Lappeenranta University of Technology
School of Business and Management
Degree Programme in Computer Science

Syed Asif Iqbal

GREEN SOFTWARE DEFINED DISTRIBUTED DATA CENTER MODEL
MASTER’S THESIS

Examiners: Professor Eric Rondeau
Professor Jari Porras
Professor Karl Anderson

Supervisors: Dr. Ah Lian Kor
Professor Colin Pattinson
ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Syed Asif Iqbal

GREEN SOFTWARE DEFINED DISTRIBUTED DATA CENTER MODEL

Master’s Thesis- 2018

Examiners: Professor Eric Rondeau
Professor Jari Porras
Professor Karl Anderson

Keywords: Software Defined Data Center, Software Defined Networking, GLBPP, Geographical Load Balancing, Distributed Data Center.

With the advent of cloud computing and IoT, the demand for large capacity data centers are increasing rapidly. Large capacity datacenters are predominantly dependent on fossil fuel energy. Designing a distributed data center architecture with high availability, scalability and very small CO2 footprint is a challenge faced by the industry. The Software Defined Systems (SDSys) paradigm comes as a solution to provide reduced complexity of control and management in distributed data center designs. Software Defined Networking (SDN) particularly brings flexibility in network management and can be deployed to reduce energy consumption as well as better fault tolerance. The software-defined distributed green data center model provides a new approach to both the implementation of cloud data center services and the efficient use of green energy resources. This research is a comparative study on Distributed Software Defined Data Center (SDDC), by evaluating the traditional approach and proposing a method to reduce non-renewable energy consumption with better fault tolerance and robustness. In this report, we introduce a
physically simulated software defined distributed data center model while examining opportunity to employ load balancing technique centrally to reduce brown energy consumption. We use OpenStack as the application and management layer along with OpenDaylight controller and Mininet emulator to simulate the model. We benchmark different performance metrics of the distributed system and compare them against standard.
ACKNOWLEDGEMENTS

This research work was undertaken at Leeds Beckett University under the supervision of Dr. Ah Lian Kor and Professor Colin Pattinson. It was financially supported by Erasmus Mundus Joint Master Degree in Pervasive Computing and Communications for Sustainable Development (PERCCOM) [1] and European Commission.

I would like to thank Gavin Weiner and Jamie Childe for providing the network support for the private cloud setup and my supervisors Ah Lian and Colin Pattinson for their kind guidance and support.

Thanks a lot, to the PERCCOM consortium for granting me the opportunity to pursue this one of kind program. I would like to thank Prof. Eric Rondeau, Prof. Jean Phillipe Georges, Prof. Karl Anderson, Prof. Jari Porras along with all the professors and supporting member from UL, LUT, LTU, LBU and ITMO for giving their best for us.

Finally, I would like to thank my family for their constant support. They have been my corner stone in my every achievement. Last two years in PERCCOM, I had to go through a lot. My family and my PERCCOM family were always there.

Syed Asif Iqbal
Lappeenranta, September 15, 2018
# TABLE OF CONTENTS

1 INTRODUCTION ................................................................................................................. 4
  1.1 BACKGROUND ........................................................................................................... 4
  1.2 RESEARCH OBJECTIVES AND QUESTIONS ......................................................... 6
  1.3 SCOPE AND DELIMITATIONS ............................................................................... 6
  1.4 SUSTAINABILITY ASPECTS ............................................................................... 7
  1.5 STRUCTURE OF THE THESIS ............................................................................. 9

2 BACKGROUND AND LITERATURE REVIEW ............................................................... 10
  2.1 SOFTWARE DEFINED DATA CENTER ................................................................. 10
  2.2 SOFTWARE DEFINED NETWORKING (SDN) ...................................................... 11
  2.3 SOFTWARE DEFINED STORAGE (SDSTORE) .................................................. 12
  2.4 SOFTWARE DEFINED SECURITY (SDSec) ....................................................... 12
  2.5 SOFTWARE DEFINED COMPUTE (SDCompute) ............................................... 14
  2.6 ENERGY EFFICIENCY IN SDN BASED DATA CENTER NETWORK ................. 14
  2.7 RIGHT SIZING GEO-DISTRIBUTED DATA CENTERS ................................... 15
  2.8 SDN BASED LOAD BALANCING IN DATA CENTER NETWORK ....................... 16
  2.9 GEOGRAPHICAL LOAD BALANCING IN DISTRIBUTED DATA CENTER ......... 17
  2.10 SUMMARY ....................................................................................................... 17

3 METHODOLOGIES AND SYSTEM IMPLEMENTATION .............................................. 18
  3.1 SIMULATION AND MODELING METHODOLOGY .............................................. 18
  3.2 SYSTEM ARCHITECTURE ............................................................................... 20
  3.3 EXPERIMENTAL SETUP ................................................................................. 24

4 RESULTS AND DISCUSSION ......................................................................................... 28

5 CONCLUSION AND FUTURE WORK .......................................................................... 34

REFERENCES .................................................................................................................. 35

APPENDIX
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>Data Center</td>
</tr>
<tr>
<td>GLB</td>
<td>Geographical Load Balancing</td>
</tr>
<tr>
<td>IaaS</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>ODL</td>
<td>OpenDaylight</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SDDC</td>
<td>Software Defined Data Center</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SDSys</td>
<td>Software Defined Systems</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Networking</td>
</tr>
<tr>
<td>VTN</td>
<td>Virtual Tenant Network</td>
</tr>
<tr>
<td>vBridge</td>
<td>virtual Bridge</td>
</tr>
<tr>
<td>vRouter</td>
<td>virtual Router</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure No</th>
<th>Figure Title</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Distributed Software Defined Green Data Center</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Immediate, enabling and structural effects of Green Software Defined Data Center in Becker's Sustainability Model</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Software Defined Data Center Architecture</td>
<td>10</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Software Defined Networking Architecture</td>
<td>11</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Software Defined Security Architecture</td>
<td>13</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Modeling and Simulation Life cycle</td>
<td>18</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Adaptation of simulation &amp; modeling methodology</td>
<td>19</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Distributed SDCC model (Hub and spoke)</td>
<td>21</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Geographical Load Balancing Period by Period flow diagram</td>
<td>23</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Experimental Setup Architecture</td>
<td>25</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Two Node SDCC Setup</td>
<td>26</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Cluster Monitor Interface</td>
<td>27</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Flow Handling of ODL controller (C11)</td>
<td>29</td>
</tr>
<tr>
<td>Figure 14</td>
<td>RAM utilization of ODL controller (C11)</td>
<td>30</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Normal Distribution of Transferred Data</td>
<td>32</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Normal Distribution of Throughput</td>
<td>33</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

This chapter provides the context of this research work. It describes the scenario of distributed data center, defines energy consumption problem along with stating the motivation and sustainability facet of this research work. The section also features the research objectives and questions and contributions made by the author. The limitations and scopes of the thesis is also discussed. The chapter finishes providing an overall structure of the entire research work.

1.1 Background

Today most computational workload is adopting cloud-based model. The industry has introduced “as-a-service” models like: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), which has accelerated cloud computing adaptation tremendously [2]. Now small enterprises or even individuals can have access to data center infrastructure within few clicks on the computer. This has forced to build large scale distributed data centers around the world. Distributed data centers provide high availability, better responsiveness, high level of flexibility and least probability of bottlenecking the system [3]. Although, there have been attempts to reduce strain on the data center infrastructure at the same time providing faster services with the introduction of edge and fog computing [4], but it has resulted in more distributed approach of handling traffic.

Nowadays, social media, mobile devices, and cloud computing are pushing the traditional network traffic beyond its limit. Global computing power demand from IoT devices, data analysis using Bigdata, high resolution video streaming, video surveillance and a new generation of smart TVs is increasing 20% a year [5]. “There will be 8.4bn connected things in 2017, setting the stage for 20.4bn internet of things devices to be deployed by 2020.” according to internet analyst firm Gartner. More than 65% of enterprises are expected to adopt IoT services by 2020. [6]. In this scenario, 5G technology with its incredible bandwidth capacity and paradigm shifting wireless architecture is emerging. All of this is contributing to create a perfect data storm for the existing communication and data management system. To meet the needs of this data storm, ICT infrastructure like network systems and data centers are emerging like mushrooms all over the world, which
are using vast amount of brown energy (sourced from fossil fuels) resulting in emission of massive amount of CO2 into the atmosphere [7]. Only the communication industry is predicted to use 20% of the total electricity by 2025. That increases CO2 emission caused by ICT industry to reach 3.5% of the global emission by 2020, crossing aviation and shipping industry [5].

For our proposed scenario we consider a collection of mini DC’s spread over a large geographical area connected with each other. Mini DC’s are co-located with renewable energy sources to power them [Figure 1]. The data centers in this model are-

• compact and cost effective
• co-located with stranded renewable energy sources
• has ability to strategically shifting computational load from one small data center to next one to cope with non-availability of intermittent renewable power
• DC’s can be connected as p2p or spoke and hub system

Figure 1: Distributed Software Defined Green Data Center
The primary underlying concept of this research is a duplicative and resilient computing system. SDSys technology gives us global view of the distributed data center system as well as affording more control and flexibility.

1.2 Research objectives and questions

The previous section mentioned the effects of large-scale cloud data centers on energy consumption and CO2 emission. Therefore, the main aim of this research is to propose a novel distributed SDDC model which is capable of leveraging load balancing technique based on energy mix for distributed DC’s. The following objectives will help support the achievement of the aim:

- **Research Objective 1:** Survey available SDDC technologies
  - Research Question 1.1: What existing technologies can be employed for the development of a distributed software defined data center model?

- **Research Objective 2:** Develop a small scale Distributed SDDC model and assess performance of the model
  - Research Question 2.1: How to model a Distributed SDDC?
  - Research Question 2.2: What are the performance metrics?
  - Research Question 2.3: What are the approaches to evaluate performance?

- **Design a distributed SDDC architecture which is capable of reallocate load to reduce non-green energy consumption**
  - Research Question 3.1: What are the available geographical load balancing techniques?
  - Research Question 3.2: How could geographical load balancing reduce non-green energy consumption in Distributed SDDC?

1.3 Scope and delimitations

The contribution of this thesis is to model software defined distributed data center, which will have the capability to shift computational load from one data center to another based on available resource and energy mix. The important aspects of the solution include- use of a distributed system, using software defined networking concept rather than traditional networking and facilitating sustainable system development. As sustainable development
and green ICT is the main agenda of PERCCOM consortium, this research outlines the importance of geographical load balancing for greener data centers.

As we did not have access to actual data center network, the experiments conducted for this research work executed in a small-scale private cloud setup and the controllers used in the SDDC model are running in a virtual machine. Due to lack of customizable load balancing service in current SDN controllers, the implementation GLBPP was not possible.

This thesis aims to shed light on the use of green energy in data center environment and how it can build a pathway towards self-sustainable data center systems. Therefore, a full-scale study of QoS in self-sustainable off-grid data center network will be interesting.

1.4 Sustainability Aspects

Sustainability as a term has been defined and redefined over the years. The word sustainability and development was regarded as conflicting ideas in the mid 20th century. Over the time, the concept of sustainable development emerged as a balance between development and conservation of limited natural resources. According to World Commission on Environment and Development (WCED) report published on 1987, sustainability is “The development that can address the need of present without trading off the need of future generations” [8]. It is a framework, that promotes development in a fair way while avoiding damage to natural resources. The framework is based on three principal pillars – social, environmental and economic. In this research, sustainability is regarded as the core doctrine of the Green SDDC model.

This research is a part of Erasmus Mundus PERCCOM program, which focuses on the sustainability aspects regarding the use of ICT for development. Therefore, to analyze the sustainability impact of this research we are using the Becker’s model defined in the 'Karlskrona Manifesto' [9]. The Baker model is used to examine the sustainability impact of a software system. The model divides the impact of a system into five(5) key dimensions, namely- social, individual, environmental, technical and economic. The social dimension includes the relationship between individuals in a group and how it affects their behavior.
Figure 2: Immediate, enabling and structural effects of Green Software Defined Data Center in Becker’s Sustainability Model.

as a unit. The individual dimension focuses on the individual’s ability to exercise their freedom of expression and rights. The environmental dimension prioritizes the balance of the local ecosystem and reduction of natural resources to control climate change and adverse environmental effects. The technical dimension mainly focuses on the resilience of the system, ease of maintenance and evolution of the system. Finally the economic dimension concerns the capital growth, financial feasibility and liquidity of the wealth.

The effect of a system on the above mentioned dimensions can be categorized into three phases namely- immediate, enabling and structural [10]. The life cycle of the system and direct consequences can be considered as immediate effect, whether the long term effects which facilitates important changes are considered as enabling effects. The structural effects are the recognizable shifts triggered by the system in the bigger picture. The green software defined data center model focuses on minimizing green energy consumption which has immediate positive impact on CO₂ emission and enables reduction of brown
energy consumption. As the structural effect in environmental dimension, Green SDDC reduces global warming. The system reduces costly non-green energy consumption which enables reduction of energy cost, that leads to building self-sufficient data centers. In the technical dimension the system reduces the effort of maintainance using the management layer of SDDC and provide higher level of flexibility and control, which will enable system evolution. As of social area, Green SDDC promotes the ability of choosing green computing, which encourages citizens to change their usage behavior. This can help to create a sustainable community in terms of technology usage behavior. In terms of individual behaviour, the data centers are counted as individual entities. The immediate effect of green SDDC reduces manual operation and maintenance, which enables the data centers to have better operational flexibility.

1.5 Structure of the thesis

This master thesis is organized as follows:

- The Introduction section provides an understanding of the reasons for and necessity of the work described in this thesis.
- In section Related Work, we explore various aspects relevant to the project, including SDDC architecture, SDN based load balancing and geographical load balancing techniques.
- The Methodology section goes into detail aspects of the project. It dives into the details of SDDC architecture and how it is implemented. The section also provides a model system for Geographical Load balancing for distributed SDDC.
- The Results and Discussion section includes description of experiments, screenshot images of the system, benchmarking results and statistical analysis of the results are provided.
- The Conclusion presents a brief summary of what has been presented.
2 BACKGROUND AND LITERATURE REVIEW

This chapter discusses Software Defined Data Center (SDDC) technology and how it is changing traditional distributed data center scenario. Then it explains the core components of SDDC and their roles. This is followed by the investigation of load balancing techniques in distributed data centers and how does SDN fits into it. The chapter also discusses about geographical load balancing and how does it affect the overall energy consumption in data centers. Finally, the chapter ends with a comprehensive review of the available geographical load balancing techniques.

2.1 Software Defined Data Center

Software Defined Data Center (SDDC) [11] is a new paradigm which has been referred to as the next evolution of cloud computing and virtualization. In SDDC all the infrastructure (compute, storage, network and security) is virtualized and provided as-a-service. SDDC is a collection of different software defined systems which are the Software Defined Network (SDN) or management of traffic, Software Defined Storage (SDStore) or management of data, Software Defined Security (SDSec) and Software Defined Compute (SDCompute) or management of workloads. This high layer of abstraction provides extreme flexibility while assigning resources to clients in multi-tenant data center architecture.
2.2 Software Defined Networking (SDN)

Software Defined Networking (SDN) is the next revolution in the networking landscape. SDN comes as a solution to the challenge posed by the traditional network, which is more susceptible to failure due to its multiple disconnected brain design. SDN streamlines the network architecture by detaching control plane from data plane. SDN is defined as “The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices” [12].

From the figure 4 we can see the SDN architecture is divided into three layers. The topmost layer is the application layer which consists of one or more applications, each has control over a set of resources revealed by the lower layers. The second layer is control layer, which is responsible for making decisions how the packets should be forwarded by

![Figure 4: Software Defined Networking Architecture](image)

Figure 4: Software Defined Networking Architecture
the lower layer devices. It contains the control logic for routing, forwarding, clustering etc. Control layer is responsible for topology discovery, path failover procedure and route selection. At the bottom we have data layer, where all the network equipment works. Data layer incorporates the network components that interacts with the users. The separation of control plane and data plane makes network switches just a forwarding device following instructions from the control layer.

SDN capitalizes Flow-based routing, or QoS routing. A flow is a one way stream of related packets that meet the same specification and share the same attributes [13] on the contrary to traditional packet-based or stateless routing, where each packet is treated discretely. Isolating control plane from the data plane provides higher level of control, flexibility and faster decision making while routing them. It mainly depends on a centralized controller or a group of controllers connected to all network devices, which uses southbound communication protocols like OpenFlow, NETCONF or OVSDB.

2.3 Software Defined Storage (SDStore)

SDStore is one of the most significant subsystems in SDDC architecture. SDStore is the storage infrastructure automated and maintained by software. It creates a large storage pool out of heterogeneous resources and dynamically allocates them based on policy to match applications. SDStore handles lakes of data in storage systems by detaching control plane from data storage plane. As the control plane is seperated from the infrastructure plane, SDStore can be deployed on any industry standard servers and disks [14].

SDStore provides unified management, automated policy driven provisioning and self service. The ability to integrate with external interfaces through use of RESTful APIs makes it open and extensible. The key benifits of SDStore are- operational efficiency, agility, cloud optimized, simplified storage environment.

2.4 Software Defined Security (SDSec)

SDSec is a crucial component of the SDDC architecture. Classical network security devices are inherently incompatible with the virtualized data center networks. By design traditional security devices simply can’t protect the components of virtualized
environment, as they are unable to see through the security threats. Therefore, to meet the challenge virtualized security scheme or SDSec is being brought forward. SDSec has the similar architecture like SDN, where data plane is separated from control plane and application layer is used as the management plane. From the figure 5, we can see the SDSec architecture. Network and security devices are part of the physical layer which is connected to control layer by southbound API’s (Like: Openflow, Netconf). The control layer is composed of SDN controller and security controller. The security controller pools packet flow and passes security command to SC agent in SDN controller. The control layer is connected to application layer by northbound API’s.

Figure 5: Software Defined Security Architecture

SDSec provides some vital features comparing to classical security devices like: abstraction, automation, elasticity, visibility, portability [15]. SDSec abstracts the security policies from the hardware devices and enables users to manipulate the policies from management layer. The automation feature ensures hassle free configurations and maintenance of components according to pre-defined policies. SDSec solutions are elastic, therefore, easy to scale up and down. The Cloud Security Alliance (CSA) have brought
Software Defined Perimeter (SDP) [16], which complements SDN while reducing security threats on network applications.

2.5 Software Defined Compute (SDCompute)

Software Defined Compute is the single most important component of the SDDC architecture. In the traditional data center environment computational workload is handled in bottom up approach, which takes substantive amount of time and effort to set-up, configure and optimize. In SDCompute environment the approach is top-down, which brings higher level of agility, flexibility and optimization based to workload profiling. It recognizes the workload requirement, decides on the appropriate components needed in the middleware, allocate them from its heterogenous resource pool.

Agility is achieved in SDCompute environment by offering elasticity and high scalability based on analyzed performance observation. SDCompute provides high level of independence to the data center network as it is built on open source systems like- Openstack [35].

2.6 Energy Efficiency in SDN based Data Center Network

There have been researches to develop solutions to reduce energy consumption of network but network hardware is still quite a far from being energy optimized. In this scenario, it is crucial to consider solutions that can cut down links while maintaining a functional network. Authors in [17 ] proposed a method to reduce upto 41% links during peak traffic of a SDN based network. They have used Shared Path First (SPF) algorithm to aggregate traffic flow and put off the excess links, which results in putting of network devices. They have compared this algorithm to Dijkstra algorithm for power gain. This technique does not cause device oscillation unlike shortest path first in Dijkstra Algorithm. However, they have only considered power gain, which is not so important in data center network where the link distance is quite small.

Authors in [18] proposed a Correlation-Aware Power Optimization Algorithm (CARPO), which combines traffic flows while cutting down extraneous links to reduce energy consumption. CARPO concentrates on reducing DCN energy consumption rather than
server energy consumption. The algorithm is implemented by employing a central power manager and OpenFlow switches. As network hardware account for the major portion of the energy consumed by the network, shutting them down along with unnecessary links is the most effective way to reduce energy consumption. Using the constraints of maximum link capacity and the equality of incoming and outgoing data rate, consolidation procedure tries to minimize energy consumption by turning off switches as much as possible. This approach can be utilized to maximize the power efficiency for Green Distributed SDDC.

2.7 Right Sizing Geo-Distributed Data Centers

Right sizing data center infrastructure capacity for internet scale applications like media streaming, online gaming, social networks etc. has been a challenge. While designing the architecture we must keep in mind about geo-distribution, high-availability, latency along with provision capacity. Latency is a critical factor while serving internet-scale applications [19] [20]. The usual approach is to get close to the users by increasing the number of geo-distributed data centers [21]. As the client demand vary with time and there is little to none probability to get peak traffic at the same time in these DC’s, it becomes quite challenging to optimize the capacity across multiple DC’s. Geo-distributed DC’s also increases the total cloud capacity required.

Larumbe et al. have proposed a Cloud Location and Routing Problem (CLRP), which focuses on mathematically solving the complexity of data center location, software component location and routing [22]. Although CLRP shows, positioning data centers in right position along with intelligent routing theoretically reduces average delay substantially, it doesn’t address routing scenario where control plane and forwarding plane is separated.

Narayanan et al. have proposed a geo-distributed capacity planning framework based on crucial aspects like capacity, latency, heterogeneous application demands and availability [23]. The framework consists of edge and core DC’s for high availability, where edge DC’s are designed to serve nearby users with low latency. In case of failure all the requests are redirected to nearest core DC. The availability SLA is designed to support Tier 2 and
Tier 4 availability standard. This research work focuses on planning the capacity on the time horizon but doesn’t provide any optimization for already running DC’s. Also, it doesn’t consider energy mix and over capacity management of the DC’s.

2.8 SDN Based Load Balancing in Data Center Network

Due to highly distributed nature of cloud data centers, it is a key challenge to schedule task and resource for internet scale applications. The loosely coupled architecture of cloud data centers makes it more complicated. Yong et al. [24] have proposed an SDN-based dynamic Load Balance solution (SDN-LB) with applying the SDN technology to the cloud data center and solve the load balance problem by employing SDN architecture. For this purpose, they have employed plug-n-server [25] as SDN architecture, which minimizes response time by controlling the load on the network and the servers using customized flow routing. The plug-n-server utilizes an integrated optimization algorithm called LOBUS (Load-Balancing over Unstructured networks).

The plug-n-server has three components- Openflow based switching network, SDN controller, underlying program for communication between server and clients. The SDN controller uses four modules: traffic detection, dynamic flow scheduling, load calculation and flow management module. The traffic detection module is deployed to detect traffic dynamically, provide monitoring service and statistics. The load calculation module uses the statistics provided by traffic detection module and provide real time load estimation for the cloud environment. The dynamic flow scheduling module is a hybrid of traditional static load balancing and dynamic load balancing algorithm. The reason for using a hybrid algorithm is to achieve high-performance load balancing while keeping complexity of deployment in cloud environment to minimum. The simulation results reflect better throughput performance than the traditional static load balancing. This method is centralized, therefore comes with a single point of failure issue. Also the centralized architecture causes low scalability, low reliability and limited flexibility.
2.9 Geographical Load Balancing in Distributed Data Center

Researches has shown that, Geographical Load Balancing (GLB) can be used to exploit the variety of Internet-scale service and provide economic sustainability for distributed data center [26] [27]. Although, reducing the cost doesn’t always mean less energy consumption, rather it can be just consuming cheaper energy or consuming high amount in a burst period. However, if data centers are co-located with renewable sources GLB provides an incredible opportunity by leveraging “follow the renewables” routing [28]. The benefits of GLB comes from dynamically adjusting the routing and computing capacity at each location. Although, the benefit comes with a significant “switching cost”, in terms of latency, energy consumption etc. The technique can be counter-productive without the knowledge of future workload, renewable energy availability, and energy prices, as they are unusable beyond the short term. Thus, online algorithms like Receding Horizon Control (RHC) and Averaging Fixed Horizon Control (AFHC) are proposed to predict the traffic pattern, renewable energy availability etc.

2.10 Summary

All these above mentioned DCN energy efficiency mechanisms focus on either, energy efficiency within DCN or proposing algorithm for better predictability of DCN metrics for optimized load balancing within distributed DCN. In my thesis we have made a comparative survey of the available Distributed Data Center models and proposed for a framework to reduce non-renewable energy consumption leveraging SDSys technology.
3 METHODOLOGIES AND SYSTEM IMPLEMENTATION

This chapter discusses simulation & modeling methodology and how it has been used in this research to model Green SDDC. In addition, this chapter also discuss the system architecture and framework of green distributed SDDC model. Finally, the experimental setup is described.

3.1 Simulation and Modeling Methodology

Simulation and modeling allow us to develop a level of understanding of the interaction with a part of the system or system as a whole. Simulation is one of the most potent evaluation tools available for designing a complex system [29]. Simulation and modeling provide a much-needed peek in advance at potential system flaws before it’s actual development, thus allows us to rectify and improve.

Figure 6: Modeling and Simulation Life cycle
For this research we begin by identifying the requirement of the experiment to have a proof of concept for Software Defined Distributed Data Center and thereby developing the scenario. The model must leverage SDSys concept. It is necessary for the model architecture to be distributed and each single entity must be connected to each other as a peer-to-peer connection or connected through a central control system (proposed framework).

As the research is done for academic purpose and without access to real data center environment, we have to simulate the SDDC setup and run experiments. For this purpose, we need to decide on the simulation environment based on its high fidelity to the real-world scenario. Therefore, we created a simulated environment in our small-scale private cloud setup consisted of large number of components simulated in SDN simulator tool. The system components have customization options according to the demands of the experiment. We must decide on the process of collecting data and establishing the performance metrics for the experimental setup. Each experiment is run at least 10 times so that we can get a mean and normalized value of the different dimensions of the performance.

![Figure 7: Adaptation of simulation & modeling methodology](image)

The results are analyzed and compared to the expected values. Based on the comparative analysis we have considered ways to improve performance which could be fed into the
creation of the scenario and restart the loop. Once, when the simulation/emulation result is satisfactory we move on to physical implementation.

### 3.2 System Architecture

We are proposing a framework that can deploy Geographical Load balancing within a logical central point of decision of the Green Software Defined Distributed Data Center Model [Figure 8]. The framework utilizes GLBPP algorithm [4] which has been tested to reduce brown energy consumption up to 42% [30]. The proposed architecture is composed of small scale SDDC units [Figure 4] co-located with renewable energy sources, which are connected to other DC’s through high speed VPN connections. Each DC uses a cluster of SDN controllers for high availability and robustness. In Hub and spoke topology, DC’s are connected to a central hub.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{VM}$</td>
<td>the power draw of a VM in Watt (W)</td>
</tr>
<tr>
<td>$E_{GVM}$</td>
<td>energy consumption of the creation of a VM (in W.h)</td>
</tr>
<tr>
<td>$P_{PR}$</td>
<td>the power draw of a running physical server (in W)</td>
</tr>
<tr>
<td>$P_{PS}$</td>
<td>the power draw of a physical server in sleep mode (in W),</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>time duration</td>
</tr>
<tr>
<td>$E_{PSR}$</td>
<td>energy consumption for a physical server to go from sleep mode to running mode (W.h)</td>
</tr>
<tr>
<td>$P_{i}(R)$</td>
<td>the power draw of all the non-IT infrastructure of DC i, with R requests to process (in W),</td>
</tr>
<tr>
<td>$Et_{i}^{1}(\Delta t, t)$</td>
<td>Total Energy Consumption (TEC) by DC i for the treatment of 1 request of duration $\Delta t$ at time t (in W.h),</td>
</tr>
<tr>
<td>$P_{g_i}^{\delta}(t)$</td>
<td>green power available at DC i, at time t</td>
</tr>
<tr>
<td>$En_{i}^{1}(\Delta t, t)$</td>
<td>Non-Green Energy Consumption (NGEC) by DC i for the treatment of 1 request of duration $\Delta t$ at time t (in W.h),</td>
</tr>
<tr>
<td>$e_{ij}$</td>
<td>energy consumption of sending a request from DC i to DC j (in W.h).</td>
</tr>
<tr>
<td>MR ($\Delta t$, t, i, j)</td>
<td>(for Migration Reward), the non-green energy saved if a request of duration $\Delta t$ is migrated at time t from DC i to DC j</td>
</tr>
</tbody>
</table>

| Table 1: Parameters for GLBPP |

20
Figure 8: Distributed SDDC model (Hub and spoke)
The data center management application stores instantaneous information about the availability of resources (compute, storage, network) along with available energy mix (green and non-green) and the cost of transferring computational load to other connected DC’s [Table-1]. In case of Openstack, the resource availability informations can be monitored and collected using Ironic Inspector API [31] and SystemInfo API [32]. Each DC sends this information to a regional hub, where the information is updated frequently. Each traffic request that comes to any DC goes through GLBPP processing flow [Figure 9], where the DC requests for required information for calculating MR (Migration Reward) to the regional hub for making further decision. Based on the MR value calculated, the traffic is processed or forwarded to next DC.

Computing Total Energy Consumption [TEC] \( [E_t^i(\Delta t, t)] \) by DC i for the treatment of 1 request:

If at least one physical server is running and not fully loaded:

\[
E_t^i(\Delta t, t) = P_{VM} \times \Delta t + E_{CVM} + P_{S}(1) \times \Delta t
\]

# If all the running servers are fully loaded:

If a server is in sleep mode,

\[
E_t^i(\Delta t, t) = P_{VM} \times \Delta t + E_{CVM} + P_{S}(1) \times \Delta t + E_{PSR} + (P_{PR} - P_{PS}) \times \Delta t
\]

If there is no available server, \( E_t^i(\Delta t, t) = \alpha \)

Computing Non-Green Energy Consumption(NGEC) \( E_{n}^i(\Delta t, t) \) by DC i for the treatment of 1 request:

If available green energy is sufficient to process the request, \( E_{n}^i(\Delta t, t) = 0 \)

If the green energy is not enough, \( E_{n}^i(\Delta t, t) = E_t^i(\Delta t, t) - P^G(t) \times \Delta t \)

Migration Reward Calculation:

\[
MR(\Delta t, t, i, j) = E_{n}^i(\Delta t, t) - [e_{ij} + E_{n}^j(\Delta t, t)]
\]

Migration Reward (shifting traffic from i to j) = (non-green energy consumed by DC i) – [(migration cost from i to j) + (non-green energy consumed by DC j)]
When a request (web traffic) arrives at a DC(i) node, the DC management system checks the availability of physical server into the system (whether all servers are fully loaded or not). If a physical server is available, the system checks whether there is any virtual machine (VM) (running web server) available or not. If the VM that are running do not have the capacity to serve the request, a new VM will be created to serve the request or wake up a sleeping VM. In the scenario where a VM is available, the system calculates the cost to serve the request. The cost is calculated in terms of total energy, non-green energy and green energy consumed to serve the request. Based on these values and the

![Figure 9: Geographical Load Balancing Period by Period flow diagram](image)

When a request (web traffic) arrives at a DC(i) node, the DC management system checks the availability of physical server into the system (whether all servers are fully loaded or not). If a physical server is available, the system checks whether there is any virtual machine (VM) (running web server) available or not. If the VM that are running do not have the capacity to serve the request, a new VM will be created to serve the request or wake up a sleeping VM. In the scenario where a VM is available, the system calculates the cost to serve the request. The cost is calculated in terms of total energy, non-green energy and green energy consumed to serve the request. Based on these values and the
information (available green energy at different nodes, load transfer cost to different nodes) collected from the central hub, Migration Reward (MR) is calculated for all connected DCs. If the MR value is positive for any possible DC node, the web traffic is transferred for processing to the DC(j) (DC with highest MR), otherwise the request is processed at DC(i).

The load migration algorithm focuses on green energy availability. Although, in case of shifting load to a distant DC where the green energy availability is minimum (but available), can be counter-productive. According to Carlinet and Perrot [33], the algorithm reduces maximum brown energy when the green energy covers 78% of the required energy. Moreover, ability to predict traffic brings the scope to improve system performance drastically.

3.3 Experimental Setup

The main goal of this experiment is to explore a working software defined data center model benchmark the system parameters and deploy basic load balancing within the system. For this purpose we setup a three node Openstack private cloud environment, where the SDN network services are controlled by Opendaylight controller and in the application layer load balancing is facilitated by LbaaS.

This study relies on widely implemented Google Fat tree [34] topology to emulate datacenter units. This specific topology has been selected due to its high bisection bandwidth, rapid network deployment and performance scalability to keep up with the agile nature of applications running in the data centers [34]. Fat tree topology is fixed tier and deterministic therefore better with complex routing algorithms. This topology is switch centric and indirect as a result, only SDN controllers forward the flow to the switches and switches are connected to hosts.
We use OpenStack Newton to setup a cloud infrastructure. OpenStack is an opensource cloud operating system [35] which comprises a set of tools (which provides an array of services) that controls large pools of compute, storage, and networking resources in a datacenter scenario. Large pool of resources are managed through a dashboard that gives administrators control over the whole system. The system can configured as a multi-tenant system and users can self provision resources through a web interface.

OpenStack abstracts compute, storage, network, security and provides them as a service through different API’s. Our OpenStack setup is a 4-node setup where 1 controller node and 3 compute nodes. We have Nova, Glance, Cinder, Keystone, Horizon, Neutron and LBaaS services running. For the purpose of network virtualization services of OpenStack, we use OpenDaylight (Nitrogen SR1). OpenDaylight supports Neutron API of OpenStack via networking-odl driver. For the purpose of emulating infrastructure layer of the SDN architecture we use Mininet [36], Open vSwitch and Zodiac FX [37] SDN switches. For this experiment we have setup clusters of OpenDaylight controller.
A single data center topology is emulated in a single instance of Mininet and connected to a cluster of 3(three) OpenDaylight controllers. We consider this setup as unit DC in our architecture [Figure 11]. Clustering is done to achieve redundancy and robustness within the system. We deployed the cluster using the Akka framework, which is a high availability model used to build message driven applications on Java Virtual Machine (JVM) environment for distributed systems. The main reasons for choosing Akka framework is, it provides a simple yet powerful solution to handle concurrent messaging between actors (in our case nodes) and it allows to create systems comprised of multiple JVM’s. We use the components of Akka framework, namely- remoting, clustering and persitance [38].

For our Opendaylight setup Akka remoting is used for peer-to-peer communication between nodes. It provides the basic functionalities required for Akka clustering. In Akka remoting the actors in the JVM does not form cluster togger.

Akka clustering is basically an ensemble of JVM’s , each individually running as an actor system while configured as a cluster provider. Although multiple actor systems can run in a single JVM. Akka cluster is an exceptional way for setting up a reactive system. Akka clustering design is responsive, resilient, elastic and message driven. Therefore, it
satisfies the fundamental properties of a reactive system. It involves nodes, clusters and leaders, where nodes are members of the cluster. Where, cluster is a set of nodes joined though the cluster service. Each cluster elects a node as the leader, who acts as a leader. This way Akka clustering provides fault tolerant p2p based cluster service.

Akka persistance synchronizes the stateful actors running in different JVMs, after a crash or triggered by a supervisor. Here the key principle is that, it never alters the current state rather the internal states are persisted. Akka persistance allows to add redundancy to endure failure, load balances over multiple servers provide better responsiveness.

To achieve fault tolerance by deploying OpenDaylight clustering technology, we needed at least three controllers in a cluster.

![Cluster Monitor Interface](image)

Figure 12: Cluster Monitor Interface

The cluster is deployed such a way that, one of them is working as a leader node. Leader nodes are used to connect to leader nodes from other clusters. We use Virtual Tenant Network (VTN) to connect two DC unit using virtual nodes (vBridge, vRouters) and virtual interfaces and links where one controller is setup as VTN manager and the other one as VTN coordinator [39].

We run the topology script in Mininet connected to the cluster and measure throughput, latency from one host to another using iPerf [40] and Wcbench [41] tools. The OpenDaylight controller is stress tested for handling large amount of flows. The bandwidth test is conducted 10 times in a loop each to get normalized result.
4 RESULTS AND DISCUSSION

We evaluate the performance of our Distributed SDDC setup in terms of Bandwidth, RAM utilization of SDN controller and flow handling limit under stress. For our test purpose we predominantly used the controller node and 1 compute node of the 4-node setup. The controller node is running on Intel Xeon ES2620 -24 core CPU with 70GB of RAM and the compute node is running on Intel Core i7 3779 CPU with 16 GB of RAM. The ODL controllers (2 cluster of 3) are running as VM in the controller node on KVM Hypervisor with 4 CPU cores and 6GB of RAM and Ubuntu 14.04. Two Mininet nodes are running in the compute node with 2 CPU cores and 4 GB of RAM each.

<table>
<thead>
<tr>
<th>Flow Handling</th>
<th>Ram Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Number</td>
<td>Flows/Sec</td>
</tr>
<tr>
<td>1</td>
<td>5222</td>
</tr>
<tr>
<td>2</td>
<td>5222</td>
</tr>
<tr>
<td>3</td>
<td>5109</td>
</tr>
<tr>
<td>4</td>
<td>5131</td>
</tr>
<tr>
<td>5</td>
<td>5152</td>
</tr>
<tr>
<td>6</td>
<td>5132</td>
</tr>
<tr>
<td>7</td>
<td>4132</td>
</tr>
<tr>
<td>8</td>
<td>5149</td>
</tr>
<tr>
<td>9</td>
<td>4141</td>
</tr>
<tr>
<td>10</td>
<td>5210</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1691</td>
<td>5244</td>
</tr>
<tr>
<td>1692</td>
<td>5239</td>
</tr>
<tr>
<td>1693</td>
<td>5227</td>
</tr>
<tr>
<td>1694</td>
<td>5102</td>
</tr>
<tr>
<td>1695</td>
<td>5173</td>
</tr>
<tr>
<td>1696</td>
<td>5184</td>
</tr>
<tr>
<td>1697</td>
<td>5207</td>
</tr>
<tr>
<td>1698</td>
<td>3245</td>
</tr>
<tr>
<td>1699</td>
<td>4763</td>
</tr>
<tr>
<td>1700</td>
<td>5195</td>
</tr>
</tbody>
</table>

Table 2: SDN Controller (C11) benchmark results

We begin our experiment by benchmarking the ODL controllers. We name the controller C11(cluster1, controller1) and run Webench against it in a loop of 1700 times. We collect the RAM utilization and flow handling data using the script [41] and plot them against run times. We noted that the number of flows handled by the controller during the test remained almost in a specific region (4950-5100 flows/second) [see Table 2]. Although, there were some outliers depicted in Figure 13 below, we can assume the limit the
controllers can reach is around 5190 at this configuration. We ran the same experiment for other controllers (C12, C21 and C22) in the setup and got similar results.

![Flow Handling of ODL controller (C11)](image)

We also collect RAM utilization data during the stress test over 1700 times. We noticed that, the utilization of RAM by the controller (C11) increases at a linear rate over the experiment. It starts from 74% in the idle state to reach maximum 93% of the allocated RAM [see Figure 14] for the VM over the test running period. We have run the same test on each controller of the cluster (C12 and C13) and observed pretty much similar pattern of performance. We have noticed some sudden spike and dip of utilization, but overall the pattern remains same.

For our experiment we have setup two DC units (emulated topology with a cluster of controller). Two units are connected by Virtual Tenant Network (VTN) using OpenStack Neutron service. We have tested connectivity from one network to another and measured latency. The latency is similar to the one within the same network. We realized the reason is, the VM’s are hosted in the same machine, therefore the latency difference is negligible.
Initially we tested the throughput within the hosts of the system using iPerf. The results varied on each attempt and the average BW was 20.96 Gb/sec. We realized it can be improved by adopting load balancing into the network. For the load balancing within the networks we have adopted Dijkstra algorithm to explore different paths to a destination and based on the path, we adopt least load method to achieve load balancing. Then, we measured the effective bandwidth from one host to another after deploying load balancing rule. We ran the experiment for 10 times each and took average of the throughput from host to host.

From the Table 2, we can observe that, there is substantial change in bandwidth after load balancing. We achieved around 33% improvement [see Table 3] of the throughput using the load balancing on an average.
<table>
<thead>
<tr>
<th>Before Load Balancing</th>
<th>After Load Balancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transferred Data(GB)</td>
<td>Throughput(Gb/s)</td>
</tr>
<tr>
<td>14.57</td>
<td>12.03</td>
</tr>
<tr>
<td>16.40</td>
<td>14.20</td>
</tr>
<tr>
<td>20.77</td>
<td>17.33</td>
</tr>
<tr>
<td>21.17</td>
<td>17.63</td>
</tr>
<tr>
<td>22.60</td>
<td>19.50</td>
</tr>
<tr>
<td>23.00</td>
<td>19.80</td>
</tr>
<tr>
<td>23.47</td>
<td>19.63</td>
</tr>
<tr>
<td>25.30</td>
<td>21.80</td>
</tr>
<tr>
<td>38.67</td>
<td>32.73</td>
</tr>
<tr>
<td>40.50</td>
<td>34.90</td>
</tr>
</tbody>
</table>

### Table 3: Effect of Load balancing on bandwidth

We analyze the throughput and transferred data statistics using gaussian distribution. To normalize the data we have calculated mean ($\mu$) and standard deviation ($\sigma$) of the results. We use the normal distribution formula to get the normalized value of the experiment results.

$$Z = \frac{X - \mu}{\sigma}$$

Here, $Z$ = normalized value; $X$ = value being normalized; $\mu$ = mean of the values; $\sigma$ = standard deviation

<table>
<thead>
<tr>
<th>Transferred Data (GB)</th>
<th>Normalized Value</th>
<th>Befor Loadbalancing</th>
<th>Transferred Data (GB)</th>
<th>Normalized Value</th>
<th>After Loadbalancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.57</td>
<td>0.02</td>
<td></td>
<td>13.28</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>16.40</td>
<td>0.03</td>
<td></td>
<td>17.00</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>20.77</td>
<td>0.04</td>
<td></td>
<td>24.38</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>21.17</td>
<td>0.04</td>
<td></td>
<td>28.10</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>22.60</td>
<td>0.05</td>
<td></td>
<td>34.98</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>23.00</td>
<td>0.05</td>
<td></td>
<td>37.28</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>23.47</td>
<td>0.05</td>
<td></td>
<td>37.58</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>25.30</td>
<td>0.05</td>
<td></td>
<td>38.70</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>38.67</td>
<td>0.01</td>
<td></td>
<td>41.16</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>40.50</td>
<td>0.01</td>
<td></td>
<td>41.30</td>
<td>0.02</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4: Normalized value of transferred data results using Gaussian distribution.
We can observe from [figure 15] the gaussian distribution of the transferred data before and after load balancing, that there is substantial change in the performance of the system. The standard deviation of the values have reduced noticeably. The mean value improved from 24.65 GB to 31.38 GB.

![Gaussian Distribution of Transferred Data](image.png)

**Figure 15: Normal Distribution of Transferred Data**

<table>
<thead>
<tr>
<th>Befor Loadbalancing</th>
<th>After Loadbalancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput(GB/s)</td>
<td>Normalized Value</td>
</tr>
<tr>
<td>12.03</td>
<td>0.03</td>
</tr>
<tr>
<td>14.20</td>
<td>0.04</td>
</tr>
<tr>
<td>17.33</td>
<td>0.05</td>
</tr>
<tr>
<td>17.63</td>
<td>0.05</td>
</tr>
<tr>
<td>19.50</td>
<td>0.05</td>
</tr>
<tr>
<td>19.63</td>
<td>0.05</td>
</tr>
<tr>
<td>19.80</td>
<td>0.05</td>
</tr>
<tr>
<td>21.80</td>
<td>0.05</td>
</tr>
<tr>
<td>32.73</td>
<td>0.02</td>
</tr>
<tr>
<td>34.90</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Table 5: Normalized values of throughput results using Gaussian distribution.**
Figure 16: Normal Distribution of Throughput

From the Gaussian Distribution of Throughput graph [figure 16] we can see that, large percentage of the results after implementation of load balancing has shifted towards right and the average throughput has increased from 20.96 Gb/s to 27.87 Gb/s.
5 CONCLUSION AND FUTURE WORK

In this thesis work, we have surveyed the state of the art of SDDC and implemented a proof of concept in a small-scale setup and evaluated different performance metrics. The numerical results show that, the least-load based, load balancing can improve the throughput up to 33%. Along with that, we have proposed a SDDC model which has the potential to reduce non-renewable energy consumption substantially.

While designing a distributed data center network traffic forwarding delay due to computational load shifting is a critical factor. The forwarding delay plays an vital role in terms of saving energy cost and ensuring QoS. As the traffic load changes in the network over the time, the delay also changes. Therefore, it is not possible to estimate the delay theoretically accurate enough, before sending the traffic. Further researches in this area capitalizing real data center network can give us more insight and help us to improve traffic routing policy in case of load shifting.

The proposed system needs a central point of control with the knowledge of all the connected DC’-s. At this moment, SDN controllers have the visibility of the connected systems but doesn’t offer customizability to deploy the proposed algorithm for geographical load balancing. Also developing a “Super controller” to preside over local controllers or to manage them from a higher level is also a crucial scope of work. The topology considered here is simple and deterministic, developing a more robust load balancing algorithm which can cope with non-deterministic and relatively complex topologies can be a future scope of this research.

Although, Geographical load balancing can reduce the overall non-renewable energy consumption to a substantial amount, it is important to have local energy optimization algorithms like: CARPO [17] or ElasticTree [42] to cut down overall energy consumption by a huge margin which can be done in the future along with QoS implementation.
REFERENCES

1. An international Master's program in green ICT as a contribution to sustainable development, A Klimova, E Rondeau, K Andersson, J Porras, A Rybin, A Zaslavsky, Journal of Cleaner Production 135, 223-239


3. Cisco Data Center Networking: Enterprise Distributed Data Centers Solutions Reference Network Design March, 2003


F. Larumbe and B. Sans’o. Optimal location of data centers and software


APPENDIX 1. Network Configuration for Openstack Setup

Interfaces:

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eno1
iface eno1 inet static
address 192.168.1.3
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
dns-nameservers 8.8.8.8

auto eno2
iface eno2 inet manual
up ifconfig $IFACE 0.0.0.0 up
up ip link set $IFACE promisc on
down ip link set $IFACE promisc off
down ifconfig $IFACE down

auto br-provider
iface br-provider inet static
address 10.0.0.2
netmask 255.255.255.0
gateway 10.0.0.6
dns-nameservers 8.8.8.8
APPENDIX 2. Screenshots of the Openstack cloud setup

Compute Node List:

Segmented Network in Openstack
Network Agent List

Running Service List