Network Virtualisation in Baadal

A thesis submitted in partial fulfillment
of the requirements for the degree of

MASTER OF TECHNOLOGY

in

Computer Science & Engineering

by

Deepak Koli

2011CS50278

Under the guidance of

Prof. S.C. Gupta

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi.

May 2016.

Certificate

This is to certify that the work presented under this thesis titled **NETWORK VIRTUALISATION IN BAADAL** being submitted by **Deepak Koli** for the award of **Master of Technology** in **Computer Science & Engineering** is a record of bona fide work carried out by him under my guidance and supervision at the **Department of Computer Science and Engineering**. The work presented in this thesis has not been submitted elsewhere either in part or full, for the award of any other degree or diploma.

Prof. S.C. Gupta
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Abstract

In this project, we introduce network virtualisation solution in the IIT Delhi private cloud. Virtualisation is creating logical components from available physical resources. Here we have used Software Defined networking (SDN) as one of the techniques of Network virtualisation. We went through networking architechture of Baadal, and suggested a policy control design for the communication between VMs belonging to different VLAN. By default, the intervlan VM communication is disabled in the baadal architecture. But there may be requirement of communication between two VMs of different vlan sharing workspace for an application. Policy design application make it possible to enable communication between two intervlan VMs. Since we have used Floodlight as SDN controller, it also provides feature of enabling and disabling it through terminal using REST API. We also developed a load balancing application by using statistics of bytes transfer between VMs. Finally we compared the results of both these applications with the traditional networking. We found 100 times increase in bandwidth in case of intervlan VMs transfer.

Acknowledgment

I would like to thank my project supervisor Dr. SC Gupta for giving me opportunity to work under his guidance and for providing the valuable suggestions, discussions and continuous support throughout the course of this project. He guided us to understand the field of Software Defined Networking through the weekly meetings and the course associated with Cloud and virtualisation. The weekly meetings kept us motivated to achieve more learning in the field and deliver the tasks in timely manner. The Floodlight community, especially Ryan Izard, was also very helpful in sorting out the bugs in the technical part of the project. We would also like to thank the baadal team especially Prateek, Kanika for their support throughout the project.

DEEPAK KOLI

Contents

List of Figures

List of Tables

Chapter 1

PROJECT OVERVIEW

In this project we will present a Network Virtualisation solution using technique called Software Defined Network (SDN). We will start with the motivation behind the project and the problem statement subsequently.

1.1 Motivation

The motivation behind the project lies within the benefits that network virtualisation provides over traditional networking. The advantages of virtualisation has already been seen in the server and storage virtualisation. Some of the benefits are as follows:-

- provides the centralised control over the network
- avails use of less money, time and effort on the hardware
- Technical skills requirement is lesser
- Application delivery time is reduced considerably
- Security is improved
- Reduction in recovery time in case of hardware failure

1.2 Problem Statement

After getting acquired with baadal networking stack, virtual network management in baadal and various virtual network management schemes, we came up with the following problem statement:-

1.2.1 Understanding Baadal networking architecture

In this part, we studied the baadal networking architecture so that we can see the applicability of the solution of the SDN to it.

1.2.2 Introducing the SDN based control in Baadal using Floodlight

As different SDN controllers are available for implementing SDN based solution, we studied various controllers like POX, OpenDayLight (ODL), FLoodLight etc. We experimented initially with POX, then ODL and finally moved to floodlight.

1.2.3 Policy control design for the InterVLan communication

In Baadal, VMs belonging to the different VLAN cannot communicate by default. But there can be a requirement of communication between two specific VMs belonging to different Vlan. So we designed a solution which enables intervlan communication between two specific VMs.

1.2.4 Traffic Measurement among VMs and VM consolidation

Under this, we measured the traffic between different VM using flows installed by the floodlight, and then consolidating VMs talking frequently on a single host.

1.2.5 Observation and Comparison of Results

After implementing the SDN based solution in baadal sandbox, we compared the performance of the intervlan networking with the default (Non SDN based) solution in terms of bandwidth.

Chapter 2

Introduction and Related Work

In this section, we will present the basic concepts necessary for understanding the Network virtualisation and the work that we will be presenting afterwards. We will start with concept of Virtualisation in general, subsequently moving onto the network virtualisation, software defined networking (SDN), virtual network management schemes in data centers and other related things.

2.1 Virtualisation

Virtualisation is the process of creating logical computing resources from available physical resources. It provides an abstraction layer between workloads and the underlying physical hardware by means of virtualisation software. The virtualised computing resources such as CPUs, storage, network, disk I/O, memory are pooled. These are then provisioned to workloads without worrrying about physical location within a data centre.

It also provides encapsulation so that workload can access only the resources assigned to them. In this way several independent workloads can be supported in a virtualised system.

2.1.1 Types of Virtualisation

Various virtualisation technologies are as follows:-

- **Server Virtualisation** - it provides abstraction to the server physical computing resources from logical resources that are created. The user specifies the number of CPUs he requires and it is the virtualisation layer that maps it to the actual physical hardware. It increases the server utilisation.

- **Storage Virtualiastion** - it abstracts and pools the storage physical resources among workloads, thereby increasing storage utilisation. The user can specify the number and size of hard disk as part of parameters of it.

- **Network Virtualisation** - it allows to have multiple small logical network from large physical network and provision of large logical network from these multiple small networks. Network administrators can improve network traffic control, and security using network virtualisation.

2.1.2 Benefits of Virtualisation

There are several advantages of Virtualisation. Major ones are stated as:-

- **Save Expenditure** Network virtualisation gives consolidation of the resources. Therefore, less hardware is required hence it saves both capex and opex. The percentage through which utilisation of the resources can be increased is nearly 80%

- **Provision of Network much faster** It offers faster provision of the network topologies. Unlike hardware assembly, ready templates can be designed for faster provision. They can be used when requested

- **Improved Disaster Recovery** The abstraction provided by the virtualisation enables us to use any commodity hardware, which makes possible to use cheaper hardware to built backup facility which is rarely used. Hence reduction in overall cost.

- **Utilisation of Resources** It has been found that on an average most of the systems operate at less than 15% of their full capacity. Virtualisation increases this efficiency by making utilisation upto 80% of the full capacity

- **Centralisation of Management** Administration becomes easy with virtualisation as it comes with the centralised and remote management

2.2 Network Virtualisation

The purpose of virtualisation is to simulate hardware services in software. All functionalities such as storage (in case of software defined storage) or network resources (in this case) are separated from the hardware and simulated in the software. The hardware platform can be any off the shelf general platform which allows for a more portable, scalable and cheaper solution.

When a network is virtualised, it creates a logical software-based view of the actual physical network and the functionalities of the network hardware boxes like routers, switches, etc. are implemented in the software itself. In this case physical device is only responsible for forwarding the packets, acting like a dumb switch, and all the routing and forwarding decisions are taken by the software.

Azodolmolky et. al. [3] have studied different implementation of virtualization technologies and the table below summarizes the pro and cons of these technologies.

Traditionally, network virtualization has been implemented through the virtual segments called VLANs. But these are limited in number to 4096 and therefore scability is an issue. We can categorise differenct techniques of network virtualisaton based on architechture listed below

1. Hypervisor having dumb virtual switch plus normal physical switch
2. Hypervisor having dumb virtual switch with intelligent physical switch
3. intelligent virtual switch is provided with typical (L2/L3) physical switch

Table 2.1 lists different Network virtualisation Technologies. We will discuss some of these technologies.

2.2.1 VLAN based virtualisation

This is the traditional method of network virtualisation. In this technology, virtual segments are created which act as actual LAN segments. These offer

Technology	Bridging	All host flooding	vNet flooding	VLAN 4k limit	VM MAC visible	State kept in network
VLANs	Yes	Yes	Yes	Yes	Yes	Yes
VM-aware networking	Yes	No	Yes	Yes	Yes	Yes
vCDNI	Yes	Yes	Yes	No	No	MAC of hypervisors
VXLAN	No	Only to some hosts	Yes	No	No	Multicast groups
Nicira NVP	No	No	Some	No	No	No

Table 2.1: Different network virtualization techniques

applications like isolation and security etc. These are limited to 4096 and hence they are not scalable.

2.2.2 VM-aware networking

VM-aware VLAN based implementation scales better. Here the basic idea is that the VLAN list on the physical switch to the hypervisor link is dynamically adjusted based on the server need.

2.2.3 VCDNI

Its principle is based on a virtual distributed switch, isolated from the rest of the network. It is controlled by vCloud director and used MAC-in-MAC encapsulation instead of VLAN. Therefore VM MAC address not visible in physical network. Here 4k limitation of VLANs is no more intact because of longer header in vCDNI protocol.

2.3 Software Defined Networking

2.3.1 Overview

Software defined networking is referred to as the splitting of the network plane into a data plane and a control plane. The control plane consists solely the control logic and controls several devices in the data plane. The data plane, on the other hand, solely consists of forwarding logic such as switches. The control plane is considered to be dumb as it just forwards packets based on the instructions given by the control plane.

2.3.2 Need for SDN

Software defined networking is a framework which allows network programmers to automatically manage and control a large number of network devices, services, topology, traffic paths using high level languages and APIs. SDN pulls out the abstractions and allows one to look at the bigger picture.

- Virtualization: It is use of network resources without worrying about where it is physically located.
- Orchestration: Ability to control and manage thousands of devices with a few commands.
- Programmable: Ability to change behaviour dynamically without the need to suspend services.
- Visibiltiy: It allows the monitoring of resources and figure is the network is disrupted or the presence of any faults.
- Performance: It also allows traffic engineering (bandwidth management), load balancing, high utilization and error handling.
- Multi-tenancy: The tenants have a complete control over their addresses, topology, routing and security.

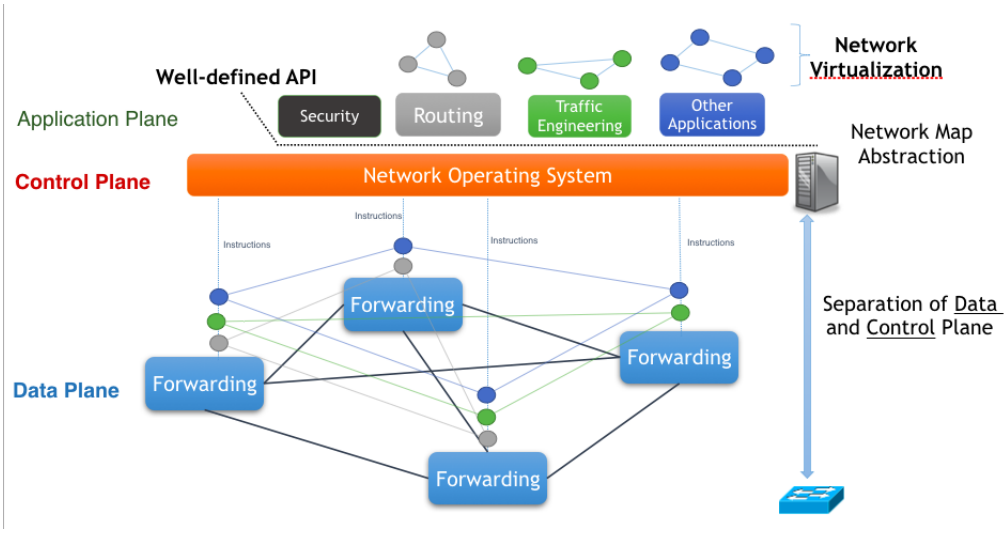

Figure 2.1: Separation of data and control planes

2.3.3 Data and Control planes

SDN consists of two separate planes which are data plane and control plane. In the traditional network where the forwarding logic is distributed to in the network. One example of this type of distribution is in the linked state routers where routers exchange information with each other to jointly create and image of the network while none of them have the complete view of the network. Unlike the traditional network, in SDN, the network intelligence is and states are logically abstracted out. The underlying network infrastructure is abstracted out from applications.

- Data plane: The forwarding logic like switches run of general purpose commodity hardware. It is decoupled from specific networking hardware
- Control plane: The data plane is controlled, maintained and programmed from a central entity called a control plane

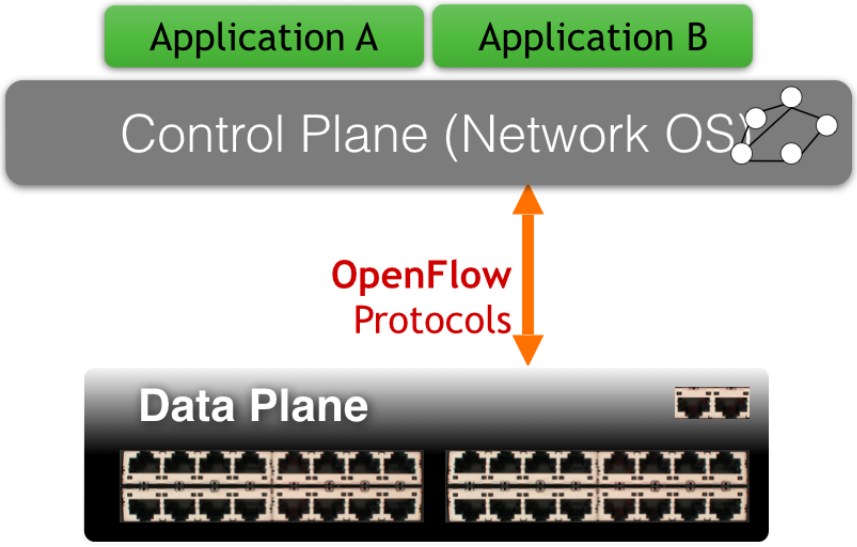

Figure 2.2: OpenFlow

2.3.4 OpenFlow

OpenFlow is a software defined networking standard.It allows network programmers to determine the path of network packets through a network of switches. It is a communication interface between the control plane and data plane of an SDN architecture. It allows direct access to as well as manipulation of the forwarding plane of network devices such as switches, routers, both physical and virtual. It can also be thought as a protocol for switches and controller interfaces.

Working of OpenFlow

OpenFlow manages the traffic (network flows) by manipulating a flow table at switches. Instructions are stored in flow tables. When a packet arrives at a switch, it matches the header fields with flow entries in a flow table. If any entry is matched, it performs indicated actions and updates the counters. If, however, it is not matches then the switch asks the controller by sending a message with the packet header.

2.3.5 SDN Controller

SDN controller is a software which communicated to the hardware switches using the above mentioned OpenFlow protocols. A controller creates rules as to how a packet must be forwarded and installs these rules in hardware switches. A switch then simply forwards the packets according to the rules set up by the controller.

Now we briefly give a description of different SDN controller that are available today.

Nox

Nox was the first SDN controller that was developed. It is a C++ based controller and many other controller like pox have been developed over it. It is not in development now.

Pox

Pox is developed over Nox controller. It is a simple to use python based controller which can be used for rapid protyping of any SDN application.

Ryu

Ryu is a component based SDN framework with a well defined API. It supports various protocols other that OpenFLow like Netconf, OF-config.

OpenDaylight

OpenDaylight is an open source Java based project under Linux distribution. It has a huge community and contributors to its code base. Although a bit difficult to learn, it has a wide variety of features and is also used for commercial products.

Floodlight

Floodlight is also a Java based controller that we have used for developing our application. It has a growing community of developers and provides a rich set of northbound and southbound APIs to create an application module.

2.4 Baadal: IIT Delhi's Private Cloud

"Baadal is a cloud orchestration and virtualization management software developed at IITD that can work with multiple virtualization technologies like KVM, Xen, and VMWare" [1]. Its main features include

- Dynamic resource scheduling and power management
- An integrated work flow system for request of virtual machines
- Virtual machines can be powered on, powered off, resumed or suspended as per user requirement.
- Different costs for resources for VMs depending upon the requirement.

Currently Baadal is deployed in IITD on 48 blade servers with 500 cores and 50 TB of storage (which is virtualized storage based on Network attached storage or NAS). Out of these over 60 machines are being used for high performance computing in the campus.

According to [2] the technical specification of baadal are as follows:

- 32 blade servers with 2x6 core Intel(R) Xeon(R) CPU X5670 @ 2.93GHz and 16 GB RAM.
- 16 blade servers with 2x4 core Intel(R) Xeon(R) CPU E5540 @ 2.53GHz and 12 GB RAM.
- A 10Gbps ethernet backbone.
- 50 TB of virtualized storage on a NetApp 3210V NAS and HP EVA6400 SAN with FC disks.
- Open source virtualization technology based on KVM.

2.4.1 Baadal sandbox

To facilitate easy development and testing of application for use in Baadal deployment this sandbox environment is provided. It simulates the actual baadal architecture and configuration in a single machine without using any more physical hosts. Once an application is tested on this sandbox then it can be ported on the actual Baadal with little modifications.

2.4.2 Netmap

Netmap is a framework for high speed packet I/O. It is implement as a single kernel module and supports access to network cards, host stack, virtual ports and netmap pipes. It uses optimizations like batch processing of packets and using a circular queue as both input and output buffers. This reduces packet processing delays and eliminates copying delays.

2.5 Related Work

2.5.1 SDN based Scheme for Virtual network management

As we have seen in virtualisation services like storage, network, computing are provided using cloud. It required usage of virtual network. As part of this project, we suggested some of the changes in virtual network management schemes. But before moving onto the changes suggested and implemented we should have an idea of management schemes for the virtual network.

- Three important aspects of management scheme are how to transfer virtual network packets, tenant isolation and adaptability of virtual network to topology changes.
- Considering these three factors management scheme can be divided into two types. One is **traditional bridging** which binds VMs NIC with physical NIC on a virtual bridge and transfer VMs packets via

the physical NICs. This has advantage of good performance but it lacks flexibility. The other type based on **overlay network** which encapsulates the virtual network packets into the hosts packets. It has advantage of flexibility but it leads to performance loss.

- Using above two traditional methods we have to make a trade off between flexible management and network performance. But SDN allows us to accomplish management logic conveniently which helps to achieve both flexibility and performance improvement.

- We have designed management scheme for vlan solution inspired from a scheme known as FENet.[9]. FENet is SDN based approach for management scheme, creates virtual network upon devices which support OpenFlow protocol and SDN controller programs are developed to manage them.

- It provides improved network utilisation and lower latency than the scheme based on traditional bridging.

- Nicira Network virtualisation platform (NVP) provides a solution which combines idea of SDN and IP tunnelling. One of the other solution proposed for tenant isolation based on SDN, is conditional on the fact that hosts are in the layer 2 network.[11]. It replaces the packet destination destination MAC address with the host MAC address while the destination IP address is still the VM's so that the packet routing does not happen across the physical network.

2.5.2 Online traffic aware virtual machine placement in data center networks

Dias et. al.[5] have presented a virtual machine placement (VMP) algorithm to reallocate virtual machines in the data center servers based on the current traffic matrix, CPU, and memory usage. VMP takes into account the current data center work load, CPU and memory usage of virtual machines to avoid bottlenecks. The basic idea is to consolidate virtual machines that have correlated services. VMP operation is divided into four parts:

- Data acquisition: The traffic between each pair of virtual machines is used to create a traffic matrix. It also takes the CPU and memory usage of each virtual machine as input as well as the topology of the data center. The most used topologies used are similar as all of them have a core which is responsible for connecting big segments of the network. Traffic engineering tries to push the traffic to the edges of the network thereby reducing it in the higher layers.

- Server partitioning: They also have proposed an algorithm for partitioning of servers. The servers which have high degree of connectivity are clubbed together. Most suitable candidates for this type of aggregation are the servers which are placed at the same layers.

- Clustering virtual machines: They [5] have used a community finding algorithm to find communities withing the virtual machines which clusters the virtual machines into groups which have high degree of connectivity.

- VM assignment: In the final step the virtual machines (which are clustered into communities by now) are allocated server partitions which were obtained in the server partitioning step.

In the simulations they have shown that their method is scalable to big data centers and provides an improvement of 12.5% over no management.

2.5.3 Traffic Engineering

Jiang et. al. [8] have proposed a solution to the problem of network load balancing by means of a joint tenant placement and route selection by exploiting multipath routing capability and dynamic virtual machine migration. They have proposed an offline algorithm that solves a static problem given a network snapshot, and an online solution for a dynamic environment with changing traffic. They have leveraged the technique of Markov approximation what required a very small number of virtual machine migrations. In simulations done by them they have evaluated the performance of online

algorithms with real workload obtained from large computing clusters and have shown that their algorithm incurs the minimum performance cost in comparison to all other tested algorithms.

Biran et. al. [4] have also studied this problem and said that VM placement should consider the aggregate resource consumption of co-located VMs in order to obey service level agreements at lower possible cost. Also, the traffic patterns are not stable in nature and vary over a period of time. They have addressed this problem by trying to allocate a placement that not only satisfies the predicted communication demand but also is resilient to demand time-variations. They have defined the problem as a min cut ratio-aware VM placement which is NP-hard in general but they have employed several heuristics to solve it. Through their simulations they have shown that the placements based on their algorithm increase data scalability, while being able to support time-varying traffic demands with a reduced number of dropped packets.

2.5.4 Network Function Virtualization

Middleboxes like proxy servers, firewall, NAT, etc. have become indispensable for today's networks. However, they come with problems like being expensive, difficult to manage and scale. Many of these problems arise because of the fact that they are hardware based devices. As a solution to this problem a recent trend toward Network Function Virtualization has started to turn these middleboxes into software-based entities.

Matins et. al. [10] have introduced a virtualized software middlebox platform called ClickOS. It is based on Xen sice it provides para-virtualized VMs to build a low delay and high throughput platform. Through their evaluation they have shown that ClickOS can hundreds of middleboxes on commodity hardware, offers millions of packet per seconds processing and provide low packet delays. They have shown that even low-end servers can forward packets at 30Gbps.

Ge et. al. [6] have built a consolidated framework OpenANFV for speeding up the performances of virutalized middleboxes. When a middlebox is

needed its resources are orchestrated by OpenStack and is instantiated as a virtual machines on a common platform. They have a Network Functions Acceleration Platform (NFAP) which provides a FPGA card which accelerates various functions. In their evaluations they have shown orders of magnitude of improvement in throughput in virtualized middleboxes like deep packet inspection, network address translation, etc. using NFAP as compared to without NFAP.

Han et. al. [7] have explained the requirements, architectural framework and use cases for NFV and have also discussed the challenges in this area. They argue that although virtualization impacts performance negatively in terms of throughput and latency, these effect must be kept to a minimum. Migrating from the existing architecture to NFV based solution is also a major hurdle that network carriers face. They have presented a architectural framework for NFV which includes four components - orchestrator, VNF manager, virtualization layer and virtualized infrastructure manager. They have also given various use cases for NFV like virtualization of cellular base station, virtualization of cellular core network and virtualization of home network.

Chapter 3

Simulation tools and Introduction to Baadal Architecture

Getting started with theory of Software defined networking, we tested our grasp on literature by performing simulations with basic topologies and scripts.

For basic experimentation, we started with the mininet for creating customised topologies using python scripts. Later on we created the baadal architecture using physical host (baadal sandbox). We will also discuss the baadal networking architecture as a prerequisite to move forward with the installation of baadal sandbox and further development.

3.1 Mininet

Mininet is a network emulator. It creates a network of virtual hosts, switches, controllers, and links. It runs standard Linux network software and supports OpenFlow for flexible custom routing and software defined networking. Some of the features of mininet are:

- A simple and inexpensive **network testbed** provision for developing OpenFlow applications
- Enables complex topology testing, without the need to wire up a physical network
- Enables multiple concurrent developers to work independently on the same topology
- Provides an extensible Python API for network creation and experimentation

3.2 Previous work done using mininet: Mini data center simulations

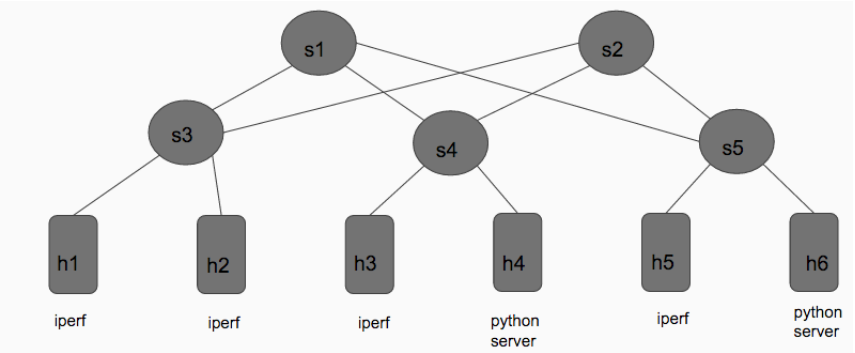

Figure 3.1: Data center simulation in mininet

3.2.1 Mini data center in mininet

Using the python API of mininet a simulation of a mini data center was done. The topology used is shown in figure 3.1. It is a classic fat tree topology which is used in many data centers in which the links become thicker as we go from bottom to up. h_1 to h_6 are hosts which are running different applications . s_1 to s_5 are the switches present out of which s_1 and s_2 act as core switches and s_3 to s_5 act as edge switches. A core switch is a backbone device, a switch that is central to a networks successful operation. We use it to connect to servers, the Internet service provider (ISP) via a router, and to aggregate all switches that a company uses to connect crucial pieces of equipment that a company cant afford to lose to downtime. As a result, a core switch should always be a fast, full-featured managed switch. Edge switches, on the other hand, connect client devices, such as laptops, desktops, security cameras, and wireless access points, to the network.

All the hosts except h_4 and h_6 are running iperf which is a tool to measure bandwidth and quality of network links. h_4 and h_6 are running a python server and client respectively. Using the controller different statistics were

obtained such as the number of bytes transmitted and received by different hosts. In figure 3.2 the bandwidth share between different hosts is plotted and in figure 3.3 the variation of number of bytes exchanged by the hosts is plotted with respect to time.

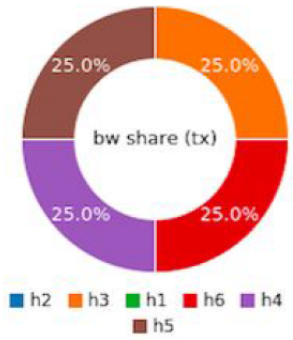

Figure 3.2: Bandwidth share between different hosts

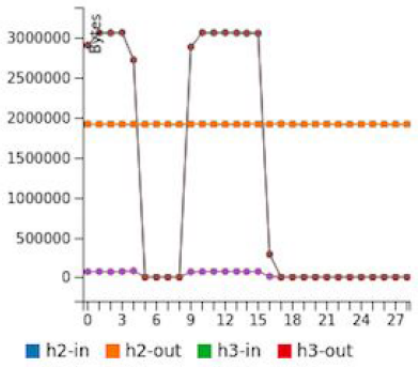

Figure 3.3: Number of bytes transmitted and received by different hosts

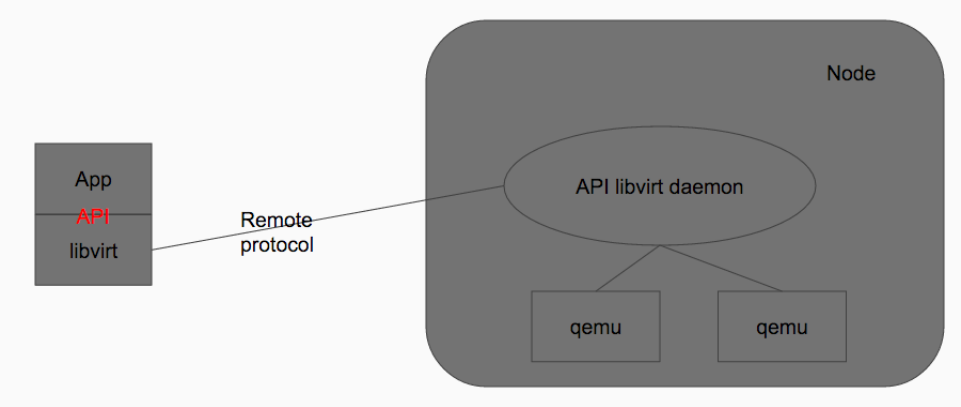

Figure 3.4: Libvirt architecture

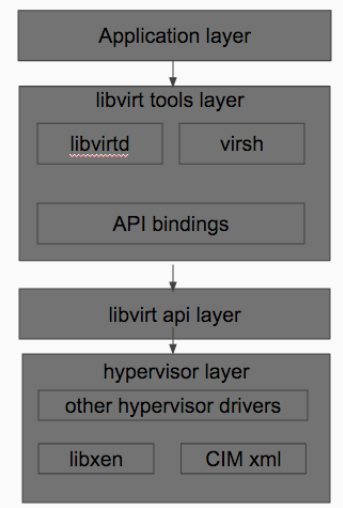

Figure 3.5: Libvirt stack

3.2.2 Libvirt

The libvirt toolkit provides a higher-level VM management interface for tools and applications, that is:

- A set of command line utilities for interacting with the virtualization capabilities of the OS

- A consistent set of APIs in C with the aim to provide support across different virtualization tools

The goals of libvirt are as follows:

- To provide all the operations needed to manage guests or domains running on a single physical node
- To supply a stable interface that isolates upper-level software from changes in the underlying virtualization layer

Libvirt architecture

The basic architecture of libvirt is shown in figure 3.4 which shows that the API libvirt daemon runs on a node (a node is a physical machine). It controls the qemu hypervisor on which guests are run. It also has the provision of connecting to the domains over remote protocol call which is a secure link.

Figure 3.5 shows the libvirt stack:

- libvirt: It is the core API layer
- virsh: It is a command line program which provides a shell environment and a management user interface using which we can create, pause, list, migrate and shutdown the domains.
- libvirtd: It is a daemon (a process which runs in the banckground) for managing guest instances and libvirt virtual networks.

Libvirt connection

Libvirt makes use of URI to specify which driver a connection refers to driver[+transport]://[username@][hostname][:port]/[path][?extraparameters]. The parameters are defined as follows:

- driver : the virtualization technology to interact with
- username : the credentials to use for connecting to the driver.

- hostname : the (possible remote) host where the virtualization technology resides
- port : the port where the virtualization technology listens for connections
- path : a driver dependent path (e.g. the path to a Unix domain socket)
- extraparameters : additional optional parameters
- transport : the transport layer to use for connecting to the driver

3.2.3 Hands on with libvirt

Libvirt was installed in the same vagrant box that was used to do the above experiment related to network statistics. Along with that python-libvirt (the python APi for libvirt), qemu, and virt-manager were also installed. virt-manager is a GUI based domain manager. Then a very basic linux distribution - tiny core linux was installed. Tiny core linux is a bare minimum linux distribution. Figure 3.6 shown the installation of tiny core linux using the virt-manager gui.

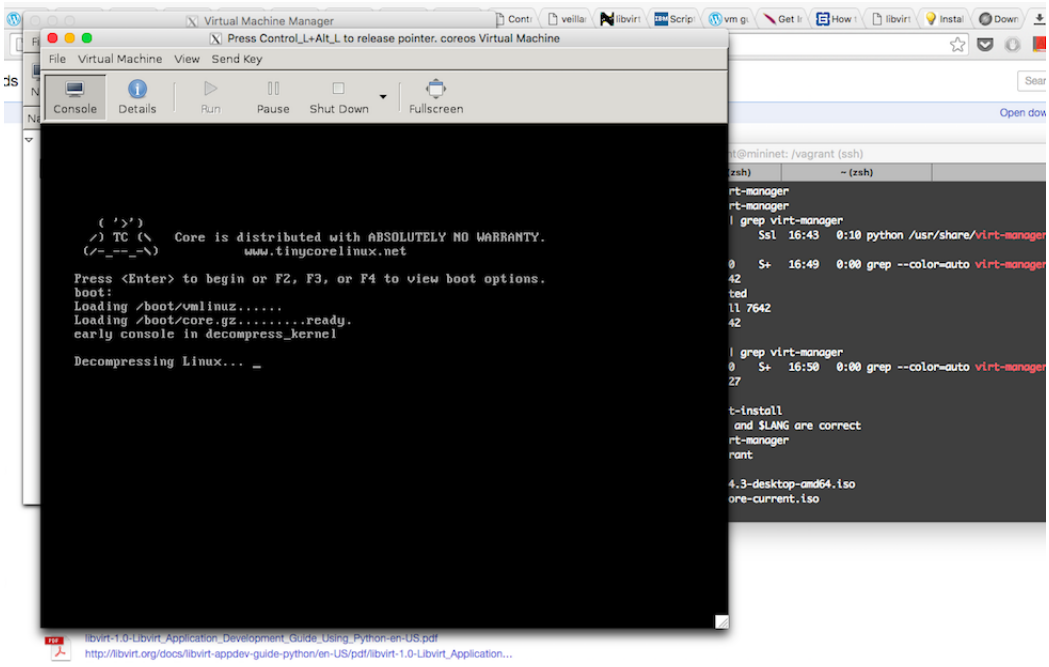

Figure 3.6: Installation of tiny core linux using virt-manager

CPU statistics

To get the CPU statistics of the domains using libvirt's python API the function getCPUStats was used. It gives out the time that the domain has consumed out of the CPUs time. This time is in nanoseconds. To convert it into a percentage utilization of CPU we sample the output of getCPUStats every t seconds. Let $cputTimeDiff$ be the change in CPU time over this time t i.e., $cpuTimeDiff = cpuTime_{now} - cpuTime_{now-t}$

Then percentage utilization is calculated as follows $\%CPUutilization = 100 * cpuTimeDiff/(t * numberofcores * 10^9)$

Memory statistics

To get the memory statistics of the domain using libvirt's python API the function memoryStats can be used. It gives the actual memory that is allotted to the domain, the swap memory and the resident set size is given as the output. Resident set size is the portion of memory occupied by that domain in the main memory (RAM).The rest of the memory may be in swap space or file system.

3.3 Overview of Baadal Networking Architecture

In this section we will present the overview of the baadal networking architecture. The components comprising the architecture are mentioned below:-

- **NAT** Network Address translator is the component which act as firewall for the baadal network. All the traffic of the baadal network pass through it. Internal baadal nodes communicate to the internet via this interface only. It is implemented on an OpenVswitch which is hosted in a physical machine with Ubuntu as its OS.

- **Hardware Switch** It is a physical switch which is connected to all the other networking components including the physical hosts, Baadal Controller, Filer and NAT.

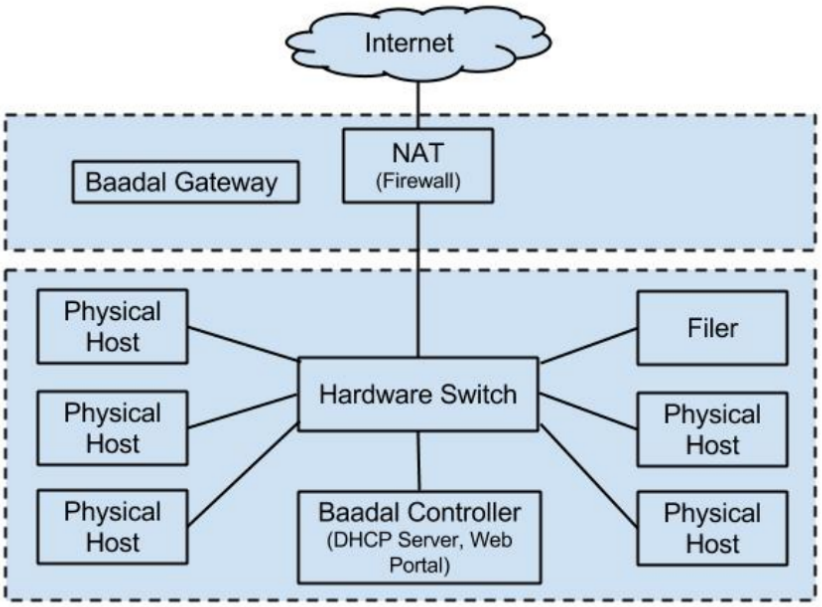

Figure 3.7: Overview of Baadal Architecture

- **Baadal Controller** This component is responsible for providing web portal where a user can request a VM, a facutly supervisor approves it and an administrator can commission it. The main function of it is to be central controlling element alongwith hosting components like DHCP server, TFTP server, scheduler etc.

- **Filer** For all the physical hosts Filer is the network file manager. On request of virtual machines, controller hosts the virtual hard disk. But additional hard disk is allocated in the filer machine.

- **Physical host** These are responsible for hosting all the VMs created by the user. KVM is the hypervisor layer in the physical host.

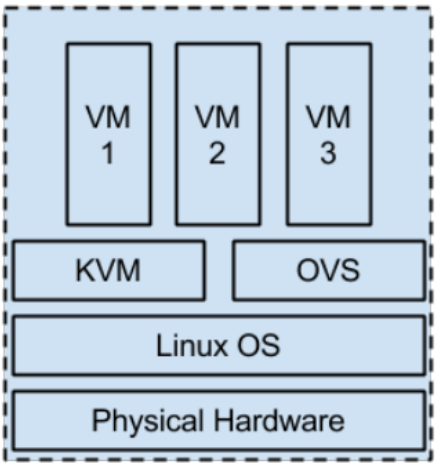

Figure 3.8: Physical host architecture

Chapter 4

Baadal Sandbox Setup Usage

This chapter will provide the necessary commmands, interfaces and methodology to use baadal sandbox. We thought of including this as we spent much of the effort in understanding baadal system. There is no proper documentation on usage of baadal sandbox. Hence we put necessary screenshots in this chapter to enable others use baadal sandbox in convenient manner.

Following are the steps to start with sandbox server we established:-

1. Login to server used for baadal sandbox setup

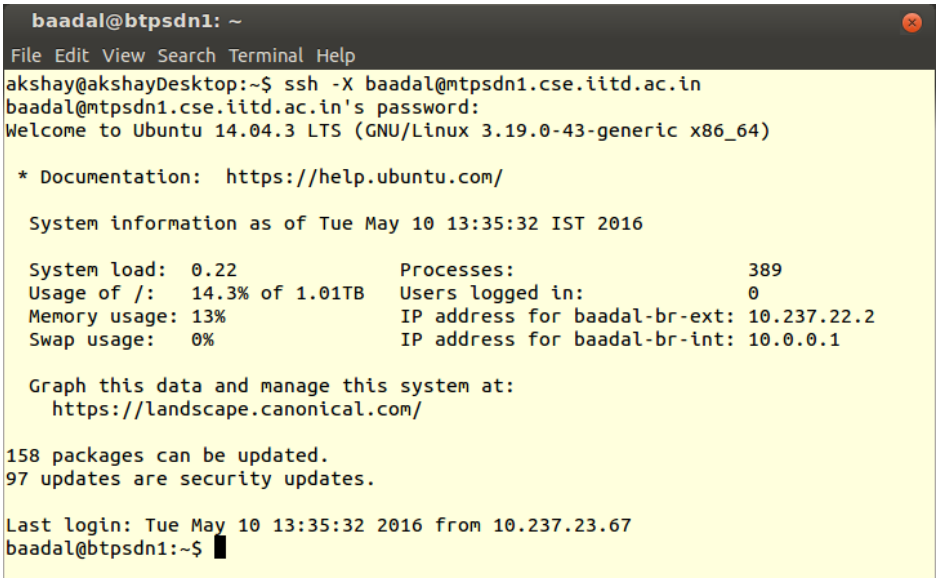

Figure 4.1: Login to the baadal sandbox server

2. For status of VMs running in sandbox we can use *sudo virsh list –all*

```
baadal@btpsdn1:~$ sudo virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     baadal_controller              running
 4     baadal_host_2                  running
 5     baadal_host_3                  running
 6     baadal_nat                     running
 -     baadal-sdn-controller          shut off
```

Figure 4.2: Different VMs running in the sandbox

3. Using *ifconfig* we can see the network configuration for the server.

```
baadal@btpsdn1:~$ ifconfig
baadal-br-ext Link encap:Ethernet  HWaddr 44:a8:42:29:59:ff
          inet addr:10.237.22.2  Bcast:10.237.23.255  Mask:255.255.252.0
          inet6 addr: fe80::a8ba:bdff:fe5e:a83b/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:7794066 errors:0 dropped:0 overruns:0 frame:0
          TX packets:43380 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1434514200 (1.4 GB)  TX bytes:2328631 (2.3 MB)

baadal-br-int Link encap:Ethernet  HWaddr 16:87:82:b3:a4:4d
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::e078:f9ff:fee2:171/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:405 errors:0 dropped:0 overruns:0 frame:0
          TX packets:164 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:34866 (34.8 KB)  TX bytes:18780 (18.7 KB)

em1       Link encap:Ethernet  HWaddr 44:a8:42:29:59:ff
          inet6 addr: fe80::46a8:42ff:fe29:59ff/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:79630140 errors:0 dropped:0 overruns:0 frame:0
          TX packets:81703 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6642002534 (6.6 GB)  TX bytes:5669950 (5.6 MB)
          Interrupt:18

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:277925 errors:0 dropped:0 overruns:0 frame:0
          TX packets:277925 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:13896312 (13.8 MB)  TX bytes:13896312 (13.8 MB)

vnet0     Link encap:Ethernet  HWaddr fe:52:00:01:15:02
          inet6 addr: fe80::fc52:ff:fe01:1502/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:264 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1555 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:19236 (19.2 KB)  TX bytes:132342 (132.3 KB)

vnet1     Link encap:Ethernet  HWaddr fe:52:00:01:15:06
          inet6 addr: fe80::fc52:ff:fe01:1506/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:111 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1651 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
```

Figure 4.3: Output of ifconfig command on sandbox server

4. login inside the controller

```
baadal@btpsdn1:~$ ssh baadal@10.0.0.2
baadal@10.0.0.2's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.16.0-30-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue May 10 07:28:07 IST 2016

  System load: 0.0                Memory usage: 1%   Processes:       96
  Usage of /:  38.5% of 9.63GB    Swap usage:   0%   Users logged in: 0

  Graph this data and manage this system at:
    https://landscape.canonical.com/

152 packages can be updated.
88 updates are security updates.

Last login: Tue Mar  8 08:42:22 2016 from 10.0.0.1
baadal@baadal-controller:~$
```

Figure 4.4: Login in the baadal controller

```
vlan250   Link encap:Ethernet  HWaddr 62:cf:54:8f:67:50
          inet addr:10.0.250.2  Bcast:10.0.250.255  Mask:255.255.255.0
          inet6 addr: fe80::60cf:54ff:fe8f:6750/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:390 (390.0 B)  TX bytes:648 (648.0 B)

vlan251   Link encap:Ethernet  HWaddr 9e:c9:9f:57:a4:ee
          inet addr:10.0.251.2  Bcast:10.0.251.255  Mask:255.255.255.0
          inet6 addr: fe80::9cc9:9fff:fe57:a4ee/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:468 (468.0 B)  TX bytes:648 (648.0 B)

vlan252   Link encap:Ethernet  HWaddr ae:4a:ff:9d:75:c0
          inet addr:10.0.252.2  Bcast:10.0.252.255  Mask:255.255.255.0
          inet6 addr: fe80::ac4a:ffff:fe9d:75c0/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:558 (558.0 B)  TX bytes:648 (648.0 B)

vlan253   Link encap:Ethernet  HWaddr 32:6b:ec:8a:a6:3e
          inet addr:10.0.253.2  Bcast:10.0.253.255  Mask:255.255.255.0
          inet6 addr: fe80::306b:ecff:fe8a:a63e/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:390 (390.0 B)  TX bytes:648 (648.0 B)

vlan254   Link encap:Ethernet  HWaddr 6a:f4:30:02:ae:4d
          inet addr:10.0.254.2  Bcast:10.0.254.255  Mask:255.255.255.0
          inet6 addr: fe80::68f4:30ff:fe02:ae4d/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:390 (390.0 B)  TX bytes:648 (648.0 B)

vlan255   Link encap:Ethernet  HWaddr 06:0e:62:e3:ed:20
          inet addr:10.0.255.2  Bcast:10.0.255.255  Mask:255.255.255.0
          inet6 addr: fe80::40e:62ff:fee3:ed20/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:230 (230.0 B)  TX bytes:648 (648.0 B)

baadal@baadal-controller:~$
```

Figure 4.5: ifconfig showing 255 fake bridges inside the controller

```
Last login: Tue May 10 07:33:27 2016 from 10.0.0.1
baadal@baadal-controller:~$ ps aux | grep web2py
www-data 18719  0.0  0.6 121476 25336 ?       Sl   07:28   0:00 python web2py.py -K baadal:vm_task,baadal:vm_sanity,baadal:host_task,baadal:vm_rrd,baadal:snapshot_task
www-data 18775  0.2  1.1 371844 46200 ?       Sl   07:28   0:01 python web2py.py -K baadal:vm_task,baadal:vm_sanity,baadal:host_task,baadal:vm_rrd,baadal:snapshot_task
www-data 18779  0.3  1.1 370796 44964 ?       Sl   07:28   0:01 python web2py.py -K baadal:vm_task,baadal:vm_sanity,baadal:host_task,baadal:vm_rrd,baadal:snapshot_task
www-data 18878  0.2  1.1 370772 44848 ?       Sl   07:28   0:01 python web2py.py -K baadal:vm_task,baadal:vm_sanity,baadal:host_task,baadal:vm_rrd,baadal:snapshot_task
www-data 18883  0.2  1.1 370772 44844 ?       Sl   07:28   0:01 python web2py.py -K baadal:vm_task,baadal:vm_sanity,baadal:host_task,baadal:vm_rrd,baadal:snapshot_task
www-data 18888  0.2  1.1 370796 44968 ?       Sl   07:28   0:01 python web2py.py -K baadal:vm_task,baadal:vm_sanity,baadal:host_task,baadal:vm_rrd,baadal:snapshot_task
baadal   19154  0.0  0.0  10712  2172 pts/0   S+   07:36   0:00 grep --color=auto web2py
```

Figure 4.6: Web2py Processes Running inside controller

5. Login inside the nat is shown as

```
baadal@btpsdn1:~$ ssh baadal@10.0.0.3
baadal@10.0.0.3's password:
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.16.0-30-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue May 10 07:29:19 IST 2016

  System load: 0.0                Memory usage: 1%   Processes:       141
  Usage of /:  12.7% of 19.47GB   Swap usage:   0%   Users logged in: 0

  Graph this data and manage this system at:
    https://landscape.canonical.com/

199 packages can be updated.
105 updates are security updates.

Last login: Mon Apr 18 09:06:19 2016 from 10.0.0.1
baadal@baadal-nat:~$
```

Figure 4.7: Login inside the nat

6. Login inside host
7. *virsh list –all* shows the VMs running inside 10.0.0.6
8. *virsh list –all* shows the VMs running inside 10.0.0.7
9. *virt-viewer domain_name* will open the GUI for the VM whose domain name is specified. Below figure shows for VM *ADMIN_admin_newvm25*

```
vlan250   Link encap:Ethernet  HWaddr 0e:5c:dc:55:12:3d
          inet addr:10.0.250.1  Bcast:10.0.250.255  Mask:255.255.255.0
          inet6 addr: fe80::c5c:dcff:fe55:123d/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)

vlan251   Link encap:Ethernet  HWaddr 4e:4a:36:a9:de:af
          inet addr:10.0.251.1  Bcast:10.0.251.255  Mask:255.255.255.0
          inet6 addr: fe80::4c4a:36ff:fea9:deaf/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)

vlan252   Link encap:Ethernet  HWaddr 1a:b1:f0:d5:b2:0e
          inet addr:10.0.252.1  Bcast:10.0.252.255  Mask:255.255.255.0
          inet6 addr: fe80::18b1:f0ff:fed5:b20e/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)

vlan253   Link encap:Ethernet  HWaddr 7a:09:3e:d6:8e:dc
          inet addr:10.0.253.1  Bcast:10.0.253.255  Mask:255.255.255.0
          inet6 addr: fe80::7809:3eff:fed6:8edc/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)

vlan254   Link encap:Ethernet  HWaddr 12:2f:d1:89:2c:4d
          inet addr:10.0.254.1  Bcast:10.0.254.255  Mask:255.255.255.0
          inet6 addr: fe80::102f:d1ff:fe89:2c4d/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)

vlan255   Link encap:Ethernet  HWaddr 2e:f7:ba:e2:7c:e0
          inet addr:10.0.255.1  Bcast:10.0.255.255  Mask:255.255.255.0
          inet6 addr: fe80::2cf7:baff:fee2:7ce0/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)

baadal@baadal-nat:~$
```

Figure 4.8: ifconfig showing 255 fake bridges inside the NAT

```
baadal@btpsdn1:~$ ssh root@10.0.0.6
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.16.0-30-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Apr 19 07:42:23 2016 from 10.0.0.1
root@host-10-0-0-6:~#
```

Figure 4.9: Login inside host 10.0.0.6

```
root@host-10-0-0-6:~# ifconfig
baadal-br-int Link encap:Ethernet  HWaddr 52:52:00:01:15:06
          inet addr:10.0.0.6  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::c087:69ff:fe7b:b0c8/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:95642 errors:0 dropped:0 overruns:0 frame:0
          TX packets:113954 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:647471697 (647.4 MB)  TX bytes:103875100 (103.8 MB)

eth0      Link encap:Ethernet  HWaddr 52:52:00:01:15:06
          inet6 addr: fe80::5052:ff:fe01:1506/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:489421 errors:0 dropped:216 overruns:0 frame:0
          TX packets:145396 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:666607536 (666.6 MB)  TX bytes:105955845 (105.9 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:72 errors:0 dropped:0 overruns:0 frame:0
          TX packets:72 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5696 (5.6 KB)  TX bytes:5696 (5.6 KB)

vnet0     Link encap:Ethernet  HWaddr fe:00:00:e8:30:77
          inet6 addr: fe80::fc00:ff:fee8:3077/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:65 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:9931 (9.9 KB)

vnet1     Link encap:Ethernet  HWaddr fe:00:00:a3:90:5c
          inet6 addr: fe80::fc00:ff:fea3:905c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:57 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:9283 (9.2 KB)  TX bytes:648 (648.0 B)

vnet2     Link encap:Ethernet  HWaddr fe:00:00:fc:a9:8b
          inet6 addr: fe80::fc00:ff:fefc:a98b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)
```

Figure 4.10: Network configuration of host 10.0.0.6

```
baadal@btpsdn1:~$ ssh root@10.0.0.7
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.16.0-30-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Apr 19 07:42:56 2016 from 10.0.0.1
root@host-10-0-0-7:~#
```

Figure 4.11: Login inside host 10.0.0.7

```
root@host-10-0-0-7:~# ifconfig
baadal-br-int Link encap:Ethernet  HWaddr 52:52:00:01:15:07
          inet addr:10.0.0.7  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::1814:4cff:fe02:32db/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:196258 errors:0 dropped:0 overruns:0 frame:0
          TX packets:205128 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1652301293 (1.6 GB)  TX bytes:159599342 (159.5 MB)

eth0      Link encap:Ethernet  HWaddr 52:52:00:01:15:07
          inet6 addr: fe80::5052:ff:fe01:1507/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1207880 errors:0 dropped:2559 overruns:0 frame:0
          TX packets:278423 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1702153866 (1.7 GB)  TX bytes:164462997 (164.4 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:56 errors:0 dropped:0 overruns:0 frame:0
          TX packets:56 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4294 (4.2 KB)  TX bytes:4294 (4.2 KB)

vnet0     Link encap:Ethernet  HWaddr fe:00:00:e5:7f:6f
          inet6 addr: fe80::fc00:ff:fee5:7f6f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:76 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:12567 (12.5 KB)  TX bytes:1951 (1.9 KB)

vnet1     Link encap:Ethernet  HWaddr fe:00:00:6c:f8:ec
          inet6 addr: fe80::fc00:ff:fe6c:f8ec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:66 errors:0 dropped:0 overruns:0 frame:0
          TX packets:86 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:14536 (14.5 KB)  TX bytes:13358 (13.3 KB)

vnet2     Link encap:Ethernet  HWaddr fe:00:00:79:3d:56
          inet6 addr: fe80::fc00:ff:fe79:3d56/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:78 errors:0 dropped:0 overruns:0 frame:0
          TX packets:74 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:12710 (12.7 KB)  TX bytes:15184 (15.1 KB)

vnet3     Link encap:Ethernet  HWaddr fe:00:00:94:e0:de
          inet6 addr: fe80::fc00:ff:fe94:e0de/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

Figure 4.12: Network configuration of host 10.0.0.6

```
root@host-10-0-0-6:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 -     ADMIN_admin_newvmm22           shut off
 -     ADMIN_admin_newvmm23           shut off
 -     ADMIN_admin_newvmm24           shut off
```

Figure 4.13: VMs shutdown status inside host 10.0.0.6

```
root@host-10-0-0-6:~# virsh start ADMIN_admin_newvmm22
Domain ADMIN_admin_newvmm22 started

root@host-10-0-0-6:~# virsh start ADMIN_admin_newvmm24
Domain ADMIN_admin_newvmm24 started

root@host-10-0-0-6:~# virsh start ADMIN_admin_newvmm23
Domain ADMIN_admin_newvmm23 started
```

Figure 4.14: Starting VMs inside host 10.0.0.6

```
root@host-10-0-0-6:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     ADMIN_admin_newvmm22           running
 3     ADMIN_admin_newvmm24           running
 4     ADMIN_admin_newvmm23           running
```

Figure 4.15: Running VMs inside host 10.0.0.6

```
root@host-10-0-0-7:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 -     ADMIN_admin_newvm13            shut off
 -     ADMIN_admin_newvm14            shut off
 -     ADMIN_admin_newvm25            shut off
 -     ADMIN_admin_newvmm222          shut off

root@host-10-0-0-7:~#
```

Figure 4.16: VMs shutdown status inside host 10.0.0.7

```
root@host-10-0-0-7:~# virsh start ADMIN_admin_newvm13
Domain ADMIN_admin_newvm13 started

root@host-10-0-0-7:~# virsh start ADMIN_admin_newvm14
Domain ADMIN_admin_newvm14 started

root@host-10-0-0-7:~# virsh start ADMIN_admin_newvm25
Domain ADMIN_admin_newvm25 started

root@host-10-0-0-7:~# virsh start ADMIN_admin_newvmm222
Domain ADMIN_admin_newvmm222 started
```

Figure 4.17: Starting VMs inside host 10.0.0.7

```
root@host-10-0-0-7:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     ADMIN_admin_newvm13            running
 3     ADMIN_admin_newvm14            running
 4     ADMIN_admin_newvm25            running
 5     ADMIN_admin_newvmm222          running
```

Figure 4.18: Running VMs inside host 10.0.0.7

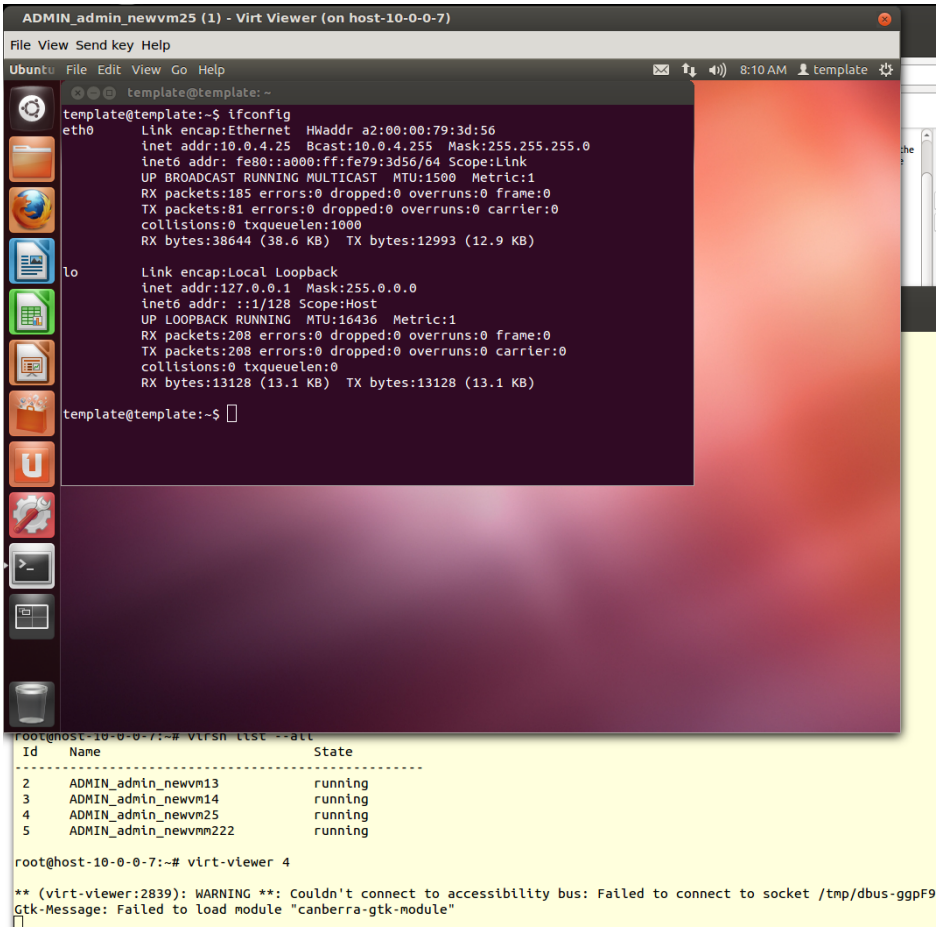

Figure 4.19: Console output for VM 10.0.4.25 after running virt-viewer

Chapter 5

Introduction to SDN based Application

We start this chapter with the introduction of the SDN controller Floodlight, using which we made policy control and load balancing application. We will see the sandbox setup to understand the networking without SDN and with SDN. The algorithm used in the policy control application for intervlan VMs is also explained.

5.1 Introduction to Floodlight

Floodlight is an open source java-based SDN controller and was the controller of choice for this project because of reasons such as - it supports switches which are OpenFlow enabled, it is Apache licensed and can be used for almost any purpose, it has good documentation and a very active community of professional developers who were able to resolve any issues that we needed help with.

5.2 Floodlight Architecture

- **OpenFlow** It works with both physical- and virtual- switches that support the OpenFlow protocol
- Offers a module loading system that make it simple to extend and enhance.
- **Open community** It being developed by an open community of developers, it welcomes code contributions from all active users and share information on project status, roadmap, bugs, etc.

- **Easy to Use**- It provides with a documentation link, using which we can learn to use it according to our application
- **Tested and Supported** It is thoroughly tested by its community and is compatible with other frameworks.

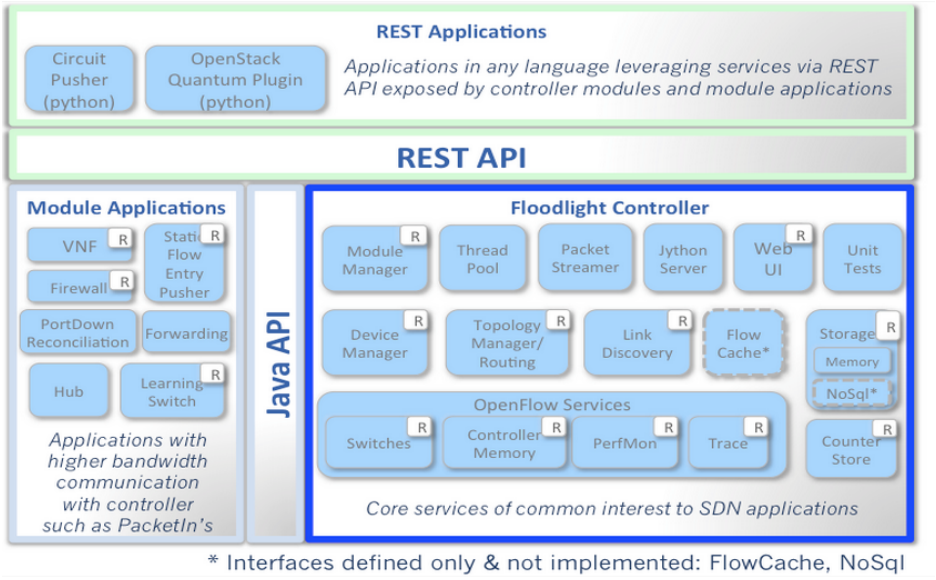

Figure 5.1: Floodlight Architecture

5.3 Baadal Sandbox setup

For testing the simulation of the application, we need an environment like baadal. Baadal sandbox provides an environment by replicating actual Baadal architecture and configurations. It is fast and easy way to test the solution of network virtualisation.

If tested successfully, we can subsequently deploy the solution in actual Baadal with some modifications.

The differences with which sandbox mimics baadal architecture are:-

- Physical hosts, baadal controller, filer, NAT are implemented as VMs on a single physical machine

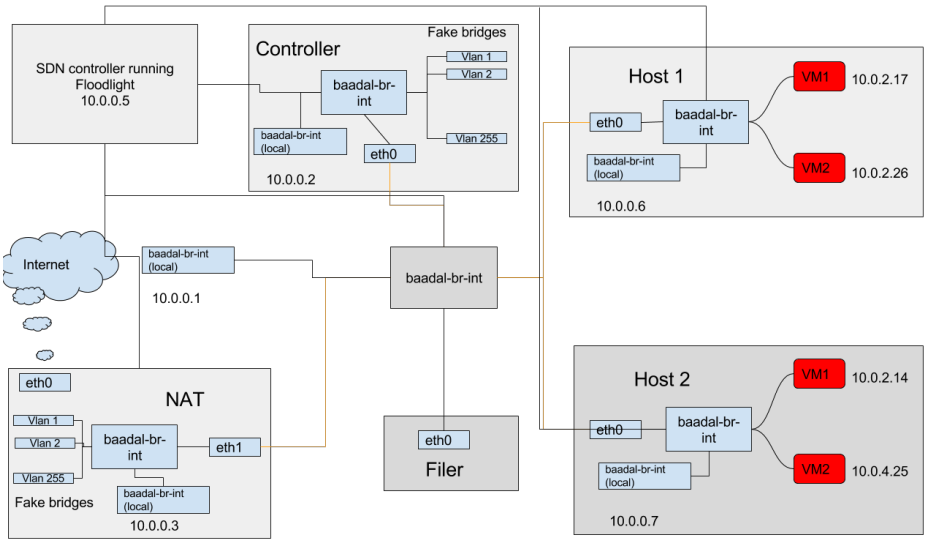

Figure 5.2: Setup of Baadal Sandbox Installed

- OVS is the switch used for implementing hardware switches
- All the VMs are implemented into the physical hosts which makes two level of virtualisation

5.4 Inter-VLan Issues and Modifications suggested

In following subsections we will present the networking details of baadal sandbox with their limitations. We will suggest the modification based on which we derived our SDN solution.

5.4.1 How do VMs communicate?

In the existing Baadal network as explained above, the inter VM communication takes place as follows:

- **VMs in same host and same VLAN/subnet** In this case, the OVS bridge in the host would simply be able to let them communicate by learning their mac-port pair as they fall in the same broadcast domain.

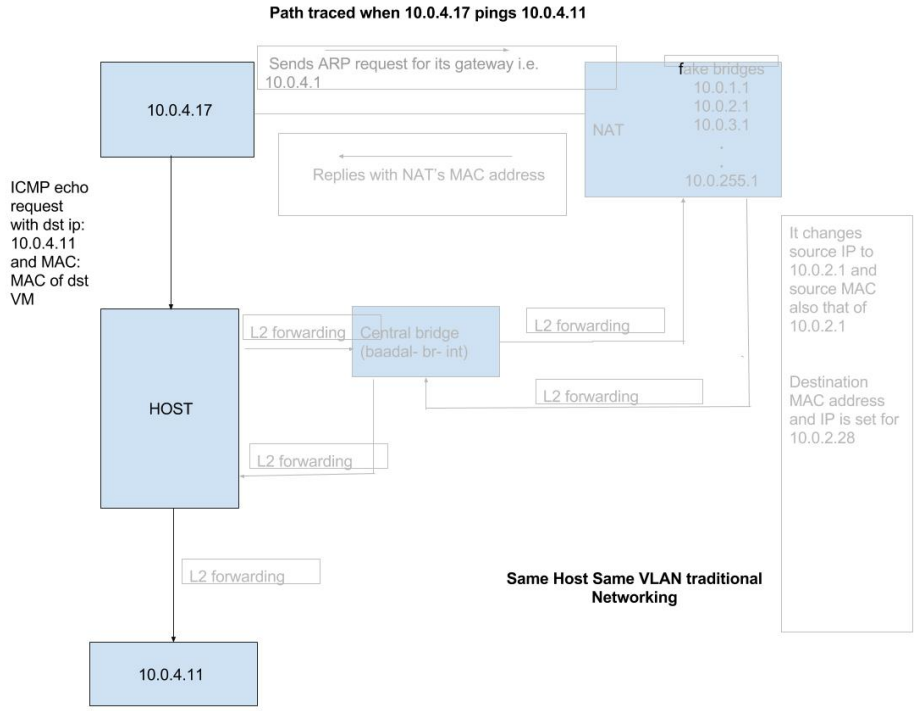

Figure 5.3: Routing in Same Host and Same VLAN

The VLAN tag corresponding to the VLAN will be attached at the ingress port while the egress port will strip it off before sending the packets out.

- **VMs in different host but same VLAN/subnet** In this case, still, the broad- cast domain is the same. So, in addition to the host OVS bridge, the central switch will come into picture. The process will be same as above case except for the extra central switch through which the packets would travel.

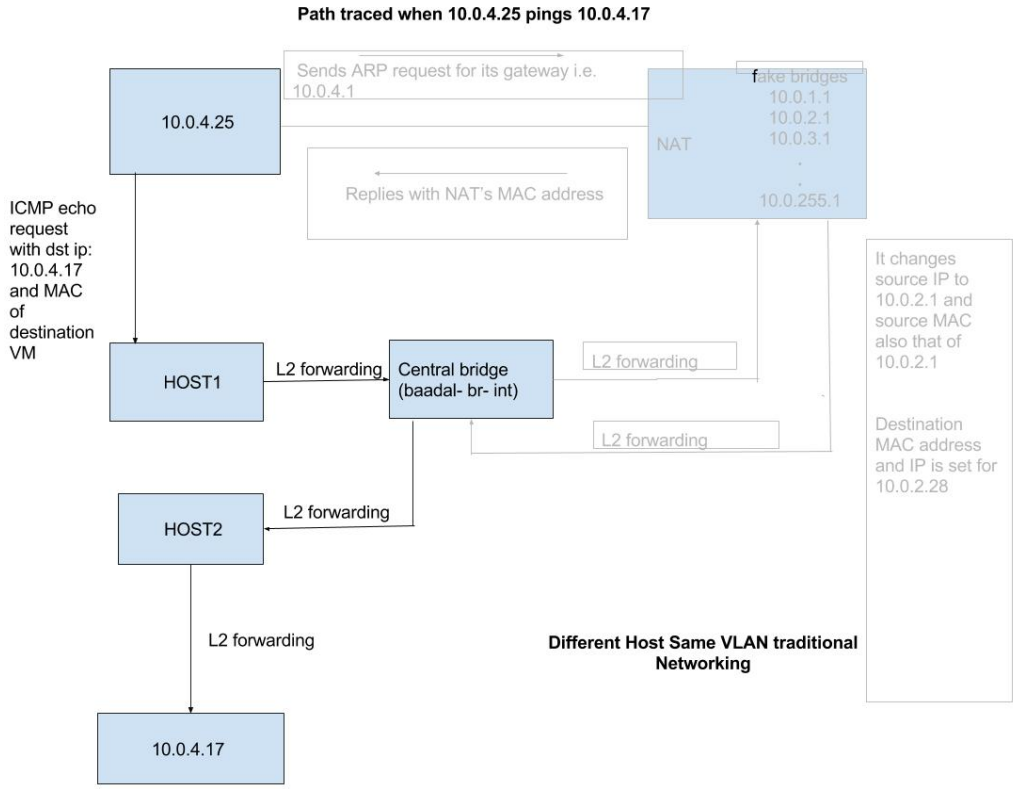

Figure 5.4: Routing in Same Host and Same VLAN

- **VMs in the same host but different VLANs/subnets**The host OVS bridge, in this case, would not be able to forward the packets as the broadcast domains are different. As the gateway is fake bridge for that VLAN, the packets go all the way up to the NAT machine passing through the central switch. The fake bridges functions to modify the VLAN tag from the in- put VLAN to the output VLAN. The modified packets are then forwarded at the appropriate port of NAT OVS bridge and then they come back to the same host. Finally, the OVS bridge in the host strip off the VLAN tag and send it out the appropriate port.

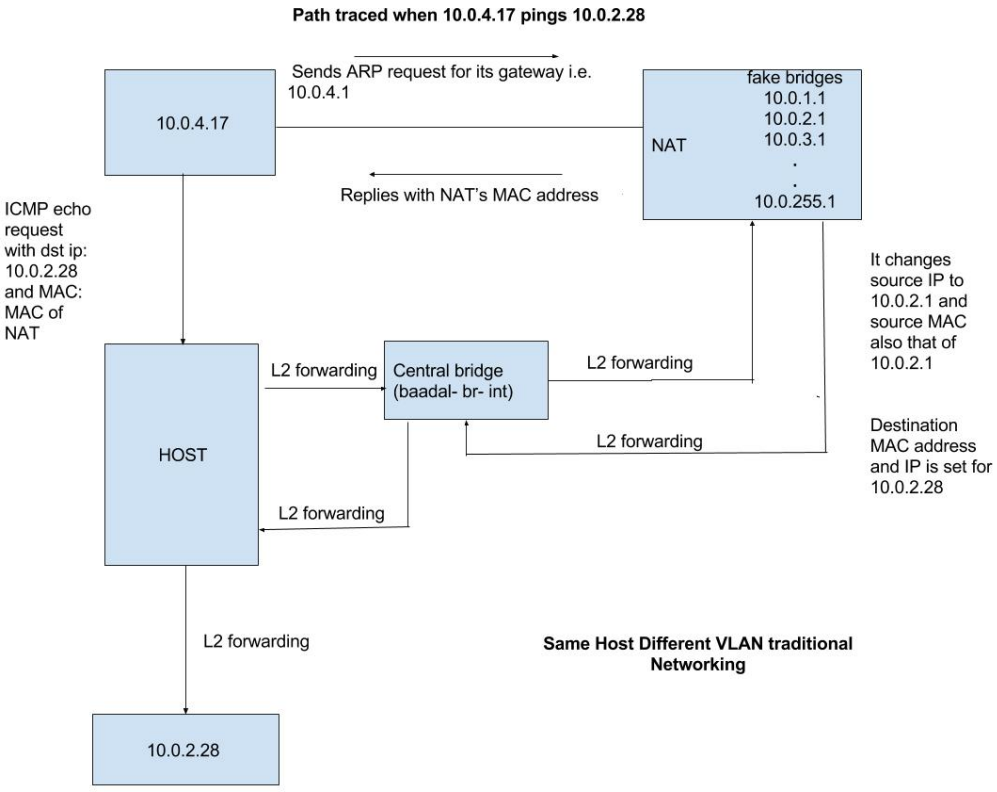

Figure 5.5: Routing in Same Host and Different VLAN

- **VMs in different hosts and different VLANs/subnets** the process is ex- actly similar as in the previous case. The only change - different destina- tion host - would be taken care off by the central switch as a part of its normal l2-learning functionality.

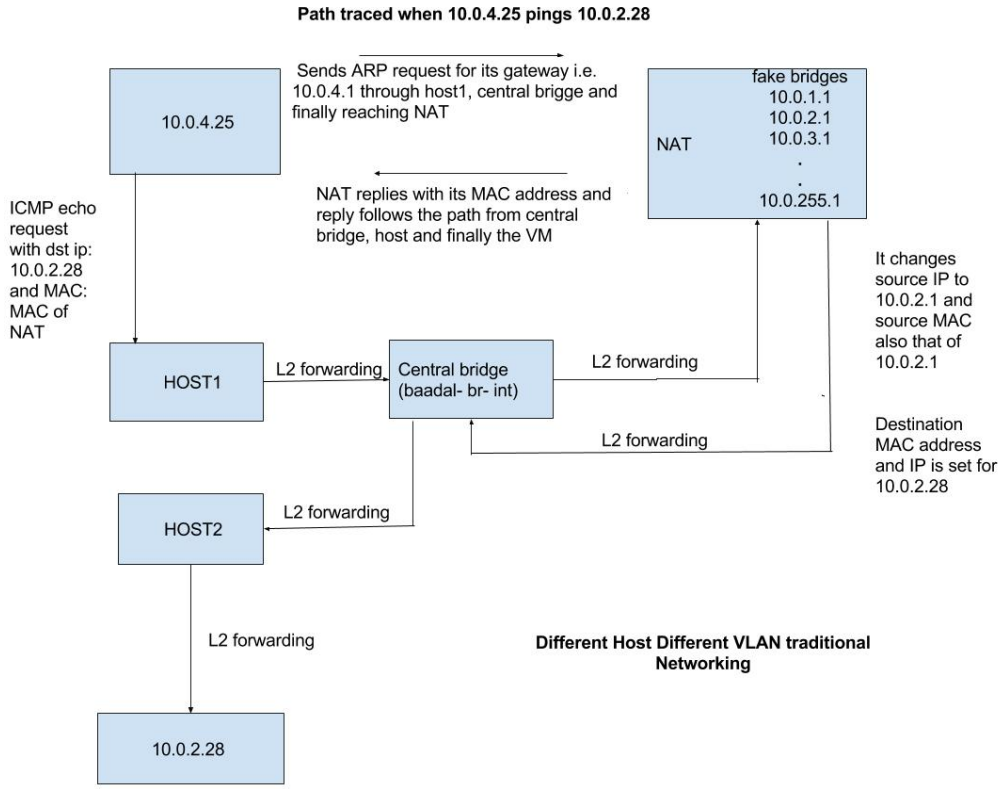

Figure 5.6: Routing in Different Host and Different VLAN

5.4.2 Limitation in present architecture

We find the following limitations in the present architecture.

- **Inter-VLAN or subnet routing** In the above architecture, fake bridges are used as the mean to enable this feature. The issue is when VMs in the same host but different subnet want to communicate. The packets have to go all the way upto NAT machine and come back. This seems unnecessary strain on the network and large latencies.
- **Controlling inter-VLAN routing** Currently, there is no way of turning VLAN routing on or off in Baadal. It is enabled by default. A granular level control of this feature with the current network architecture seems non-trivial and difficult to implement
- **Maximum number of VLANs** This architecture has an upper boung of 255 VLANs because of the one to one correspondence between the subnets and the VLANs.
- **Number of VMs in a VLAN** The number of VMs in each VLAN in the cur- rent implementation is upper bound at 255. This is when considering the addressing scheme of 192.168.x.y or 172.16.x.y with 255x255 addresses available.

5.4.3 Modification in Network Architecture

We proposed and implementing the following modifications in the network architecture.

- **No subnets** The reason subnets are used is because they define the broadcast domains. The domains restrict the hops that broadcast packets travel and helps in controlling traffic volume in network. But, with a SDN controller application, the same thing could be acheived without having the subnet restriction. The reason we want to remove it is bacause the fake bridges are required only to enable inter subnet routing. Fake bridges being in NAT requires packet to cover all the

way upto NAT and back. Thus, with just one subnet, the local OVS bridges would be able to forward the packets.

- **VLANs implemented using VLAN tags only** As opposed to having both subnets and VLAN tags to implement VLANs, we use only VLAN tags. The IEEE 802.1q protocol provides 12 bit VLAN tag amounting to 4094 tags. This would enable us to have 4096 VLANs as opposed to just 255! We can further extend this to even larger numbers if we use both inner and outer VLAN tags. Note that the VMs and the hosts are not VLAN aware in the current as well as the proposed network architecture.

5.5 Introduction to Policy routing application

The virtual machines running in Baadal are allotted a security domain (a vlan) to segregate traffic of a certain vlan from other vlans. With using the traditional networking approach there is no way to disable it (inter-vlan routing is disabled by default in Baadal). With the SDN solution we have the capability to enable or disable it using a REST API. Moreover, there is also a provision for enabling/disabling communication between two virtual machines irrespective of their security domain. The use case for this feature would be a scenario in which people working with virtual machines in different vlans (because of them being in different departments or areas of the campus) need to collaborate with each other.

5.6 Application that we are working on: Functionalities

The application that we are working on implements modified network architecture as suggested above. The important tasks that will be handled in the application are as follows:

- Tagging the packets if they ingress at an access port in a host and egress from the trunk port.
- Untagging the packet when they are to egress to an access port and they have a tag attached
- The application handles ARP and DHCP packets which are special kind of packets. The details of how it is being done is presented in the following sections.
- The application takes care of security by restricting which kind of packets reach where.
- It takes care of forwarding normal traffic from one node to other.
- It handles inter-VLAN routing as well.

5.7 Floodlight modules

Floodlight has modular architecture to implement any application over it and control its features. There are two kind of modules in floodlight

- **Controller Modules** these implement core network services a software defined network should expose to applications
- **Application Modules** these implement solutions for different purposes like firewall, static flow pusher, L2 learning switch etc.

Module loading system has a JAVA interface IFloodlightModule that realises this framework.

5.7.1 Module Loading System

The following are the objectives of the this system

- What are the modules that are to be loaded using a configuration file

- Modify the implementation of modules without affecting other modules that are dependent on them
- Creating a platform and API to extend Floodlight
- Code Modularity is enforced

The main parts of this system are

- **Modules** It is a class that implements the IFloodlightModule interface
- **Services** these are the services that the module provides. these are implemented as interface that extends the IFloodlightService interface
- **Configuration file** It contains the modules that are to be loaded
- **module file** It contains list of all classes used by JAVA service loader

5.7.2 How to write a Module

Following are main steps for writing a Floodlight Module

1. Add a class to implement IOFMessageListener and IFloodlightModule interfaces
2. Import Module Dependencies and set up Initialization
3. Register a listener for PACKET_IN Messages
4. Ordering the modules for processing OpenFlow Messages before or after a particular module as required by the application
5. Register the module in configuration file to load the module on startup

5.7.3 Module: Baadal

It implements IOFMessageListener, IFloodlightModule, and IBaadalService. The data structures used are

- **gateways** list of ipv4 addresses for routing between different subnets.
- **ipToTag** stores mapping of ipv4 addresses to VLAN tags
- **interVmPolicy** Stores the status(enabled or disabled or default) of policy routing between a pair of VMs

The main functions implemented are

- **processPacketIn()** sends the packet for processing according to the source switch that it is coming from
- **init()** initialises variables and data structures (ipToTag, interVmPolicy etc.)
- **TimerTask()** schedules a task to clear cached entries in data structures at regular intervals
- **getIpToTag()** it returns mapping between IP address and corresponding VLAN tag
- **addIpToTag()** adds entries for mapping between IP address and VLAN tag
- **getInterVmPolicy** it returns the status of policy control for different pair of VMs
- **addInterVmPolicy** it adds entries into inter VM policy table as per requirement

5.8 How do VMs communicate after implementation of application

After running the policy control application, the modifications in the routing of different cases are

- **Same host and same VLAN** In this case routing design will be same as per traditional routing in baadal where the host will simply forward the packet based on the MAC address.

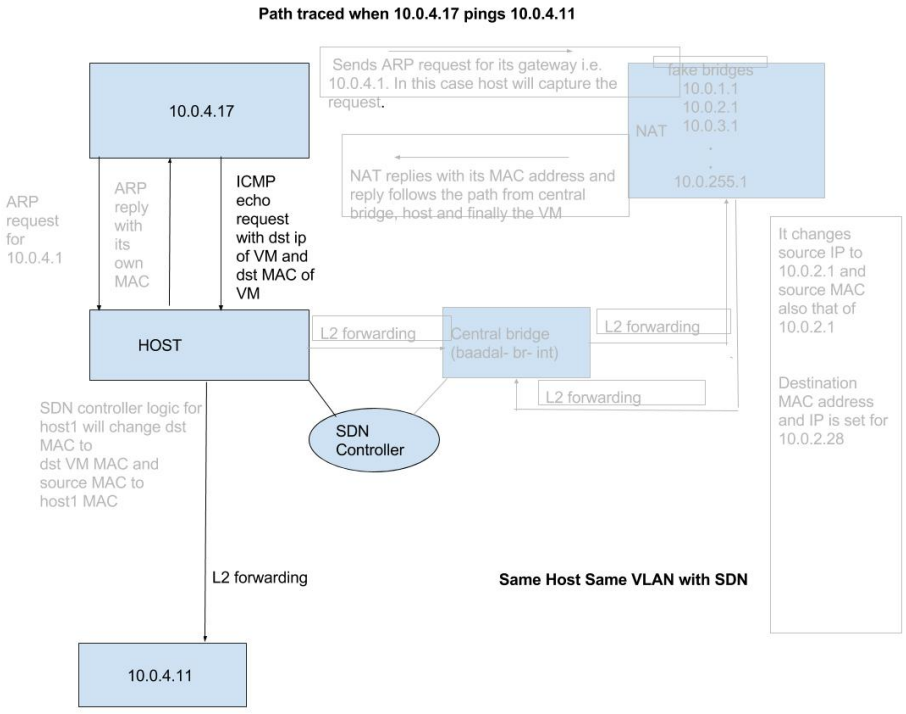

Figure 5.7: Routing in Same Host and Same VLAN using SDN

- **Same host and Different VLAN** In this case host itself will reply to the ARP request, sent by the VM for its gateway. In traditional routing, this reply was generated by the NAT. Now the host will forward the packet to the destination VM by changing the ethernet frame (setting the destination MAC address to Destination VM MAC address and Source MAC address to MAC address of the host)

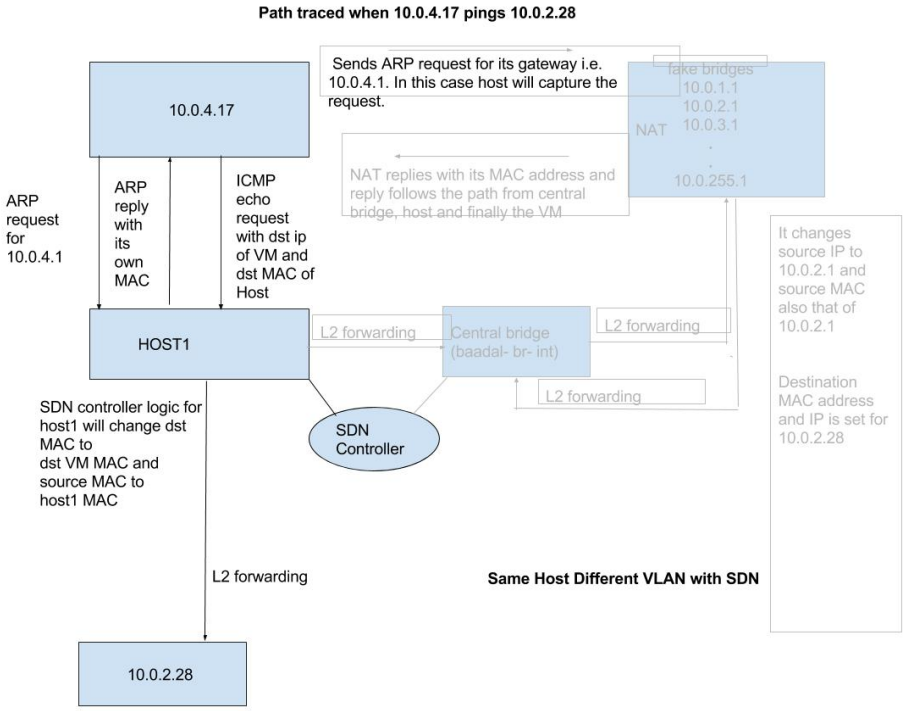

Figure 5.8: Routing in Same Host and different VLAN using SDN

- **Different host and Different VLAN** In this case host itself will reply to the ARP request, sent by the VM for its gateway. In traditional routing, this reply was generated by the NAT. But now Central bridge will come into the picture to forward the packet to the other host2 unlike previous case. The modification in the Ethernet frame here is done by the host1 only. Host2 will simply forward the packet to the destination VM.

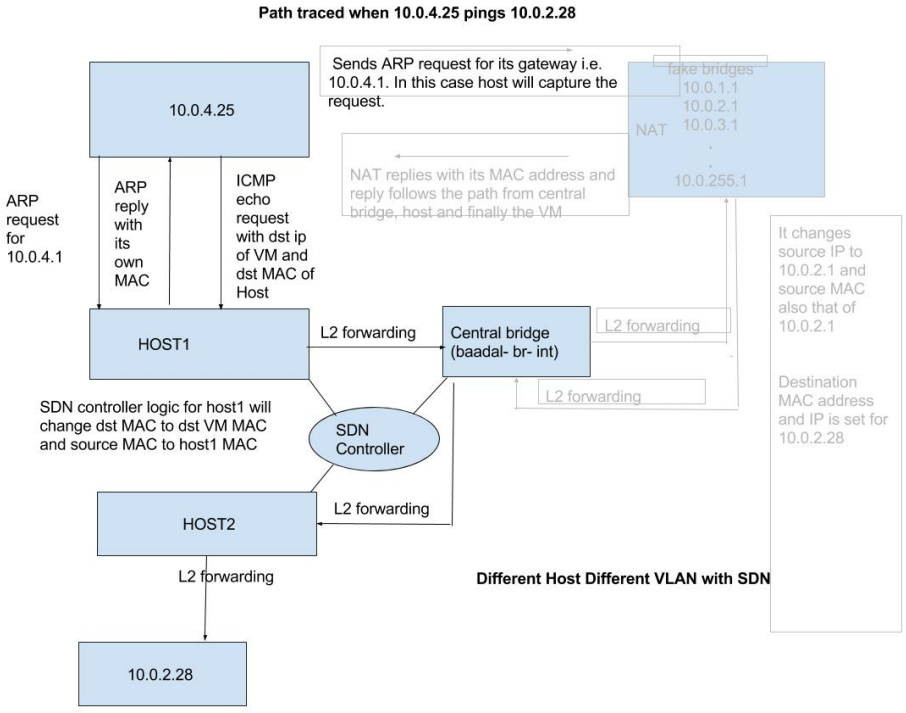

Figure 5.9: Routing in Different Host and Different VLAN using SDN

- **Different host and Same VLAN** In this, the packet will be forwarded by the host on the basis of the MAC address to the central bridge.The central bridge will further forward it to the other host which subsequently make the packet reach destination VM.

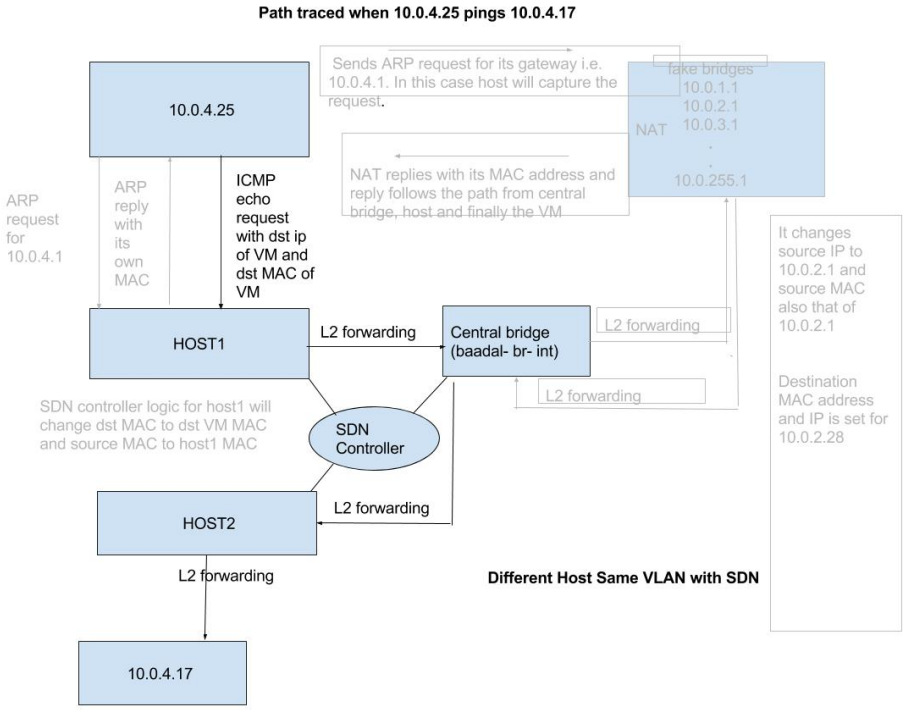

Figure 5.10: Routing in Different Host and Same VLAN using SDN

5.9 Application pseudo code

Algorithm 1: Handling arp requests at hosts

```
if packet type == IPv6_TYPE then
    Drop the packet;
    Install flow to drop matching packets;
end
if packet type== ARP_TYPE then
    if ARP type == REQUEST then
        if destinationProtocolAddress is one of the gateways then
            send reply with host's mac address;
            don't send the packet for further processing;
        else
            send the packet for further processing;
        end
    end
    if ARP type == REPLY then
        store the incoming ip to mac mapping;
        send the packet for further processing;
    end
end
```

Algorithm 2: Routing packets coming from local port of hosts

```
if input port = local port then
    if mac corresponding to destination ip is not known then
        send ARP request for destination ip;
        wait for 100ms;
    end
    if mac corresponding to destination ip is still not known then
        drop the packet;
    else
        replace the destination mac address of the packet with the mac
        corresponding to the destination ip read from the table;
    end
    if output port is known then
        send the packet out;
        install flow for all the above actions;
    else
        drop the packet;
    end
end
```

Algorithm 3: Routing packets coming from trunk port of hosts

```
if input port = trunk port then
    if output port corresponding to destination mac is not known then
        send ARP request for destination ip and wait for 100ms;
    end
    if output port is still not known then
        drop the packet;
    else
        if output port is local or trunk then
            send packet out and install flow;
        else
            if packet is untagged then
                send packet out and install flow;
            else
                get policy decision between source and target ip address;
                if decision == ALLOW then
                    pop vlan, send packet out and install flow;
                end
                if decision == DISALLOW then
                    drop packet and install flow;
                end
                if decision == DEFAULT then
                    if inter vlan routing is enabled then
                        pop vlan, send packet out and install flow;
                    else
                        if vlan tag of packet is same destination ip's tag
                        then
                            pop vlan, send packet out and install flow;
                        end
                        drop the packet and install flow;
                    end
                end
            end
        end
    end
end
```

Algorithm 4: Routing packets coming from access ports of hosts

```
if input port = access port then
    figure out if the packet is intended for the host or for any other
    machine;
    if destination ip is NOT of host AND destination mac is of host
    then
        if mac corresponding to destination ip is not known then
            send ARP request for destination ip;
            wait for 100ms;
        end
        if mac corresponding to destination ip is still not known then
            drop the packet;
        else
            set the destination mac address to the mac corresponding
            to destination ip from the table;
            set the source mac to host's mac;
        end
        if output port is not known then
            send ARP request for destination ip address;
            wait for 100 ms;
        end
    else
        if output port is not known then
            send ARP request for destination ip address;
            wait for 100 ms;
        end
        if output port is known then
            if output port is TRUNK then
                push vlan tag, send packet out and install flow;
            end
            if output port is LOCAL then
                send packet out and install flow;
            end
            if output port is ACCESS then
                route according to inter vm policy;
                check next page for details;
            end
        else
            drop packet and install flow;
        end
    end
end
```

Algorithm 5: Routing according to inter vm policy when both input and output ports are access ports

```
if output port is ACCESS then
    get policy decision between source and target ip addresses;
    if decision == ALLOW then
        send packet out and install flow;
    end
    if decision == DISALLOW then
        drop packet and install flow;
    end
    if decision == DEFAULT then
        if inter vlan routing is enabled then
            send packet out and install flow;
        else
            if vlan tags of source and target ip are same then
                send packet out and install flow;
            else
                drop packet and install flow;
            end
        end
    end
end
```

Chapter 6

Results, Conclusion and Future Work

In this chapter we will discuss the results of the VLAN application in terms of bandwidth for all the four different cases. We have also shown the results for the load balancing application. Finally we will conclude the results of both the applications and suggest future work which can be done to further extend the application into a full fledged SDN based solution for Baadal.

6.1 Comparison of SDN based solution in terms of Bandwidth

Figure 6.1 shows the comparison of bandwidth between two virtual machines belonging to same host and same VLan. It shows that the bandwidth attained for different sizes of packets is almost same for both the traditional and SDN based solution

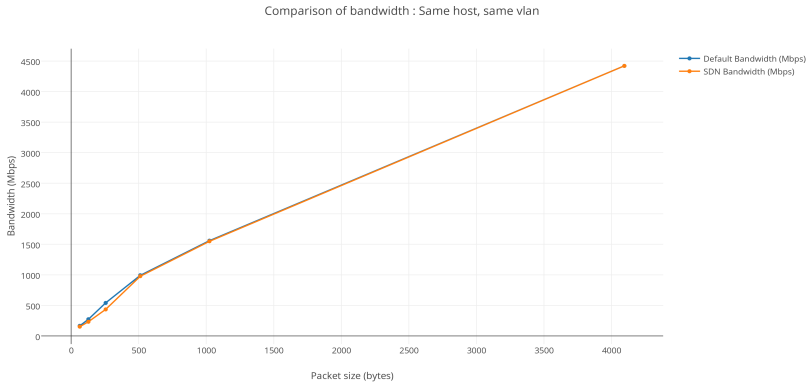

Figure 6.1: Bandwidth comparison of traditional and SDN metworking in Same host Same VLan

Figure 6.2 shows the bandwidth comparison for VMs belonging to different host and same vlan. We can see that the bandwidth for different packet sizes achieved is much higher in case of SDN even though uniformity is less in traditional networking.

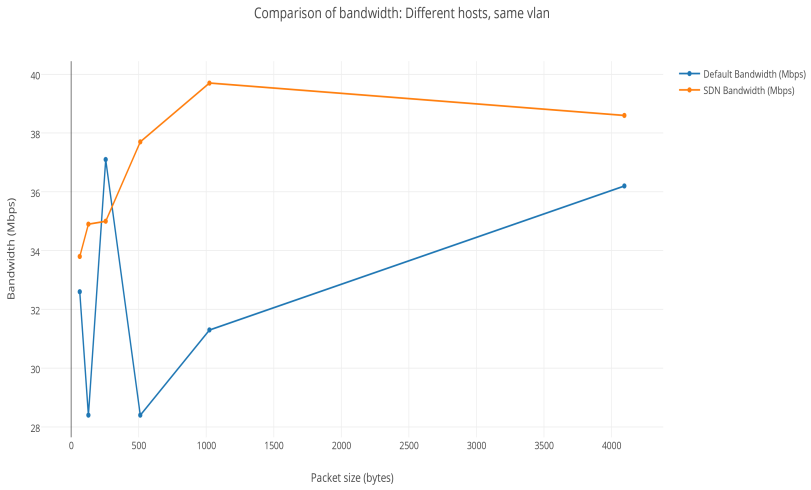

Figure 6.2: Bandwidth comparison of traditional and SDN metworking in Different host Same VLan

The next case of comparison is for the VMs belonging to same host and different vlan. The plot 6.3 shows a huge improvement in bandwidth achieved with SDN solution. It is observed that with increasing packet size the SDN bandwidth increases while the traditional networking bandwidth almost remains same.

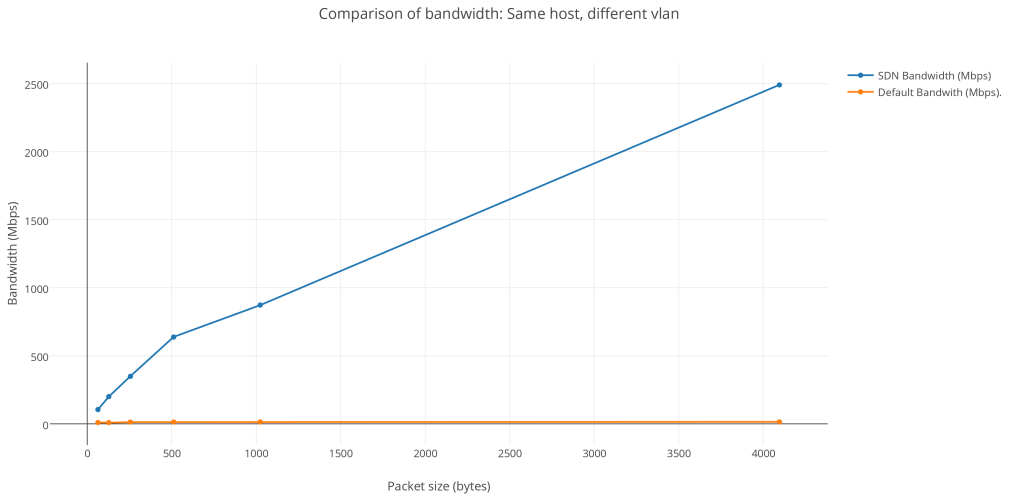

Figure 6.3: Bandwidth comparison of traditional and SDN networking in Same host Different VLan

The last case for the comparison is between VMs belonging to different hosts and different vlan. From the plot 6.4 it can be observed that the difference in the bandwidth between SDN based solution and traditional networking is higher for the smaller packet size unlike previous case.

6.2 Results for VMs consolidation application

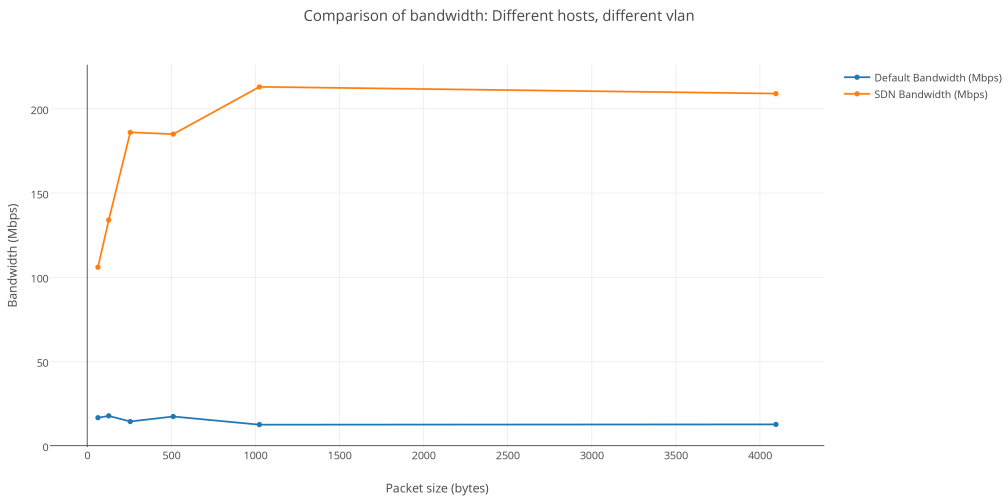

Figure 6.4: Bandwidth comparison of traditional and SDN metworking in Different host Different VLan

6.3 Load balancing application

Algorithm 6: Load balancing algorithm

```
while True do
    Read flows from the floodlight server;
    Compute the bandwidth between every pair of VMs;
    count = the number of times the bandwidth between any two VMs
    crosses a certain threshold in a certain time interval;
    if count exceeds pre decided number then
        find a new host for the VMs;
        start the migration;
    else
        set the number of times the threshold has been crossed to zero;
        continue;
    end
end
```

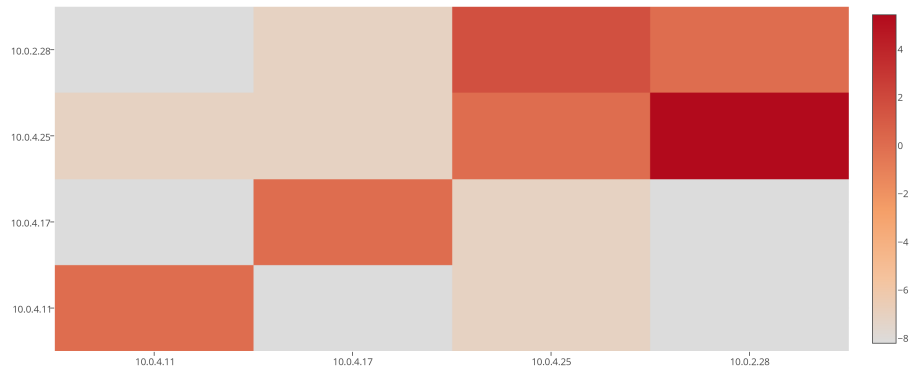

Figure 6.5: Heatmap showing network traffic between virtual machines

6.4 Conclusion

With the requisite understanding of the Network virtualisation and Software Defined Networking, we studied networking architecture of the baadal. We learnt about various virtual network management schemes. Based on these, we implemented some of the networking changes in the baadal. After implementing the changes, we found that there is significant improvement in the performance of the baadal in terms of bandwidth. Similarly load balancing application also brought improvemmnt in bytes transfer if two frequent talking VMs are consolidated on the same hosts. Hence we can say that SDN can be used to bring more applications in the baadal as it provides easier administratiion and control of the networking.

6.5 Future Work

We now propose the following tasks as future work so that the SDN application can be further extended

- Currently we do not handle broadcast and multicast packets in our routing scheme. These packets can be simply flooded to every output port (except the input port) if the incoming packet does not have a vlan

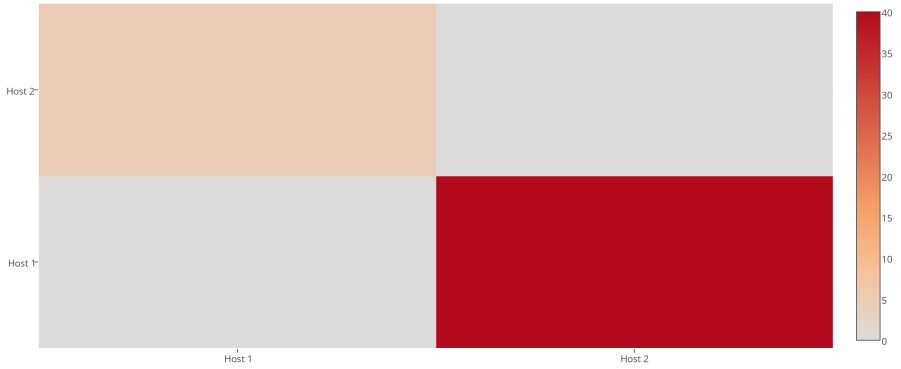

Figure 6.6: Heatmap showing network traffic between different hosts before VM migration

tag. If, on the other hand, the incoming broadcast/multicast packet has a vlan tag then it can be flooded only in its vlan. The output ports corresponding to the concerned vlan can be found out using tables *ipToTag*, *ipToMac* and *macToPort*.

- The connection to outside network is not established yet.It can be done by sending any packets which are destined to an outer network to the NAT machine as NAT is the only machine which is connected to the outside network. We also have to keep a track of all the fake bridges which are there for each vlan and must make sure that the packets going to outside network go through the concerned fake bridge.

- The load balancing module can be improved by experimenting with different vm migration algorithms.

- A front-end can be built which can be used to control the services that are implemented in this SDN application. Currently they are controller using the implemented REST APIs. The front-end will be able to make these REST API calls via a GUI.

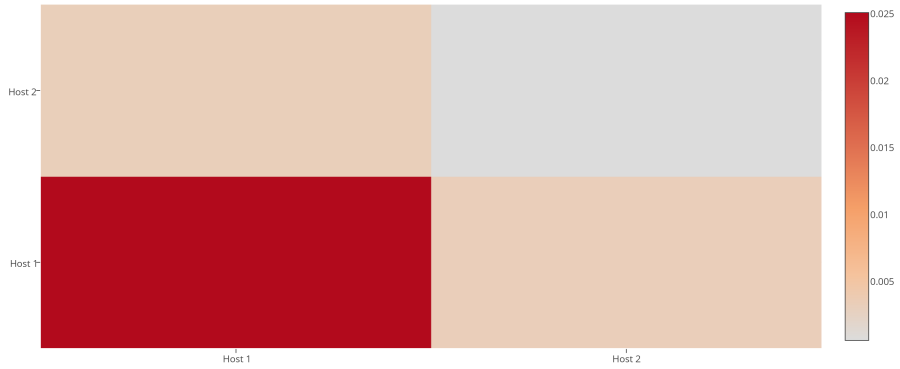

Figure 6.7: Heatmap showing network traffic between different hosts after VM migration

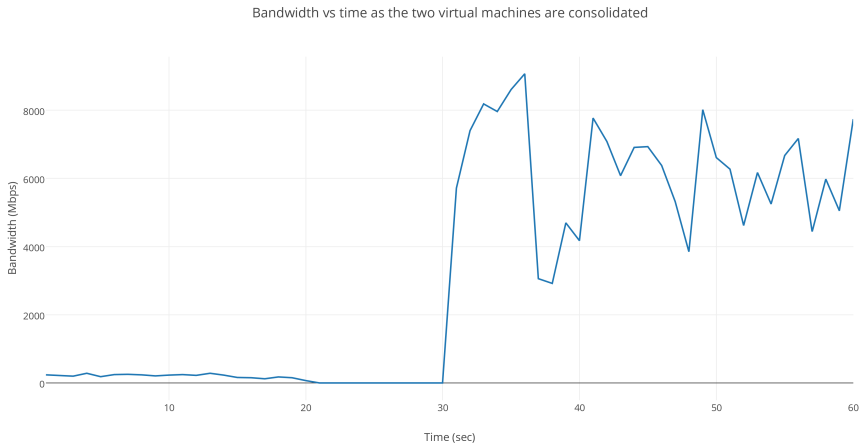

Figure 6.8: Bandwidth vs time as two VMs are being consolidated on a same host

Bibliography

[1] Baadal: the iitd computing cloud. `http://www.iitd.ac.in/content/baadal-iitd-computing-cloud`. Accessed: 2016-05-11.

[2] Baadal: the iitd computing cloud. `http://www.cc.iitd.ernet.in/CSC/index.php?option=com_content&view=article&id=123`. Accessed: 2016-05-11.

[3] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour. Sdn-based cloud computing networking. In *Transparent Optical Networks (ICTON), 2013 15th International Conference on*, pages 1–4. IEEE, 2013.

[4] Ofer Biran, Antonio Corradi, Mario Fanelli, Luca Foschini, Alexander Nus, Danny Raz, and Ezra Silvera. A stable network-aware vm placement for cloud systems. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 498–506. IEEE Computer Society, 2012.

[5] Daniel S Dias and Luis Henrique MK Costa. Online traffic-aware virtual machine placement in data center networks. In *Global Information Infrastructure and Networking Symposium (GIIS), 2012*, pages 1–8. IEEE, 2012.

[6] Xiongzi Ge, Yi Liu, David HC Du, Liang Zhang, Hongguang Guan, Jian Chen, Yuping Zhao, and Xinyu Hu. Openanfv: accelerating network function virtualization with a consolidated framework in openstack. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 353–354. ACM, 2014.

[7] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *Communications Magazine, IEEE*, 53(2):90–97, 2015.

[8] Joe Wenjie Jiang, Tian Lan, Sangtae Ha, Minghua Chen, and Mung Chiang. Joint vm placement and routing for data center traffic engineering. In *INFOCOM, 2012 Proceedings IEEE*, pages 2876–2880. IEEE, 2012.

[9] Kun Liu, Tianyu Wo, Lei Cui, Bin Shi, and Jie Xu. Fenet: An sdn-based scheme for virtual network management. In *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, pages 249–256. IEEE, 2014.

[10] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pages 459–473. USENIX Association, 2014.

[11] Rogerio V Nunes, Raphael L Pontes, and Dorgival Guedes. Virtualized network isolation using software defined networks. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 683–686. IEEE, 2013.