CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER'S THESIS

Title:	Alternative Network Layers in OpenStack
Student:	Pragya Sharma
Supervisor:	Ing. Tomáš Vondra
Study Programme:	Informatics
Study Branch:	Computer Systems and Networks
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2018/19

Instructions

The OpenStack cloud platform contains the Neutron component which controls virtual networking for virtual machines created by other components. It provides an API standard as well as its reference implementation using Linux components. There are also several competing implementations with different features and using different technologies.

Research the available implementations. With the open-source ones, focus not only on functionality but also on development status, user base, and deployment options.
Choose a few of them (e.g. 2 and the default one) and install them on your or provided hardware. An OpenStack deployment supporting the selected solutions will be required. During the installation, observe high availability features, if any.
Evaluate their features and benchmark their performance. Document the observed stability during the benchmark and prolonged operation.

References

Will be provided by the supervisor.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 10, 2017

Czech Technical University in Prague

Faculty of Information Technology

Department of computer systems

Master's thesis

Alternative Network Layers in OpenStack

Er. Pragya Sharma

Supervisor: Tomas Vondra

15th May 2018

Acknowledgements

First and foremost, I offer my sincerest gratitude to God for His affection, provision, and knowledge for my entire life. It is with incredible appreciation, that I acknowledge the help of my Supervisor Tomas Vondra, without his assistance, guidance, and direction this thesis would not be possible. I would also like to give my most grateful for my grandparents Mr. Shri Ram Sharma Acharya and Mrs. Bhagwati Devi, parents Mr. Brij Mohan Sharma and Mrs. Shashi Sharma, and siblings Dr. Nivedita Sharma and Pranav Sharma who supported me guided me and motivated me throughout my life.
THANKS

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 15th May 2018 ....

Citation of this thesis

Sharma, Pragya. *Alternative Network Layers in OpenStack.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Cloud computing zskv rostouc popularitu dky vy klovatelnosti, vt flexibilitu a snadnj dostupnost svch slueb. Cloud Computing umouje poskytovatelm slueb vytvet velk mnostv zdroj zkaznkm, aby mli uivatel k dispozici zdroje na podn. Na tento el poskytovatel slueb cloud computingu pouvaj virtualizaci, protoe to dv schopnost efektivn sdlet zdroje mezi uivateli. OpenStack, relativn nov open source cloud computing platforma, se zamuje na poskytovn st jako slubu (NaaS) vyuvajc virtualizan technologii. OpenStack slibuje rozshl klovateln cloudov infrastruktury. Bt nov, zstv prozkoumat, jak tyto schopnosti pin, a co pesn fungovn jeho vnitnch detail. Clem tohoto projektu je studium intern sov vkon OpenStack zaloenho na implementaci Neutron nebo jinch sovch plugins jako Calico. Parametry vkonu st, jako je propustnost, ztrta paket a paket zpodn bude vyhodnoceno v rmci provozu TCP a UDP pomoc benchmarkingu IPERF nstroj.

eten o modelu toku st potvrzuje, e VM na tomt sov komunikace ve vrstv 2 a komunikace v rznch stch ve vrstv 3 bez ohledu na umstn hostitele. Vsledky ze st vkonnostn experimenty ukzaly, e OpenStack zaruuje vkon s prakticky dnou sovou propustnost. Navc, vsledky ukazuj, e umstn stroj z hlediska hostitele a sov adresa ovlivuje vkon st, nicmn Calico je silnm konkurentem pro Neutron, pokud jde o vkon pedevm pro TCP..

Klíčová slova Cloud, cloud een, NaaS, veejn cloud, soukrom cloud, cloud cloud, neutron, Open VSwitch, virtualizace, Multi-Tenancy, SDN

Abstract

Cloud computing is gaining increasing popularity because of its higher scalability, more flexibility and ease of availability of its services. Cloud Computing enables service providers to build a large pool of resources to their customers so that users will have resources accessible on demand. To this end, cloud computing service providers use Virtualization, since it gives them the ability to effectively share resources among their users. OpenStack, a relatively new open source cloud computing platform, focuses on delivering network as a service (NaaS) using virtualization technology. OpenStack promises massively scalable cloud infrastructures. Being new, it remains to be investigated on how it delivers those abilities, and what the exact workings of its internal details are. The aim of this project is to study the internal network performance of OpenStack based while using Neutron implementation or other networking plugings such as Calico. Network performance parameters like throughput, packet loss and packet delay will be evaluated under TCP and UDP traffic using IPERF benchmarking tool.

The investigation about network flow pattern confirm that VMs on the same network communicate at layer 2 and those at different networks communicate at layer 3, irrespective of their host locations. The results from network performance experiments showed that OpenStack guarantees a performance with virtually no network bandwidth bottleneck. In addition, the results shows that the location of machines in terms of host and network address affect network performance, however, Calico is a strong competitor for Neutron in terms of performance mainly for TCP.

Keywords Cloud, Cloud Solutions, NaaS, Public Cloud, Private Cloud, Hybrid Cloud, Neutron, Open VSwitch, Virtualization, Multi-Tenancy, SDN

Contents

List of Figures

Introduction

The technology is like a never-ending ladder, whenever we think we are reaching to the top, something new pops-up. As soon as the IT industry was thinking that it has everything, there came an advanced technology named as "Cloud computing", which provided the best answers for all monetary and technical issues of small and big companies. Via which companies got satisfactory and desired solution regarding the issues such as the high initial investments for infrastructure, usage of full resource capacity, etc. As even in a middle-level company the set-up cost for infrastructure in terms of software and hardware components that need to be installed, configured and maintained on a regular basis is quite high, and in case if these components are proprietary then we should never forget the cost of licenses and updates to use them in future as well.

These requirements gave birth to the concept of Cloud computing. As the Internet is expanding widely and the main motivation is to connect each other, through the Internet. Companies are remotely accessing, storing and managing the computers/servers, sharing resources "On-demand", means, the provisioning of components-both hardware and software, as and when needed, Cloud computing takes resource-sharing to the next level.

This has contributed to companies that provide cloud computing service such as Amazon, Google, Microsoft, Rackspace and Justcloud to generate more revenue [1]. The main feature of cloud computing is On-demand resource allocation, through which users are charged based on usage time and amount of resources they utilize. For instance, a company or a user can be billed based on the duration of the resource usage. Furthermore, consumers get high-performance services as well as low-cost services or resources instead of building their own highly costly infrastructure [1].

If we explicitly talk about the OpenStack, this open source platform is supported by many IT giants including Cisco, HP, IBM, and Intel. As there is an outstanding and significant increase in demand and evolution of cloud environment, it is getting immensely complex. Albeit, OpenStack is an industrial venture in a prospect of a worldwide joint effort of developers and distributed computing technologists, creating the open standard distributed computing working framework for both open and private clouds. OpenStack was founded by NASA and Rackspace Hosting on 21 October 2010; approx 8 years ago, which quickly become a worldwide programming group of engineers working together on a standard and considerably adaptable open-source cloud framework. OpenStack fundamentally constitutes three programming activities, named as, OpenStack Compute, OpenStack Object Storage, and OpenStack Image Service.

OpenStack provides an arrangement concerning interrelated ventures conveying different segments for a cloud foundation arrangement and controls extensive pools of capacity, register and systems administration assets all through a data center, that all managed through a dashboard(Horizon) that gives executives control while enabling their clients to arrangement assets through a web interface.

In this thesis, the main focus is on the Neutron, which is responsible for providing network performance in the OpenStack. In order to comprehend the scalability and predictability of network performance of the OpenStack cloud computing, the network performance will be investigated using different projects and plugins.

Motivation and Objective

Cloud computing has become an inseparable part of the technology and innovation which can provide services and /or applications via the web. It can swiftly provide the information and the data from wherever the system is situated at any minute. Albeit, there might be conceivable interruptions or failures to the extreme disappointment of the system because of some specialized issues. Hence, it ought to be remembered that there is a high prerequisite for maintenance despite the pivotal benefits cloud computing can give [2].

Everyone who is spending money wants the best services no matter if the amount of money is large or small, its the same case in IT industry as well, it doesnt matter if the organization is big industry giants or small entrepreneur they all want quality of service for what they are paying off in terms of no latency, no packet loss, and fast speed. However, in order to provide best quality of service for the above-mentioned points we must make our focus more towards a good capacity of network layer (Layer3) or data link layer (Layer2) performance, which needs depth study of network performances in general.

Also, not to be forgotten that Virtualization is an essential part of the cloud computing, Wikipedia provides a very sophisticated definition of Virtualization "In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources." [3]. It is exploitation of a single hardware equipment in a constructive way, through this a single physical server, sharing same physical processor, hardware, I/O can host multiple virtual machines, there are so many companies are getting benefit from it, for example, Xen Virtualization is utilized by the most acclaimed cloud supplier Amazon EC2.

OpenStack usages this tremendous phenomenon of Virtualization, in OpenStack it has mainly three kinds of services such as computing services, networking services and storage services named as Nova, Neutron, and Cinder respectively. These services provide logical resources that can be made on-demand by cloud clients by means of HTTP commands, these logical resources could be virtual machines (VM), networks, routers, switches and firewalls, these resources established and implemented by the platform using servers, VMs, network and storage devices.

However, every organization still needs resource management, as logical resources are mapped to the underlying physical resources that host them, hence, we have to compromise with the computing process and connection performance.

As cloud computing and OpenStack turn out to be more unpredictable, resource management moves toward becoming to a greater extent of complexity. This is particularly valid for OpenStack as each service is autonomous which in turn further complicates united management. For bigger organizations and the expanded alleviation to distributed storage and computing, resource planning turns into a critical issue.

The main objective of this thesis is to find out the unsurpassed networking competence, so that the consumers can utilize OpenStack in its full extent according to their environment requirement, that knowledge is gained regarding the networking capability of Neutron by comparing it with other available Projects.

Problem Statement

Cloud computing is a hot concern during this era and researchers include excellent concepts related to this. To the current point, various cloud computing operating device comes to exist for cloud systems, variety of them are OpenStack, EC2, OpenNebula, CloudStack and so forth. Cloud computing affords many offerings, those offerings are IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS(Software as a Service). As the demand for cloud computing is growing in no time, its performance needs to be properly adequate to satisfy the need of its customers' environment, whether it's private or public cloud. One among the foremost important issue of cloud computing is to reap a more robust network performance as a result of a device without good Networking performance is kind of inconceivable to be considered a high overall performance cloud devices.

The OpenStack cloud platform carries the neutron component that controls digital networking for machines created via alternative additives. It presents an API additionally to its reference implementation using Linux elements. There are also various competitive implementations with special features and also the use of special technologies.

This Thesis will focus principally on the analysis of the offered implementation of Network nodes in OpenStack and will work intensively on Neutron comparison with OVN, Opendylight, Calico, Project Roamana and others to seek out out the Networking performance which can be achieved by completely different flavors of Network component of OpenStack.The problem statements of this studies are:-

1. To become aware of the network analysis of the various projects of the OpenStack platform.

2. To evaluate a practical comparison of overall network performance of neutron with alternative projects on OpenStack, conjointly to observe HA features, if provided in Project throughout the installation.

It's far believed that the observer can cause expect the conduct of network customers on OpenStack and customers can perceive what network overall performance capabilities they get the use of OpenStack cloud computing. So, as to try and do that we need to evaluate in their features and benchmark their performance. Document the ascertained stability throughout the benchmark and prolonged operation.

Chapter 1

State-of-the-art

Before going into the practicality, through this chapter I have tried to provide some background nitty-gritty of the main contents of this thesis. Albeit, the main concentration of this thesis is on OpenStack, however, as its born out of cloud computing, its necessary to cover all the aspects of cloud computing, however, I have tried to do it cover its all-classification in brief. Later, I focused more on the detailed description of OpenStack starting with its history and going all the way through its architectural overview with its components, means touching all the important parts related to this thesis.

1.1 Cloud Computing

Cloud Computing [1] is a modern computing paradigm that providing IT infrastructure and it is very essential requirement for the IT companies, where a distributed- computing resources are served by a network-based mechanism. As the usage of the cloud computing increased it resolved the biggest issue of the requirement of the gigantic storage with the ability of consolidation of resources for better management.

Cloud computing is a model for on-demand access to a shared pool of computing resources that can be allocated and released by the consumer. According to The NIST definition of Cloud Computing by PeterMell and Timothy Grance [4] a cloud has five essential characteristics.

On-demand self-service: *A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.*

Broad network access: *Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).*

Resource pooling: *The providers computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and vir-*

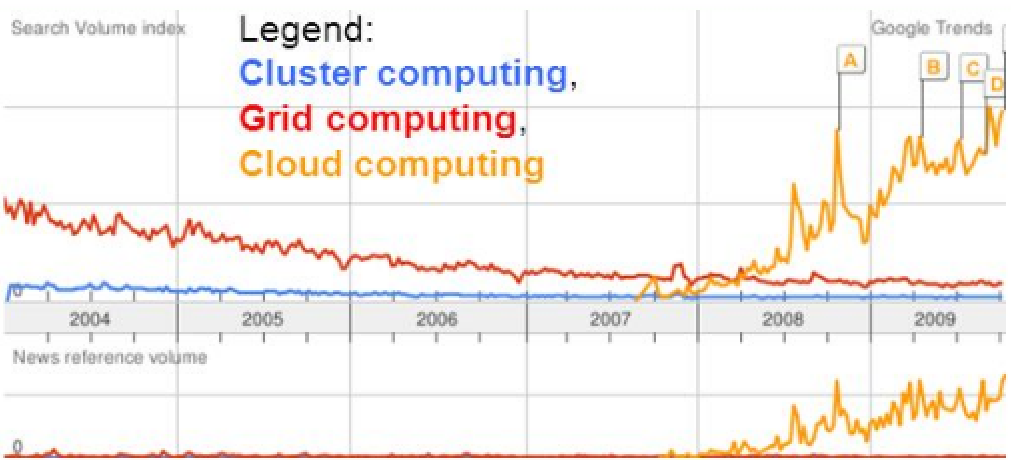


Figure 1.1: Demand of Cloud Computing

tual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

Rapid elasticity: *Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.*

Measured service: *Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.*

1.1.1 Cloud Computing History

The cloud computing first time came into lime light in early 1996, when the references to the phrase "cloud computing" appeared, where it was just an idea of computation may someday be organized as a public utility [1], until in year 2000 Amazon put his hands to it, however, Amazon started its data centers using small capacity. In the below chart the rise of cloud computing can easily be spotted even in back 2009 comparing to other computing methods.

In the below table the development of Open Source Cloud computing is shown.

Name	Year	Description	Deployment
Eucalyptus	Early 2008	AWS API compatible platform	AWS API-compatible platform
OpenNebula	Early 2008	RESERVOIR European Commission funded project	Private and hybrid clouds, and for the federation of clouds
CloudStack	In May 2010	Began at cloud.com	Public, private and hybrid cloud services
OpenStack	2010	By Rackspace and NASA	Public, private and hybrid cloud services

Table 1.1.1: Open Source Cloud Computing [1]

1.1.2 Cloud Computing Models

A consumer always looks for the comfort whenever he/she buys anything, this looks like a universal truth as while providing the cloud services makers considered the fact of consumers ease. Hence, considering this fact there are mainly 3 kinds of cloud computing models according to the consumer's requirements: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

The three models for cloud computing services in a service-oriented architecture (SOA). A brief overview follows:

SaaS

SaaS, short for Software-as-a-Service, is that the most elementary kind of cloud service, SaaS permits the users to utilize numerous applications from the cloud instead of using applications on their own PCs/Servers. The cloud service providers would sometimes give some variety of software development atmosphere to permit applications to be developed to be used within the cloud. The application programming interface (API) that the users use to access and act with the software permits the user to use the package while not having to fret regarding how or where the info is being keep or what proportion storage space is accessible because the cloud service supplier can manage this for them. Examples of SaaS are Google apps, Salesforce, Cisco WebEx, and so on.

PaaS

PaaS, short for Platform-as-a-Service, provides a development platform, PaaS

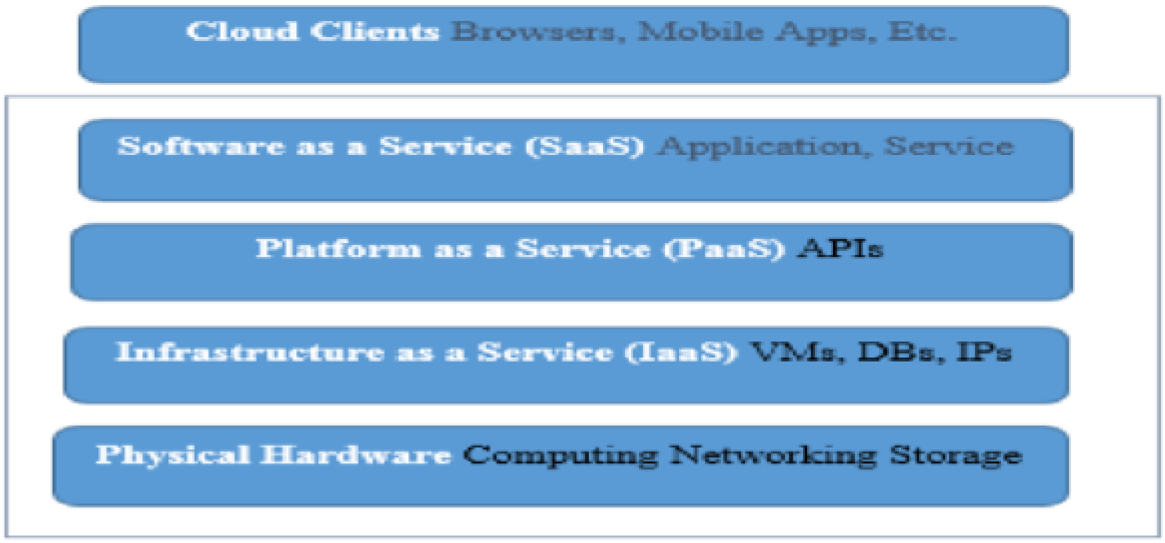

Cloud Clients Browsers, Mobile Apps, Etc.
Software as a Service (SaaS) Application, Service
Platform as a Service (PaaS) APIs
Infrastructure as a Service (IaaS) VMs, DBs, IPs
Physical Hardware Computing Networking Storage

Figure 1.2: Cloud Computing Models

operates at a lower level than the SaaS. it's accountable for the management of the storage space, bandwidth allocation and computing resources offered for the applications. It retrieves the resources required to run the software and dynamically scales up these resources once additional is required. This service holds a key attribute of the cloud mentioned as self-provisioning of resources. Examples of PaaS embody Windows Azure, Heroku, Google App Engine, and so on.

IaaS

IaaS, short for Infrastructure-as-a-Service, Infrastructure as a Service could be a single tenant cloud layer wherever the Cloud computing vendors dedicated resources solely shared with narrowed consumers at a pay-per-use fee. This greatly minimizes the necessity for large initial investment in computing hardware such as servers, networking devices, and process power. They conjointly permit variable degrees of economic and purposeful flexibility not found in internal data centers or with collocation services, as a result of computing resources may be added or removed rather more quickly and cost-effectively than in an enclosed data center or with a collocation service. IaaS and alternative associated services have enabled start-ups and alternative businesses specialize in their core competencies without concern regarding the provisioning and management of infrastructure. IaaS utterly abstracted the hardware at a lower place and allowed users to consume infrastructure as a service, while not bothering something regarding the underlying complexities. The cloud encompasses a compelling price proposition in terms of price, however out of the box, IaaS solely provides basic security (perimeter firewall, load balancing, etc.) and for applications going in the cloud want higher levels of security provided at the host. Examples of IaaS embody Amazon EC2, Google Compute Engine, and so on, OpenStack also comes in IaaS.

1.2 Overview of OpenStack

OpenStack is a briefing of programming apparatuses for building further as managing distributed computing platforms for public and private clouds. OpenStack cloud operating framework turns all arrangements of hypervisors within data center or over many data centers into pools of resources, and these pools of resources are often consumed and additionally oversaw from one element, named as dashboard. Consumers can simply visit its dashboard to play out with its utilities in a very straightforward and fast means, and one can build virtual machines (VMs), tack together networks and manage volumes. OpenStack community works along around every 6 months, for time-based release cycle with visit improvement points of reference. OpenStack is associate open source platform for cloud computing designed to work with python programming. It also can be outlined as computing, networking and storage, that has a pool of services like CPU, memory and storage. As mentioned previous it is a venture of RACKSPACE and National Aeronautics and Space Administration and since 2010, variety of releases are out. The subsequent table shows the progress of OpenStack and the completely different options included in every release.

Release name	Release date	Included Component code names
Austin	21-Oct-10	Nova, Swift
Bexar	3-Feb-11	Nova, Glance, Swift
Cactus	15-Apr-11	Nova, Glance, Swift
Diablo	22-Sep-11	Nova, Glance, Swift
Essex	5-Apr-12	Nova, Glance, Swift, Horizon, Keystone
Folsom	27-Sep-12	Nova, Glance, Swift, Horizon, Keystone, Quantum, Cinder
Grizzly	4-Apr-13	Nova, Glance, Swift, Horizon, Keystone, Quantum, Cinder
Havana	17-Oct-13	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer
Icehouse	17-Apr-14	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove
Juno	16-Oct-14	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara
Kilo	30-Apr-15	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic
Liberty	16-Oct-15	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight
Mitaka	7-Apr-16	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum
Newton	6-Oct-16	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum, aodh, cloudkitty, congress, freezer, mistral, monasca-api, monasca-log-api, murano, panko, senlin, solum, tacker, vitrage, Watcher
Ocata	22-Feb-17	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum, aodh, cloudkitty, congress, freezer, mistral, monasca-api, monasca-log-api, murano, panko, senlin, solum, tacker, vitrage, Watcher
Pike	30-Aug-17	Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum, aodh, cloudkitty, congress, freezer, mistral, monasca-api, monasca-log-api, murano, panko, senlin, solum, tacker, vitrage, Watcher

Table 1.2: OpenStack Releases [2]

1.2.1 OpenStack Architecture

OpenStack has been developing from time to time and it is been rising its design by separating its components/nodes consistent with their use. Thus, the design of OpenStack is in an exceedingly distributed fashion. The OpenStack project consists of many reticulated sub-projects that facilitate to manage totally different aspects of hardware resources together with computing, storage, networking and alternative connected services, every of that offers its own set of APIs to facilitate the combination of the full software stack.

OpenStack provides cloud managers with a web-based dashboard as a strong and versatile Application Programmable Interface (API) to regulate a collection of physical hosting servers with totally different varieties of Hypervisors and to manage the specified storage facilities and virtual network infrastructures. The OpenStack dashboard conjointly permits instantiating computing and networking resources at intervals the data center infrastructure with a high level of transparency. Figure 2.1 illustrates the OpenStack conceptual design with interactions among its software elements [5].

As an IaaS-focused cloud platform, OpenStack has VMs at its center, provisioned by the Nova module. VMs units are enclosed by alternative services together with network connectivity handled by Neutron; OS images kept by Glance; storage services provided by Swift and Cinder. Keystone is answerable for the authentication of the entire OpenStack system whereas, at a high level, Horizon provides a web-based management interface to any or all of the services. Below is the description of its components and services.

1.2.1.1 Compute Service

OpenStack compute (Nova) is intended to manage pools of computing resources and provide access to those pools via either graphical program tools (dashboard) or command-line tools or the wealthy native API sets, in short it is an instance management component. Although, dashboard is accessible only by the authenticated user. Nova works with most well-liked virtualization technologies resembling KVM (default) [7], VMware [6], Xen [8] or Hyper-V [9] as well as Linux container technologies like LXC [10]. Nova may be thought of the most important part in associate degrees of IaaS system, within which cloud users have access to VMs hosted by nodes running nova service. Inside OpenStack platform, Compute nodes use for hosting and managing cloud computing systems.

1.2.1.2 Networking Service

The Networking service, Neutron, previously known as Quantum, there were 2 major issues with Quantum and as a resolution of which Neutron born:

1. As the Networking was sub-component of Nova, limited technology "baked in" to design

2. No tenant control of network topology and addressing, no way to insert advanced network services, such as firewall.

Neutron provides an association with the abstraction of Virtual Network Infrastructure (e.g.: network, subnets, ports, routers, etc.) and services (e.g.: firewall, load balancer, virtual private network, etc.) inside the OpenStack based cluster, fundamentally it is a network management component.

Neutron primarily provides VMs (created and managed by Nova) with networking service, that before the existence of neutron were handled by Nova network. Being capable of providing solely basic networking service, development and deployment of nova-network became step by step lessened in later releases, with a long-run they arranged to remove this module from OpenStack code base. Neutron has accentuated because of the default networking module for OpenStack with a lot of versatile and full-edged abstractions of network infrastructure and services. Neutron is an API frontend that manages the Software Defined Network (SDN).

1.2.1.3 Storage service

Besides conventional storage technology (that comes in conjunction with computing resources managed by Nova), OpenStack additionally supports 2 further varieties of storage, specifically Object Storage and Block Storage.

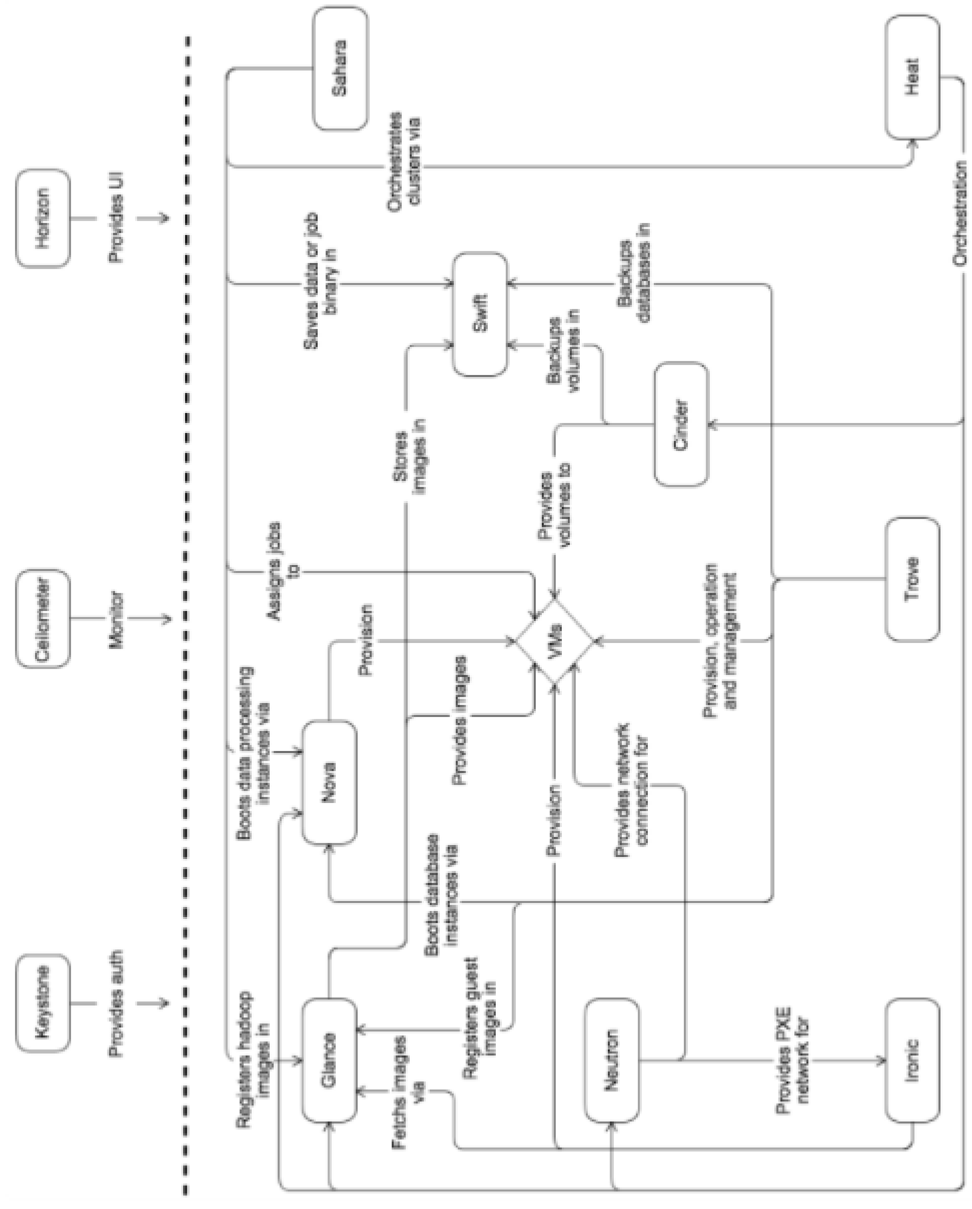


Figure 1.3: OpenStack conceptual architecture

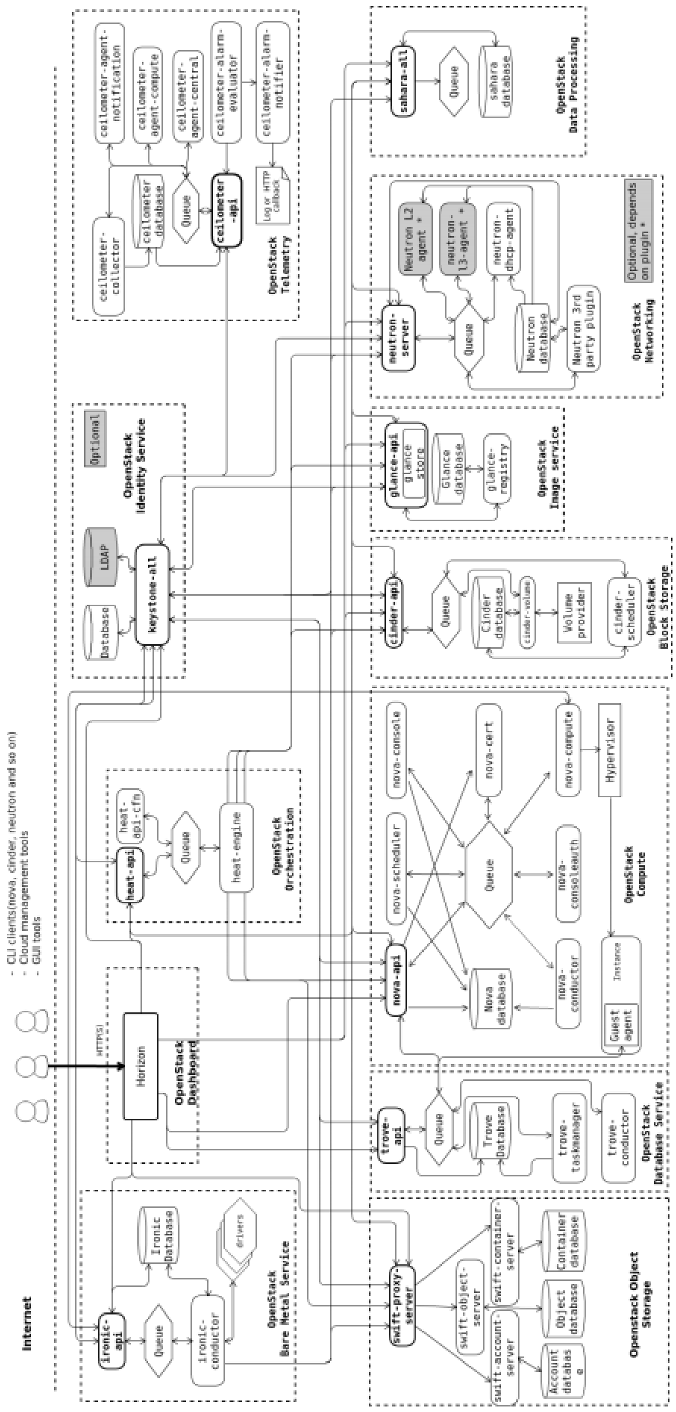


Figure 1.4: OpenStack logical architecture

Object storage (Swift) is a scalable redundant storage management component within which objects and files to keep, replicated, and distributed throughout multiple servers inside the cluster. However, files are stored without metadata, they are simply containers and files. There are 2 layers in swift for deployment: The Proxy (API Layer) and The Storage engine (Software-based storage distribution/replication). GlusterFS and Ceph are the 2 widely used software-based storage. As an example, Amazon runs its storage service S3 via its public cloud platform at massive scale.

Block storage (cinder) manages (creates/attaches/detaches) virtualized block storage pools and supply OpenStack users with access to those pools. It is to be worth notice that once the volume is created and attached to the instance, it can be used on the instance as any other block device would be used. Block storage is absolutely integrated into compute (Nova) and dashboard (Horizon) services via APIs, enabling users to consume these storage resources even with none data of the technology of the underlying storage devices. There is one more utility of Cinder it can easily handle snapshots, also these snapshots also used by instances as a boot source. Logical Volume Manager (LVM) is by default configured, Cinder also uses GlusterFS and for software-based storage.

1.2.1.4 Dashboard Service

OpenStack Dashboard (Horizon) permits users to access and manage VMs, Virtual Networks and alternative OpenStack resources via a web-based graphical users interface (written in Python). The Dashboard service presenting altogether totally different virtual machines, the virtual networks they belong to and virtual routers that connect these networks. Besides dashboard, users (mainly developers) might also move and perform administrative tasks by practice sets of native OpenStack APIs or the EC2 compatibility APIs.

1.2.1.5 Shared Services

OpenStack works with several alternative services these are utilized by the above core projects, making it easier to implement and operate on the cloud. These services together with identity, image management and a web interface - integrate the OpenStack elements with one another, furthermore as external systems to produce a unified expertise for users as they interact with totally different cloud resources.
Identity service
OpenStack Identity service (Keystone) provides a central authentication and authorization mechanism for various OpenStack services. Keystone put together provides a catalog of endpoints for all OpenStack services. the primary issue that must happen whereas connecting OpenStack deployment is authen-

tication, Keystone manages tenants, users, and roles and be a catalog of the services and endpoints for all the elements within the running cluster.

Tenant is that the cluster of objects in OpenStack, everything should reside within a tenant, let's say, users, instances, and networks. However, Project and Tenant are the same, during use of web interface its known as Project and on command line named as Tenant. From the mentioned line the difference between User, Tenant and roles would be clear Users grant a role within tenant [11].

Image service

The OpenStack Image service (Glance) allows creating, storing and retrieving disk images for VMs, that is used by compute service throughout the provisioning of VM instances. In Cloud computing there is a requirement of a registry of pre-installed disk images to boot, Glance is that registry inside OpenStack deployment [11].

Telemetry service

The OpenStack telemetry service (Ceilometer) collects resource measurement and track the services used by OpenStack users and supply billing consequently. Initially, it was designed for the billing purposes only later on after realization of its utilities it turned into a general-purpose telemetry system. Through Ceilometer one can get a statistical view of the resource measurement [11].

Orchestration service

OpenStack Orchestration (Heat) provides the facility to stipulate and automatize the deployment of infrastructure, services and applications using flexible templates. Orchestration makes it possible to launch multiple instances in one go, that is meant to work along. This happens by managing a file named as Template, that contains all info concerning what ought to be launched. It will scale up or scale down the OpenStack cluster [11].

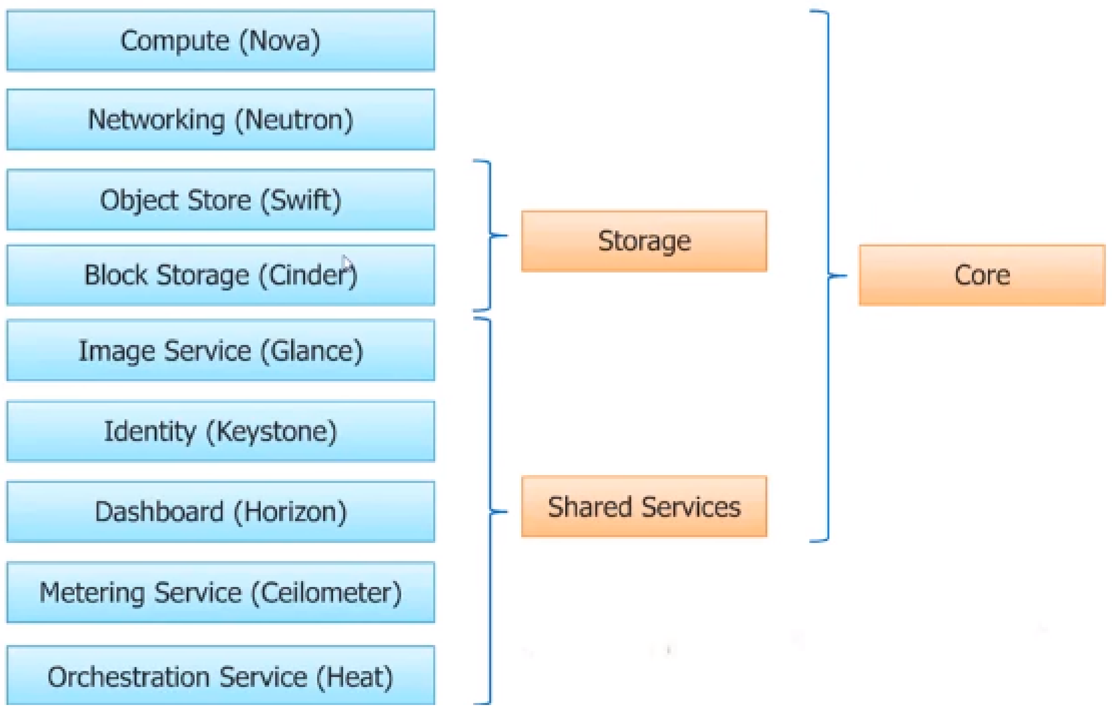


Figure 1.5: OpenStack Services

1.2.2 Extra Services

There are additional services that OpenStack uses and are needed to possess a operating cloud. These don't seem to be maintained by OpenStack and a number of other choices for every service are out there. Services used are[30]:

- **Network Time Protocol (NTP)** networking protocol for clock synchronization. All OpenStack nodes ought to have a similar system time and NTP servers are used for that. it's extremely suggested that controller nodes reference some different trusted NTP servers and every one other nodes have controller as their NTP server. One advised implementation that OpenStack uses is Chrony.

- **SQL database** Most OpenStack services store their data on SQL databases. It ought to be placed within the controller node. OpenStack supports several databases, most notably MySQL and its fork project MariaDB and conjointly different popular ones like PostgreSQL.

- **Messaging queue service (MQ)** it's utilized in distributed systems to send and receive messages. It uses a queue as a data structure so as to temporary store messages once destination service isn't responding or is presently busy. OpenStack makes use of messaging queues to coordinate data about status and operations among services. usually runs on controller node and plenty of MQ services are supported together with RabbitMQ, Qpid and ZeroMQ. Most used and supported is the RabbitMQ.

- **Memcached** distributed memory caching. OpenStack uses Memcached to cache tokens for authentication. As this service is used in authentication, only OpenStack servers ought to have access to that. this might be achieved with rewalls, authentication and encryption. the most practicality of Memcached is to create central authentication quicker.

1.2.3 Message exchange and Integration

OpenStack is an absolutely distributed system, which consists of multiple smaller projects, or modules. Every module is intended with the "Share Nothing Architecture" principle in mind and is functionally autonomous from the others. A module like Nova or Neutron is included of multiple elements that together create its functionalities. like the other distributed systems, the practicality of OpenStack as an entire depends heavily on however its inner services are integrated, that successively depends on the aptitude of its modules and elements to communicate between them. There are three main mechanisms that enable the communication and service integration of OpenStack: RESTful API, Remote Procedure call, and RabbitMQ.

1.2.3.1 RESTful API

Inter-process communication between clients is accomplished through Remote Procedure call (RPC) by which implementation of subroutines during a remote location get activated. OpenStack modules like Nova (nova-compute, nova-api, nova-scheduler), neutron (neutron-server, neutron-openvswith-agent) or cinder (cinder-scheduler, cinder-volume) build significant use of RPC for its intra-module communication, to the extent that just about everything happening in these modules is triggered by RPC calls. for instance, when Neutron's neutron-server receives a (RESTful) request to form a brand-new network, it asks the accessible plug-in (e.g ml2plugin) to successively send an RPC call to the corresponding agent (e.g. neutron-openvswith-agent) [22].

1.2.3.2 Remote Procedure Call

Through Remote Procedure call (RPC) clients are allowed for inter-process communication to trigger the execution of subroutines during a remote location. OpenStack modules like Nova (nova-compute, nova-api, nova-scheduler), neutron (neutron-server, neutron-openvswith-agent) or cinder (cinder-scheduler, cinder-volume) build significant use of RPC for its intra-module communication, to the extent that just about everything happening in these modules is triggered by RPC calls. for instance, when Neutron's neutron-server receives a (RESTful) request to form a brand-new network, it asks the accessible plug-in (e.g ml2plugin) to successively send an RPC call to the corresponding agent (e.g. neutron-openvswith-agent) [22].

1.2.3.3 RabbitMQ

RPC calls depends upon a channel or a messaging mechanism through that they're delivered to the intense processes (i.e. consumers). RPC requests are packaged into messages that are sent to a message broker that then forwards them to the customers. This is wherever a messaging broker like RabbitMQ fits into the image. RabbitMQ [22] is an open-source implementation of the Advanced Message Queue Protocol (AMQP) standard. AMQP is intended to facilitate the brokering of messages between completely different processes, applications of an equivalent system, or even between systems that communicate by message passing. In OpenStack platform, AMQP is used to determine an effective internal communication mechanism between elements of a similar OpenStack module, for instance, Nova, neutron or cinder.

1.2.4 KVM

Kernel-based Virtual Machine, or KVM, is a full virtualisation resolution for Linux and has been shipped with Linux kernel since kernel version 2.6.20. KVM enabled by running QEMU-based hardware emulation with KVM-acceleration mode enabled. KVM is a special in operation mode of QEMU that utilises processors hardware-assisted virtualisation capability (Hardware Virtual Machine, or HVM) to perform hardware virtualisation via its processor-specific kernel modules. KVM is among many hypervisor platforms compatible with OpenStack. By itself, KVM doesn't perform any emulation. Instead, it exposes the /dev/kvm interface, that a userspace host will then use to[28]:

- Set up the guest VM's address space. The host should conjointly provide a firmware image (usually a custom BIOS once emulating PCs) that the guest will use to bootstrap into its main OS.
- Feed the guest simulated I/O.
- Map the guest's visual display unit back onto the host.

On Linux, QEMU versions 0.10.1 and later is one such userspace host. QEMU uses KVM once accessible to virtualize guests at near-native speeds, however otherwise falls back to software-only emulation.

1.3 OpenStack Networking: Neutron

As this thesis is exclusively targeted at Networking of OpenStack, thus this section can cover thoroughly the OpenStack Networking module - neutron and its relevant software packages that altogether construct the Neutron-based network infrastructure.

1.3.0.1 The Neutron Project

The OpenStack neutron project, having its premiere in Havana release (October 2013), replaced nova-network to provide OpenStack with a full-featured abstraction of the Virtual Network Infrastructure still as basic and advanced network services. Thanks to Neutron, cloud users have access to essential networking infrastructure and resources like network, subnet and router objects. the elements simulate functionalities of real- world corresponding physical components: network consists of subnets connected to routers, that route traffic between totally different subnets and networks. Besides the availability of such basic network services as NAT, DHCP or routing, neutron conjointly allows users to create advanced virtual network topologies as well as services like firewalls (Firewall- as-a-Service, or FWaaS), load balancers (LoadBalancer-as-a-Service, or LBaaS), and virtual private networks (VPN-as-a-Service, or VPNaaS).

Neutron has a component on the controller node called the neutron server which accepts and routes API requests to the suitable OpenStack Networking plug-in for action, along with a bunch of OpenStack Networking agents and plugins that communicate with each other using a messaging queue by plugging/unplugging ports, creating networks or subnets, and providing IP addressing as per requirement. Depending on the type of deployment, you can choose the different agents that you want to use. Some plugins that are available today with Neutron are the following [13]:

- Open vSwitch Plugin
- Cisco UCS/Nexus Plugin
- Cisco Nexus1000v Plugin
- Linux Bridge Plugin
- Modular Layer 2 Plugin
- Nicira Network Virtualization Platform (NVP) Plugin
- Ryu OpenFlow Controller Plugin
- NEC OpenFlow Plugin
- Big Switch Controller Plugin

- Cloudbase Hyper-V Plugin
- MidoNet Plugin
- Brocade Neutron Plugin Brocade Neutron Plugin
- PLUMgrid Plugin
- Mellanox Neutron Plugin Mellanox Neutron Plugin
- Embrane Neutron Plugin
- IBM SDN-VE Plugin
- CPLANE NETWORKS CPLANE NETWORKS
- Nuage Networks Plugin
- OpenContrail OpenContrail Plugin
- Lenovo Networking Lenovo Networking Plugin
- Avaya Neutron Plugin Avaya Neutron Plugin

1.3.1 Architecture Overview

OpenStack Networking is a standalone service that usually deploys many processes across the variety of nodes and works closely with other components such as Identity, Compute, Storage, etc. The most important process of the OpenStack Networking service is neutron-server, this is central server Python daemon that exposes the OpenStack Networking API and passes tenant requests to a collection of plug-ins for receiving, dispatching API requests.

With the help of these APIs Neutron client is able to build flexible policies and network topologies. The Neutron server administers several agents responsible for host and network configuration. Communication between Neutron server and these agents relies either on RPC (over RabbitMQ) or through the standard Networking API. This section provides an overview on the architecture and main components of Neutron, as illustrated in below Figure.

1.3.1.1 Neutron Server

The neutron server daemon (neutron-server) runs on the network node, it starts up and reads the configuration files then hundreds all configured plug-ins and extensions. It implements and provides network models and IP addressing for every port consequently. It additionally exposes APIs to neutron clients (via Dashboard, command line interface or API calls), and forwards requests from the clients to configured plug-ins. These consumer requests are placed

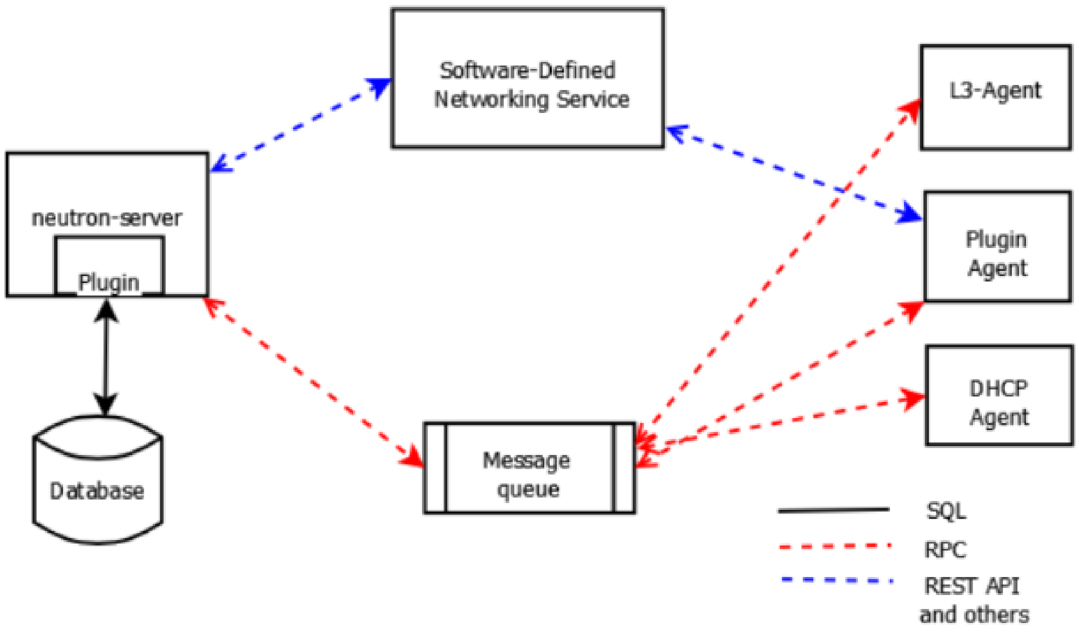


Figure 1.6: Neutron Architecture Overview

into a message queue (using the RabbitMQ messaging system and AMQP (Advanced Message Queuing Protocol)) and sent to corresponding agents (L2, L3, DHCP or alternative agents for advanced services).

1.3.1.2 Plug-in architecture

Neutron's ability to integrate with completely different underlying infrastructure and alternative networking services is enforced by a range of plug-ins. In alternative words, whereas neutron server provides its users with sets of capable APIs to manage and customize networks, it's the plug-ins that do the particular configuration tasks and enable neutron to support fast-changing network technologies from numerous vendors additionally on with efficiency deploy the Software-Defined Networking paradigm. the most task of the plug-ins is to manage agents, these agents give layer 2/3 connectivity to instances, it handles physical-virtual network transitions and additionally handles metadata. As such, the plug-in architecture brings a robust and versatile method of customizing a network's capabilities. there's just one neutron plug-in running at a time named as Module Layer 2(ML2), through this framework OpenStack Networking will at the same time use numerous forms of layer 2 networking technologies [14].

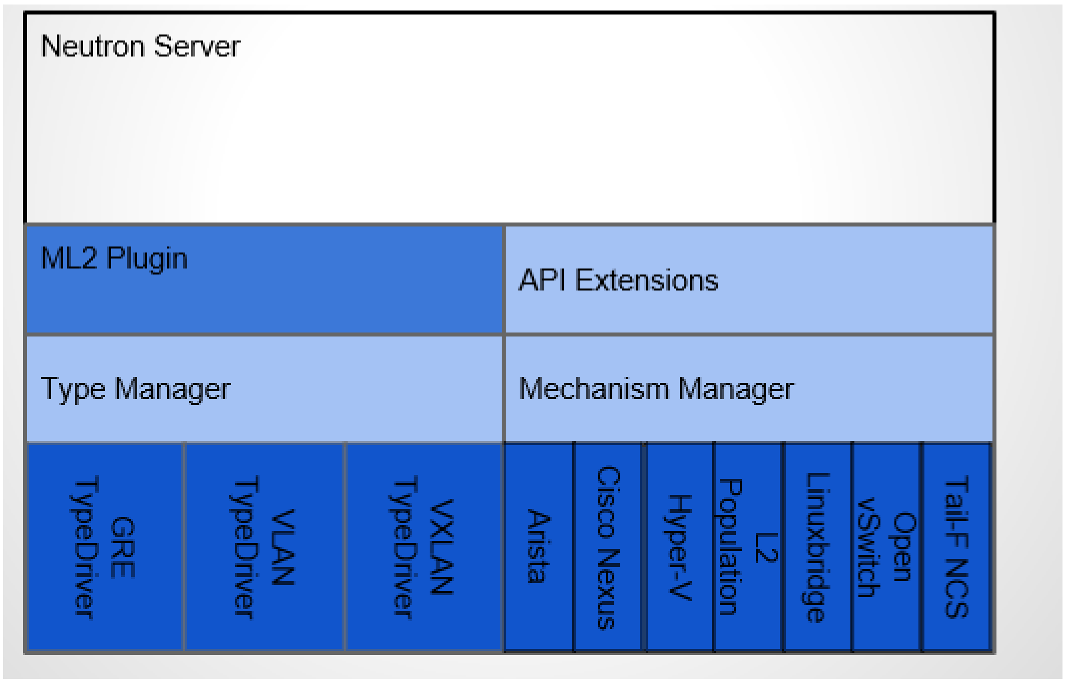


Figure 1.7: Plug-in architecture

Modular Layer 2

Modular Layer 2 (ML2) plug-in provides a framework to at the same time manage a range of Layer-2 technologies, every with a private mechanism driver. ML2 helps to handle the problem of implementation redundancy within which completely different switch technologies and vendors (e.g. OpenvSwitch, Linux bridge or Cisco) herald their own monolithic plug-ins and associated agents whereas still providing similar options and obtaining obstructed into the same environment (Neutron stack). This leads to duplication of the database, code base, and so on, besides development and maintenance efforts.

ML2 is meant to eliminate these duplication problems and to simplify the scalability potential (e.g. development of new plug-in/agent for new switch vendor). Besides, with ML2, there's still just one single plug-in allowed however multiple switch technologies may be run at the same time thanks to ML2's mechanism driver. The ML2 plug-in basically consists of kind Manager and Mechanism Manager[15].

Type manager: driver manages network state of a selected type and additionally performs supplier and tenant network validation. OpenStacks latest Pike version also supported type drivers include: local, flat, vlan, gre and vxlan.

Mechanism manager: The mechanism manager manages drivers for various underlying technologies (from completely different vendors) want to manipulate the underlying infrastructure and makes certain that they're applied in accordance with accessible type drivers.

1.3.1.3 Message queue

Similar to different OpenStack modules, neutron uses RabbitMQ as a messaging broker for communication between its internal elements by exchanging Remote Procedure call (RPC) over RabbitMQ message queueing mechanism using Advanced Message Queuing Protocol (AMQP) frameworks, which offer message queues for peer-to-peer communication. Queue implementations are generally deployed as a centralized or decentralized pool of queue servers the communication between neutron elements together with neutron-server, the OVS agent and also the OVS.

1.3.1.4 L2 Agent

L2 agents run on hypervisors (Compute nodes) and perpetually communicate with neutron server using RPC. Its main responsibility is to wire new devices (TAP interfaces created by Nova) and to set up the software bridges on the compute nodes. an L2 agent is answerable of monitoring its hosting hypervisor and informing the neutron-server of events occurring with the new or removed devices. In OpenStack, L2 property is provided using numerous mechanism drivers. L2 agents got to be present altogether compute/network nodes to make sure that L3 services are approachable by tenant VMs and subnets[17]. There are typically 2 bridges:

br-int (Integration bridge): Its the bridge that takes care of tagging and untagging the traffic that coming back in or out of the VMs. To tag the traffic, it uses local VLAN id and assigns it to the network.

br-tun (Tunneling bridge): It takes care of translating the labeled traffic. It interprets the VLAN id into segmentation and using it then for tunneling. If for instance, to utilize GRE tunnels, the segmentation id is used to specify the tunnel id. The L2 agent is additionally answerable for applying security cluster rules (firewall rules) that enforced in neutron by using IPTABLES and IP sets.

Open vSwitch Agent
As mentioned earlier, neutron needs plug-in agents (for example neutron-openvswitch-agent) to be present altogether hypervisor and networking nodes to provide local OpenvSwitch configuration. OVS agent provides layer-2 connectivity between instances and therefore the physical network infrastructure using VLAN (802.1q) tagging. It supports one untagged (flat) network and up to 4095 tagged (VLAN) networks. the particular amount of VLAN networks depends on the physical network infrastructure. After receiving a request from neutron-server OVS agent acts accordingly to configure OVS. This prin-

cipally involves putting in the integration bridge (br-int), to that all internal network services and tenant VMs are connected. The neutron-openvswitch-agent significantly depends on an OVS-specific API (ovs lib) to configure OVS and manipulate ow entries via 2 utilities ovs-vsctl and ovs-ofctl, respectively. Despite being an OpenFlow-compatible switch, OVS operates inside neutron networking as a regular L2 switch with each traditional and ow modes [16].

Linux operative agent
The Linux bridge agent configures Linux bridges to comprehend L2 networks for OpenStack resources. Configuration for the Linux bridge agent is usually done in the linuxbridge_agent.ini configuration file. ensure that on agent begin you pass this configuration file as an argument [14].

SRIOV Nic Switch agent
The sriov nic switch agent configures PCI virtual functions to understand L2 networks for OpenStack instances. Network attachments for alternative resources like routers, DHCP, then on aren't supported. Configuration for the SRIOV nic switch agent is usually done in the sriov_agent.ini configuration file. ensure that on agent begin you pass this configuration file as an argument [14].

MacVTap agent
The MacVTap agent uses kernel MacVTap devices for realizing L2 networks for OpenStack instances. Network attachments for alternative resources like routers, DHCP, then on aren't supported. Configuration for the MacVTap agent is usually done in the macvtap_agent.ini configuration file. confirm that on agent begin you pass this configuration file as an argument [14].

1.3.1.5 L3 Agent

While the L3 agent shares plenty of an equivalent architectural aspect as the L2 agent, its utterly different. With the help of L2 agent one can connect to the networks. Albeit, the L3 agent gets the routers connected to every different one. It moves data from one network to a different and from your network to the external world. Figure 2.7 captures a summary of L3 services together with virtual elements configured by L2 agents. The L3 agent creates totally different internal ports with prefix tap for DNS service, qr- for virtual router or prefix qg- for gateway to the External (public) network [17]. As its name suggests, the neutron L3 agent (neutron-l3-agent) configures its hosting a node with totally different Layer-3 networking services together with routing, NAT, and Floating IP. Traditionally, such L3 services run on Network node and rely on the L2 agent to supply layer-2 connectivity to VM instances running on compute nodes. neutron L3 agents use the Linux IP stack and iptables [18] to perform L3 forwarding and NAT. so as to support multiple routers with probably overlapping IP addresses, neutron-l3-agent defaults to using Linux network namespaces to provide isolated forwarding contexts.

Routing
A network namespace is like container or VM for the network device, in other

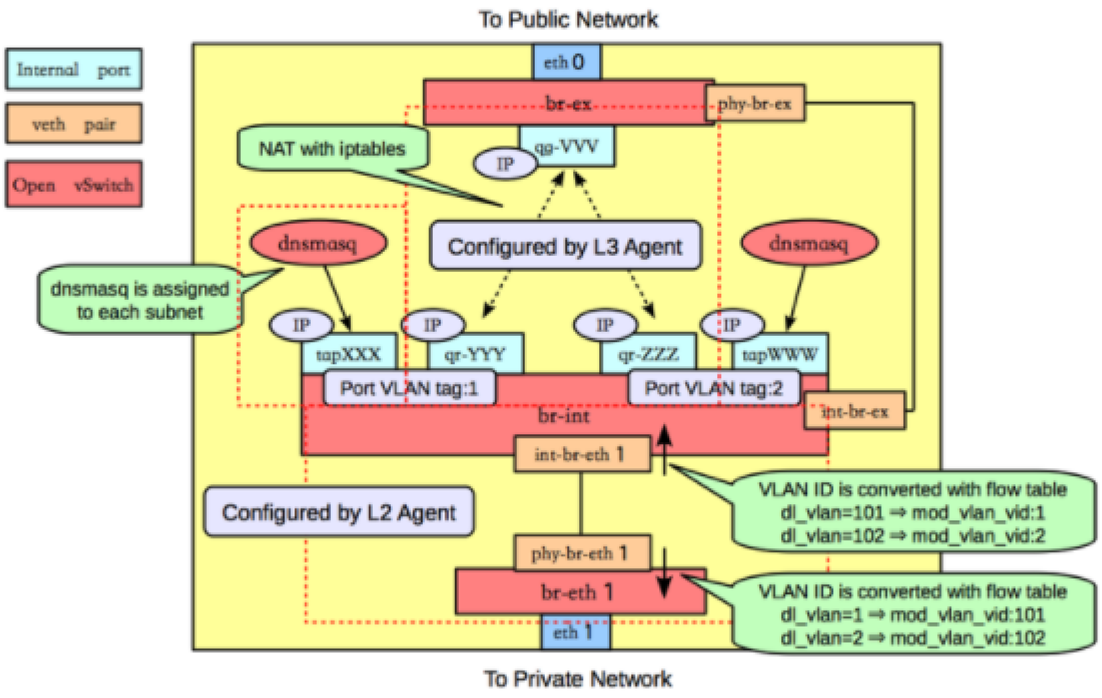


Figure 1.8: OpenStack Neutron L3 Agents

words, network stack virtualization. Similar to conventional networking, routing is required if packets are sent from one subnet to another, including traffic between VMs belonging to different subnets, or between VMs and hosts reachable via the External network. L3 depends on the L2 agent, in similar way Nova depends on the L2 agent to plug the port and perform the L2 wiring. Once those ports exist, whether its the ovs port or a veth pair (virtual cable), they can be moved into a namespace, just like taking a wire and plug it into your device. After this it's L3 agent's task to configures the IP addresses on the interfaces. It configures the routing, whether its a basic routing table or extra routes that had been configured on the router. It uses iptables to implement the floating IP functionality and provides a floating IP to every respective instance. Neutron has to find out the dedicated router for every particular instance need to go through to get to the external network and it uses NAT to implement that floating IP. Assume that there are multiple tenant networks and one shared external network and each of the former is connected to the latter via a Neutron-based virtual router. Such a router will be:

- Connected to the internal (tenant) network via \ qg- \ (gateway) interface on br-ex

- Connected to the external network via port via \ qr- \ (router) interface on br-int integration bridge
- Having a namespace (\ qrouter-\ prefix) associated with router name to avoid IP Connection between networks

On the router, there is a metadata access, shared access for instances without floating IP and some of the advanced services are also integrated into it (VPNaas and FWaas).

NAT

The neutron-l3-agent implements its router's NAT or Network Address Translation, functionality using Linux kernel iptables, enabling packets from internal (tenant) networks to succeed in an external network before going out to the web. just like routing, NAT rules of a router ought to be executed under a particular router's namespace to isolate them from host's network and alternative tenants' networks.

Floating IP

By default, server instances have a default IP assigned to them. But it is available only internally. A floating IP needs to be assigned for outside access. The Compute API provides several functionalities to access and use floating IPs. The IP is created in the network where the server is launched. The API selects the first available floating IP from a pool of IPs and associates it with the instance. If an instance dies, its floating IP is available for reuse. The virtual router provides Floating IP by NAT and iptables. This L3 service allocates and associates IP addresses from an external network to internal tenant VMs to make them directly reachable from an external network. As explained above the neutron-l3-agent implements dedicated Floating IP also by using iptables to perform NAT.

1.3.1.6 DHCP Agent

Neutron depends on its DHCP Agent, neutron-dhcp-agent, located within the Network node to provide Dynamic Host Configuration Protocol (DHCP) services to tenant networks, thus allocating IP addresses to VMs. particularly, dnsmasq [20] is used as back-end service for this purpose. For every subnet created, there's a running dnsmasq daemon attached to the int-br via port with \tap-" prefix under a DHCP namespace.

1.3.2 Network Architecture

There are 4 types of network in a standard OpenStack configuration: Management, Guest, API and External networks [21].

Management network

Used for internal communication between OpenStack elements. used for administrative communications and OpenStack internal operations such as authentication, access to internal databases (on Controller node), configurations, and so on. When the cluster is being set up, all configurations that require multi-node connectivity use the Management network. The IP addresses on this network ought to be approachable solely within the data center and is taken into account the Management Security Domain.
Guest network
Used for VM data communication inside the cloud deployment. The IP addressing necessities of this network rely on the OpenStack Networking plug-in in use and also the network configuration selections of the virtual networks created by the tenant. This network is taken into account the Guest Security Domain.
External network
Any given Networking set up has a minimum of one external network. in contrast to the other networks, the external network isn't simply a virtually defined network. Instead, it represents a view into a slice of the physical, external network accessible outside the OpenStack installation. IP addresses on the external network are accessible by anybody physically on the outside network.

In addition to external networks, any Networking set up has one or additional internal networks. These software-defined networks connect on to the VMs. solely the VMs on any given internal network, or those on subnets connected through interfaces to an identical router, will access VMs connected thereto network directly.

It is essentially the gateway that allows traffic from VM instances to reach physical networks in other word, it used to offer VMs with internet access in some deployment situations. The IP addresses on this network ought to be approachable by anyone on the net. As such, VM traffic must go through the node or nodes that have routing capability. This network is taken into account to be within the public security Domain.
API network
Exposes all OpenStack APIs, as well as the OpenStack Networking API, to tenants. The IP addresses on this network ought to be approachable by anyone on the net. this might be identical network because the external network, because it is feasible to form a subnet for the external network that uses IP allocation ranges to use solely under the total range of IP addresses in an IP block. This network is taken into account the public Security Domain.

1.4 Networks and multi-tenancy

Apart from the physical networks connecting the various nodes, in OpenStack, there are 2 kinds of networks:The tenant network and The provider network

A provider network is a network that's external to the cluster and permits to have outside connectivity, by passing through the network node. A virtual instance can even allot (and then deallocate, when it's not needed) a floating IP on this network to achieve external visibility. provider networks are created solely by the OpenStack administrator.

Instead, the tenant network (also called self-service network) is formed by the cloud user for connectivity inside projects. However, it lacks of connectivity to external networks like the internet, unless it's using a virtual router that incorporates a gateway on a provider network. By default, tenant networks are absolutely isolated and don't seem to be shared with alternative projects. within the creation phase of a virtual instance, a fixed IP is taken from this network (which won't be modifiable once the creation, as a consequence of a neutron abstraction regarding the port). Moreover, it's necessary to remark that the instance receives the IP from a DHCP server (which truly could be a dnsmasq[20] process generated by the DHCP agent) that's configured to supply thereto instance always identical fixed IP.
Neutron supports different kinds of tenant networks.

- **Flat tenant network**: Here there's no tenant support; each instance resides on a similar network, which may even be shared with the hosts.
- **Local tenant network**: during this kind of network the instances reside inside the local compute host and isolated from any external networks.
- **VLAN tenant network**: 802.1Q tags (VLAN IDs) are used in VLAN tenant network, which is pretty almost like the VLANs used in the physical network. The benefit of this network that it permits all instances to communicate with each other over the environment.
- **VXLAN or GRE tunneling tenant network**: In such networks,private communication between instances are supported by network overlays.

Inside a node, tenant flows are perpetually separated by internally assigned VLAN IDs. Then, to perform communications among physical nodes, tenant flows are separated, for example, by user-defined VLAN IDs, VXLAN IDs or GRE IDs, depending on the chosen tenant isolation technique. it's necessary to remark that whichever it's the tenant isolation mechanism used, inside a node OpenStack can perpetually create use of automatically internally assigned VLAN IDs. Indeed, despite the fact that a tunneling technique is used for the tenant networks, the VLAN IDs are still accustomed isolate the tenants traffic within the node. However, though by means that of this techniques (VLAN, VXLAN, GRE, etc.) the multi-tenancy is achieved, further mechanisms are required to let different users have overlapping networks. To do that, the virtual routers for the tenant networks, still because the processes acting as a DHCP server, are implemented within many network namespaces. The network namespace is a technology of the Linux kernel that enables isolating multiple network domains within one host, by replicating the network software stack. Therefore, a process executed during a namespace sees only specific network interfaces (e.g. those of the router), their own routing and arp tables, their own firewall and NAT rules. Namespaces are able to guarantee L3 isolation, creating doable for the various users to have to overlap IP addresses. Therefore, multi-tenancy is achieved by the use of techniques as VLAN tagging and/or tunneling (VXLAN, GRE), whereas the L3 isolation is granted by the use of many Linux kernel network namespaces. However, it's necessary to know that neutron provides multiple network abstractions to let the user be transparent to any or all these low-level details required to deliver and enhance multitenancy in this distributed system.

CHAPTER 2

Open VSwitch

This section talks about the Search Results essence of the Open vSwitch (OVS), that how an open-source software switch became a must choice for implementation in the Linux kernel and designed to work in a multiserver virtualization environment, how it emerged in the networking industry, became an inseparable part of Software-Defined Networking and where it is taking us. There are several SDN-related technologies including OpenFlow, Open vSwitch, and OpenDaylight.However, this chapter mainly focuses on Open vSwitch and SDN.A standout amongst the most pivotal parts of this section is the way and where Open vSwitch fits into the OpenStack picture.

2.1 Motivation For Open vSwitch

Hypervisors would like the flexibility to bridge traffic between VMs and with the surface world. On Linux-based hypervisors, this accustomed mean using the inbuilt L2 switch (the Linux bridge), that is quick and reliable. So, it is reasonable to ask why Open vSwitch is employed.

The answer is that Open vSwitch is targeted at multi-server virtualization deployments, a landscape that the previous stack isn't well matched. These environments are typically identified by extremely dynamic end-points, the maintenance of logical abstractions, and (sometimes) integration with or offloading to special purpose switching hardware.

Following are the features of Open vSwitch[39] which put it on the top :

- Visibility into inter-VM communication via NetFlow, sFlow(R), IPFIX, SPAN, RSPAN, and GRE-tunneled mirrors
- LACP (IEEE 802.1AX-2008)
- Standard 802.1Q VLAN model with trunking
- Multicast snooping

- IETF Auto-Attach SPBM and rudimentary required LLDP support
- BFD and 802.1ag link monitoring
- STP (IEEE 802.1D-1998) and RSTP (IEEE 802.1D-2004)
- Fine-grained QoS control
- Support for HFSC qdisc
- Per VM interface traffic policing
- NIC bonding with source-MAC load balancing, active backup, and L4 hashing
- OpenFlow protocol support (including many extensions for virtualization)
- IPv6 support
- Multiple tunneling protocols (GRE, VXLAN, STT, and Geneve, with IPsec support)
- Remote configuration protocol with C and Python bindings
- Kernel and user-space forwarding engine options
- Multi-table forwarding pipeline with flow-caching engine
- Forwarding layer abstraction to ease porting to new software and hardware platforms

2.2 Open vSwitch

Open vSwitch is an open-source project of a distributed virtual multi-layer switch that permits hypervisors to virtualize the networking layer. This caters to an oversized range of virtual machines running on one or additional physical nodes. With the help of the virtual bridges, virtual machines are able to communicate with each other on a similar physical node, as the connectivity of the VMs with ports of virtual bridges works in similar fashion to a physical server's connection to physical ports on a Layer 2 networking switch. In order to communicate outside the hypervisor node these bridges mutually connect virtual machines to the physical network. In OpenStack, each the neutron node and also the compute node (Nova) are running Open vSwitch to provide virtualized network services [23].

OVS depends on 2 things in order to forward packets between hosts: ***virtual network bridges and ow rules***. OVS has 3 main elements together with the **server module (ovsdb-server)**, the main daemon (ovs-vswitchd) and a kernel module.

- ovs-vswitchd – Open vSwitch daemon (Slow Path): this is often the software package module sometimes running in user space to talk to a control cluster, that often embodies network management modules and an SDN controller, to require remote network configuration and program that into the kernel quick path.
- ovsdb-server – Open vSwitch database server wherever OVS switch-level configuration and policy data is stored.
- openvswitch_mod.ko – kernel module (Fast Path): this is often the software package module typically running on OS or hypervisor kernel to really perform packet process.

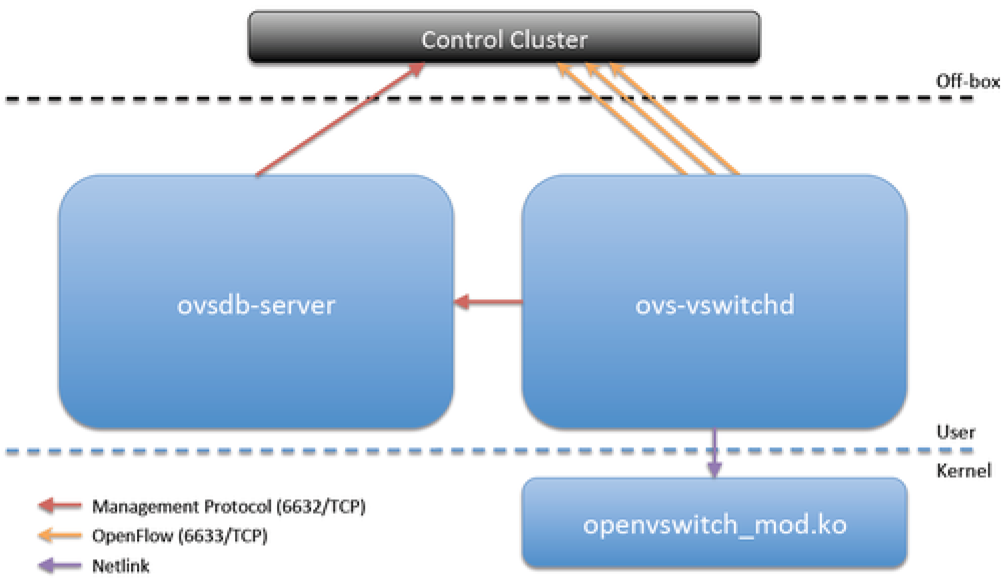


Figure 2.1: Open vSwitch Architecture

OVS provides high-level interfaces together with, among the others, ovs-vsctl and ovs-vsctl that enable OpenStack to configure and use it because of the underlying L2 switch. Specifically, Neutron's OVS agents (neutron-ovs-agent) store switch-level configurations into the OVS server module. The ovs-vswitchd daemon can browse these configurations and come upon virtual networks accordingly. The kernel module of OVS conjointly called "fast-path", plays a vital role in packet forwarding, although the forwarding choices are mostly done by the OVS daemon, as its the kernel modules and it handles the majority of the traffic. Once OVS experiences a new traffic ow, the first packet of the ow, thought of a cache miss, is handled in user area by the OVS daemon, after that, all remaining packets of the ow are forwarded by kernel module for higher performance. As user-space process is far slower than kernel- space process, we might commonly experience some delay with the first few packets.

2.3 Opn vSwitch Characteristics

Open vSwitch is one of the three prevalent virtual switches besides VMware virtual switch and Cisco Nexus 1000V. Nicira, later purchased by VMware, made Open vSwitch to address the issues of the open source community since there was none element rich virtual switch offering intended for a Linux-based hypervisor, for example, KVM and XEN. OVS has rapidly turned into the accepted virtual switch for XEN environments, and now it is having a vital impact in the OpenStack project and in addition a noticeable building obstruct in SDN environments. The following characteristics helped Open vSwitch endure the essential requirements.

2.3.1 The mobility of state

All network state related with a network element such as virtual machine ought to be effortlessly identifiable and migratablein an heterogeneous environment. This may embodied with conventional ”soft state”, L3 forwarding state, ACLs, QoS , and so forth. Open vSwitch has incorporated both configuring and migrating network state between instances.

2.3.2 Responding to network dynamics

Virtual environments are often described by high-rates of diversification in terms of adding/ removing VMs, making changes in Network configuration and so on. Open vSwitch permits a network control framework to acknowledge and adjust as environment changes.

2.3.3 Maintenance of logical tags

Distributed virtual switches for example, VMware vDS and Cisco's Nexus 1000V regularly sustain logical context within the network through appending or manipulating tags in network packets. This can be utilized to particularly recognize a VM or to hold some other setting that is just relevant in the logical domain. A significant part of the issue of building a distributed virtual switch is to effectively and accurately deal with these tags. Open vSwitch incorporates numerous techniques for determining and keeping up tagging rules, which are all available to a remote process for orchestration.

2.3.4 Hardware integration

Open vSwitch's forwarding path is intended to be managable to ”offloading” packet handling to equipment chipsets, regardless of whether it is in a classic equipment switchchassis or in an end-host NIC. This takes into account the Open vSwitch control path to have the capacity to both control a refined software implementation or an equipment switch. Open vSwitch control abstractions, both bare-metal and virtualized hosting environments that can be managed using the same mechanism for automated network control.

2.4 Software Defined Networks

Many big online and cloud companies are moving their data centers towards virtualization in order to get benefit from QoS and predictability. Likewise, highly secure and low-cost overhead networking is highly important. Therefore, service providers and networking companies requires an alternate solution to smartly and efficiently counter this emerging networking landscapes. SDN which is also known as software defined networks is transpired as the powerful technology capable of handling smart applications as well as dynamic landscape of the future networks while reducing the operational cost with simply changing the hardware and software and increasing and simplifying the management of the network by simplified software [36].An overview of SDN's architecture is shown in the figure 2.2 [40]

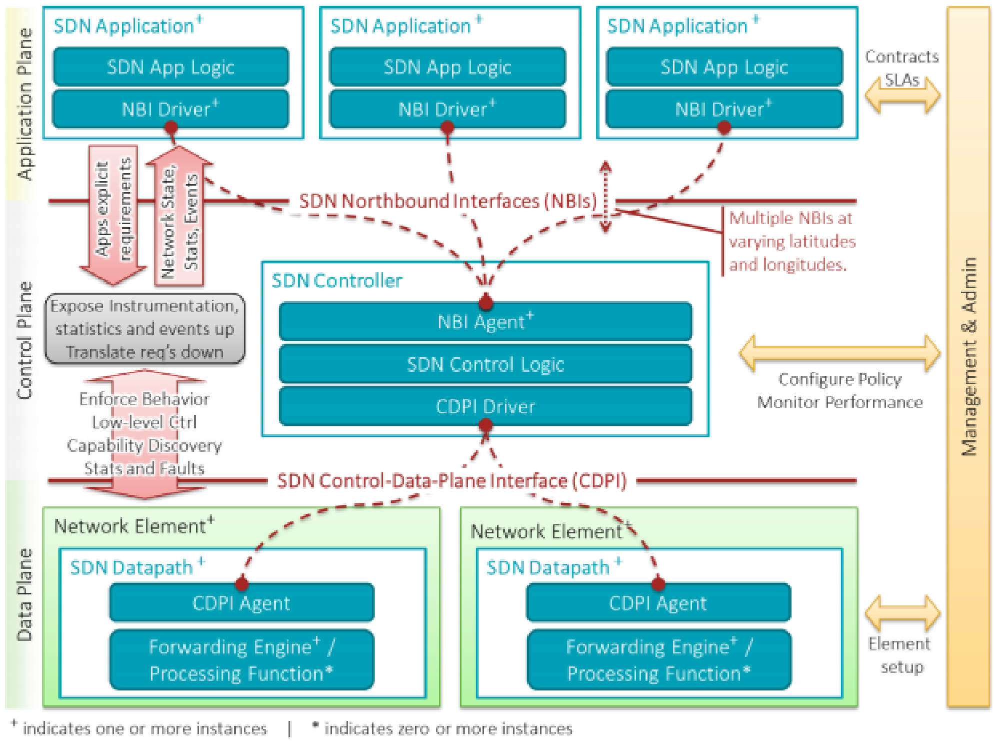

Application Plane
Control Plane
Data Plane
SDN Application+
SDN App Logic
NBI Driver+
Apps explicit requirements
Network State, Stats, Events
SDN Northbound Interfaces (NBIs)
Multiple NBIs at varying latitudes and longitudes.
Contracts SLAs
Expose Instrumentation, statistics and events up Translate req's down
Enforce Behavior Low-level Ctrl Capability Discovery Stats and Faults
SDN Controller
NBI Agent+
SDN Control Logic
CDPI Driver
Configure Policy Monitor Performance
Management & Admin
SDN Control-Data-Plane Interface (CDPI)
Network Element+
SDN Datapath+
CDPI Agent
Forwarding Engine+ / Processing Function*
Element setup
+ indicates one or more instances | * indicates zero or more instances

Figure 2.2: Overview of SDN Architecture

Setting up a network involves installations and complex configuration of routers, switches and other networking devices and requires a highly trained and skilled engineer to deal with the complexity. This approach is the system based and vendor based approach and this approach increases the provisioning cost, human resource cost in managing the multilevel and multi-vendor networks and decreasing the revenue. This is the perfect time for the service providers to go ahead with the alternatives like SDN that increase the network management and reduce the provisioning overhead [1]. Software defined networking (SDN) is the termed emerged and coined recently but the concept of SDN comes form 1996. Many companies start building and implementing SDN to make this a reality and main drivers behind the SDN are Ipsilon. Internet Engineering Task Force (IETF), OpenDaylight, Open networking foundation (ONF), Ethane and OpenFlow to name few [36]. Sometimes people refer SDN as OpenFlow but in reality OpenFlow is just an API for SDN [36].

Software defined networking is to allow network administrators to adapt quickly according to the dynamic business environment and be able to control the network from a centralized location called control console without even touching the separate console for router or switches or other networking devices and can providing the quick service and support to the network without knowing how the particular server or router is connected to other devices [37-38].

2.5 Why SDN?

The main motivation behind SDN is to streamline the network performance, provisioning, and deployment by separating the controller and data planes and enabling programmability of the networking devices and enabling network engineers to perform network operations form one console without knowing the actual complex connectivity. Usually, configuring the network devices requires a highly trained network engineer to perform the task and vendor based devices require special training to perform the same task differently on different machines which increase the cost. SDN provides open source solution for the above problem making network operations streamlined [37-38]. SDN was designed to bring more flexibility in the network and supports virtualization. It segregates control engine and forwarding engine and enabled the programmability in control engine as well as abstracted the infrastructure from applications and other services [37-38].

Software-defined networking separates the network into three main parts, application layer, control layer, and infrastructure layer. Every SDN has a controller, northbound APIs, and southbound APIs.

- Controllers are also referring as the brain of the SDN network as I give the centralized bird eye view of the whole network and allows engineers to tell the devices in the infrastructure layer (forwarding plane) how to manage and what to do with the network traffic.

- Northbound APIs are used to communicate with the top layer and allows the network engineer to deploy the services and direct the traffic programmatically. These APIs also integrates lower layer called controller so that network developer and make change pragmatically and at the same time provide an abstraction to the upper layer.

- Southbound APIs are used to relay the traffic to the routers and switches in the forwarding plane (infrastructure layer). OpenFlow is considered to be the SDN but its just a southbound API [38]. Southbound API allows the network engineer to make changes to the SDN controller with respect to the demand on the go. ONF develops the OpenFlow which is the famous South bound API for SDN.

This abstraction, separation programming capabilities were not available in vendor based hardware and network engineer cannot program the devices. So, SDN allows us to do so in real-time without touching the device.

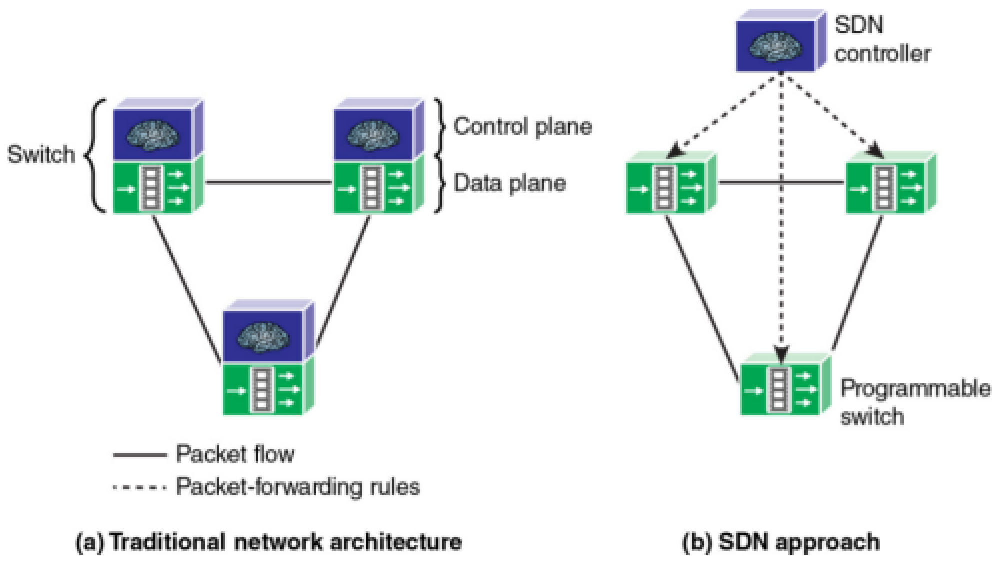


Figure 2.3: Traditional vs. SDN networking

2.6 Evolution of Software Defined-Networking

SDN was initial standardized in 2011 by the Open Networking Foundation (ONF), a user-driven organization dedicated to the promotion and adoption of SDN, and implementing SDN through open standards, necessary to move the networking industry forward. [25]. The ONF is that the entity behind the standardization of the OpenFlow protocol, one in all the foremost used approaches to perform SDN.

The scope of the SDN paradigm is to support an additional dynamic and scalable telecommunication networks environment. this is often achieved through the decoupling of the control plane (i.e. the routing, that decides a way to forward packets) from the data plane (i.e. the forwarding, that receives, stores and forwards the packets) [26].

Therefore, it's attainable to mention that SDN makes an attempt to:

- Creates flow tables controllable from the outside via a typical standardized API (Application Programming Interface), not tied to any vendor easier programmability of the network
- Affects a programming model, obligatory by the particular framework used

- Concentrates the control (i.e. the network intelligence) in a logically centralized entity, eventually physically distributed to avoid to own a single point of failure, this makes attainable to take care of a global view of the network that may seem to applications as one logical entity.

By separating the intelligence of the network from the data path, it's attainable to give to operators, developers, etc., the interface to program the network itself. Therefore, through the a-prior installation of rules within the switches, it is possible to implement a proactive way to implement selections within the network, however, if the switch isn't able to decide over an explicit packet (in general, it is possible to mention over an explicit header), the centralized control entity (the controller) are often reactively reached to determine what to do.

- The flow is truly a very general consideration: so, a flow could include an explicit application flow (e.g. all BitTorrent traffic), in addition to all the traffic associated with an explicit protocol (e.g. all ICMP traffic), in an exceedingly additional fine-grained view, it's doable additionally to contemplate it because of the traffic of an explicit user.
- Then, to every flow, there may be an explicit action to be taken: for example, this may be the redirection of traffic over an explicit node or maybe the tagging of a packet with an explicit VLAN to isolate it.
- It is exceptional to mention that the SDN is revolutionary up to an explicit degree: in specific, it's doable to examine similarities in terms of routing with Multi-Protocol Label Switching (MPLS), although the real plus point that SDN proposes is that the standardized interface that permits a neater programmability.

On the other hand, as a result of similarities, several operators are selecting to deploy MPLS over SDN to require the most effective out of the 2.

- Moreover, the centralized control is truly one thing that's not explicit to be essentially a part of SDN: however, its introduction fosters an additional global view of the network from the point of view of the network applications, which is something innovative and useful.
- Finally, SDN has not to be thought as restricted simply to specific networks, as data centers, edge networks or cloud networks: though it's not assured that SDN is a guaranteed market success, the aim of operators is to place SDN in production even in legacy networks.

2.7 SDN Controller

SDN Controller Platform widely used as SDN Controller is the "brain" of Software-Defined Network (SDN), as it is a logically centralized entity for the

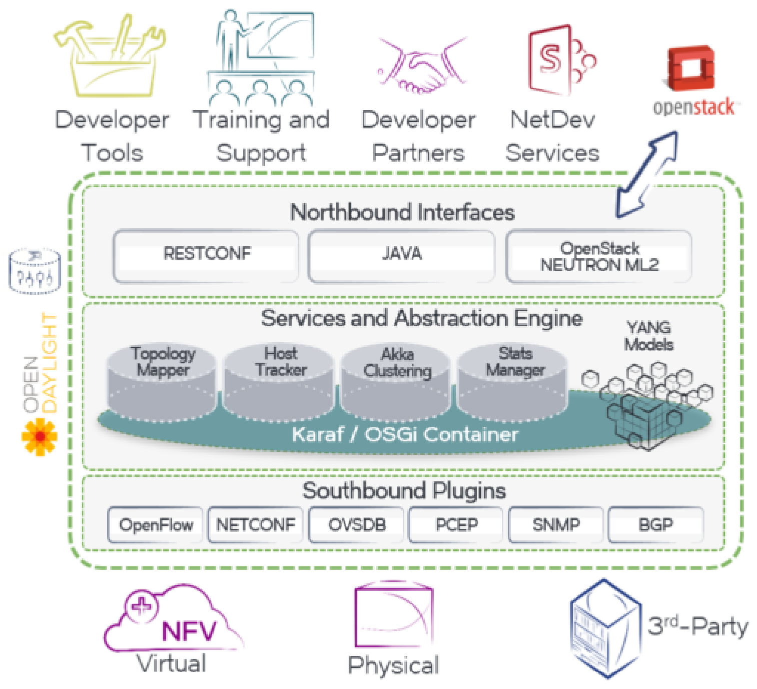


Figure 2.4: SDN Controller

control point. It mainly manages the flow control between Southbound APIs to Northbound APIs in order to deploy intelligent networks, in other words, SDN controller translates requirements from the SDN Application layer down to the SDN Datapaths and provides the SDN Applications an abstract view of the networks, as shown in the Figure 2.7 [38]. It contains a collection of the pluggable modules that can perform different network tasks, including some of the basic tasks such as gathering network statistics, inventorying devices within the network, as well as some of the high-level capabilities of running algorithms to perform analytics and orchestration of the new rules through the network.

OpenFlow and OVSDB are the most well-known protocols used by SDN Controllers, the use of the protocols can influence the entire network architecture.An SDN Controller consists of one or more NBI (SDN Northbound Interfaces) Agents, the SDN Control Logic and the CDPI driver.

2.8 SDN into OpenStack

Prior to Neutron, previously known as Quantum, OpenStack had a simple and flat networking environment without L3 or firewall support. The network constructs were baked into Nova which made it difficult to accommodate the changes happening in networking. Neutron was introduced to separate the networking part from other OpenStack service and provide different implementation choices of the abstractions in which Neutron server provides abstraction definition and management, while Neutron plugins do the actual implementation. However, Neutron has been criticized for its complexity and deficiencies on several OpenStack user surveys.

Installing and operationalizing OpenStack is still a constant struggle, especially at scale, and Neutron is a major obstacle to the scalability and resiliency of OpenStack deployments. The reason is that Neutron does not have its own layer 3 routing capability, but it uses a Linux kernel and Linux routing instead. In a large cloud environment with a lot of virtual networks, tenants, and applications, all traffic requiring routing and floating IP services need to be handled by the same Neutron L3 agent. Therefore, the agent becomes the choke point. While it is possible to deploy multiple pairs of L3 agents, it was proven to be very complicated even at moderate scale. Furthermore, there are two networks, physical and virtual, to manage, which makes correlating issues seen in virtual networks to physical networks more perplexing and time-consuming.

SDN solutions can distribute their own L2/L3 agents among OpenStack nodes to help eliminate Neutron L3 agent bottleneck issue. And SDN controllers centralize the management of physical and virtual networks, so it helps simplify managing and monitoring tasks. Additionally, SDN, as discussed above, exposes a myriad of network abstractions thanks to its centralized control plane, which makes it a perfect match for OpenStack. OpenStack supports RESTful APIs for every component. The integration of SDN into OpenStack can result in better networking abstractions and powerful programmatic APIs. The centralized management in SDN architecture also benefits a multi-vendor based cloud infrastructure like OpenStack.

Chapter 3

Approach and Methodology

This chapter describes analysis and design of neutron and alternative projects of OpenStack. as a result of all projects have nearly similar components, and hence follows similar skeleton not each part, that is why software analysis is required. These components share very similar architecture, therefore, there's no reason to speak regarding every component individually. we need to clarify the common structure and focus more on the foremost necessary elements of every sub-component. It will conjointly demonstrate however a proof-of-concept OpenStack cloud is constructed with Open vSwitch because of the main neutron ML2 plug-in. And then, OpenDaylight is integrated into that OpenStack cloud to achieve an SDN-based cloud. Also, aside from these main projects, this chapter will cover alternative projects, like Dragnflow, Calico, OpenContral and Project Romana.

3.1 Rapid Deployment Tools

To stimulate the deployment of the OpenStack cloud, loads of tools have been developed. The goal of these tools is to automate OpenStack deployment while prowling some of the complexity from the user.

The most common tools are:

- Devstack: An opensource community driven, used in this thesis.
- RDO: Driven by Red Hat.
- PackStack: A utility that uses Puppet Modules to deploy various parts of OpenStack on multiple pre-installed servers over SSH automatically.
- Fule: A rapid deployment tool which is developed by Mirantis.

3.2 OpenStack Deployment

This part covers how the OpenStack cloud is built and how to verify OpenStack functionality by launching an instance. In general, an OpenStack deployment may be of 2 types: **a production environment or a development environment.**

The ***development environment*** is incredibly straightforward to set up, adaptive to the users need and it's thought for an analysis of the platform or for the testing of the functionalities. Some acknowledge solutions are DevStack and PackStack. However, this deployment is incredibly unstable and it's not thought of running complex experiments or to incline to a customer.

Instead, the ***production environment*** is a lot of complicated to set up, but also finer-grained. Indeed, its installation needs a, typically long, planning phase wherever it's selected which nodes the components got to be deployed and with which configuration.

Then, the particular deployment would possibly occur in terms of writing configurations by-hand when installing the components, as the official OpenStack guide recommends for relative little environments, or by the utilization of automation tools as Red Hat TripleO, Ansible, Chef, Puppet, Fuel, Autopilot, etc. In each case, the educational curve is incredibly steep, therefore it is up to the Cloud administrator to determine which tools to use, considering also the dimensions of the cluster to be designed. For this thesis I am using DevStack deployment.

3.2.1 Infrastructure

The base infrastructure used to deploy the software is a basic laptop. Its principal features are:

- RAM: 16 GB
- Number of CPUs: 16
- Processor: Intel core i7-7600U

However, OpenStack needs a minimum of three nodes to work as desired. This nodes are virtual machines that will run over the physical host. The virtual infrastructure needed is the following:

- Virtual machines: Each node needed in the deployment will be a virtual machine. In this deployment, it has been used three virtual machines, one for each node named controller, network and compute.
- Virtual networks: This networks will be used to connect the virtual machines. Two networks have been defined in this deployment, one for management purpose and the other for internal information exchange between nodes.

3.2.2 Virtual Machines

In order to run VMs I am using VMWare Workstation, internally all virtual machines run over KVM hypervisor and I have installed a Ubuntu Server 16.04.3 LTS as base image. The technical features for each instance are the following:

- RAM: 8GB
- Number of CPUs: 4
- Storage space: 30GB

The number of interfaces in each node will vary depending on the node.

3.2.3 Single-Node Environment

In recent OpenStack versions, the OpenStack command is taking over and provides a generic interface to many projects, such as:

- OpenStack network: manage networking in Neutron
- OpenStack project: manage Keystone projects
- OpenStack server: manage instances in Nova
- OpenStack stack: manage stacks in Heat
- OpenStack volume: manage volumes in Cinder, etc.

However, not all the projects use it, so, occasionally, the user needs to make use of project-specific tools as well. Although, in this thesis, we are using Single host installation,as due to PC resources are limited and doesn't support multi-node environment.

OpenStack is installed on a personal PC running Windows 10 OS and VMWare Workstation is used to form Virtual Machine as guest OS. The installation relies on the Devstack[30] script.Also, in order to understand OpenStack better initial lab set up as single-host installation,single-host installation brings an initial hands-on expertise with OpenStack functionalities whereas simplifying the installation process and remaining appropriate for a PC with restricted hardware resources.

Under all-in-one mode of Devstack, basic OpenStack modules, and services (Identity, Nova, Neutron, Compute, Dashboard) are installed and run on the same host machine. While Devstack effectively provides an instant and easy method of experiencing OpenStack, it doesn't absolutely expose the installation method and is a smaller amount customisable.

This lamentably, makes it harder to understand the system, furthermore, on debug, its components should any problems occur throughout either installation or operation stage. Moreover, DevStack is taken into account a lot of appropriate for starters instead of a perfect resolution for a full featured OpenStack cluster.

3.3 DevStack

DevStack is a set of scripts and utilities to quickly deploy an OpenStack cloud environment and it is freely available on GitHub. DevStack allows developers and system administrators to automate the process of installing OpenStack on a server reducing it to a simple command for every installation. The services that are configured by default are Identity (Keystone), Object Storage (Swift), Image Storage (Glance), Block Storage (Cinder), Compute (Nova), Network (Nova), Dashboard (Horizon) and Orchestration (Heat). The main script is `stack.sh`, it does all the works, installing and configuring all the services set by the user, however, in order to use stable version `git checkout stable/pike` command used. All the required configurations, such as the Git repositories to use, the services to enable or the OS images to use, can be achieved overriding default environment variables (found in `stackrc`) through file `local.conf`.This is achieved with a `localrc` section, as shown below[30]:

```
==========================================
[[local|localrc]]
ADMIN_PASSWORD=Password1
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=192.168.214.128
==========================================
```

The environment variable `ENABLED SERVICES` is used to define the service to run,for example the Nova services to install, in a simple compute node installation, are `nova-compute, nova-api, nova-network`. By running the script `tools/install prereqs.sh` it is furthermore possible to install all the dependencies required by the configured services. Other useful scripts provided by DevStack are `unstack.sh` , that allows to stop everything that was started by `stack.sh`, and `clean.sh` that tries to remove all the traces left by the OpenStack installation performed by DevStack.

3.3.1 Networking configuration

The network node runs the Networking plug-in and a number of other agents that give tenant networks and provide switching, NAT, DHCP and routing

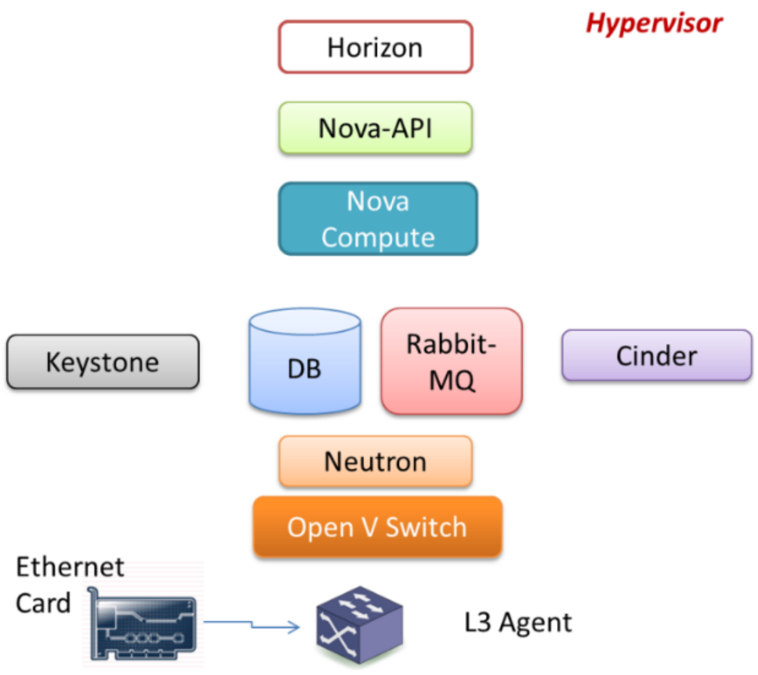


Figure 3.1: All-in-one architecture

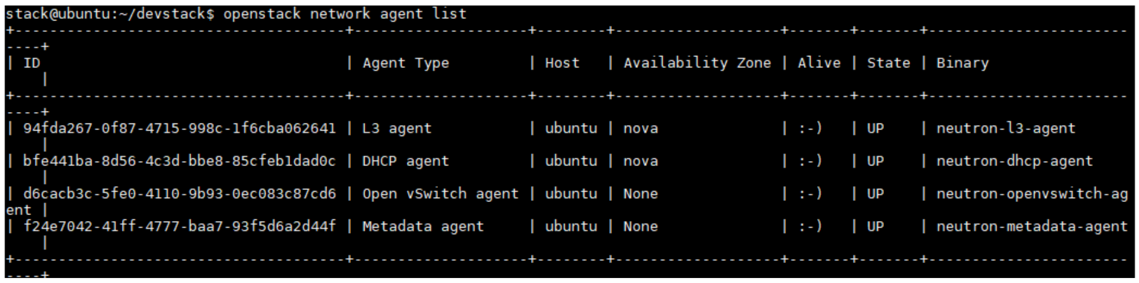
services. This node conjointly handles the external connectivity for tenant virtual machines instances.Before installing the modules, the networking configuration needs to be started in every node as it is showed in figure 3.1.5. This part is important so as to confirm the connectivity between the nodes and to the internet.

3.3.2 Network node

Networking is a big topic in OpenStack. In the past, networking was defined by Nova. However, as networking became more and more complex, a separate project was created to deal with it: Neutron. Several processes can be involved in making Neutron Networking happening in OpenStack. In order to make successful Networking in the OpenStack, Neutron needs few default agents, as well as some additional agents could be installed as per requirement[32]:
Default Agents

- **neutron-openvswitch-agent** :This agent installs the plugin for integrating openvswitch with neutron.

- **neutron-dhcp-agent** :This agent distributes IP addresses to the VMs using dnsmasq driver



```
stack@ubuntu:~/devstack$ openstack network agent list
+--------------------------------------+--------------------+--------+-------------------+-------+-------+-------------------------
----+
| ID                                   | Agent Type         | Host   | Availability Zone | Alive | State | Binary
    |
+--------------------------------------+--------------------+--------+-------------------+-------+-------+-------------------------
----+
| 94fda267-0f87-4715-998c-1f6cba062641 | L3 agent           | ubuntu | nova              | :-)   | UP    | neutron-l3-agent
    |
| bfe441ba-8d56-4c3d-bbe8-85cfeb1dad0c | DHCP agent         | ubuntu | nova              | :-)   | UP    | neutron-dhcp-agent
    |
| d6cacb3c-5fe0-4110-9b93-0ec083c87cd6 | Open vSwitch agent | ubuntu | None              | :-)   | UP    | neutron-openvswitch-ag
ent |
| f24e7042-41ff-4777-baa7-93f5d6a2d44f | Metadata agent     | ubuntu | None              | :-)   | UP    | neutron-metadata-agent
    |
+--------------------------------------+--------------------+--------+-------------------+-------+-------+-------------------------
----+
```

Figure 3.2: Neutron Default Agents

- **neutron-l3-agent** :This agent creates network devices that connect to l2 networks.
- **neutron-metadata-agent** :The metadata agent provides configuration information such as credentials to instances.

In order to run successful Neutron there are few additional agents/modules are also required:

- **neutron-server**
- **neutron-ovs-cleanup**
- **neutron-lbaas-agent**
- **neutron-plugin-ml2**
- **python-neutronclient**

3.3.2.1 Service configuration

The behavior of neutron is organized by 5 files named as below with the path where these files are stored:-

- neutron.conf stored in /etc/neutron/
- ml2_conf.ini stored in /etc/neutron/plugins/ml2/
- l3_agent.ini stored in /etc/neutron/
- dhcp_agent.ini stored in /etc/neutron/
- metadata_agent.ini stored in /etc/neutron/

As all the OpenStack configuration files, these also are distributed in many sections [nameofthesection].

neutron.conf
In order to type the configuration options.The neutron.conf which is responsible for the configuration of the database, authentication mechanism, message broker, topology change notifications, and plug-in.

Below is the default configuration of OpenStack with DevStack installation, regarding the [DEFAULT],[keystone_authtoken] and [nova] sections.
In the DEFAULT section there are few default options configured, such as:-

- service_plugins=neutron.services.l3_router_plugin.L3RouterPlugin. It sets router as the service plugin entry point to be loaded from the neutron.service_plugins.
- auth_strategy = keystone: It sets Keystone service as the authentication strategy.
- core_plugin=ml2: It sets ml2 framework as the Neutron core plugin entry point to be loaded from the neutron.core_plugins namespace.
- notify_nova_on_port_status_changes=True: It enables to send notification to nova when port status changes.
- notify_nova_on_port_data_changes=True: It enables to send notification to nova when port data, such as fixed ips or floating ip, changes so nova can update its cache.Etcs.

In the keystone_authtoken section there have been defined these options:

- project_name=service: It sets service as the Neutron admin tenant.
- user=neutron: It sets nova as the Neutron admin user.
- password=NEUTRON_PASS(Password1): It sets the Neutron users password in order to authenticate to Keystone.
- auth_url= http://IP_ADDRESS/identity: It sets the complete Identity API endpoint.
- auth_typr = password : It sets the password for the Identity API endpoint.
- auth_url= http://IP_ADDRESS/identity: It sets the complete Identity API endpoint.
- auth_typr = password : It sets the password for the Identity API endpoint.
- username=nova: It sets the username for connecting to nova in admin context.

- password=NOVA_PASS(Password1): It sets the password for connection to nova in admin context.
- region_name=RegionOne: It sets the name of nova region to use. As the configuration in Keystone only consider one region it is not very important.Etc.

Also to be noted that there are few options which are deprecated since the release of Queens version, which is used in this thesis, such as auth_uri,identity_uri, etc.

ml2_conf.ini
The second configured file is ml2_conf.ini which configures the Open vSwitch (OVS) mechanism to build the virtual networking framework for instances. In controller node there arent OVS components because it doesnt handle instance network traffic.

The ml2 section provides a list of network type and drivers on entrypoints:

- tenant_network_types=vxlan: It sets VXLAN network type as the network type to allocate as tenant networks.
- extension_drivers = port_security:It sets port_security as an extension driver entrypoints to be loaded from the neutron.ml2.extension_drivers namespace.
- mechanism_drivers=openvswitch,linuxbridge: It sets openvswitch and linuxbridge as the networking mechanism driver entrypoint to be loaded from the neutron.ml2.mechanis_drivers namespace.

The securitygroup section there have been defined these options:

- firewall_driver=iptables_hybrid: It sets the OVS iptables firewall driver.

In the ml2_type_* section there have been defined these options:

- flat_network = public: It sets public name for physical network with which flat networks is created.
- network _vlan _range = public: It sets physical network "public" without any VLAN range for provider and tenant.
- tunnel_id_ranges=1:1000: It sets from 1 to 1000 the possible GRE tunnel IDs that are available for tenant network allocation.
- vni_range = 1:1000 : It sest from 1 to 1000 the possible Geneve VNI and VXLAN VNI IDs that are available for tenant network allocation.

l3_agent.ini, **dhcp_agent.ini**, **metadata_agent.ini**
The next configured files are l3_agent.ini,dhcp_agent.ini and metadata_agent.ini these files provide routing services for virtual networks. In the DEFAULT section there have been defined these options:

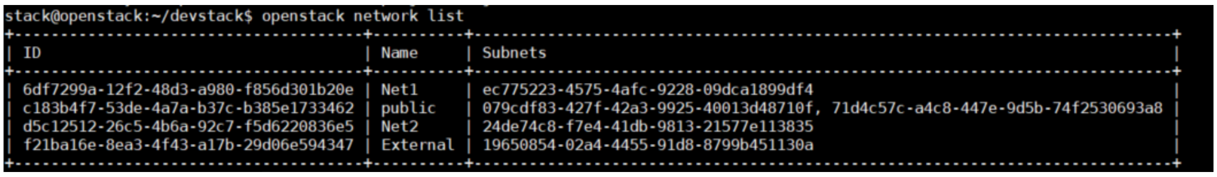
- interface_driver= openvswitch: It sets openvswitch as the driver to manage the virtual interface[34].
- ovs_use_veth = false: It sets false so that it doesn't use veth for OVS interface.
- debug = false : It sets If set to false, so that the logging level will not be set to DEBUG.

- interface_driver= openvswitch: It sets openvswitch as the driver to manage the virtual interface[33].
- ovs_use_veth = false: It sets false so that it doesn't use veth for OVS interface.
- dnsmasq_local_resolv = True: It sest true to enable the dnsmasq service to provide name resolution for instances via DNS resolvers on the host running the DHCP agent.
- debug = false : It sets If set to false, so that the logging level will not be set to DEBUG.

- metadata_workers = 2: It sets 2 as the number of separate worker processes for metadata server by default it halves the number of CPUs [35].
- nova_metadata_host = IP_ADDRESS: It sets the IP address of the Nova metadata server.
- debug = True : It sets If set to true, so that the logging level will be set to DEBUG.

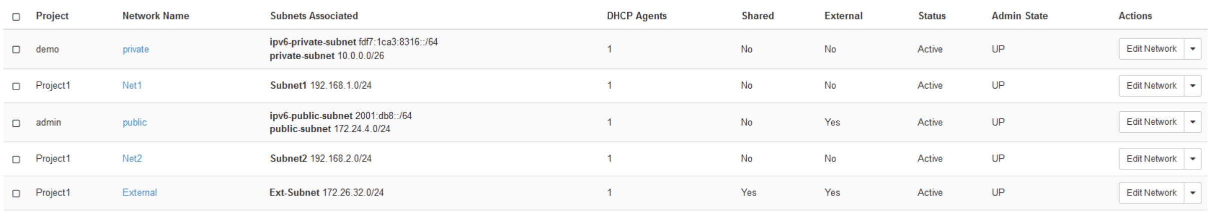
3.4 Neutron configuration

Once the Neutron service is properly working, now there could be modifications according to the requirement and it can configure a virtual network infrastructure to which instances will connect and be able to exchange data among them. To do so, it has been created two type networks, the external and the one or more internal networks which will be part of a single tenant.



```
stack@openstack:~/devstack$ openstack network list
+--------------------------------------+----------+----------------------------------------------------------------------------+
| ID                                   | Name     | Subnets                                                                    |
+--------------------------------------+----------+----------------------------------------------------------------------------+
| 6df7299a-12f2-48d3-a980-f856d301b20e | Net1     | ec775223-4575-4afc-9228-09dca1899df4                                       |
| c183b4f7-53de-4a7a-b37c-b385e1733462 | public   | 079cdf83-427f-42a3-9925-40013d48710f, 71d4c57c-a4c8-447e-9d5b-74f2530693a8 |
| d5c12512-26c5-4b6a-92c7-f5d6220836e5 | Net2     | 24de74c8-f7e4-41db-9813-21577e113835                                       |
| f21ba16e-8ea3-4f43-a17b-29d06e594347 | External | 19650854-02a4-4455-91d8-8799b451130a                                       |
+--------------------------------------+----------+----------------------------------------------------------------------------+
```

Figure 3.3: Networks with CLI



☐	Project	Network Name	Subnets Associated	DHCP Agents	Shared	External	Status	Admin State	Actions
☐	demo	private	**ipv6-private-subnet** fdf7:1ca3:8316::/64 **private-subnet** 10.0.0.0/26	1	No	No	Active	UP	Edit Network
☐	Project1	Net1	**Subnet1** 192.168.1.0/24	1	No	No	Active	UP	Edit Network
☐	admin	public	**ipv6-public-subnet** 2001:db8::/64 **public-subnet** 172.24.4.0/24	1	No	Yes	Active	UP	Edit Network
☐	Project1	Net2	**Subnet2** 192.168.2.0/24	1	No	No	Active	UP	Edit Network
☐	Project1	External	**Ext-Subnet** 172.26.32.0/24	1	Yes	Yes	Active	UP	Edit Network

Figure 3.4: Networks with Dashboard

3.4.1 External network

External networks commonly communicate over the physical networks that are publicly routable to the Internet. There can be one or more external networks to Neutron according to the requirement[41]:

- VMs can route packets from the internal network to the internet
- VMs can use floating IPs and use to the publicly communicate from the internet

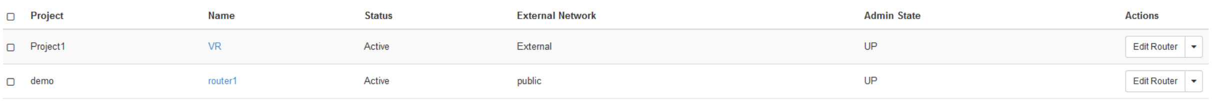
The external network provides Internet access to the instances utilizing NAT(Network Address Translation) by default, also it is exclusively connected to the network node and can be accessed using Neutron. External network has the scope for all tenants and can only be created by administrators. Tenants connect their router for external access. Mostly beneficial network types are flat (untagged) and VLAN (802.1Q tagged).

Network creation can be done from the users in a selected environment Dashboard or by using openstack CLI[42]:

Here Net1 and Net2 are configured as an Internal/Tenant Network and External Network is created for sending traffic over the cloud. In order to make Internal network communicate with the external network, Virtual Routers are used, for Project1 VR is used for routing the traffic and router1 is the default router.

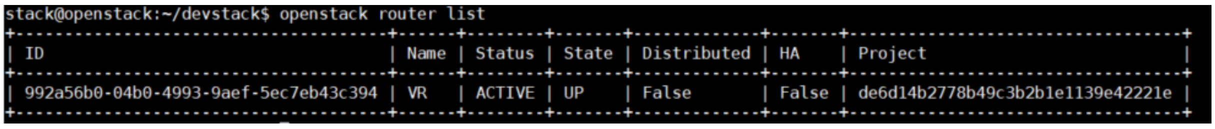
CLI provides more specific information related to the Project only.

Once the router is configured multiple qrouter and qdhcp namespaces appear on the network node, as they are the logical copy of the networking stack which owns routers, firewall rules, and network interface devices. qrouter and



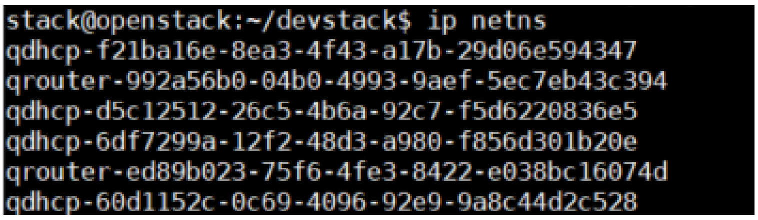
	Project	Name	Status	External Network	Admin State	Actions
☐	Project1	VR	Active	External	UP	Edit Router ▾
☐	demo	router1	Active	public	UP	Edit Router ▾

Figure 3.5: Virtual Routers



```
stack@openstack:~/devstack$ openstack router list
+--------------------------------------+------+--------+-------+-------------+-------+----------------------------------+
| ID                                   | Name | Status | State | Distributed | HA    | Project                          |
+--------------------------------------+------+--------+-------+-------------+-------+----------------------------------+
| 992a56b0-04b0-4993-9aef-5ec7eb43c394 | VR   | ACTIVE | UP    | False       | False | de6d14b2778b49c3b2b1e1139e42221e |
+--------------------------------------+------+--------+-------+-------------+-------+----------------------------------+
```

Figure 3.6: Virtual Routers for Project1



```
stack@openstack:~/devstack$ ip netns
qdhcp-f21ba16e-8ea3-4f43-a17b-29d06e594347
qrouter-992a56b0-04b0-4993-9aef-5ec7eb43c394
qdhcp-d5c12512-26c5-4b6a-92c7-f5d6220836e5
qdhcp-6df7299a-12f2-48d3-a980-f856d301b20e
qrouter-ed89b023-75f6-4fe3-8422-e038bc16074d
qdhcp-60d1152c-0c69-4096-92e9-9a8c44d2c528
```

Figure 3.7: Namespaces

qshcp namespaces have their own proprietary tasks, qrouter namespace is responsible to represent the virtual router and routes the traffic to and from the instances, however on the other hand qdhcp namespace is responsible for enabling the DHCP for the network[43].

Nevertheless, for interconnecting VMs within the Network and over the networks, OvS is utilized. Beside this Virtualization technique, all the instances have TAP interfaces works on the layer 2. Moreover, br-int receives VLAN tagged packets and recognize the network they belong to, and transfer the packet by tagging untagging it via br-ex or not according to the packet destination specification.

```
    stack@openstack: /devstack$ sudo ovs-vsctl show
d24c5356-3873-4a61-b1e4-cc541bedd351
       Manager "ptcp:6640:127.0.0.1"
           is_connected: true
       Bridge br-ex
           Controller "tcp:127.0.0.1:6633"
              is_connected: true
       fail_mode: secure
       Port "ens33"
           Interface "ens33"
       Port br-ex
           Interface br-ex
              type: internal
       Port phy-br-ex
           Interface phy-br-ex
              type: patch
```

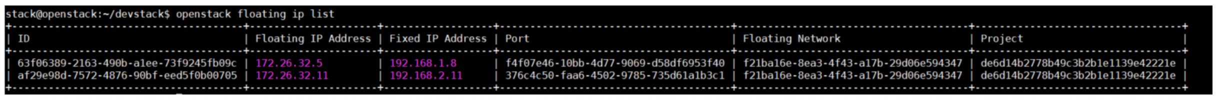
```
            options: peer=int-br-ex
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port patch-tun
            Interface patch-tun
                type: patch
                options: peer=patch-int
            Port "qvof4f07e46-10"
                tag: 3
                Interface "qvof4f07e46-10"
            Port "qr-9043aecb-62"
                tag: 3
                Interface "qr-9043aecb-62"
                        type: internal
            Port "qr-917086e6-d3"
                tag: 4
                Interface "qr-917086e6-d3"
                        type: internal
            Port "qr-e17b6cc0-04"
                tag: 1
                Interface "qr-e17b6cc0-04"
                        type: internal
            Port "tap062189fe-67"
                tag: 4
                Interface "tap062189fe-67"
                        type: internal
            Port "qr-c280739e-75"
                tag: 1
                Interface "qr-c280739e-75"
                        type: internal
            Port br-int
                Interface br-int
                        type: internal
            Port "qvo376c4c50-fa"
                tag: 4
                Interface "qvo376c4c50-fa"
            Port "tapbeff5a7c-d9"
                tag: 1
                Interface "tapbeff5a7c-d9"
                        type: internal
            Port "tapf6ef13e0-d7"
                tag: 3
                Interface "tapf6ef13e0-d7"
                        type: internal
            Port "tapbe499cd8-11"
                tag: 5
                Interface "tapbe499cd8-11"
                        type: internal
            Port "qvoa0e37be4-44"
                tag: 4
                Interface "qvoa0e37be4-44"
            Port "qg-cefb6d74-94"
                tag: 2
                Interface "qg-cefb6d74-94"
                        type: internal
            Port int-br-ex
                Interface int-br-ex
                        type: patch
                        options: peer=phy-br-ex
            Port "qg-dd957f5c-c1"
```

```
                tag: 5
                Interface "qg-dd957f5c-c1"
                      type: internal
             Port "qvoad441829-11"
                tag: 3
                Interface "qvoad441829-11"
          Bridge br-tun
             Controller "tcp:127.0.0.1:6633"
                is_connected: true
             fail_mode: secure
             Port patch-int
                Interface patch-int
                      type: patch
                      options: peer=patch-tun
             Port br-tun
                Interface br-tun
                      type: internal
          ovs_version: "2.6.1"
```

However, in order to route traffic from br-ex the below configuration commands were used[44]:

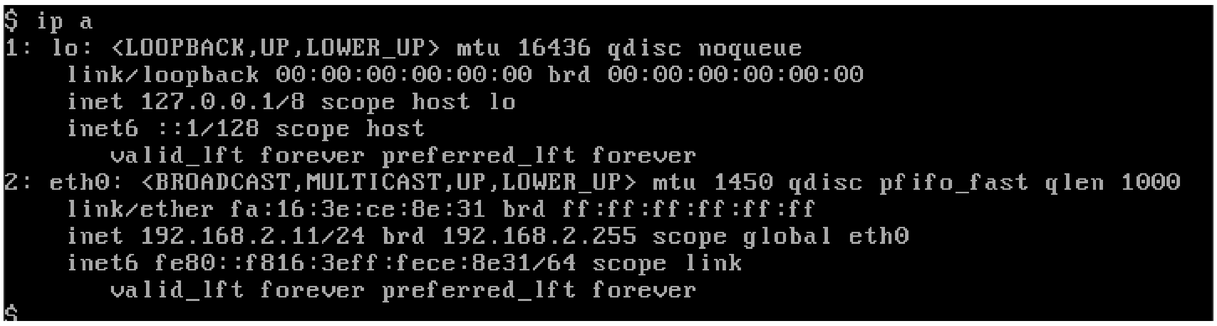
```
$sudo ovs-vsctl add-port br-ex ens33
$sudo ifconfig ens33 0
$sudo ifconfig br-ex 192.168.214.128 netmask 255.255.255.0
$sudo route add default gw 192.168.214.2 br-ex
```

Although, still getting access to the instances from the outer network we need Floating IPs and the instances have no idea regarding this Floating IP which is provided to them via their external network[43].



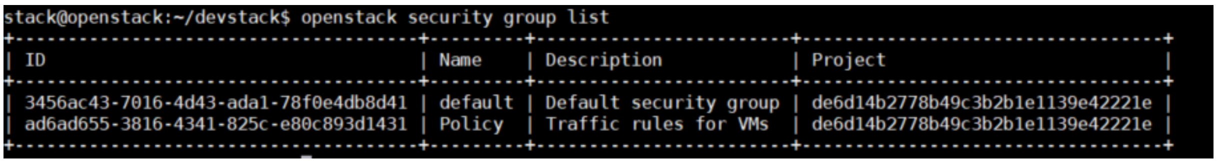
```
stack@openstack:~/devstack$ openstack floating ip list
+--------------------------------------+---------------------+------------------+--------------------------------------+--------------------------------------+----------------------------------+
| ID                                   | Floating IP Address | Fixed IP Address | Port                                 | Floating Network                     | Project                          |
+--------------------------------------+---------------------+------------------+--------------------------------------+--------------------------------------+----------------------------------+
| 63f06389-2163-490b-a1ee-73f9245fb09c | 172.26.32.5         | 192.168.1.8      | f4f07e46-10bb-4d77-9069-d58df6953f40 | f21ba16e-8ea3-4f43-a17b-29d06e594347 | de6d14b2778b49c3b2b1e1139e42221e |
| af29e98d-7572-4876-90bf-eed5f0b00705 | 172.26.32.11        | 192.168.2.11     | 376c4c50-faa6-4502-9785-735d61a1b3c1 | f21ba16e-8ea3-4f43-a17b-29d06e594347 | de6d14b2778b49c3b2b1e1139e42221e |
+--------------------------------------+---------------------+------------------+--------------------------------------+--------------------------------------+----------------------------------+
```

Figure 3.8: Floating IPs assigned to the Instances



```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc pfifo_fast qlen 1000
    link/ether fa:16:3e:ce:8e:31 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.11/24 brd 192.168.2.255 scope global eth0
    inet6 fe80::f816:3eff:fece:8e31/64 scope link
       valid_lft forever preferred_lft forever
$
```

Figure 3.9: Fixed IPs assigned to the Instances



```
stack@openstack:~/devstack$ openstack security group list
+--------------------------------------+---------+------------------------+----------------------------------+
| ID                                   | Name    | Description            | Project                          |
+--------------------------------------+---------+------------------------+----------------------------------+
| 3456ac43-7016-4d43-ada1-78f0e4db8d41 | default | Default security group | de6d14b2778b49c3b2b1e1139e42221e |
| ad6ad655-3816-4341-825c-e80c893d1431 | Policy  | Traffic rules for VMs  | de6d14b2778b49c3b2b1e1139e42221e |
+--------------------------------------+---------+------------------------+----------------------------------+
```

Figure 3.10: ecurity groups for Project

Security groups also plays an important role while routing the traffic, as the transfer of the packet depends on the "allow/deny" rules, here Policy is the assigned security group for this Project.

By default ingress and egress traffic is allowed for both IPv4 and IPv6, although either we can customize the traffic on the same security group or can create a new one.

For performing benchmarking the below topology would be used in this thesis, where the test cases would be created on the traffic flow with the below scenarios:

1. VMs in the same network and same Host
2. VMs in the different network and same Host
3. North-South traffic with Floating IPs
4. North-South traffic without Floating IP(Fixed IPs)

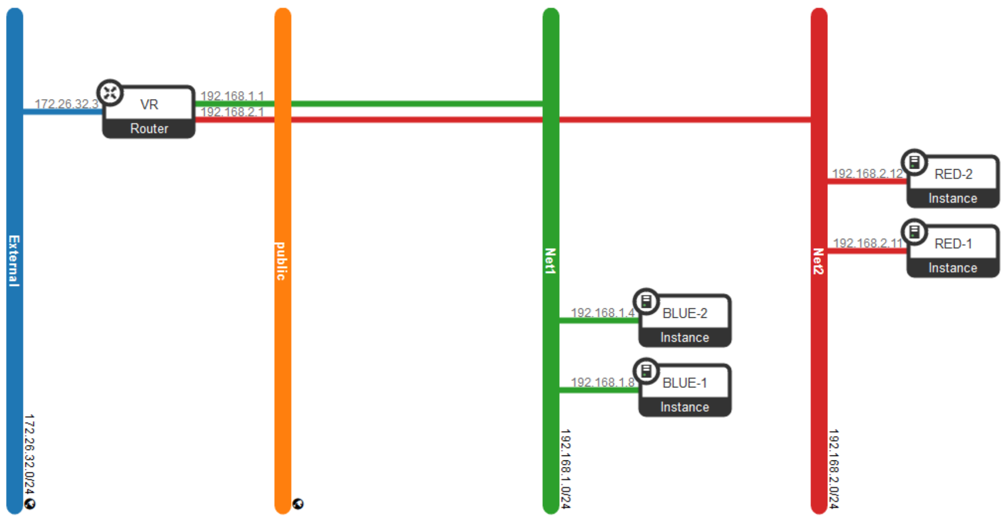


Figure 3.11: Topology for Benchmarking

3.5 Introduction of other Networking Project

As previously stated, the Neutron is in charge of the networking in OpenStack. As Neutron is an extremely complex project which grants multiple tenants to share physical resources that is transparent to the user still maintains isolation. However, Neutron is not the only Project to handle Networking in OpenStack, in this Chapter will concentrate particularly on the brief introduction of the other Networking Projects, so that the analysis of the implementation of Network nodes in OpenStack will work intensively on Neutron comparison Calico to investigate the Networking performance which can be accomplished by entirely diverse flavors of Network component of OpenStack.

3.5.1 Calico

Calico is an unofficial OpenStack project and it provides Neuton driver and DHCP agent using networking-calico project. The networking-calico is one of the numerous potential Neutron drivers which implement connectivity between different instances (VMs) as defined by the Neutron API. Metaswitch Networks is the principal company behind this project and it is maintained by Tigera team. It is an open source, Apache licensed project for virtual networking in data centers mainly to provide secure network connectivity for containers and VMs. In the early days of Networking in OpenStack, it was quite complex as if two applications needed to talk to each other, it used to put in an overlay and expose the virtual network concept to users and then start conflict isolation with network topology and as a result as the applications grew, the number of overlays grew and hence increase in the complexity, for the additional features of the traffic like east-west flows and north-south flows with virtual routing between overlays[49]. With the use of the microservices, the fundamentals of networking took a different approach and introduced the concept of IP per Pod, which assume that world is IP. So as a result of this new approach was to decouple isolation from networking and it allowed to move towards much simpler fabric and example of that is Calico. It is a pure L3 layer networking abstraction built cloud-native platforms, which creates and manages a flat layer 3 networks and assigns each VM a fully routable IP address to communicate without IP encapsulation or NAT for bare metal performance. Instead of overlays, it uses IP-In-IP tunneling approach otherwise works with other overlays for example flannel. It has flexible security policies and simple networking as it uses intentbased policy spans for private, public and hybrid clouds, even for heterogeneous workloads and orchestration environment. Its simple, highly scalable, non-overlay IP networking model with standard Linux data path and policy enforced on any underlying fabric makes it more attractive

Chapter 4

Analysis and design

The thesis will be carried out in two phases. The first phase of the experiment is to investigate traffic flow pattern and the second to study network performance in terms of throughput, packet loss, and delay. The research will be conducted by deploying virtual machines(VMs) on same and/or different hosts and networks respectively. All VMs will be using the 64-bit CirrOS operating system as their base OS.

Phase 1. Probing of The Network Traffic flow on OpenStack

Nowadays virtualization is a crucial function which has a huge benefaction in a cloud computing environment. It is needed for network transport and computing as well as storage. Network virtualization enables instances to communicate in a secure and pliable way during migration. It also creates virtual networks that provide an intellectual abstraction that makes easy to deploy and manage network services and underlying network resources.

In such system, traffic segmentation is achieved using VLAN tagging or GRE tunneling for each tenant. Thus, the different alternative of network switching system plays a big role in the complexity of networking architecture in cloud computing.

In the architectural setup of the experiment environment for this research on OpenStack, VLAN-tagging is activated for network traffic isolation. By studying different flowing pattern across the network, cloud providers will be able to find out the nature of VLAN-tagging service with other network services such as GRE Tunneling. To investigate traffic flow characteristics, a number of tools will be required. Those tools will be capable of tracing the flow of the traffic over the network. The tools that are going to be applied in this research will be traceroute or tracepath/route and tcpdump in order to observe the network flow.

Phase 2. Network Performance on OpenStack

The next task is to investigate is network performance in the OpenStack cloud computing environment, which is a high-performance computing network. In order to carry out this investigation, benchmarking tool VMTP, which is a small python application that can automatically perform ping connectivity, round trip time measurement (latency) and TCP/UDP throughput measurement for the following East/West flows on any OpenStack deployment for both TCP and UDP traffic.

4.1 Experimental setup

This experiment is conducted on the Pike OpenStack cloud and various networking components to compare with Neutron, which is already deployed on Ubuntu server 16.04. The system single node configuration with 30GB SCSI hard disk and 1 physical network card, 8GB RAM.

Cloud computing service providers provide different virtual instances which differ in their machine hardware, virtualization technology and hosting setup. Instances in the lower tier have slower CPU, fewer VCPU cores, less RAM size and less amount of disk size than instances in the higher tier. Therefore, the experiments done should be able to show how the difference in parameters affects network performance, as in this case, i.e. Open Stack, experiments will be performed on customized flavour Ubuntu for virtual machines.Their specifications are as in table:

Virtual Machine Type	**CPU**	**Memory(RAM) size**	**Total(Disk size)**
Ubuntu	1 VCPU	2GB	12GB

Table 5.1: Virtual Machine Specification

4.2 Experimental Design

The detailed experimental plan for the investigation task is as follows

4.2.1 Topology Case1 Experiment: Same Host Same Network

Experiment will be done to investigate the traffic flow between two virtual machines located at same host and same network.

4.2.2 Topology Case2 Experiment: Same Host Different Network

This test will be performed in order to study the traffic flow between two virtual machines located at same compute node but different network. The two different network can be attached to the same router or they can attached to a router.

4.2.3 Topology Case3 Experiment: North-South traffic with Floating IPs

In this case traffic flow goes from VMs using floating IP will leave the external bridge gateway from the local instance via the distributed virtual router.

4.2.4 Topology Case4 Experiment: North-South traffic without Floating IP(Fixed IPs)

In this scenario, we will look at the traffic headed towards external destination from VM which is not using floating IP hence to get outside it will rely on SNAT on the centralized network node traffic.

4.3 Studying the Network Performance

This section will evaluate the network behavior between instances within OpenStack cloud. Due to the use of different type of network equipment like switches, VLAN configuration and so forth and different networking projects, network properties within one cloud and network between different clouds have quite different behavior. Many providers promise high bandwidth from Mbps to Gbps inside their cloud infrastructure. To compare the network performance, matrices like throughput, latency, and loss will be measured. Both TCP and UDP will be used to measure the throughput.

In the experiment, the network throughput, which is a major factor, will be evaluated. It will also examine the behavior of bandwidth sharing in case of multiple Virtual machines in one compute node. Simultaneously round-trip packet delay and packet loss will be measured between two specified virtual machines. In order to carry out the throughput and packet loss experiments, Iperf which is a benchmark tool that generates TCP and UDP traffic will be used.Iperf uses a default bandwidth of 1.05 Mbps in case of UDP traffic. However in order to have accuracy in measuring achievable bandwidth, the maximum available bandwidth which is 10 Gbps will be set using b option. The default maximum transmission unit (MTU) and UDP buffer size is 1500 bytes and 224 Kbytes respectively. For TCP, default TCP window size is 23.5 Kbytes

The default the maximum transmission unit (MTU) and UDP buffer size are 1500 bytes and 224 Kbytes respectively. For TCP, default TCP window size is 23.5 Kbytes. To minimize complexity in measuring and evaluating the network performance, all factors that can affect the performance are kept as their defaults values. Factors like TCP window size, maximum transmission unit, UDP buffer size, datagram length, and parallel transmission will keep unchanged because the problem statement address how is the bandwidth sharing between instances with their defaults. Furthermore, the experiment will be huge if experiments are done by changing those parameters.

To measure packet round-trip delay (RTT), ping tool will be used. Basically, the ping command is used to check the existence of a network connection of a remote host by sending ICMP packets. If the host is reachable then it tells how much time it uses until the response comes back to its source that is the delay time of the packet. For better effectiveness, the tool will send ping packet for 1500 times and collect the average delay time.

Chapter 5

Results

This chapter includes the test result found from the actual experimental set up to address the problem statement section.To summarize the experiments setup for Ubuntu flavoured instances and their IPs addresses for the different scenarios are mentioned as VM1 and VM2 are in the same host that belongs to the same network and is taken as Case1. While VM3 and VM4 have different network address but they belong to the same host and treated as Case2. VM5 and VM6 reside on different network but using floating IP, which is Case3 where as VM7 and VM8 are on different network using Fixed IPs and is considered as Case4.

5.1 Results for Network Traffic flow

For each case as mentioned on prior, one VM sent continuously ping packet to its respective destination. At the same time, packets were tracked on both internal and external bridge interfaces of compute node and network node using tcpdump in order to see the traffic flow as in the following command:-

```
tcpdump -n -e -i interface host ip address
-n To display addressess by names
-e To display link-level header
-i To specify interface name
```

5.2 Results for Network Performance

This section explains the calculated average values of all network performance parameters considered in this project.

5.2.1 Average TCP Throughput

To measure TCP throughput, IPERF was executed for 15 minutes and data was collected every 5 seconds. The command was executed on the client side.

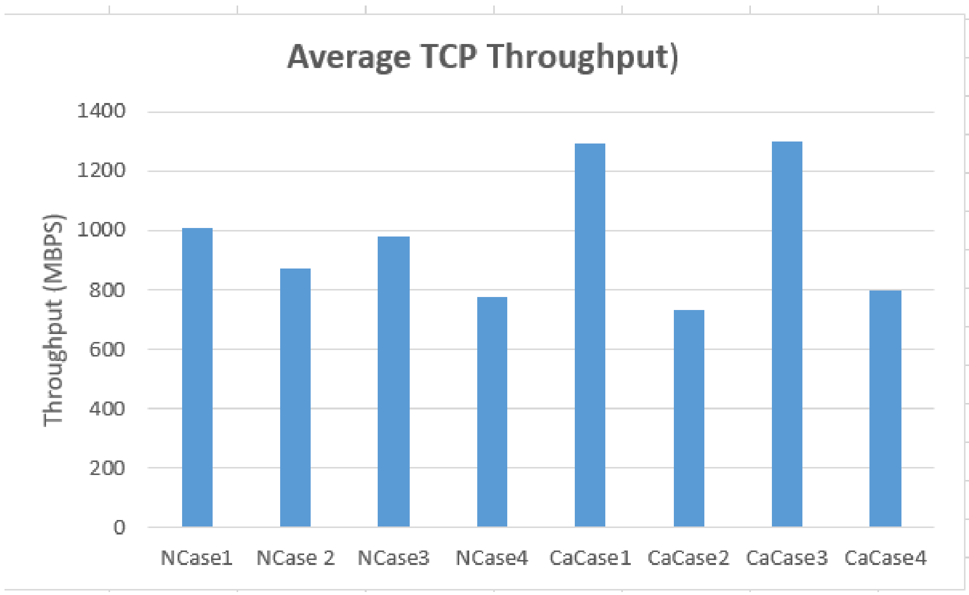


Figure 5.1: Average TCP throughput found for all scenarios

While collecting data for one scenario, the other scenarios were not running.On this experiment IPERF run for 900 seconds that collected throughput for every 5 seconds and then average value is calculated.

```
iperf -c server_ip -i 5 -t 900
-c To starts iPerf in client mode and connects with the iPerf server
-t To set the duration of the connection in seconds (default:  10 seconds)
-i To report intervals
```

5.2.2 Average UDP Throughput

In order to collect throughput and loss for UDP traffic, the following command was executed for 15 minutes and it reported data every 5 seconds.

```
iperf -c server_ip 5 -t 900 -b 10G
-c To starts iPerf in client mode and connects with the iPerf server
-t To set the duration of the connection in seconds (default:  10 seconds)
-i To report intervals
-b To set up bandwidth (default:  1.05 Gbps for UDP)
```

5.2.3 Average Packet Delay (Latency)

Ping command was executed on the client side for 1500 times in order to measure round-trip delay. During collection of data, no other processes were running on the VM.

```
ping -c 1500 server_ip
-c To starts ping in client mode and connects with the server
```

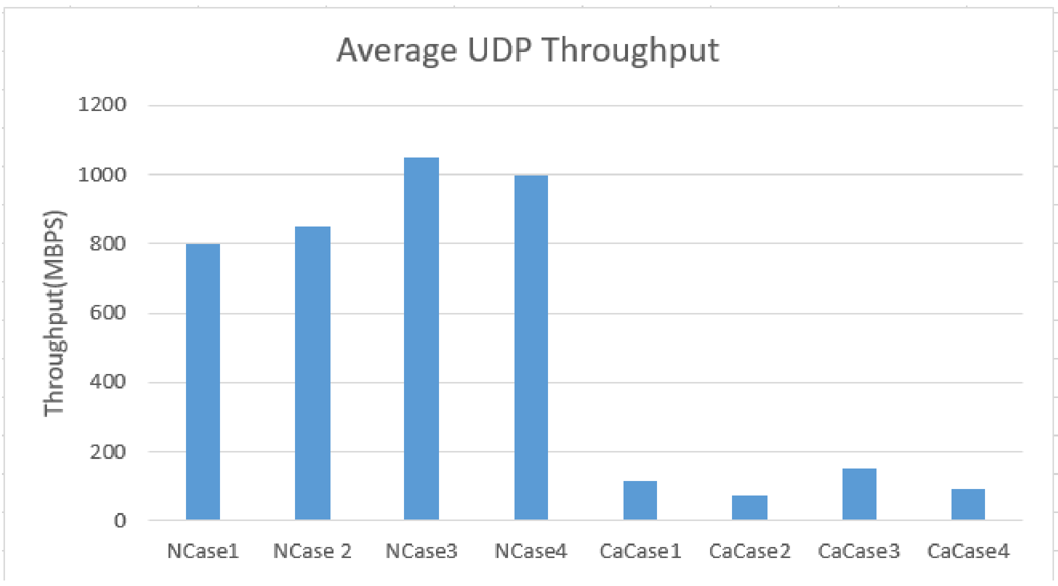


Figure 5.2: Average UDP throughput found for all scenarios

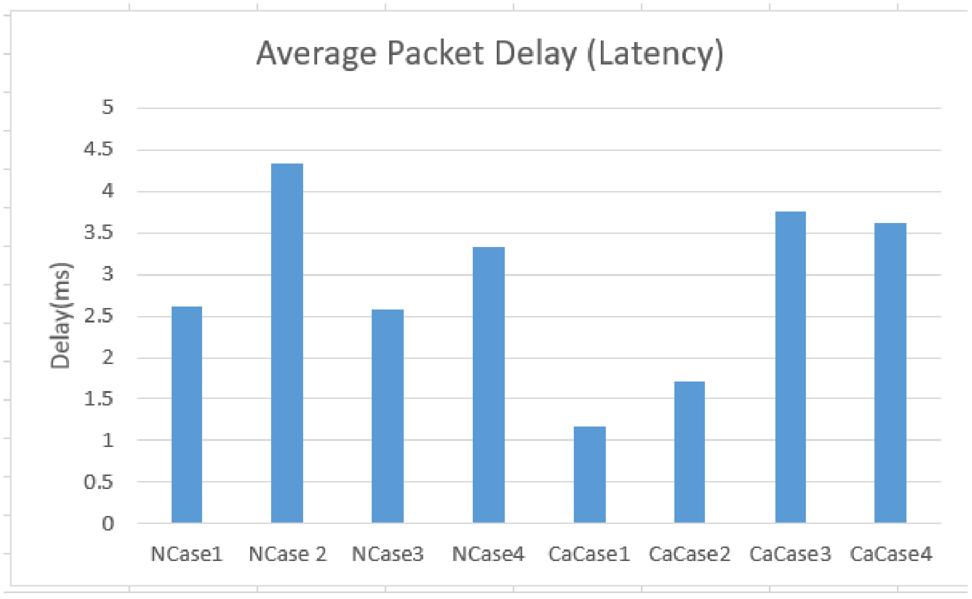


Figure 5.3: Average packet delay for all scenarios

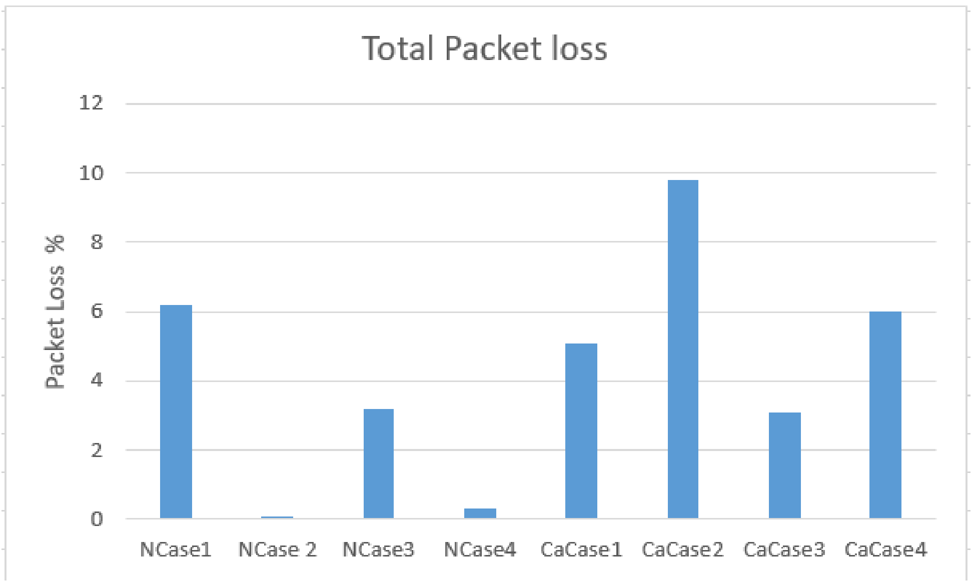


Figure 5.4: Total packet loss for all scenarios

5.2.4 Total Packet loss

Packet loss was obtained during the collection of UDP throughput.

5.3 Comparision between differnt Projects

As it has been illustrated by the figure 6.1 the throughput for VMs on the same host and same network is much higher than the TCP throughput for all other cases in both Neutron and Calico, however, Calico showed better performance with TCP data in comparison to Neutorn.However, on the contrary for UDP throughput Neutron gave better results and notability in Case 3 showed in figure 6.2. In terms of Latency Calico sent packets with less latency, moreover for case 1 it was quite less in comparison to all other cases respectively for Neutron and Calico , showed in figure 6.3 . Although, as can be seen easily from figure 6.4 packet loss is high in Calico project.

So from all eight scenarios, where the important parameters investigated were:- TCP throughput UDP throughput Packet delay Datagram loss From the network performance study, when two VMs are located on the same host and same network, they perform better than when they are on the same host but different network. Thus, the measurement showed that VMs on the same host and same network achieved an average TCP throughput better than VMs on same host but different network. On the other hand, there is no significant difference between their average UDP throughputs. In the case of packet delay, VMs on the same host but different network have higher values than VMs that are located on the same host and same network. This result is expected because when VMs are on different.

The measurement of packet (datagram) loss showed different results in case of both Neutron and Calico, in case Neutron it showed higest recorded packet loss when VMs are in same host and same Network, although for Calico it was VMs from the differnt network but on same host communicatin with each other. This result seems contradictory with the results of UDP throughput. This is because, if UDP throughput is similar between the two cases, so should their UDP loss be similar too. But, the result deviated from expectation. Therefore, a more thorough investigation becomes necessary to find the root cause of this deviation. First, the error may be due to defect in the VMs. For this case, different VMs were created and tested several times, but the result remained the same. Next, the compute node was changed several times, and the tests done. Still, the result remained the same. Then, the VMs were tested in a different scenario, and the results were confirming that there was no defect on the VMs or the compute nodes. Finally, the VMs were attached to a floating ip and the tests conducted.This is because, in this research, only private IPs were used throughout the whole experiment. This time, i.e. using floating ips, the loss came significantly down, even though the delay increased as expected. But this decrease in loss does not clarify why the previous losses were very high. More investigation is needed to conclusively explain this result,which could not be covered in the time scope of this thesis.

Conclusion and Future work

Owing to its importance and impact on improving network and distributed services, there is an ongoing research and innovation in cloud computing. As new innovations arrive, it is tantamount to evaluate new technologies applied on cloud computing environment, so that to enhance the correct understanding about the new technology. This research investigated the possible internal traffic flow pattern and evaluated network performance of each pattern on OpenStack cloud computing environment. Openstack falls into this category of new arrivals in cloud computing. Thus, this research aimed to add some contribution towards understanding the detail workings of Openstack. To that end, it investigated the possible internal traffic flow pattern and evaluated network performance of each pattern on OpenStack cloud computing environment. From the investigation, it can be confirmed that when Virtual machines (VMs) with private IPs are located on the same network, then they only use switches in order to communicate with each other independent of their location on the nodes. The results showed that the location of machines in terms of compute node and network address matters for the network performance. Thus, when VMs are on the same node and the same network , they perform better than other scenarios. This is because the transmission path, for example, is shorter (in case of delay) than the other scenarios. In OpenStack, bandwidth is unlimited in principle. What exactly is the effect of this unlimited bandwidth on performance? Since it is intuitive that network performance cannot be unlimited in practice, what exactly did Openstack provide by making unlimited bandwidth available? This needed an investigation to understand, and thus the investigation on network performance was conducted. And, the results from the study of network performance showed that, by providing unlimited network bandwidth, OpenStack didnt assure unlimited network performance. Rather it ensured that there will be no network bandwidth bottleneck.

Albeit the results showed that the future projects are clearly trying hard to provide better network performance than Neutron, however, still there are few developments needing as can easily seen in UDP throughput scenario for Calico.

Future Work

In this research, unforeseen results have been seen and unanswered questions were popped up. Having experienced the unknown challenges as well as considering the importance of dealing with some problems, it can be suggested further research work on the following events:-

1. Since both Neutron and Calico gives lower performance in terms of bandwidth for each VM, it would be important to study how to improve its network performance.
2. The study can be done in more wider prospective by adding other Networking plugins such as OVN,ODL etc.
3. As Neutron and Calico gave higher bandwidth for different VMs, it would be important to investigate CPU and memory usage of VMs.

Bibliography

[1] Cloud computing. Openstack.org [online]. 2018, 18 March 2018 [cit. 2018-03-27]. https://en.wikipedia.org/wiki/Cloud_computing[cit. 2018-03-27] .
[2]En.wikipedia.org. (2018). OpenStack. [online] Available at: https://en.wikipedia.org/wiki/OpenStack[cit. 2018-03-27].
[3] En.wikipedia.org. (2018). Virtualization. [online] Available at: https://en.wikipedia.org/wiki/Virtualization[cit. 2018-03-27].
[4] Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. [online] Nvlpubs.nist.gov. Available at: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf [cit. 2018-03-27].
[5] Docs.openstack.org. (2018). OpenStack Docs: Conceptual architecture. [online] Available at: https://docs.openstack.org/install-guide/get-started-conceptual-architecture.html[cit. 2018-03-27].
[6] VMWare. (2018). VMware Official Site. [online] Available at: https://www.vmware.com[cit. 2018-03-27].
[7] Linux-kvm.org. (2018). KVM. [online] Available at: https://www.linux-kvm.org/page/Main_Page[cit. 2018-03-27].
[8] Xenproject.org. (2018). VS16: Video Spotlight with Xen Project's Lars Kurth. [online] Available at: http://www.xenproject.org/[cit. 2018-03-27].
[9] En.wikipedia.org. (2018). Hyper-V. [online] Available at: https://en.wikipedia.org /wiki/Hyper-V[cit. 2018-03-27].
[10] Linuxcontainers.org. (2018). Linux Containers - LXC - Introduction. [online] Available at: https://linuxcontainers.org/lxc/introduction/[cit. 2018-03-27].
[11] RADEZ, Dan. OpenStack Essentials [online]. 3rd. Birmingham: Packt Publishing Limited, 2015 [cit. 2018-03-27]. ISBN 978-1-78398-708-5. Available from: https://www.amazon.com/OpenStack-Essentials-Dan-Radez/dp/1783987081
[12] Docs.openstack.org. (2018). OpenStack Docs: Networking service overview. [online] Available at: https://docs.openstack.org/neutron/latest/install/common/get-

started-networking.html [cit. 2018-03-27].
[13] Wiki.openstack.org. (2018). Neutron - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/Neutron [cit. 2018-03-27].
[14] Docs.openstack.org. (2018). OpenStack Docs: ML2 plug-in. [online] Available at: https://docs.openstack.org/neutron/pike/admin/config-ml2.html [cit. 2018-03-27].
[15] Mestery, K. and Kukura, R. (2018). Modular Layer 2 In OpenStack Neutron. [online] Openstack.org. Available at: https://www.openstack.org/assets/presentation-media/ML2-Past-Present-and-Future.pptx [cit. 2018-03-27].
[16] Docs.openstack.org. (2018). OpenStack Docs: Open vSwitch: Provider networks. [online] Available at: https://docs.openstack.org/ocata/networking-guide/deploy-ovs-provider.html [cit. 2018-03-27].
[17] Arie Bregman. (2018). Openstack Neutron: L2 L3 agents. [online] Available at: http://abregman.com/2016/01/03/openstack-neutron-l2-l3-agents/ [cit. 2018-03-27].
[18] En.wikipedia.org. (2018). Iptables. [online] Available at: https://en.wikipedia.org/wiki/Iptables[cit. 2018-03-27].
[19] Docs.openstack.org. (2018). OpenStack Docs: Layer 3 Networking in Neutron - via Layer 3 agent OpenVSwitch. [online] Available at: https://docs.openstack.org/neutron/pike/contributor/internals/layer3.html [cit. 2018-03-27].
[20] Thekelleys.org.uk. (2018). Dnsmasq - network services for small networks.. [online] Available at: http://www.thekelleys.org.uk/dnsmasq/doc.html [cit. 2018-03-27].
[21] Docs.openstack.org. (2018). OpenStack Docs: Networking architecture. [online] Available at: https://docs.openstack.org/security-guide/networking/architecture.html [Accessed 27 Mar. 2018].
[22] VIETSTACK TEAM. (2018). RPC in OpenStack. [online] Available at: https://vietstack.wordpress.com/2015/09/17/rpc-in-openstack/ [cit. 2018-03-27].
[23] Superuser. (2018). Understanding Open vSwitch, an OpenStack SDN component - Superuser. [online] Available at: http://superuser.openstack.org/articles/openvswitch-openstack-sdn/ [cit. 2018-03-27].
[24] Docs.ocselected.org. (2018). Tenant and provider networks - OpenStack Networking Guide - current. [online] Available at: http://docs.ocselected.org/openstack-manuals/kilo/networking-guide/content/tenant-provider-networks.html [cit. 2018-03-27].
[25] Opennetworking.org. (2018). [online] Available at: https://www.opennetworking.org/images/stor
resources/white-papers/wp-sdn-newnorm.pdf [cit. 2018-03-27].
[26] F. Hu, Q. Hao, and K. Bao. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. In: IEEE Communications Surveys Tutorials 16.4 (2014), pp. 21812206. issn: 1553-877X. doi: 10.1109/COMST.2014.2326417. [online] Available at: http://labs.xjtudlc.com/labs/wldmt/reading%20list/papers/Multimedia

%20and%20Networking/HHB14.pdf [cit. 2018-03-27].
[27] William Stallings. Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud. Addison-Wesley Professional, 2016, ISBN-13: 978-0134175393
[28] En.wikipedia.org. (2018). Kernel-based Virtual Machine. [online] Available at: https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine [cit. 2018-03-27].
[29]Penguin, T. (2018). Installing DevStack on Ubuntu 16.04 - The Urban Penguin. [online] The Urban Penguin. Available at: https://www.theurbanpenguin.com/installing-devstack-on-ubuntu-16-04 [cit. 2018-03-27].
[30] Docs.openstack.org. (2018). OpenStack Docs: Multi-Node Lab. [online] Available at: https://docs.openstack.org/devstack/latest/guides/multinode-lab.html[cit. 2018-03-27].
[31] Docs.openstack.org. (2018). OpenStack Docs: Manage projects, users, and roles. [online] Available at: https://docs.openstack.org/keystone/pike/admin/cli-manage-projects-users-and-roles.html [cit. 2018-03-27].
[32] Docs.openstack.org. (2018). OpenStack Docs: Install and configure controller node. [online] Available at: https://docs.openstack.org/mitaka/install-guide-rdo/neutron-controller-install.html [cit. 2018-03-27].
[33] Dhcp_agent.ini. Openstack.org [online]. 2018, 2018-02-08 [cit. 2018-03-27]. Available from:
https://docs.openstack.org/ocata/config-reference/networking/samples/dhcp_agent.ini.html.
[34] L3_agent.ini. Openstack.org [online]. 2018, 2018-02-08 [cit. 2018-03-27].
Available from:
https://docs.openstack.org/ocata/config-reference/networking/samples/l3_agent.ini.html
[35] Keystone.conf. Openstack.org [online]. 2018, 2018-02-08 [cit. 2018-03-27]. Available from:
https://docs.openstack.org/ocata/config-reference/identity/samples/keystone.conf.html
[36]SEZER, Sakir, Pushpinder Kaur CHOUHAN and Sandra SCOTT-HAYWARD. Are we ready for SDN? - Implementation Challenges for Software-Defined Networks [online]. IEEE Communications Magazine, 2013, 36-43 [cit. 2018-03-27]. DOI: 10.1109 / MCOM.2013.6553676. Available from:
https://pure.qub.ac.uk/portal/files/14448878/CommsMag_Final.pdf
[37] ROUSE, Margaret. Software-defined networking (SDN). TechTarget [online]. August 2015 [cit. 2018-03-27]. Available from:
http://searchsdn.techtarget.com/definition/software-defined-networking-SDN
[38]What is Software Defined Networking (SDN) ?. SDxCentral [online]. 2012-2018 [cit. 2018-03-27]. Available from: https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/
[39] Features - Open vSwitch [online]. Linux Foundation Collaborative Project, 2016 [cit. 2018-03-27]. Available from: http://www.openvswitch.org//features/
[40] BAILEY, Stuart, Deepak BANSAL and Linda DUNBAR. SDN Architecture Overview [online]. December 12, 2013, 3-5 [cit. 2018-03-27]. Available from: https://www.opennetworking.org/images/stories/downloads/sdn-

resources/technical-reports/SDN-architecture-overview-1.0.pdf
[41]Wikipedia. Wikipedia [online]. [cit. 2018-04-20]. Available from: https://en.wikipedia.org/wiki/Network_address_translation
[42]Openstack. Wikipedia [online]. [feeling. 2018-04-20]. Available from: https://developer.openstack.org/firstapp-libcloud/networking.html
[43]IGBE, Damian. Identifying and Troubleshooting Neutron Namespaces [online]. November 21, 2013 [cit. 2018-04-20]. Available from: https://www.mirantis.com/blog/identifying-and-troubleshooting-neutron-namespaces/
[44]IGBE, Damian. How to add eth0 and wlan0 to OpenvSwitch Bridge. Fosshelp [online]. October 30, 2014 [cit. 2018-04-20]. Dostupn z: http://fosshelp.blogspot.cz/2014/10/add-eth0-wlan0-openvswitch-bridge.html [45]Wikipedia [online]. [cit. 2018-04-22]. Dostupn z: https://en.wikipedia.org/wiki/OpenDaylight_Project
[46]Opendaylight [online]. [cit. 2018-04-22]. Dostupn z: http://docs.opendaylight.org/en/stable-oxygen/getting-started-guide/introduction.html [47]Sdxcentral. Sdxcentral [online]. [cit. 2018-04-20]. Dostupn z: https://www.sdxcentral.com/projects/romana/
[48]Openstack. Openstack [online]. [cit. 2018-04-20]. Dostupn z: https://docs.openstack.org/networking-ovn/latest/admin/ovn.html
[49]Projectcalico [online]. [cit. 2018-04-24]. Dostupn z: https://docs.projectcalico.org/v3.1/introduction/

Appendix **A**

Acronyms

OVN	Open Virtual Network
SDN	Software-defined networking
ODL	OpenDaylight
SaaS	Software as a Service
NaaS	Network as a Service
PaaS	Platform as a Service
OVS	Open vSwich

Appendix B

Contents of enclosed CD

```
│ readme.txt ....................... the file with CD contents description
├── exe ..................................... the directory with executables
├── src .....................................the directory of source codes
│   ├── wbdcm ..................................... implementation sources
│   └── thesis .............the directory of LaTeX source codes of the thesis
└── text .........................................the thesis text directory
    ├── thesis.pdf..........................the thesis text in PDF format
    └── thesis.ps............................the thesis text in PS format
```