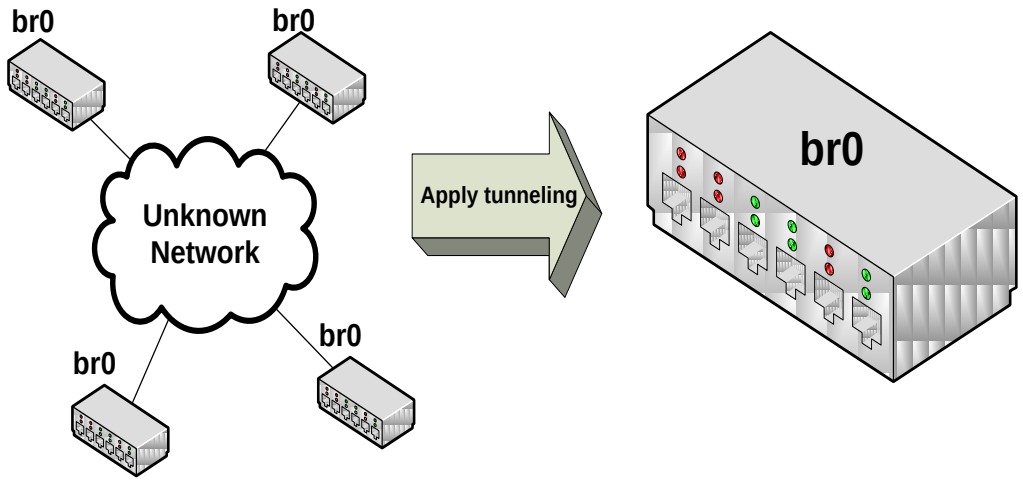


As you know, we are having a switch that is physically distributed over several machines shown in Figure 15. Between these switches there is a network that could be unknown, or at least it shouldn't matter. The problem starts at the point when we want to have VMs that are not on the same physical machine and also in the same time being connected to the same virtual switch. All the distributed switches should then logically only be looking as one switch. This is somewhat a problem for us. The packets must be sent between the Xen nodes for it to be able to look as only one switch. In the background the packet can actually be sent a long distance but in the virtual world this should be seen as the virtual switch was just sending it out on one of its ports.

2.4.1 Tunneling

Shown in (Figure 13) below the distributed switches becomes one by applying tunneling to the problem. Each of all the switches will have ports where they can send out packets to other switches via the unknown network. Each switch will have a unique port and the interface connected to this port is a tunnel interface that is directly connected to the other switch via a tunnel.

Figure 13 – Describing the solution to having distributed switches to look as one



If a packet arrives on a port on one machine that has a destination address for a Xen machine being located on another Xen machine the packet needs to travel over the network to the next switch and will then be sent out to the correct switch port. But all this is hidden, because when we applied the tunneling, we made it look like it actually is one switch. This is also how it actually is supposed to be – Only one switch. After the tunneling was applied we now have a virtual switch that can be used as any normal physical switch would be, though the packets is actually in the background being sent between the Xen nodes.

2.4.2 Broadcast messages with STP turned off

The STP is required to be turned off on all switches because we want to send data on all the ports on the switch. This leaves us with a network topology where loops are possible and that needs to be solved by us. The loops start to occur when we have more than two Xen nodes. Because then, on each Xen node there will be more than one path a packet can go to reach another Xen node. When a broadcast message then is being sent to one node it will result in the packet being sent out on all the other tunnel interfaces leading to other Xen nodes. This pattern will continue on all nodes causing loops.

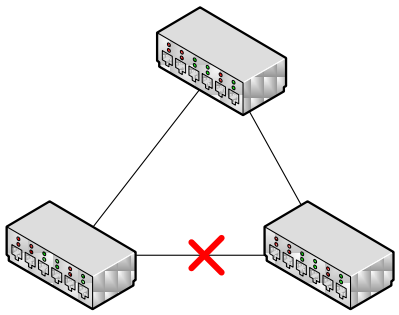
Without STP, loops in the network become a problem when sending broadcast messages. Since all the distributed switches are having STP turned off the broadcast messages need to be taken care of. Our solution to this is to create a spanning tree only for the broadcast messages. All other packages are being sent normally since they will not create any loops in the network.

Shown in Figure 14 there will be a spanning tree for broadcast messages after one link have been removed. This network topology will now never create loops and in the same time all links can be used for non-broadcast traffic. What this means, looking at Figure 14, is that when a broadcast message arrives on a switch it cannot send that packet to other switches except the tree root node who can reach everyone. Loops will now never occur and this will also work with more than three nodes but then more links needs to be cut off.

With a correct tree structure for the broadcast messages the following tunneling solution will work perfectly. All the nodes can reach each other using a tunnel but broadcast messages is cut off to disable loops.

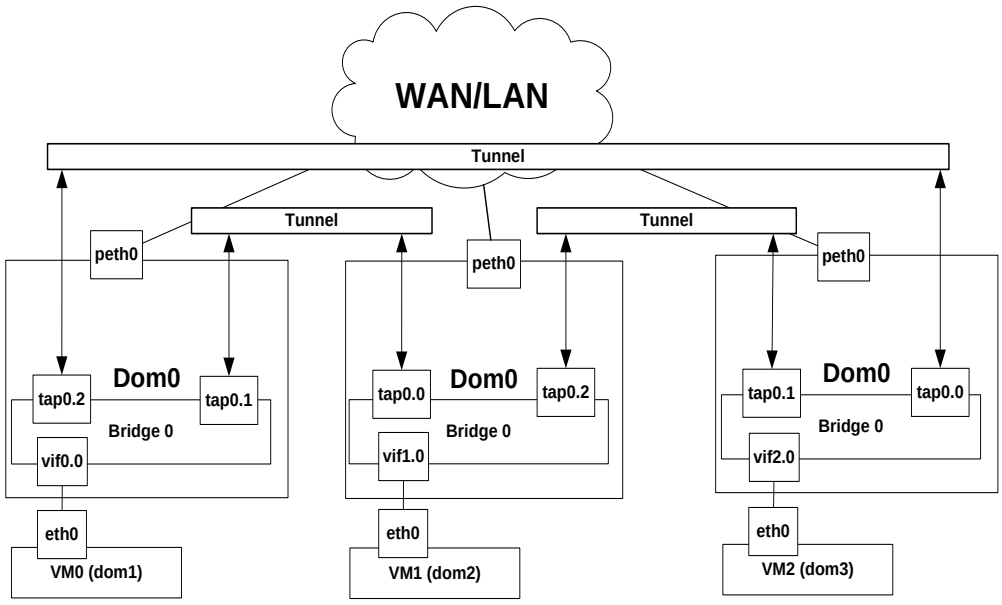
How did we prevent loops from happening when STP was turned off? We used OpenFlow and it is described further in chapter (2.5.2).

Figure 14 – Removed link between two switches for broadcast messages only



In Figure 15 we have our final version presented. It describes how we created a virtual network that had VMs spread out over three Xen hypervisor nodes and yet they were connected to the same virtual switch.

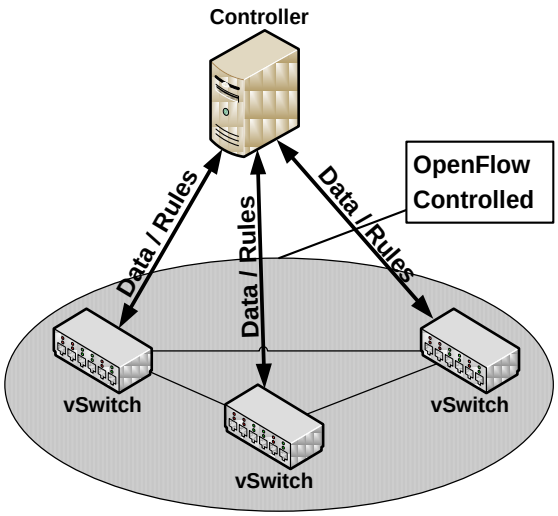
Figure 15 – Complete solution to our distributed virtualized network problem.



2.5 Adding OpenFlow into a virtualized world using Open vSwitch

This topic turned out to be one of our main goals in this thesis. It was at first put at low priority because we were to concentrate more on the roles, shared disk and other parts. But since this is such an interesting topic and is currently at a very early stage in the developing process we saw it as a more appealing topic to work on and decided to prioritize it more. It turned out good. We had a lot of use of OpenFlow and have found a great Open Source implementation of it (Open vSwitch) which we are using on both as a replacement for the Linux bridge and also as a dedicated PC modified to act as a switch. Shown below in Figure 16 it is illustrated how OpenFlow switches and controller can interact.

Figure 16 – OpenFlow controlled switches



The flow of a packet is decided by either the switches or the controller. When a packet arrives at a port on a switch it'll first check if there is an OpenFlow rule that matches the packet on the local switch and if it's a match it will perform the given action on that packet. If such a rule does not exist, the switch will send the packet to the controller which will look at the packet and make up a rule for the packet. If its going to be send out on a switch port, it will send it back to the switch again and also set a rule for that type of packet so the next time a packet arrives to the switch, it will directly know what to do with it. It is better if the switch make the decision because sending packets to the controller increases the latency of that packet.

2.5.1 Controlling flows by rules

The flows are being controlled by using rules. The rules can only be applied on incoming port and they decide which output port the packet should be sent to. It could also not be sent anywhere and thrown away or be sent to the controller and let it take the decision. The OpenFlow rule with Open vSwitch is set by using "ovs-ofctl" command. The following example command gives better understanding of how the rules can look like. This command will add a rule on the switch which will first match on IP traffic and the "dl_type=0x0800" in the rule does that. Then it matches on the destination MAC address "dl_dst=02-1E-EC-25-6A-68". By default the rule will only be active as long as packets are coming which is matching the rule, so by setting "idle_timeout=0" the rule will never get removed even if no packets matches the rule. The last "actions=output:2" will send out the packet on port two of the switch.

```
ovs-ofctl add-flow tcp:<switch ip> dl_type=0x0800,dl_dst=02-1E-EC-
25-6A-68,idle_timeout=0,actions=output:2
```

The different ways of OpenFlow to match packets on are many. Following table shows what Open vSwitch currently supports.

Table of ways of filtering traffic with Open vSwitch

Command	Description
in_port=port_no	The port on switch packets arrive
dl_vlan=vlan	IEEE 802.1q Virtual LAN tag. Specify a number between 0 and 4095
dl_src=mac	MAC address of Ethernet source
dl_dst=mac	MAC address of Ethernet destination
dl_type=ethertype	Ethernet types: IP=0x0800 and ARP=0x0806
nw_src=ip[/netmask]	Source IP address
nw_dst=ip[/netmask]	Destination IP address
nw_proto=proto	Protocol defined in number between 0 and 255. ICMP=1, TCP=6, UDP=17
tp_src=port	UDP or TCP source port
tp_dst=port	UDP or TCP destination port
icmp_type=type	ICMP message with type. Specified with a number between 0 and 255
icmp_code=code	ICMP message with code.

All above mentioned can also be modified in the packet.

2.5.2 Using OpenFlow to prevent broadcast traffic to loop

In OpenFlow there are no options to control outgoing traffic so there is no possibility to use an OpenFlow rule to block certain ports sending out broadcast traffic. Blocking broadcasts on incoming traffic will lead to that no one will get the broadcasts. Allowing broadcasts on incoming traffic only for VMs that are supposed to get them will lead to a lot of rules and complications during migration of a VM.

There are not any rules to control outgoing traffic; however there are settings on the switch that can be set to block all FLOOD (Open vSwitch terminology for broadcast) packets for outgoing traffic. The modification on the port is called "noflood" and is set on every port that should not send FLOOD messages.

2.5.3 Installing Open vSwitch in bridged mode

In our case we are using Xen as hypervisor. When installing Open vSwitch it will replace the current virtual switches used by Xen, which is using the Linux bridging code [17]. As the Linux Bridge is an Ethernet Switch all the interfaces attached to this bridge can be seen as it was ports on a real physical switch.

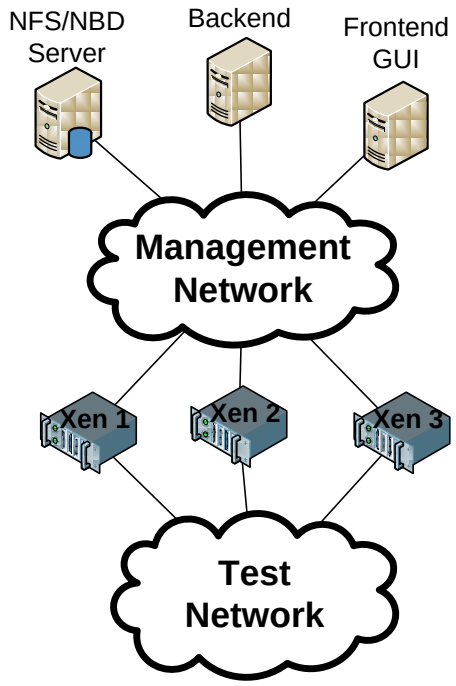
2.5.4 Simulate bandwidth and latency using OpenFlow

If the paths can be controlled, the paths can also be made to simulate traffic between VMs. If it in a real network contains a wireless access point then one path can be made with higher latency or even use a real wireless connection. By letting all the VMs in the virtual network that are talking with this access point take that simulated path, there would be a more real-like simulation of the network. All this can be controlled by just adding the correct OpenFlow rules to the switches.

2.6 Network topology used in our test environment

The following figure (Figure 17) shows our topology used in this thesis. In the topology we have a Frontend with a GUI and a Backend which is in charge of creating the virtual network topology drawn in the GUI. We also have a NFS and NBD server running on one dedicated server. More about the NFS solution can be read in chapter 2.9 and the NBD solution in chapter 2.8. As you can see in Figure 17, the Xen nodes have two connections. One of them is to the management network and the other one is to the test network. This is because we want to isolate the Test Network as much as possible so it doesn't get influenced by any other traffic. We didn't start out with all these different machines; they kept increasing as the thesis project evolved. At first we had only one Xen machine. The backend and frontend was run on our laptops and everything was connected together with a switch. Then as the Thesis progressed we increased the number of Xen machines to three and also added a dedicated NFS/NBD server. All the machines used can be seen as normal PCs with no special hardware. The Xen nodes were having between two and fore CPU cores and between 2GB and 8GB of RAM.

Figure 17 – Network topology



2.6.1 Open vSwitch topology in the Test Network

In the test network seen in Figure 17 we had a topology consisting of three switches. All these switches were in fact a customized PC. Two of these switches were used in a test setup seen in Figure 18. A network PCI card with four ports was added to all of them so we could connect more machines to them and they could also have more than one connection between each other. More about this and why will be discussed later.

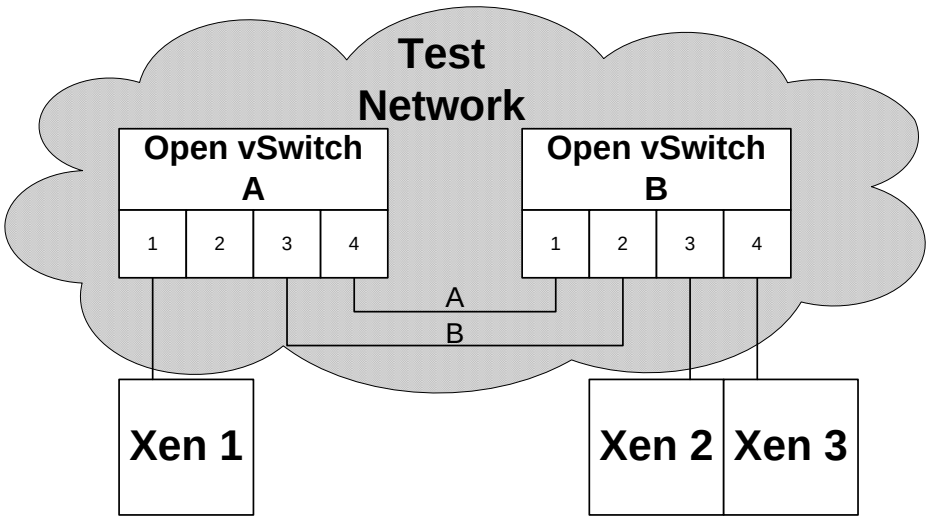
The PC was installed with a Linux distribution together with the necessary pre required software to compile and install Open vSwitch.

The following test setup in Figure 18 we have two PC's with Open vSwitch running. They were connected to each other with two ports. When using two connections between the switches we could make experiments with flows in the network by applying OpenFlow rules. The flows could be controlled to take different paths, A and B. The traffic was monitored on all ports on switch A and displayed in a graph. The graph displayed each ports amount of traffic so we could see if it worked as intended. The traffic could with OpenFlow rules be set to only send on connection A, which would on the graph be displaying traffic on A and no at all on B.

The reason to have more than one connection is that we can simulate bandwidth/latency in the virtual network. One path could be 10Mbps and another 100Mbps. It could even be a wireless link between the two switches, which would simulate the latency and packet loss of a real wireless network.

We are of course also still having our tunnels going through in the test network and they need to be taken into consideration when adding the OpenFlow rules as the traffic is not showing the VMs MAC/IP but the TAP interfaces MAC/IP.

Figure 18 – Example of Open vSwitch test setup.



2.7 Backend

Our frontend is the GUI and our backend is where the core services are. The frontend is communication with the backend which in turn is communicating with all the Xen nodes. The reason of using a backend .c program is to have the support of multiple Xen hosts, which means that in our setup it's supported to use a cluster of hypervisor nodes.

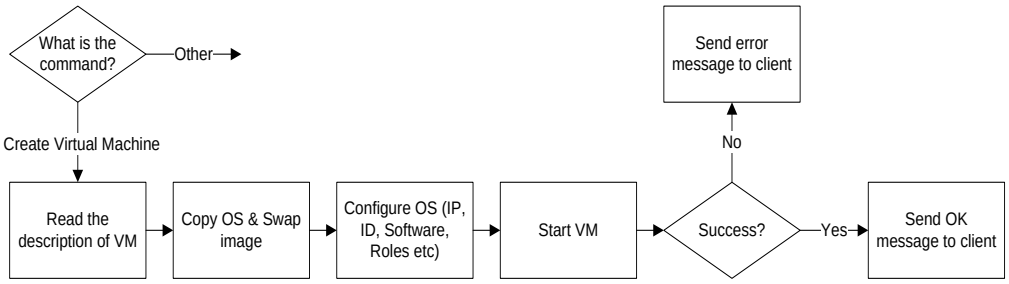
Implementing a backend makes it also possible to faster apply a change if necessary. When applying a change in the backend, no changes have to be made in GUI. An example could be; we want to change the way we start VM. Because the GUI only tells the backend to start a VM it's up to backend to determine how this should be done. Making these two parts not to rely on each other was an important task to enable future work of what we have implemented. Everything of this could also be reversed. The layout of the GUI can be changed without having to change in the backend.

The software on backend was built from scratch with no use of existing code. It's a multithreaded application that handles each request in a new thread utilizing the pthread library. It's modular and highly customizable. Each module such as, VTUN, OpenVPN, OpenFlow etc, is written in an individual file. New modules can be added and old can be deleted fairly easily. This proved to be of great help when we switched tunneling software from using OpenVPN to VTUN. The backend software has an interface to each Xen node with the Libvirt library. This library is a powerful tool when handling with VMs. Since libvirt not only support Xen hypervisor we're not locked to a specific hypervisor but can use all other that Libvirt supports.

2.7.1 How do we setup new virtual machines?

When a new VM is about to setup the backend will receive a series of instruction commands from the frontend. With this information received, the backend will do the necessary operations to get the VM started on a Xen node. The operations are read description, copy images, configure and finally start the VM. After the VM is started it sends back a report message to let the frontend know the result. It's programmed to handle multiple requests at the same time so each request is handled in a thread.

Figure 19 – Flowchart showing how the process of setting up a VM is made on the backend.



To get a live migration to work the Xen nodes needs to be able to read from the same filesystem. The images are therefore being copied on the NFS server which is shared by all Xen nodes. When the image can be accessed by all the Xen nodes the migration will only take a few seconds, compared with if it would have to copy the image before migrating.

After images have been copied they are going to be configured. The ID of the OS image is written to "/etc/myid" on Linux VMs. This ID is being used by the VM to identify itself. With this ID the scripts on the OS image will know where it can find the configuration for the specific VM in the shared disk. The shared disk includes IP address and basic roles configuration so far of what we have implemented. The directory structure of the configuration on the shared disk is the same as on the OS. When the IP is to be copied, it copies from the shared disk the file "<Config_directory>/etc/network/interfaces". In Linux, the IP of a system can be found in the file "/etc/network/interfaces". When this file has been copied it'll startup the interface with the newly added configuration. The script will continuously check to this shared storage looking for changes. If any changes have been made it will copy the changes to the OS image.

Regarding the roles, we have not put our focus on that in this Thesis. Though, we think that it is an important task to finish because it decreases the setup time of virtual networks. What we have

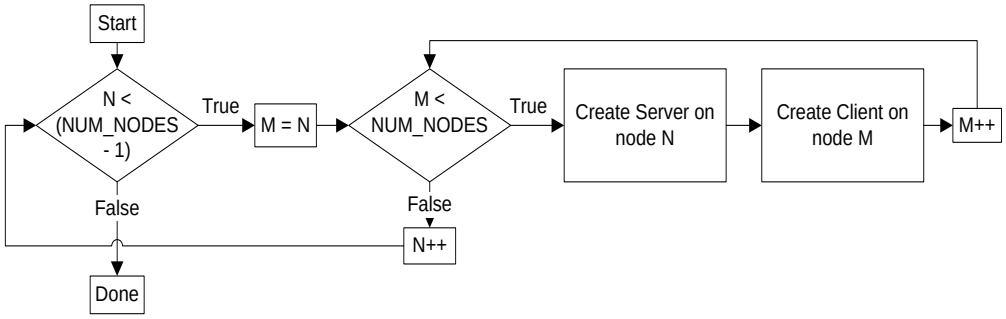
completed is that different roles can be different OS's, IP, RAM and CPU. The intension of the roles is to also install the necessary software's for the specific role. Let's say we have chosen to start a VM with the role FTP. Then we also want it to have installed a FTP server and to start up the FTP server when the VM boots with the configuration from the shared storage. The shared disk is already prepared for having all the software's and scripts put there but it is required further developing in the configuration scripts.

The VM is started after all configurations have been made to it. The installation of software is meant to be done after it has started using the scripts and software on the shared disk. The startup of a VM is taken care of using Libvirt. The Libvirt library is being used by our backend program and it can communicate using the API with all the Xen nodes in the cluster. To startup a VM it needs a XML file describing the VM. The description of the VM was sent to the backend in the first step and was thereafter converted to a XML file. It waits for the status reported by libvirt and the reports it back to the frontend.

2.7.2 VTUN setup Algorithm

VTUN is setup with one tunnel between all Xen nodes. All data passed between the nodes will pass through this tunnel. The tunnel is there to ensure that the packets will get to the correct destination. The algorithm used to setup VTUN is described in (Figure 20) using three Xen nodes. The algorithm starts at the first node and sets up a tunnel between that one and all the nodes above. Then it goes to the next one and does the same, and so on until it has reached the next last one. First is the server started then the client, this is to make sure that the client can make a connection to the server directly when it gets started.

Figure 20 – Flowchart describing the algorithm to setup VTUN.



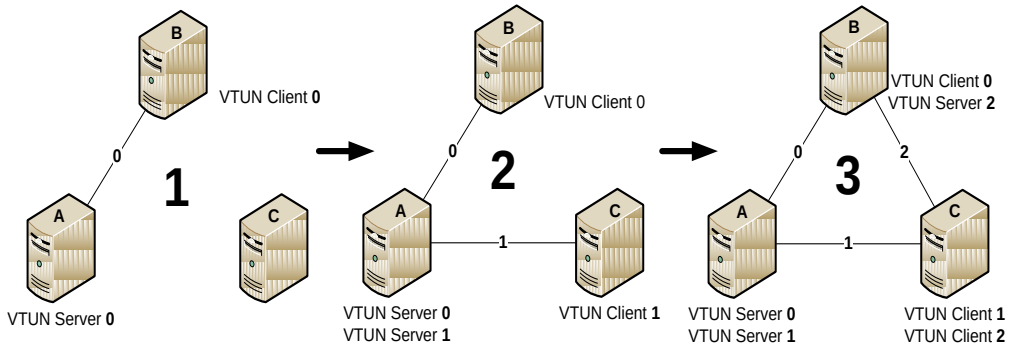
Now, to get a better understanding we will continue describe the algorithm by using an example (Figure 21). In this example we're using three Xen nodes A, B and C. To set everything up it takes three steps per bridge. Below is described step-by-step what happens.

Step 1: VTUN is setup between A (Server) and B (Client).

Step 2: VTUN is setup between A (Server) and C (Client).

Step 3: VTUN is setup between B (Server) and C (Client).

Figure 21 – Describing Algorithm for setting up VTUN using an example with three Xen nodes.



2.8 Shared Storage between virtual machines

2.8.1 Network Block Device (NBD)

With Xen it's possible to attach and detach block devices to a VM. NBD is a client/server software that makes it possible to import a disk over the network as a block device. The imported device will for the clients system look as a normal local block device. This gives us then the opportunity to import a shared disk over the network and then attach it to all the VM's using Xen's built in technology. NBD do though come with a problem. Due to the fact that the filesystem is not supporting more than one mount at a time, there is not possible to use NBD as a safe shared storage solution. The risk of two VMs having the disk mounted at the same time is too big.

2.8.2 GNBD with GFS

To solve the problem with NBD not being suited for several clients to mount the filesystem at the same time, I started to look into GNBD (Global NBD). Using GNBD I can mount the device on more than one client at a time and everyone can see the changes directly. But for this to work it's required to use a special file system; GFS (Global File System). All clients that mount this file system must have a cluster manager (CMAN). This manager together with the correct configuration is required to mount this file system. If a client doesn't have this setup it's rejected upon mounting the device. All clients need therefore to be connectable with each other.

GFS works as such that all the nodes that mount the device must have a cluster manager and a lock manager (DLM in my case). When GFS is created on a device, information like the following is needed; lock or no lock, name of the cluster and number of journals. "nolock" is used for local usage. Before mounting GFS, the CMAN daemon need to be started which adds the node to the cluster.

Because we can't promise that all the VM's are connectable with each other GNBD/GFS cannot be used as a shared disk solution. The reason for why we don't want to have the VMs connectable to the outer world is that we want to keep the simulated environment as real as possible. If we would add an interface to outer world we think that we would loose a great part of the simulation. It could also cause us more problems with the internal networking conflicting with the other and so on.

There are alternatives to GFS such as OCFS but they all have the common part where they need to have connectivity to an either central server or to all the peers. Which would as mentioned before not work in our case.

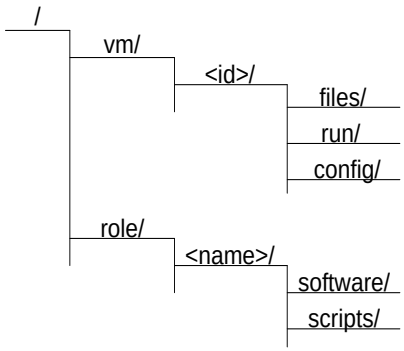
Then how about mounting an image file (shared.img) that is put on in a shared NFS file system? No this does not either work. Because when changes are made inside an image, the NFS will not detect any changes on the file and it will not be updated on the nodes. Because all the changes are being made inside the image, it will for the system look like nothing ever changes on this file.

2.8.3 Directory Structure on the NBD

The NBD is hosting all the files needed for the VM and follows a specific structure (Figure 22). It includes configuration, software and roles. The configuration files are located under "/vm/<id>/config/". Here all the files will be located with the same structure as they would be on the system. At first boot, all files are copied from the "config" directory to the local system of the VM. After first boot, a script will be checking this directory for changes and copy the changed configuration to the local filesystem. If some extra files are needed it can be put under "files" directory. All programs that should be started should be put under "run" directory.

For the roles, they contain two folders, software and scripts. The scripts are for installing the software on the system. Software's can be source files or executable binary files, it's up to the script to figure that out and install. The software is specific for each role. A role can be FTP, Mail, DNS, Router etc.

Figure 22 – Directory structure on NBD file server



2.8.4 A VM's ID

A VM is always having a shared disk attached to it, but how can it know where to look in this disk since it's being used by other VMs? Having an ID solves this problem, but then comes another question. How can a VM know its ID?

There are several solutions to this problem, good and bad. The first one I got to think of was having a unique name of the shared disk when it was attached to the VM. The VM could then look under all the deivces attached and get the ID from there.

But, this was causing more problems than solving the one I had, renaming devices to whatever you want is not a good looking solution, so I moved on and started to think about other solutions.

My other solution I came up with and the solution I ended up using are, before the hard drive Image of the VM is loaded and started, I add the ID to its filesystem. When the VM then is started it knows exactly where to look for the ID in the filesystem (in "/etc/myid"). This solution worked out great and is the one I ended up using.

2.8.5 Router role

In a network it is of big importance to have router role. A router in a virtual network can be a normal Linux distribution with a running router daemon, or in a smaller network it is also possible to only use Linux IPTables to route traffic. The router role was implemented by using a modified Linux distribution with the necessary software's already installed.

2.9 The NFS Image Storage Solution

The NFS server is for sharing Images containing operating systems. They are big so a dedicated NFS server is needed. Every started VM needs an Image file, which means that every time a VM starts the OS image will get copied. The NFS server needs to be reachable by all Xen machines. All Xen machines needs to have a NFS client installed and always be connected to the NFS server. They will mount the NFS storage at "/mnt/nfs/" on their local system. Under this directory all the shared images can be found.

We used the NFS server with two different setups described here below.

2.9.1 Sharing IMGs with NFS with less network congestion

This alternative is good when you don't want to migrate VMs between Xen nodes. The Image is only copied once to the Xen node over the network. After that it uses the local copy to distribute all its VMs. Because its using a local copy the VM can not be migrated. It's required for the other VMs to see the image.

2.9.2 Sharing IMGs with NFS with high network congestion

This alternative is good when you want to support migration between Xen nodes. Here all Xen nodes must know all the running VMs Images. The image that is running in a VM is therefore visible for all the Xen nodes. Every time a new VM gets started a copy of the image over the network needs

to be made. When the image is copied on the NFS server all the Xen nodes can see the image and a migration is possible.

2.10 Communication protocol frontend to backend

Since a thesis don't include so much time really for programming, we did in our developing of the software not put much time on implementing a good protocol, but we have of course a plan for what would be the best suited solution. The software we built was also supposed to be a proof-of-concept evaluation and the communication protocol does not have any effect on the outcome of our built software. What we came up with to be the best communication protocol is to send XML which is much better than sending unformatted text strings between frontend and backend.

The benefits of using XML are that you get something that is dynamic. Adding or removing new functionality is easy and goes smoothly when parsing the XML correctly. It's a protocol that can evolve with the product, any upgrade on backend or frontend can be done without changing in the way they communicate. Even if only one side is upgraded it's not necessary to do any changes on the other side. If we would've included this type of protocol in our thesis project it would have cost us time that we could have spent on implementing better and more important things. We kept it simple by only sending text strings.

The XML protocol is meant to be working as following; In the GUI the topology of the virtual network will be represented in a XML file. This file will follow predefined rules. When the virtual system is deployed the XML file is sent over to the backend. The backend will parse through the XML file and setup the virtual network according to the description. The XML description of the topology could also be used to save/load topologies in the GUI.

2.11 Tunneling software used

We used OpenSource software's to tunnel traffic between Linux hosts. They are described here below and what we think would be the best solution.

2.11.1 OpenVPN

OpenVPN serves our purposes well and was the first tunneling to be used. It's a widely used tunneling solution. It was our first choice as a tunneling solution. At first we setup only one tunnel between all the Xen nodes. But because all client-to-client traffic needs to be routed through the server we saw it as a possible network problem and made a unique tunnel between all nodes. It has no option to turn off the encryption which is a drawback for us as we don't need any encryption when sending on LAN.

2.11.2 VTUN

VTUN (Virtual Tunnel) [19]is an easy way of creating tunnels over TCP/IP. It can be configured to not use any encryption, which benefits us. We don't need encrypted traffic and it will only cause extra unnecessary load on the servers, but it's good to also be able to turn it on in case we want to send traffic over Internet.

2.11.3 "The perfect solution" with tunneling

There are plenty of software's out that can handle tunneling well but they all offer only tunneling. What If we wanted to do some other modifications to the packet combined with tunneling? We could use another program for that, or we could build our own tunneling software and put all the parts we want into that. It would be good to be able to look at a packet and depending on what protocol it's, send it out with different priorities. The tunneling software we use requires an IP to be set to the interface. We don't need this IP and it was shown in tests that the IP on the tunnel interface were conflicting with the VMs IPs.

2.12 Boot order of VMs

Building a virtualized network is almost the same as building it with real hardware. When a virtual network is setup completely and is set off to be started, a special start order needs to be done to get the network setup correctly. First all switches will be created, in the real world this means plug the power in, but in the virtual world this means to create all the Linux bridges on all the Xen nodes and also setup the tunneling. When the switches are created we can now start to go into the virtual machines. The VMs starting order depends on what role they are. The starting order of these are important because a VM can be configured to be a router role. All routers need to be booted first of all VMs. They need also to start the routing daemon and learn the network topology. After the routers, all the server type roles should be started such as FTP, Mail and NFS servers. They should be started because the server roles could be depending to be running when a client is started. A client could for example require mounting its filesystem from a server and therefore it's important to have that server already up and running. After all servers have been booted, the clients can safely boot and we should have a fully working virtualized network.

To be able to set the boot order is an essential task to get a fully working network. Every role should have a boot priority and also each role should have an intergroup boot priority. The priorities are set by the creation of the roles and the intergroup priorities are set if necessary when the VM's are drawn in the GUI.

2.13 GUI

Having something that the user easily can interact with is one of our main goals in this thesis. Building something that is graphical and not a CLI is important in this case. The GUI was made in a way that the user can easily get started because we used a familiar interface layout. This layout gives the user the "big picture" over how the VM's are configured. We don't provide more info than necessary to not confuse the user in some case. The work we've done "behind the scene" is something that we only will provide to developers and others being interested. Keeping everything simple will help us making a user-friendly GUI.

Figure 23 shows an example of a virtual network graph which is drawn in the GUI. To the left there is a panel to choose what type of VM to draw. In the drawing graph area it is possible to move around the VM freely and connect it to a switch. Each VM can be configured in the GUI. When the topology is finished, then by pressing the green play button it will connect to the backend server and start up the virtual network on the Xen nodes. In Figure 24 we have the physical view. To be able to control the OpenFlow switches in the physical network it is required to draw the topology in this graph. The left panel offers different types of switches and number of ports. When creating a new switch it is required to fill in the management IP so that backend will know where to send the OpenFlow rules. The flow paths are drawn in the GUI and are shown as arrows. To apply the changes on the switches there is an "Apply button" to press and the OpenFlow rules will be set on the switches by the backend server.

Figure 23 – Virtual network topology in GUI

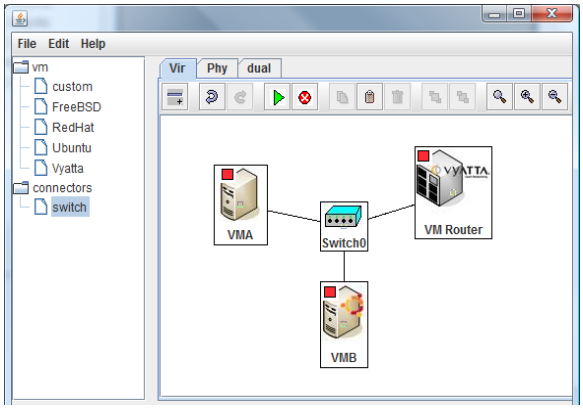
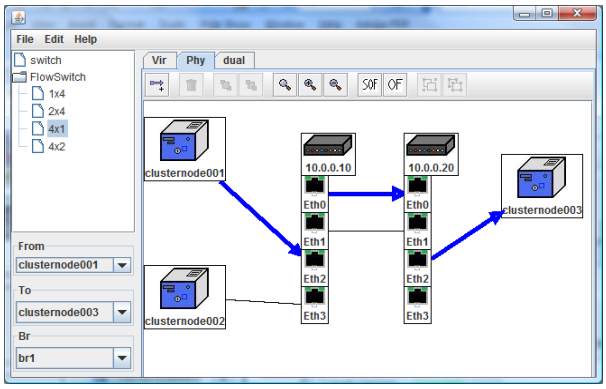


Figure 24 – Physical network topology in GUI



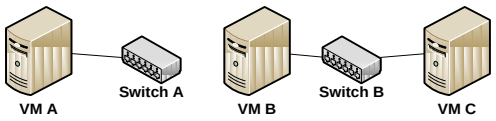
3 Test and verification

3.1 Verify traffic leakage using one Xen machine

Test description

The topology in Figure 25 will verify that VMs that are not connected to each other can also not send any data to each other. Three VMs were used A, B and C. Only VM B and C should be able to communicate with each other and VM A should not be able to communicate with anyone.

Figure 25 – Topology used to verify packets were sent correctly



Results

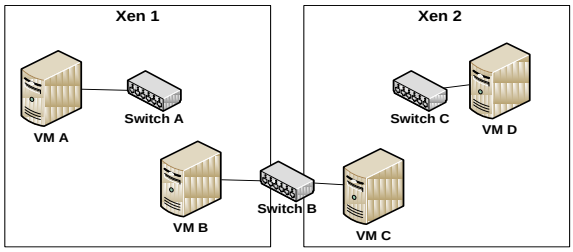
Results show that the VM A can not communicate with B or C. It also shows that B and C can communicate like expected. We did not expect anything to be sent wrong because we only have one Xen machine and no packets need to leave the machine.

3.2 Verify traffic leakage using two Xen machines

Test description

Now when we have two machines, the same switch could be located on both machines like the Switch B shown in Figure 26. This test will verify that a switch being located on more than one machine will work and not leak data to another VM. VM A and VM D should not be able to communicate with another VM. Communication should only be possible between VM B and VM C which are located on different machine.

Figure 26 – Two Xen machines used to verify no traffic leakage



Results

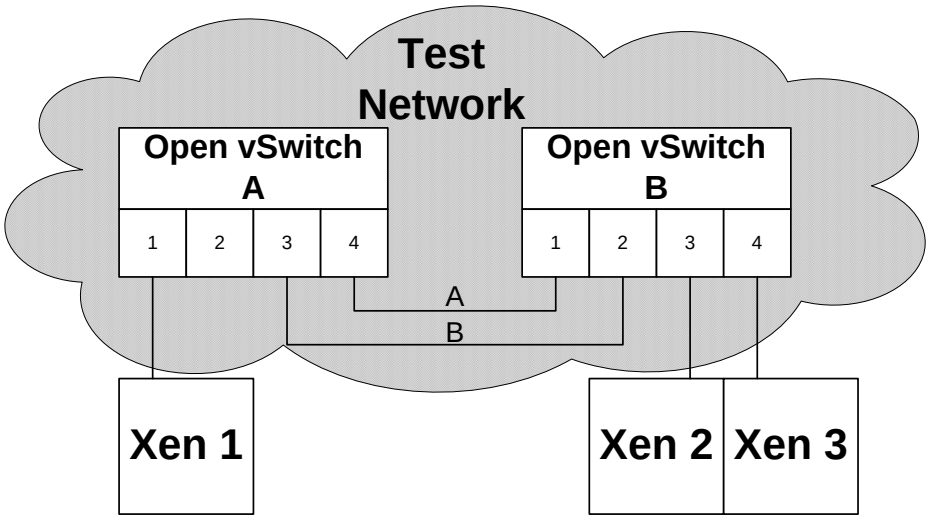
In this test traffic could be sent wrongly because the packets needed to be sent between the Xen machines. When only using two Xen machines we don't have the same complexity that we would have when using three. In this test we found that everything was working as the drawn network topology shows.

3.3 Test setup with two Open vSwitches and three Xen nodes

The following test environment was setup to test and verify that the implemented things were working. To verify that it was working a graphical traffic plotter was setup on Open vSwitch A. This plotter was showing traffic activity on the ports 1, 3 and 4. Between switch A and B the traffic could take either path A or B. The path was decided with an OpenFlow rule and was made from the GUI.

VMs was placed on all of the Xen nodes and connected together with a switch. Packets were sent between the Xen nodes and different OpenFlow rules were set on the switches to confirm that everything was working as it should.

Figure 27 – Test network used to verify our implemented software



To test our software implementation fully, we setup test cases.

Test case 1 – Verify Tunneling and possible Loops:

1. Put VM A on Xen 1, VM B on Xen 2 and VM C on Xen 3
2. From each VM ping the other two VMs
3. Make sure all VMs are reachable
4. Make sure there are no loops in the network

Results – Test Case 1

By looking at the controller's log of packets going through all switches, we saw that there were no loops in the network. By using ping on the VM's we determined if the VM's were reachable or not, and in test case 1 all the VM's could ping each other.

Test case 2 – Verify OpenFlow rules and migration:

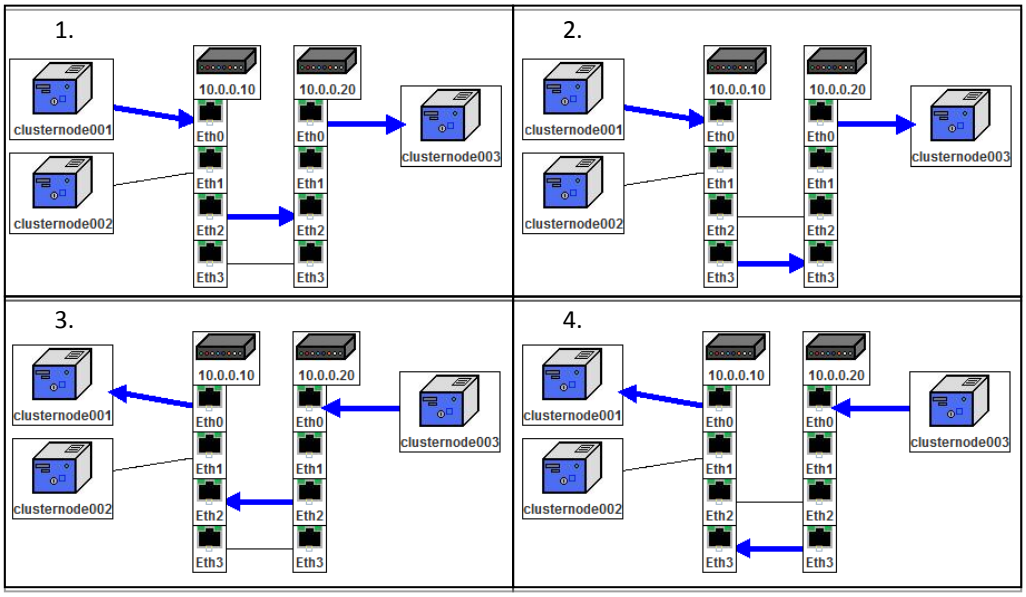
1. Put VM A on Xen 1
2. Put VM B on Xen 2
3. From VM A ping VM B
4. Make sure that the packets are sent on path B
5. Switch path to A by changing the path in the GUI by applying OpenFlow rules
6. Make sure packets now is sent on path A and that the ping is working
7. Live Migrate VM B from Xen 2 to Xen 1
8. Make sure there is no traffic on path A and B and that the ping is working

Results – Test Case 2

The OpenFlow rules worked as expected and we could clearly see on our graph monitoring switch A that the traffic was sent correctly. The migration worked but with some limitations. We could not migrate to Xen 2 from any other Xen machine. However Xen 2 could migrate to all other and back to itself again. This drawback was found to be a problem in the hardware on Xen 2 not being compatible with the other two Xen nodes we used.

If we take a further look at point 5 described above in test case 2. In Figure 28 it is shown the GUI of the physical test network and how it looks like when the paths are changed. From image 1 to 2 and 3 to 4 the paths are changed to take another interface on the switches. They were taking eth2 and are now taking eth3. This will result in that this specific flow will change its path.

Figure 28 – Change OpenFlow paths in the GUI



3.4 IP conflicts between TAP and Virtual Network

During the test phase of our implemented software we discovered that there could be an IP conflict between the TAP and one of the VMs. The TAP interface has to be set using an IP. This IP could though create conflicts with the virtual network. If the IP is set to the same as an IP of a VM in the virtual network, the TAP interface would take all packets sent to the VM. A quick solution to this problem is to set an IP address of all TAP interfaces with an IP that cannot be used in the virtual network.

4 Conclusions

The first month of the thesis we experimented and tested with Xen hypervisor and management software's. During this period we got familiar with everything. We learned a lot during this time and this experience was used during the later part of the thesis work when we were implementing the software. It was a well spent time. Due to this I think that we could build much better software. Because we knew what important parts that should be included and by looking at other management software's we also knew what we wanted that we thought was missing from them. A plus to this thesis is that we have been using only open source software's.

I and Daniel worked together as a team developing the software. I was in charge of the backend part and Daniel the frontend. We however worked towards the same goals and spawned new ideas together. Working together made it easier to solve problems. It is good to bring any problem up to discussion with someone and explain what the problem is about and share thoughts and knowledge.

4.1 Distribution

The distribution problem was a big challenge because we ran into some unexpected problems. However the final solution to the distribution dilemma is really good. We have shown that there is a fairly easy way to make a virtual network spread out over several physical machines. A lot of effort was put into making it very easy to setup a virtual network from the GUI. We managed to implement a lot and we implemented more functionality than what we expected from the beginning. We started out with only one Xen hypervisor but ended up using three in the end. It was good to have three machines to test on because we were then forced to make it work on all three. Otherwise it is a big risk that what we have implemented never would work with several machines if we would only be working and testing with one machine. The complexity increased significantly when we used more than one machine and we encountered some unexpected problems. But it was however never any problems that made us end up in a dead end, the work was continued and we had a clear goal of how to achieve our goals and to get there it was not only possible in one way but there was several ways to get there. As for example, we knew that we would had to tunnel our traffic, but we were never locked to use only one tunneling solution.

We added the tunneling concept to our distribution problem. This was due to that packets that were sent in the virtual network could be seen in the physical network leading to problems of having multiple IP subnets in the network between the nodes. Tunneling the packets solved that. Using the tunneling got us into some other problems like there could be more than one tunnel for the packet to take to get to its destination. We wanted a direct tunnel to the next machine that the packet would go through, but it was also possible to also go through another machine to get there. We think that the tunneling is the best approach to this problem. The way it's being done could be improved if possible to allow the broadcast traffic to not go in a spanning tree. The only good way we found to prevent loops was to create a spanning tree for the broadcast messages.

4.2 OpenFlow

OpenFlow is only in the research stage and no one really knows if it is going to be a product or not. From our experience we think that it is a really good concept and have a lot of possibilities. We got introduced to OpenFlow by our supervisors at Ericsson and they thought it would be good to add it into our virtual network and try it out. We added it to both our virtual network and the testing network between the Xen machines.

We used OpenFlow-enabled switches to solve a big problem also. By replacing the Linux bridge with Open vSwitch switching software we could disable broadcast messages and thereby also remove any loops in the network.

4.3 Shared storage

The best solution to this problem would be to have Xen add more support to add different types of disks to a VM.

A solution that would work is that all the Xen nodes which already are connected with each other can have the functionality to run GFS. Then they would distribute the files from the GFS device into another local device which is then attached to the VM's. But having this rather "bad solution" is not what we're looking for at the moment. So there will not be any good shared disk implementation.

Because we don't want to expose the VM's to any other network than the test network, we couldn't really get any good solution to this problem at that moment.

5 Improvements and future work

5.1 Test network

In the test network we have laid the base for doing simulations with latency and bandwidth. If these things would be added to the test network the simulations could be done with more precise results.

The test network is currently being drawn by hand in the GUI. A requirement is that it must be looking exactly the same in the GUI as it is in the real network. This can though cause errors if it's a big and complex test network, and therefore it would be good to have a network topology identifier. This identifier would automatically look up how the network is connected and draw it in the graph.

5.2 Communication protocol

Come up with a good communication protocol for the frontend-backend communication. We suggest using XML but there could be other technologies to look into.

5.3 Roles

The work done here can be further developed since we more or less stopped after a while working on this and instead concentrated on OpenFlow. The concept is done. We have come up with a good idea how this should be done and also made preparations and some basic implementation.

A role is like having a template defining everything for a VM. The roles are used as a help for the configuration of a VM. When a VM is started with a certain role all the necessary software's will get installed and configured. The roles are made to fasten up the setup time of a VM. The goal is to have roles where there is no need to configure a VM after it has started. With a perfect configured role there would be an easier job to setup bigger networks. To crate or modify a role should be done easy with a good user-friendly interface.

5.4 Shared disk

Here improvements can be made on how the automatic lookup for changes is done on the shared disk. To be able to discover changes there need to be a solid working solution. The current one we provide is not fulfilling that requirement. A solution where there is no shared disk between the VMs and all of them have a unique disk is a possible way of fixing this.

5.5 Mount over network

To start a VM we used a disk image. These images is however a bottleneck. They take enormous amount of time to be created and copied over the network and the time increases with the number of VMs. There are of course faster ways and that is what we propose to be done as future work.

We know for a fact a way of doing this, which is also supported by Xen and we would suggest starting looking into this to begin with.

5.6 Graphical User Interface

An even more user-friendly interface could be done. Since it is the first thing the user sees its important to have a good looking interface so the first impression of the software will be that it is good and stable.

6 References

[1] Xen documentation
http://en.wikipedia.org/wiki/Xen, Last accessed 2010-01-26

[2] Xen documentation. Citrix Systems, Inc
http://www.xen.org/support/documentation.html, Last accessed 2010-01-26

[3] Xen user manual. Citrix Systems, Inc
http://www.xen.org/files/xen_user_manual.pdf, Last accessed 2010-01-26

[4] Xen networking. Citrix Systems, Inc
http://wiki.xensource.com/xenwiki/XenNetworking/, Last accessed 2010-02-02

[5] History of Virtualization
http://en.wikipedia.org/wiki/Timeline_of_virtualization_development, Last accessed 2010-01-26

[6] Google Appenginge. Google
http://code.google.com/appengine/, Last accessed 2010-01-26

[7] Amazon EC2
http://aws.amazon.com/ec2/, Last accessed 2010-01-26

[8] Virtualization. Advanced Micro Devices, Inc
http://www.dell.com/downloads/global/products/misc/ServerVirtual.pdf, Last accessed 2010-01-26

[9] Hypervisor
http://en.wikipedia.org/wiki/Hypervisor, Last accessed 2010-01-26

[10] OpenNebula. Distributed Systems Architecture Group
http://www.opennebula.org/, Last accessed 2010-01-26

[11] Libvirt
http://www.libvirt.org, Last accessed 2010-02-02

[12] Anatomy of the libvirt virtualization library. M. Tim Jones
http://www.ibm.com/developerworks/linux/library/l-libvirt/, Last accessed 2010-02-17

[13] Eucalyptus. Eucalyptus Systems, Inc
http://www.eucalyptus.com/, Last accessed 2010-02-02

[14] Open vSwtich
http://openvswitch.org, Last accessed 2010-01-26

[15] OpenFlow. OpenFlow Consortium
http://www.openflowswitch.org/wp/documents/, Last accessed 2010-01-26

[16] OpenFlow specification
http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf, Last accessed 2010-02-02

[17] Extending Networking into the Virtualization Layer. Oct 2009. B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, S. Shenker.
http://openvswitch.org/papers/hotnets2009.pdf, Last accessed 2010-02-02

[18] Amazon EC2
http://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud, Last accessed 2010-01-26

[19] VTUN. Maxim Krasnyansky
http://vtun.sourceforge.net/, Last accessed 2010-01-26