CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

# A NAT Traversal Method for Interconnecting Networks in an SDN Environment

by

Shahila Sadiq

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2020

*I dedicate my thesis to My Parents  Brother Tariq Mehmood Jentsch. I have a special feeling of gratitude for my beloved parents, siblings and friends. Special thanks to my supervisor whose uncountable confidence enabled me to reach this milestone.*

**CERTIFICATE OF APPROVAL**

**A NAT Traversal Method for Interconnecting Networks in an SDN Environment**

by

Shahila Sadiq

(MCS181020)

**THESIS EXAMINING COMMITTEE**

| S. No. | Examiner | Name | Organization |
|---|---|---|---|
| (a) | External Examiner | Dr. Muhammad Yousaf | Riphah IU, Islamabad |
| (b) | Internal Examiner | Dr. Nadeem Anjum | CUST, Islamabad |
| (c) | Supervisor | Dr. Amir Qayyum | CUST, Islamabad |

Dr. Amir Qayyum
Thesis Supervisor
December, 2020

| | |
|---|---|
| Dr. Nayyer Masood | Dr. Muhammad Abdul Qadir |
| Head | Dean |
| Dept. of Computer Science | Faculty of Computing |
| December, 2020 | December, 2020 |

# *Author's Declaration*

I, **Shahila Sadiq** hereby state that my MS thesis titled "**A NAT Traversal Method for Interconnecting Networks in an SDN Environment**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

**(Shahila Sadiq)**

Registration No:MCS181020

# *Plagiarism Undertaking*

**(Shahila Sadiq)**

Registration No:MCS181020

# *Acknowledgements*

In the name of Allah, the Most Gracious and the Most Merciful Alhamdulillah, all praises to Creator and Sustainer of the universe for the strengths and His blessing in completing this thesis.

I would like to express my sincerest appreciation to my enthusiastic supervisor, **Dr. Amir Qayyum** for his supervision, assistance, and immense knowledge. I am sincerely thankful to him for his constant support, motivation, and patience.

I am grateful to all members of CORENET group especially Dr. Mudassir Tufail, Dr. Naveed and Sir Umar for their kind help and guidance during my research work.

My deepest gratitude goes to my beloved parents and siblings for tolerating my mood swings and being patient with me. A very special thanks to my brother Tariq Mehmood Jentsch for believing in me, for giving me the determination to overcome many trying moments to pursue my dreams. Heartily thanks to my parents for their love and moral support. Sincere thanks to my sisters and brother for their support, kindness and moral support during my study.

A true friend is one who thinks you are a good egg even if you are half-cracked. A very special thanks to my friend Asia Shahab who supported and encouraged me when I was about to give up.

Last but not the least and without hesitation, I would like to thank tea, for helping me survive all the stress.

**(Shahila Sadiq)**

Registration No:MCS181020

# *Abstract*

Emergence of technology, applications and services demands for new networking requirements.With the rise of social media, mobile, cloud and big data, the demand for ubiquitous accessibility, dynamic management, high bandwidth and IPV4 addresses have increased. Assigning unique IP addresses to growing number of network devices is impossible from IPV4 address range because it was not imaginable at the time of designing IP that internet will grow to such huggable size. Network Address Translation (NAT) is designed for conserving IP address and to solve the problem of IP exhaustion. As networking paradigm is moving to Software Defined Networking (SDN) now so the need of developing application for SDN is increasing. SDN has changed the way of designing and managing the networks. It emphases on improving the limitations of conventional network and decouple the forwarding or data plane from control plane making the management process easy. This decoupling feature makes the network directly programmable by keeping the basic infrastructure abstracted for services and applications.

Different techniques exist for developing application on SDN but most of them use NAT gateways for IP conversion adds workload and delay on these gateways. In this study, we aim to focus on this underlying feature of SDN for developing application at software level. In this thesis, we propose a new approach for interconnecting networks where there is no need to have unique IP addresses and controller handles the process of address translation. Our proposed solution is based on using Software Defined Networking technology to help networks to interconnect with each other as per their need. If an organization is looking to consume a service in much flexible fashion with high availability and best possible rate without worrying about whether that service is coming one network or other then the proposed solution alleviates the problem of establishing connectivity by automating the process. Controller is the main entity in this paradigm which acts as brain of network and takes decision about packets. The controller used in this thesis is Floodlight controller because it has centralized architecture with modularity and a lot of developer's community support. The proposed approach implements NAT registry service on controller which is maintained centrally. So instead of organizations to maintain registry for keeping unique IPs and for communicating with others this task is done centrally using controller and existing technologies i.e. private networks and private networking schemes. With the help of this NAT registry, the process of IP conversion is automated. Results have shown that by automating the process of IP conversion using controller, the need for processing on hardware (at NAT gateways) has mitigated. Organizations can now get services locally or globally using their private networking schemes and with this IP converting NAT registry on controller, no two organizations have same IP on network.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **API** | Application Program Interface |
| **BGP** | Border Gateway Protocol |
| **COB** | Continuity of Business |
| **Co-Lo** | Co-Location |
| **CSPs** | Cloud Service Providers |
| **IP** | Internet Protocol |
| **IPV4** | Internet Protocol version 4 |
| **L3** | Layer Three |
| **MAC** | Medium Access Control |
| **MPLS** | Multiprotocol Label Switching |
| **NAT** | Network Address Translation |
| **NBI** | Northbound Interface |
| **NFV** | Network Function Virtualization |
| **ODL** | Open Daylight controller |
| **OF** | OpenFlow |
| **QoS** | Quality of Service |
| **RTT** | Round Trip Time |
| **SBI** | Southbound Interface |
| **SDN** | Software Defined Networking |
| **SD-WAN** | Software Defined Wide Area Network |
| **TCP** | Transmission Control Protocol |
| **VPN** | Virtual Private Network |

# Chapter 1

# Introduction

With the rise of the World Wide Web (WWW), the prime illustration of Internet moved away from the educational community to user oriented use case. Internet was first used in the educational community for the file transfer and email purpose only but when Internet start growing in the last era of 20th century, the need for new and innovative protocols rise [1].

With the emergence of the social media, mobile, cloud and big data, the demand for ubiquitous accessibility, dynamic management, high bandwidth and Internet Protocol version 4 (IPV4) addresses have increased [2].

Different type of Internet of Things (IoT) devices have also invented with the development of Internet and IoT technology. Wang et al. [3] stated that Internet Protocol (IP) address configuration is needed by all the devices for connecting to the Internet. Because of increasing number of devices IP addresses are lacking gradually in Internet Protocol version 4 (IPV4) network environment. Besides new type of applications developed for the end users generate their own features that need to be provided by Internet [4].

It was not imaginable at the time of designing IP that internet will grow to such huggable size. Assigning unique IP addresses to growing number of network devices is impossible from IPV4 address range [3].

IPV6 was developed to overcome the address problem that was faced in IPV4 but there was no built-in backwards compatibility with IPv4 which was hindering the communication of IPV6 and IPV4 networks [6].

With increasing number of devices different applications have also developed with different requirements for making delivery successful over internet for example applications like VOIP are sensitive to delay whereas applications like video conferencing require certain bandwidth for their flow.

Conventional networking works on the basis of decentralization because every device working in conventional networking has its own control plane. If there are changes to be made according to any specific applications, then every network devices device has to be configured manually [4].

It is quite obvious that the changing application's requirements demand for the network to be flexible and scalable and new IoT devices need more unique addresses to connect with internet.

Network Address Translation (NAT) is designed for conserving IP address and to translate IP between public and private IP minimizing the need for registering IPs or having unique IPs [3].

An SDN-based NAT is mostly implemented at Wi-Fi access point and problem of Internet Protocol (IP) exhaustion can be solved temporarily but still Peer-to-Peer (P2P) session cannot be established between devices located in different private networks [5].

New requirements demanded by the applications can be fulfilled by making the centrally controlled network. Software Defined Networking (SDN) can be considered in this regard as it has changed the way of designing and managing the networks. It emphases on improving the limitations of conventional network and decouple the forwarding or data plane from control plane making the management process easy [57]. This decoupling feature makes the network directly programmable by keeping the basic infrastructure abstracted for services and applications [7] as illustrated in Figure 1.1 [8].

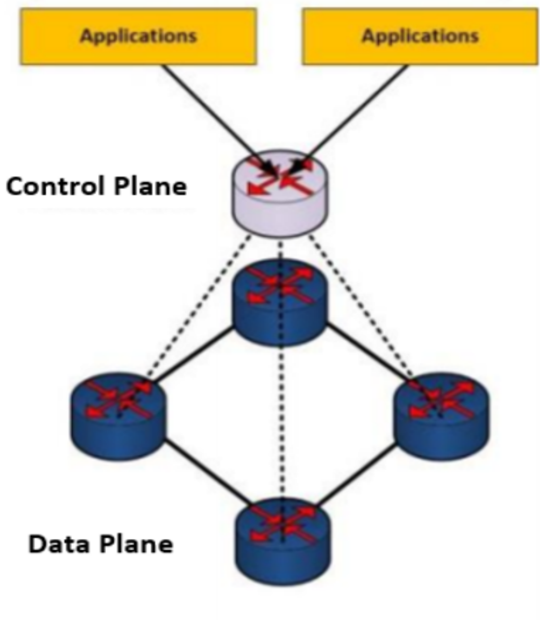


Figure 1.1: Architecture of SDN

The control plane decides all the policies and rules related to the packet that where and how the packet will be forwarded and data plane handles the packet according to the rules dictated by the control plane [1]. The control logic is centralized and the controller, working as central entity enables network programmability which automates the network manageability through programs. Network traffic can be adjusted dynamically because controller has a global view of the network. Control on forwarding devices is achieved through OPenFlow protocol which enables the communication between controller and forwarding devices [10]. Data plane devices can be designed in more general way because they are separated from control plane. In other words it can be said that these devices can become unintelligent and simple boxes with a lone responsibility of handling and forwarding the packets as per the rules defined by control plane [1].

In recent times there has been an increased concentration in SDN for wide-area network scenarios called as SD-WAN. The aim is to let network administrator centrally control the network without changing hardware. Traditional networking can be improved by defining network policies and developing applications i.e. NAT in a centralized way. It also allows quick optimization and management of network resources through dynamic and automated SDN program [3]. By developing applications like NAT on SDN, the problem of increasing networking devices and new requirements can be diminished.

## 1.1 Traditional NAT Traversal Schemes

Different type of NAT traversal schemes exist that support P2P communication. Some of these schemes are Relaying, Connection Reversal and hole Punching [13]. In relaying scheme, every packet is forwarded by a relay server located in public network. P2P communication can be supported by this scheme stably, but causes workload on relay servers and adds transmission delay [56]. In connection reversal scheme, one device locates private network and another device locates public network. In this way packets do not have to move through relay server rather packets move through the device that uses public IP. It has advantage over relay scheme but at the same time flexibility is degraded by public IP address limitation. In hole punching scheme, different devices trace several private networks and then P2P communication is performed. Fig 1.2 represents the traditional hole-punching scheme [5].

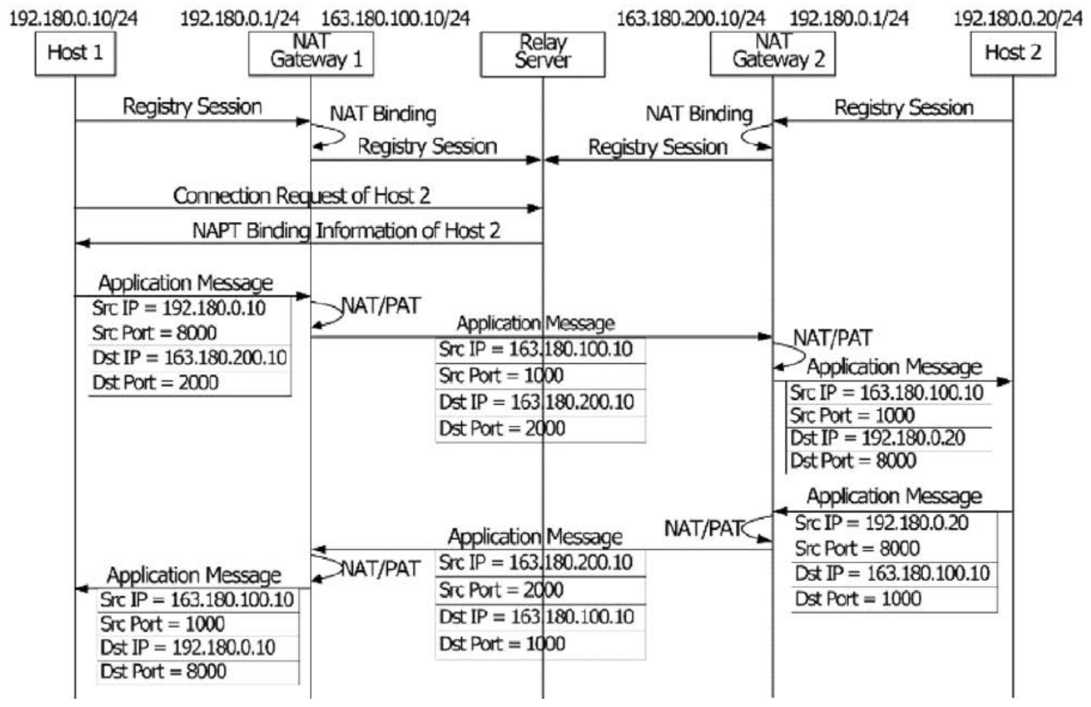


FIGURE 1.2: Traditional P2P Communication Based on Hole Punching

Devices exchange their own NAT binding information to relay server and procuring the NAT binding information of another device that wants to do P2P communication. As private network have many devices so network performance is degraded by NAT traversal schemes because of packet modification and NAT processing [5].

## 1.2 SDN based NAT Traversal Scheme

As SDN provides central control over network, this property can be used for improving traditional NAT traversal method. As applications are implemented on SDN, so NAT application can also be developed on SDN. The controller in SDN can handle the mapping from private to public network in a central place rather than collecting from every data plane.

As controller has global view of whole network topology, so the information of changing network is provided to control plane periodically. Appropriate flow entries can be configured on Openflow switches according to the rules dictated by the control plane [3].

As whole process is handled by the controller, so it is required by the hosts to be notified as in case of hole-punching.

## 1.3 Terminologies about SDN

In this section, we will define some common terms that are used in SDN. The definitions of the terms provide an easy understanding and overview of the Software Defined Networking (SDN).

### 1.3.1 Control Plane

The part of network which carries information about making necessary decisions for establishing and controlling the network is called control plane [14]. It can also be said that control plane is the main place where routers and switches make decision about packet that where to forward this packet keeping in view the global view of network [15]. It is also called as brain of the router.

### 1.3.2 Data Plane

Another name for the data plane is forwarding plane. Data plane have routers and switches used for forwarding the packet. In SDN these are not concerned with making decision. Their only purpose is to forward packet according to rules given by control plane [16].

### 1.3.3 Controller

The main entity working in the control plane of Software Defined Networking (SDN) is controller. For deploying intelligent network it manages flow control through Southbound API below and business and application logic through Northbound API above. Controller usually contains different modules that perform different tasks [17].

### 1.3.4 OpenFlow

OpenFlow is the communication protocol in SDN. Because of this protocol SDN controller can connect directly with the forwarding plane devices i.e. router and switches of network. Devices must support OpenFlow protocol for communicating with controller if they are working in an OpenFlow Environment. Controller can inject rules in network devices using this protocol [18].

### 1.3.5 Southbound Interface

Southbound API devices that how the controller should communicate with its data plane. Controller communicates with its data plane for making changes in the network. This communication is made possible through the OpenFLow protocol which is connected to the southbound interface controller [19].

### 1.3.6 Northbound Interface

The communication of controller with application plane is made possible through its northbound API. Firewall, security, virtual gateways and load balancing are different applications and services. Application developers can easily connect with network and can make changes according to applications without knowing inner working of network. This interaction is made possible with help of northbound interface [19].

### 1.3.7 RESTful API

RESTful API or REST API is the abbreviation of the representational state transfer (REST) application program interface (API). It is frequently used for the communication in the web service development. It uses less bandwidth so it is preferred for the less internet usage. It is used for the building APIs and this way user can connect with the cloud services. HTTP request is used for the different actions [20].

## 1.4 Problem Statement

Existing literature defines different types of techniques for IP conversion. In those studies, Network Address Translation (NAT) is performed at NAT gateway or Wi-Fi access points. Different studies have also proposed that how applications can be developed for SDN environment, but even in such network NAT gateways are used for converting IPs adding workload on these gateways. Besides, some approaches have considered including extra physical hardware in SDN for address translation purpose causing delay and hindering real time transmission.

## 1.5 Research Questions

Based on our problem described above we have identified following research questions:

**RQ 1:** How organizations can be interconnected using their private IPs and without collision due to same addresses?

NAT registry application is implemented on controller which establishes and control network in a central way. Hence changes on network and address translation can easily be handled by controller which ensures that no two devices have same address on network.

**RQ 2:** Which SDN controller can perform best when interconnecting organizations? What could be parameters for comparing and selecting controller?

Through in depth study of different controllers that are used in SDN environment, parameters are formulated for comparing and selecting controller.

**RQ 3:** How service availability and disaster management can be assured in case of particular service provider goes down?

Our approach focuses on automating the process of link convergence in case particular service provider goes down. Multipath routing is used for accessing service from the alternative path.

## 1.6 Research Methodology

1. In the first phase, we did literature review to find the common techniques that are relevant to connecting clouds and organizations and converting their IPs. After studying different techniques we concluded that NAT gateways are used for converting IPs. As networking paradigm is moving to SDN which is based on centrally controlling the network still NAT gateways are used for converting IPs adding work load and causing delay.

2. In order to overcome the gap in existing approaches, we have proposed implementing NAT registry application on SDN controller which reduces delay and workload on NAT gateways and supports communication between hosts in private networks. .

3. NAT application is implemented after studying details about the controller's documentation that how a new module can be implemented and integrated with the existing modules and how application installs rules on OpenFlow switches.

4. Steps performed in our approach are discussed below:

   (a) Firstly, we selected Floodlight controller for our implementation because it was easy to implement any new module and integrate it with existing ones.

   (b) Next we implemented NAT registry application for controller.

   (c) As NAT works with Layer3 routing application, so we used existing application of Layer3 Routing Application of floodlight controller and integrated with our application.

   (d) For checking the correctness of implemented application, we emulated the topology using emulator Mininet

      i. We wrote a python script for topology with 2 hosts, three switches and one controller.

      ii. For checking the connectivity between hosts, Pingall command was used.

      iii. After initiating communication between hosts, Wireshark was used for finding whether IPs were converted correctly or not.

      iv. Afterwards, Iperf command was used for finding jitter and packet out of order % for different values of Bandwidth and Time Interval. These parameters are also used for comparison.

   (e) Multipath routing is also used for converging traffic in case particular link goes down.

5. After performing all these steps, we have compared our results with dataset of NAPT [8].

## 1.7 Research Contribution

In this research work, we have proposed a technique of implementing NAT application on controller. Global view of network topology is maintained by controller in SDN and after implementing NAT service, translation process of IPs for different devices is handled centrally. Workload from network devices is reduced by central NAT which distributes workload among multiple devices. Parameters have also formulated for finding the suitable controller for any scenario i.e. architecture of controller, how easily a new module can be implemented and integrated with existing modules, language supported by controller, community support for controller that how many people are using that controller and whether it can be used for research purpose or not. Multipath routing is used for automating the process of link convergence. By this process, traffic can be converted to alternative path in case one link goes down.

## 1.8 Thesis Structure

The Thesis is organized in five chapters.

- After introduction chapter, chapter 2 presents the literature review related to SDN and its controllers.
- Chapter 3 presents methodology and implementation details.
- Chapter 4 presents details about testing environment and results of the work.
- Chapter 5 finishes up the entire work and gives future research directions.

# Chapter 2

# Literature Review

In order to understand the context of this thesis, it is important to understand Software-Defined Networking paradigm in a better way. The main objective of this section is to assess the available literature relating to Software-Defined Networking and how this approach can be effective for controlling network and developing applications in a centralized way. The literature review is presented in multiple sections starting with the explanation SDN and applications development in SDN in following section 2.1.

Section 2.2 explains the controllers that are used in SDN.

Section 2.3 concludes and summaries the chapter.

## 2.1 SDN and Applications Development on SDN

The conventional WAN technologies used for connecting Data Centers have some issues and this infrastructure is not a good solution for online services like Microsoft, Google and Amazon. The devices used for interconnection are not flexible enough to deal with different types of data packets and dynamically executes the changes occurring in paths [21]. WAN needs to be intelligent enough that it can adapt changes according to the state of network, at any time so that, it can utilize links and nodes fully. The solution to handle this problem is the separation of data handling module from forwarding module. So the control plane is extracted from software and implemented as software which enables a centralized control over the network and increase resource optimization and efficiency. This idea is the base of new paradigm called as Software Defined Networking (SDN). SDN was first deployed by Nicria in 2010 with NTT and Google as co-developers [22].

The emerging paradigm SDN separates the control plane from data plane making the switches simple forwarding devices and whereas logic is implemented centrally in control plane.It is changing the way of designing and managing the network. The forwarding plane works according to the rules dictated by the control plane. Two main characteristics of SDN are: separation of control plane and data plane which allows the network to be adaptable, cost-effective, dynamic, software programmable and easily manageable. This separation between the control and data plane allows for data plane devices to be designed in a generic and simplified way. Secondly, a single software control program controls multiple data plane elements because an SDN combines the control plane. The controller controls the data plane with well-defined API [23]. Controller of SDN is like the brain of network which manages the flow to the switches below and business logic and applications above [24].

Kim et al. [25] presented that how network can be configured and managed by using SDN. Policies can be defined in high language and can easily determine the problem occurring in network. SDN separates the data plane from control plane and makes the network switches as simply forwarding devices. Entire network is then controlled by centralized software. Reasons due to which networks become difficult to manage are: Network state change frequently and Configuration of network on every device. The conventional methods makes it impossible for network operators to define such configuration policies that responses to low level events occurring in networks automatically. By introducing SDN the continual change to network state can be handled automatically by increasing level of abstraction, designing programming languages, and network configuration. The need to configure individual device is not needed now because network operators can now make network wide decision in a single logical location with a global view of whole network state. In this approach limited number of control domains were considered for implementing polices ignoring the automation of IP conversion.

Jammal et al. [9] presented the advantages of using SDN in multiple environments such as Data Centers and the challenges faced by SDN. Locating servers and application through IP addresses can work fine in static networks where every device can be identified by IP address but cannot work in environment where network state changes continually. Managing such a sizable network through traditional approach is time consuming and expensive. The target of SDN is to simplify the network architecture by using the idea of centralizing the handling the L2 switching intelligence and L3 routing. Though SDN is one of the promising solutions in IT network yet it faces certain challenges like reliability and scalability. No way is defined in approach for handling controller failure. There must be some way that maintains the network reliability and in case of link failure, Controller should support multipath and reroute to active paths. Besides Routers maintain registry for IP conversion causing more delay.

Nunes [26] presented the historic perspective of programmable networks continuing with idea of SDN and its applications and then the future of SDN. For simplifying network evaluation, programmable network's idea is considered. SDN promises to significantly simplify the network innovation, management and evaluation. The key idea is to let software developers rely on network resources the same way as they rely on storage and computation resources. In SDN network devices act as simple forwarding devices and whole intelligence is shifted towards central logical controller. Network devices can be controlled through open Interface that is Open Flow or ForCES. Using single controller can become reason for whole network failure so switch can connect to multiple controllers via OPenFlow and can work these controllers can work as backup. Functionalities can be moved from middle box and can be implemented on controller in SDN. Another example of SDN related applications is the deployment from Google for connecting its data centers. In future SDN can support in different type of networks like cloud service, heterogeneous networks and Information Centric networks.

Raghavan et al. [27] proposed the outline of new architecture called as Software Defined Internet Architecture SDIA).The main purpose is that architectural evolution can become a part of software's task instead of hardware. In current network scenario, architecture is coupled with infrastructure which means that if change is to be made in architecture, which adds huge cost both for development and deployment and hence architecture evolution becomes difficult. SDIA combines and applies the main idea from software forwarding, MPLS and SDN. As a result the services can be implemented as software and approach becomes flexible and modular. The architecture follows top- down approach where internet services are decoupled in tasks and then the implementation related to these tasks is focused. As this architectural implementation is basically related to software so it can be standardized through open source effort and hardware standardization from large standardized bodies is not required. Multiple perspectives were considered on this approach but there was no automation for services offered by middle box.

Feamster et al. [57] presented the link between SDN and other technologies like Network Function Virtualization (NFV) along with history of programmable network. Computer networks are very difficult and hard to manage because they are made up of different type of equipment that include switches, routers and middle boxes like load balancer, intrusion detection systems and network address translators. The software that run on these routers and switches is most of the time closed and proprietary and have gone through many years for standardization and interoperability testing.

These devices are configured individually by the network administrator for different vendors and sometimes for different products of same vendor. Deploying a new service in such case is difficult because it needs configuration of all network devices again. If networks are made programmable then deploying new services and making new innovations can become easy. The main idea of the programmable networks started from active networks but was lacking incremental deployment. Then vision the moved to the separation of control and data plane. SDN focuses on this approach because it is changing the way of designing and managing the network.

Sahba [28] presented basics of SDN and technique through which switch can communicate with controller. SDN helps frequently changing network states by providing a centralized control plane. In cloud infrastructure, SDN use network centralization and network virtualization. Sin SDN, network devices can be programmed and virtualized networks can be developed by using existing hardware infrastructure. New virtual networks can be setup, upgrade and modified very easily and cheaply using SDN. In order to let communication happen between forwarding devices and controller, OpenFlow protocol is used. This protocol is used for decoupling the switch from controller. This is one of first SDN communication protocol. Controller can access flow table and can add, delete and update entries with the help of protocol. Main purpose of this protocol is to make communication between switches and controller standardized in SDN based architecture. Increasing functionality and performance and decreasing cost and network complexity can be achieved using protocol. OpenFlow ports are used for connecting switches with each other but there were no rules defined for converting IP.

Sharma et al. [29] proposed blockchain based cloud architecture using SDN and fog computing. High performance and cost effective computing can be achieved using this cloud based infrastructure. Blockchain technology is now considered as important because using this applications can be operated in distributed way. SDN decouples the forwarding plane from control plane making control more centralized. With the arrival of Iot, all intelligent thing like sensors, mobile, smart cars, laptops are connected to internet and have data analysis capability. Combination of IoT and cloud computing makes Fog computing that can provide cloud services like computing, network capabilities and storage to the users of system. The proposed architecture is efficient in offloading the data to cloud and accomplish design principles with less overhead. For offloading the data to cloud, NAT gateways are used for IP conversion adding workload on gateways.

Ali et al. [30] proposed source routing by making use of MPLS label. The solution is applicable both on SD-LAN and SD-WAN. SDN offers virtualization, resource sharing and flexibility by separating the control and forwarding plane. Network is divided in many sections using clustering algorithm called as MaxHop. Each section of network have different tradeoff overhead like bandwidth, traffic and flow table so this multi objective optimization problem is solve with linear weighted scalarization. Using routing with MPLS have many benefits and support heterogeneous switches with many ports. The proposed solution reduced bandwidth overhead to 79% as compared to traditional MPLS based forwarding scheme that needs complete encapsulated forwarding information at ingress switches. By dividing the network on basis of routing information in sub sections, setting the contacted switches and dividing the routing information along switches, the proposed solution outperforms the existing technique.

Gallegos [31] presented the evolution of SD-WAN for organizing business network by interconnecting two data centers. Emergence of new technologies like IoT, microsegmentation, virtualization and cloud based technologies generated new demand for Internet community. Traditional WAN technologies lack the requirements that are needed for real time applications and are supported by cloud based infrastructure. These services demand for high bandwidth. The routing algorithm working in WAN is TCP/IP for layer 3 or 4 and has high failure recovery time and inefficient resource utilization. The problem can be solved by using SDN solution in WAN called as SD-WAN. Automation and Orchestration feature can achieved using SD-WAN. The centralized control and management makes it more flexible to react more quickly, and programmable interface makes it dynamic enough to adapt any infrastructure. Results of proposed approach demonstrated that adequate level of QoS can be achieved by providing services efficiently in cloud through SD-WAN. Beside this new technologies like Iot, services on demand and cloud based services easily be enabled by using SD-WAN. But IP conversion process at NAT gateways caused workload on gateways and added delay.

Huiracocha et al. [32] proposed the use of SD-WAN for interconnecting two Data Centers with pre-fined QoS and traffic prioritization. Traditional network is facing challenges with the emergence of mobile, cloud computing and big data because these applications and services require network to be more dynamic and manageable. One solution is to invest more for increasing capacity of existing infrastructure but the size of network, inactive time in network and heterogeneous nature of services makes this solution invalid. SDN makes management process very simple and allows innovation and evolution. The proposed solution tries to solve any of the problems that occur while interconnecting data centers over traditional WAN. Limitations of existing networks like traffic prioritization and lack of efficient bandwidth management can be solved by using software defined technologies in wide area network. Better decision can be made regarding resource allocation and without depending on manufacturer, a global view of infrastructure can be provided. Results proved that resources can be utilized efficiently and sufficient level of QoS and traffic prioritization can be achieved using SD-WAN for interconnecting data centers but this approach was limited to two datacenters only so for interconnecting more datacenters IP conversion process is needed.

Bano et al. [12] proposed a solution called as Miracle that uses Software Defined Networking and Network Virtualization and provide benefit like cost reduction and agility to businesses who want to take benefit of colocation services. Cloud service providers follow different data models and provide various cloud services. These CSPs host their hardware in co-location points. As these co-location points do not have every type of service so organizations have to connect with multiple co-location points for multiple services. This is not a cost effective way for offering and managing storage and big data. The proposed solution creates logical elements by offering network virtualization. Instead of keeping multiple hardware in multiple co-location points, the Miracle platform takes service direct from the cloud and forwards it to requested organization, this way the need for organization to invest on multiple hardware in order to get services from multiple co-location points is mitigated. The solution is beneficial for those organizations who want to migrate to cloud to offer their services. Custom functionalities, flexibility and abstraction can be achieved with ease. The approach proposed colo-connectivity but there was no process defined for address translation increasing the possibility of overlapping customer's addresses.

Lallo et al. [11] proposed an Software Defined Networking (SDN) based approach for supporting end to end connectivity between collaborating partners. Communication between federated partners was made possible using SDN on network edge.

Approach was based on Network address and port translation with IP routing and flexibility and scalability was achieved by leveraging SDN. Although this approach showed its effectiveness in various network scenarios yet there was chance of overlapping customer address space because there was no mechanism defiend for dealing with this.

Jain et al. [33] presented design, implementation and evolution of a private WAN that was connecting Google's data centers present geographically. This private WAN is called B4. Deployment of new rapid network is enabled by separating control and data plane. Traditional WAN utilizes link almost 30-40% because all applications are treated equally WAN places premium on high availability but B4 solution utilized 100% link with cost effective bandwidth.

It meets application bandwidth demand more efficiently than any other solution. Bottleneck and future challenges in SDN is protocol bridging from data plane to control plane and the programming that makes SDN dynamic. B4 introduced hybrid approach that took benefit from existing routing protocols and new traffic engineering services and demonstrated that how SDN can be introduced in existing infrastructure gradually.

Yang et al. [2] presented the architecture and advances in SD-WAN. Advent of new applications and services demand more requirements from Wide Area Network. The base of Wide Area Network was best effort mentality which had no concern to provide guarantee for service quality. Beside this Wide Area Network is not easy to upgrade ion case of change occur in requirements. SD-WAN applies the idea of SDN in Wide Area Network.

It simplifies the building and managing perspective of connection in large geographical area and gives flexibility, central monitoring, control and management with very less cost. Two main features of SD-WAN are: no need for manually configuring each device as policies are defined centrally and applications that are already working in centralized manner can be hosted through inherent programmatic way. In the proposed approach, NAT traversal method caused workload on servers which added further delay.

Kim et al. [5] presented the idea of fuuly NAT traversal scheme using SDN environment. The approach was based on packet switching rather than packet modification. Because of the packet switching the delay can be reduced because packet header do not have to be modified in order to process further. IoT global connectivity service can be provided by internet service provider and the approach also takes advantage of separated control and data plane for achieving manageability.

Monir et al. [8] presented an approach for making network translation process easy in SDN environment by introducing a middleware called as NAPT. The middle ware used was programmable and its purpose was to make translation process easy and effective. Several calculation were made for delay and out of order packet percentage by using bandwidth and time interval as other parameter. The inclusion of additional hardware added more delay and increased cost.

## 2.2 SDN Controllers

This section gives a detail about SDN controllers and the services they are already providing and comparison of main features of popular controllers.

In order to implement network services there are some desired characteristics required for controller like it must be well documented and widely used by community. Following section will discuss most popular SDN controllers.

### 2.2.1 OpenDaylight

OpenDaylight is a modular open platform for customizing and automating networks of any size and scale. The OpenDaylight Project arose out of the SDN movement, with a clear focus on network programmability. It was designed from the outset as a foundation for commercial solutions that address a variety of use cases in existing network environments [34]. With this mission ODL was introduced in 2013.Originally led by Cisco and IBM, but now it is hosted under Linux Foundation Project. It is one of the most extensively deployed SDN controllers and has received industry wide support. Its code can be exploited in number of services [35]. Considering the architecture of ODL, Model-Driven Service Abstraction Layer (MD-SAL) is the fundamental of ODL platform. The interaction between network devices and network applications is handled in SAL. The model used by ODL is YANG which represent network devices and applications and SAL is used for exchanging data and adapting mechanisms between YANG models [34]. YANG is a data modeling language that was initially designed by the IETF NETCONF Data Modeling Language Working Group [36]. YANG data model is used in ODL so that micro services can be created and combined in order to solve more complex problems. Because of the modularity and flexibility of ODL it can be used by end users easily and they are allowed to create controller with whatever feature they need.

### 2.2.2 NOX

The first SDN controller was NOX which was initially developed by Nicira networks. It was first controller that supported OpenFlow protocols. In 2008 it was given to research community and became basis for many projects on SDN. NOX initially used cooperative threading to process events in a single threaded manner [37]. Its popular applications are Ethane and SANE (represent network as file system). Different versions of NOX are NOX, NOX-MT, and POX. Novel NOX supports C++ only and if compared with old NOX its network applications are fewer. NOX-MT is quiet altered version of NOX[38]. Different optimizations techniques have been introduced in it so that multithreading process can be introduced in it and response time of NOX can be improved. Another version based on the Python is POX. Main idea behind this was to create a separate python based platform. Fundamental purpose of POX is research [23]. Considering the model, NOX is assembled on an event-based programming model. It accepts a simple model of programming interfaces that rotates around three pillars: events, network view and namespace. Events can be generated by directly OpenFlow messages or by NOX applications. As far as namespaces and the network view are concerned, NOX consists of number of basic applications that construct the network view and maintain a high-level namespace that can be used by other applications [39].

### 2.2.3 Beacon

A Java-based open source OpenFlow controller created in 2010 by Stanford University and Big Switch Network [40]. Beacon was widely used for teaching, research, and as the basis of Floodlight. It was one of the cross platform, fast and modular SDN controller and supported both event and threaded based operations [41]. The main reason behind its popularity was that it was considered important in research community. Three main objectives behind the development of Beacon were: attain high performance, deliver developer side productivity and surge run time capability to start and stop application [40]. For maximizing code reuse and to mitigate the development burden for both controller and application using it, Beacon took advantage from many off the shelf libraries [42]. Basic architectural component of Beacon is bundle. Bundles are like jar file which contain metadata, java classes, resources and others. Bundles consume other packages, extended other bundles, run codes, and share their resources. These code bundles are independent and can be installed at startup or runtime so this thing makes Beacon modular and configurable without much problem [40].

### 2.2.4 ONOS

For building next generation SDN solutions ONOS is considered as important SDN controller. This controller can meet the needs by offering the adaptability to make and deploy new dynamic network services with easy programmatic interface. Configuration and real-time control of network are supported by this controller which eradicate the desire to run switching control protocol and routing protocol inside network fabric. Using this controller end users can create new applications without changing the data plane system [43]. ONOS has been designed keeping in mind the following goals: Code modularity, separation of concern, configurability and protocol agnosticism. More precisely, it should always be possible that new functionalities can be introduced without any difficulty as they are self-contained units. For achieving this, there must be clear boundaries between subsystems. In addition, there must be possibility of loading or unloading features at startup or even at run-time. Moreover an application must not be bounded with specific protocol library or any implementation [44]. Distributed architecture of ONOS is a key design principle. It can be deployed as pool of servers which might coordinate with each other and can provide much better capability e.g. if one of the controller instance fails, then distribution can provide fault tolerance and resilience. Scalability feature can also be considered as if workloads increase then whole system can handle this workload in a much better way as compared to single instance [45]. There are certain problems in achieving the above criteria and one of them is cluster coordination. Cluster Coordination in ONOS can be achieved by introducing distribution mechanism in different subsystems that generates events. Events are generated in stores and are shared among all the nodes in the clusters through distributed mechanism. These distributed mechanisms are built in various services' distributed store [44].

### 2.2.5 Floodlight

Floodlight is an enterprise class, Java based controller which is supported by a big community of developers. It is designed to work with routers, switches, virtual machines and access points that provisions openFlow standard.It is easy to setup with very less dependencies and offers modules loading system so that it can be extended and enhanced [46]. It supports virtual switches so it is easy to develop and test modules many times in a virtual environment. It is modular, runs over multiple platform and manages memory [47].

Architecture of floodlight is modular and has different modules like topology management, device/end-station management, path/route computation, infrastructure for web access (management), counter store (OpenFlow counters), and a state storage system, that are well stitched a by module-management system [48]. The

Floodlight OpenFlow controller can interoperate with any element agent that supports OpenFlow, but Big Switch also provides an open source agent that has been incorporated into commercial products [49].

### 2.2.6 RYU

Ryu is and open source controller written in Python, developed and supported by NTT cloud data centers. It supports NETCONF and OF-config network management protocols, as well as OpenFlow [50]. User of platform can write code in order to utilize the supporting infrastructure. This feature provide flexibility of SDN solution. For scalability feature Ryu does not have any kind of clustering ability inherently so external tools are required so that network state can be shared and failover between cluster members can be allowed. Ryu has component based architecture where existing components can be combined to form new application or existing components can be modified and new can be implemented [51].It provisions OenFlow protocol up to latest version and has an OpenFlow encoder and decoder library. Like other SDN controllers, it can create and send OpenFlow messages, listen to asynchronous events and can parse and handler incoming packets [50].

Table 2.1 compares the popular SDN controllers based on different parameters.

Table 2.1: Comparison of SDN Controllers

| **Controller Attribute** | **ODL** | **Flood-Light** | **Ryu** | **Beacon** | **ONOS** | **NOX** |
|---|---|---|---|---|---|---|
| **Year** | 2013 | 2013 | 2013 | 2010 | 2014 | 2009 |
| **GUI** | Web Based CLI | Web Based Java | CLI | Web Based CLI | Web Based | CLI Web Based UI |

| **Controller Attribute** | **ODL** | **Flood-Light** | **Ryu** | **Beacon** | **ONOS** | **NOX** |
|---|---|---|---|---|---|---|
| **Programming Language** | Java | Java | Python | Java | Java | C++ |
| **Supported Platform** | Linux, MacOS Windows | Linux, MacOS Windows | Linux | Linux, MacOS Windows | Linux, MacOS Windows | Linux |
| **Open Stack** | Y | N | Y | N | Y | N |
| **South-bound APIs** | OPen Flow 1.0,1.3 | Open Flow 1.0-1.5 | Open Flow 1.0-1.5 | Open Flow 1.0 | Open Flow 1.0, 1.3 | Open Flow 1.0 |
| **North-bound APIs** | REST, REST-CONF XMPP, NET-CONF | REST Java RPC | Quantum REST | Ad-hoc | REST, Neutron | Ad-hoc |
| **Architecture** | Flat Distributed | Centralized | Centralized | Centralized | Flat Distributed | Centralized |
| **Modularity** | High | High | Fair | Fair | High | Low |
| **Documentation** | Good | Good | Medium | Fair | Good | Limited |

| Controller Attribute | ODL | Flood-Light | Ryu | Beacon | ONOS | NOX |
|---|---|---|---|---|---|---|
| **Community Support** | Linux Foundation | Big Switch Networks | NTT | Stanford University | Linux Foundation | Nicira Networks |
| **Licence** | EPL 1.0 | Apache 2.0 | Apache 2.0 | GPL 2.0 | Apache 2.0 | GPL 3.0 |
| **Consistency** | Yes | Yes | Yes | No | Yes | No |
| **Multi-threading** | Yes | Yes | Yes | Yes | Yes | Yes (NOX-MT) |

## 2.3 Comparison and Conclusion

The main focus of this study is to identify the way for interconnecting the private networks by implementing the application of NAT on SDN. Existing approaches are based on the interconnecting networks by introducing some translating hardware in the Software Defined Networking (SDN) environment which adds more delay.

For reducing the delay the implementation is to be done on software level which takes advantage of the underlying concept of Software Define networking.

Several issues are discussed in the existing studies related to the translating process and interconnecting networks. Comparison of the techniques proposed in the literature is enlisted in Table 2.2 where publication year, objective of the study, limitations and strengths are mentioned.

Table 2.2: Literature Review

| **Ref.** | **Pub. Year** | **Objectives** | **Strengths** | **Weaknesses** |
|---|---|---|---|---|
| [27] | 2012 | Decoupling Architecture from infrastructure | • Ease the adoption of various new Internet architectures | • No automation for services offered by middle box. |
| [25] | 2013 | Improve various aspects of network management | • Enable network operators to implement network policies in a high-level language | • Limited number of control domains for implementing polices. |
| [9] | 2014 | Benefits of using SDN in a multitude of environments | • SDN can be used for simplifying the management process of big networks | • Routers maintain registry for IP conversion.<br>• Add delay |
| [11] | 2016 | Support end to end connectivity between collaborating partners | • Made communication possible using SDN on network edge | • No mechanism for dealing with overlapping customer address space. |
| [31] | 2017 | Use SDN to interconnect clouds | • Resource sharing between clouds made easy | • Workload at NAT gateways caused delay |
| [30] | 2017 | Implement efficient routing scheme for Sd-WAN | • Learning the best path for routing. | • Less used links are pruned causing congestion on commonly used links. |

| **Ref.** | **Pub. Year** | **Objectives** | **Strengths** | **Weaknesses** |
|---|---|---|---|---|
| [28] | 2018 | Find suitable protocol for SDN | • OpenFlow protocol setup communication between controller and forwarding plane of network | • No rules defined for converting IP. |
| [29] | 2018 | Provides low-cost, secure, access to competitive computing infrastructures | • Efficiently manage the raw data streams produced by large IoT devices | • Difficult to establish Peer-to-Peer session. |
| [2] | 2019 | Explore the possibility of applying new techniques for networking on SD-WAN | • SD-WAN is a promising architecture of next generation design of wide area network. • Cope with requirements of applications | • High load on servers for NAT traversal |
| [32] | 2019 | Interconnect datacentres in SDN environment | • Programmability within router • Successful translation of addresses. • Resource sharing | • Limited to two data-centres • No process defined for IP conversion |
| [12] | 2019 | Provide agility and cost reduction to organizations. | • Significantly ease the adoption of colocation services. | • No process defined for address translation. |

| **Ref.** | **Year** | **Technique used** | **Objective** | **Weaknesses** |
|---|---|---|---|---|
| [8] | 2019 | Interconnecting organizations in SDN environment using NAPT middleware | • Programmability within router<br>• Successful translation of addresses.<br>• Resource sharing | • Additional physical hardware.<br>• Yielded more delays<br>• Costly |

Most of the proposed approaches are based on interconnecting organization using traditional technologies. Some approaches have used SDN for IP conversion process but they have used NAT gateways for this purpose or have introduced hardware for conversion process. In order to cope with current requirements of services and applications, network needs to be dynamic and centrally manageable. Our work is different from current approaches in terms of interconnecting organization/clouds by implementing the translating service at software level. For example, Kim et al. [25] considered implementing polices in SDN for automating the services but the mechanism for automating the services offered by middlebox like Firewall and NAT. Raghavan et al. [27] proposed various aspects of network management using SDN but not defined any policy for IP conversion process. The approach considered limited number of control domains like time, network flow and data usage for implementing policies. Jammal et al. [9] proposed using SDN for multitude of environments and showed that SDN can simplify the management process but here routers were used the maintain the NAT registry which hindered the scalability feature of SDN. Approaches presented in [31] and [2] used SDN for implementing various aspects but increased workload at NAT gateways. The approaches proposed by [11] and [12] supported end to end connectivity using SDN and provided agility and cost reduction to organizations aiming to leverage colocation services respectively but not defined any mechanism for dealing with overlapping customers' space. Another approach proposed by Monir et al. [8] added additional hardware of IP translation purpose and made resource sharing successful but because of this extra hardware the transmission delay increased making the situation even worse.

After critically analyzing the literature, we adopted an approach which considers interconnecting hosts residing in private networks by making NAT program on controller. The approach has used the concept of SDN where controller is considered as main entity. Choice of controller to be used depends on different parameters like architecture, modularity, programming language, developer community support, documentation available and low learning curve. Controller should have centralized architecture with good developer's community support and documentation. Keeping in view these parameters Floodlight Controller is considered a good choice to be used in this approach because it supports centralized architecture. Good documentation and developer's community support is also available for this controller which makes it easy to learn and integrate new feature in existing one.

# Chapter 3

# Proposed Approach

This chapter contains detail about research methodology and implementation details.

## 3.1 Introduction

In the literature section, we have identified most commonly used technique for interconnecting hosts between private networks. NAT is performed at NAT gateway or Wi-Fi access points adding workload on these gateways. As networking paradigm is now moving to SDN still IP conversion is taking place at NAT gateways or hardware points.

To overcome the gap in existing techniques, we have proposed a new technique which takes advantage of the underlying concept of SDN and implements NAT module at software level.

## 3.2 Proposed Architecture

Flow chart of our proposed approach is shown in Figure 3.1.

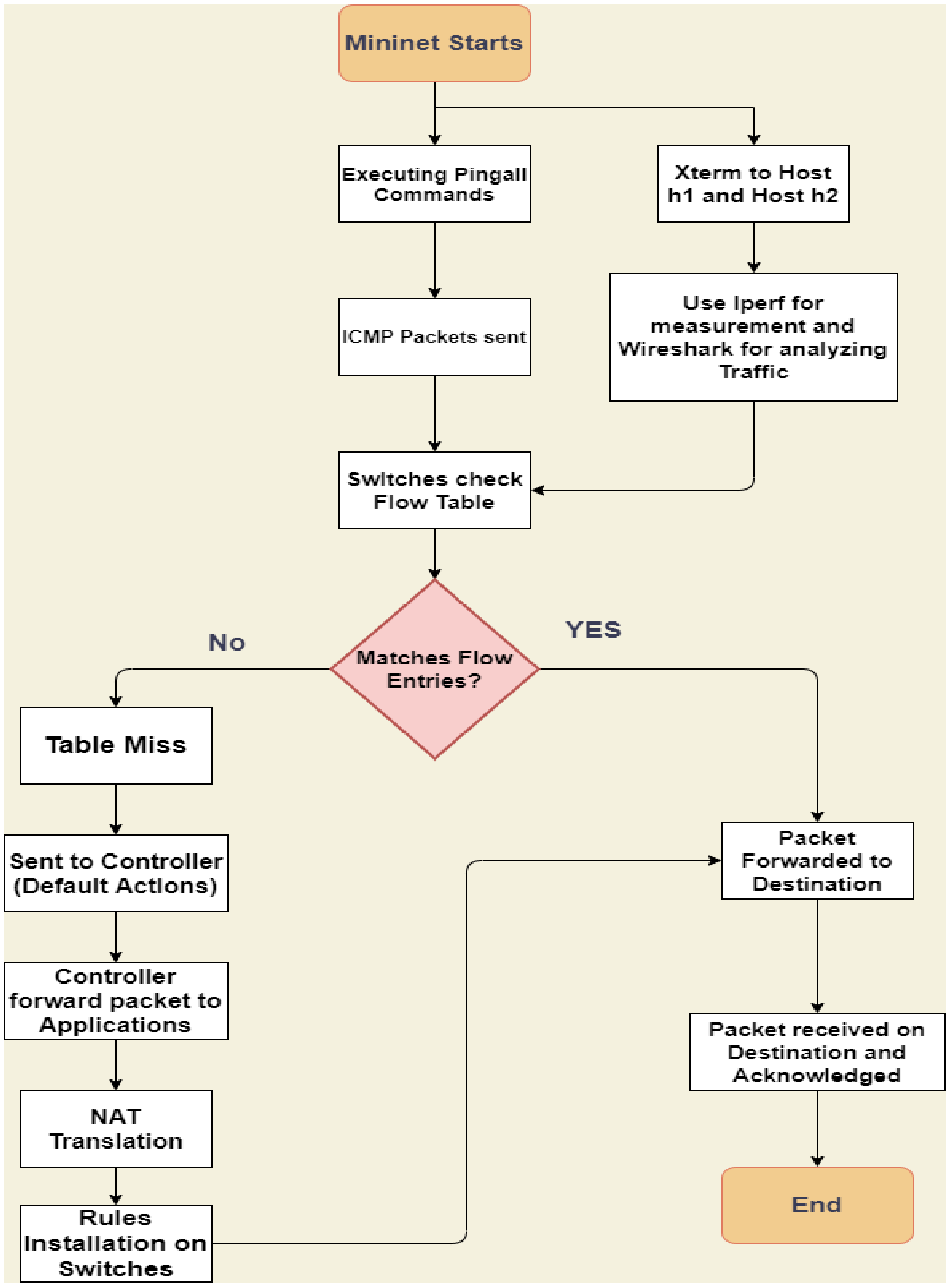

Mininet Starts
Executing Pingall Commands
Xterm to Host h1 and Host h2
ICMP Packets sent
Use Iperf for measurement and Wireshark for analyzing Traffic
Switches check Flow Table
Matches Flow Entries?
No
YES
Table Miss
Sent to Controller (Default Actions)
Controller forward packet to Applications
NAT Translation
Rules Installation on Switches
Packet Forwarded to Destination
Packet received on Destination and Acknowledged
End

FIGURE 3.1: Flowchart of Proposed Approach

### 3.2.1 Mininet

Our experiment started with Mininet. Mininet is one of the network emulation tool which is used for creating networking scenarios. This may include controllers, switches, virtual hosts and links. Linux network software is run on its hosts whereas OpenFlow protocol is supported by their switch which is important for SDN. A complete experimental network scenario can be prototyped using Mininet and can be used for testing debugging and any other research task [54]. Mininet is used in this research for creating topology. Graphical user interface of Mininet called as MiniEdit is used for creating graphical image of topology.

The topology created for experiment is presented in Figure 3.2.

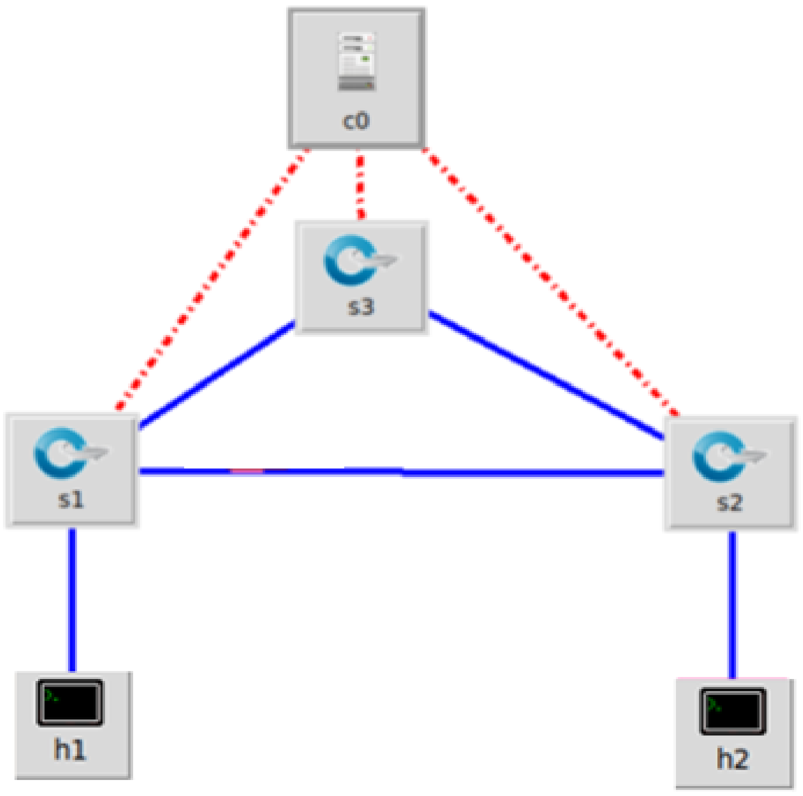
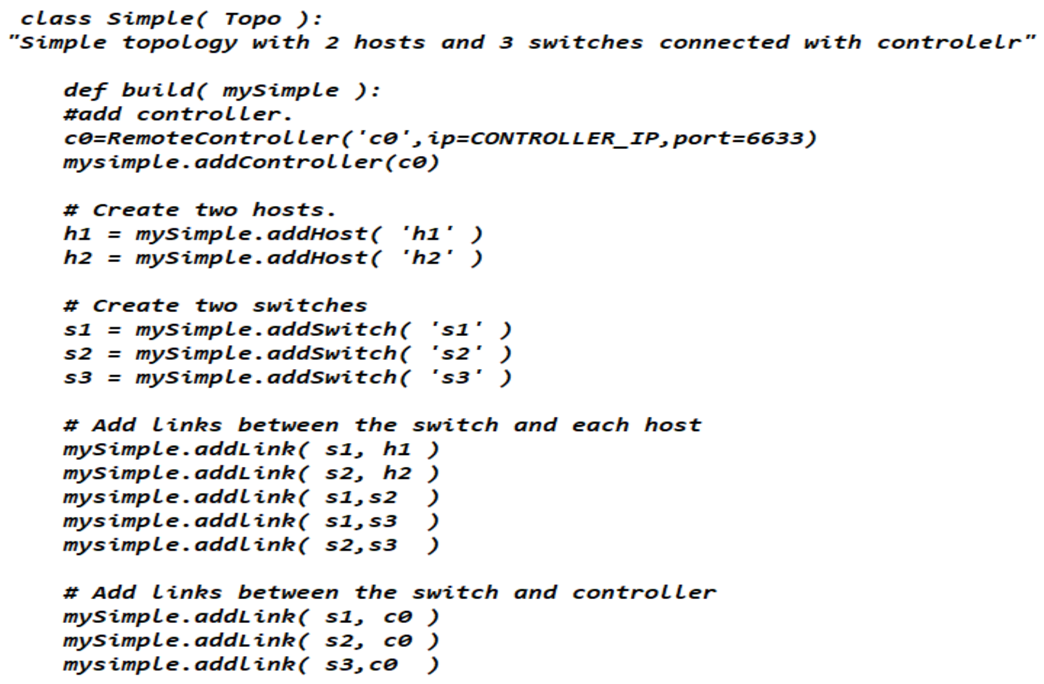


FIGURE 3.2: Implemented Topology

In the above figure:

- Hosts are represented as h1 and h2
- Switches are s1,s2 and s3
- SDN controller is represented as c0 where NAT program is implemented

Python can also be used for creating custom network topologies that imitates real world networks. The above network topology can also be created using Python and instantiated using Mininet as given in Figure 3.3.



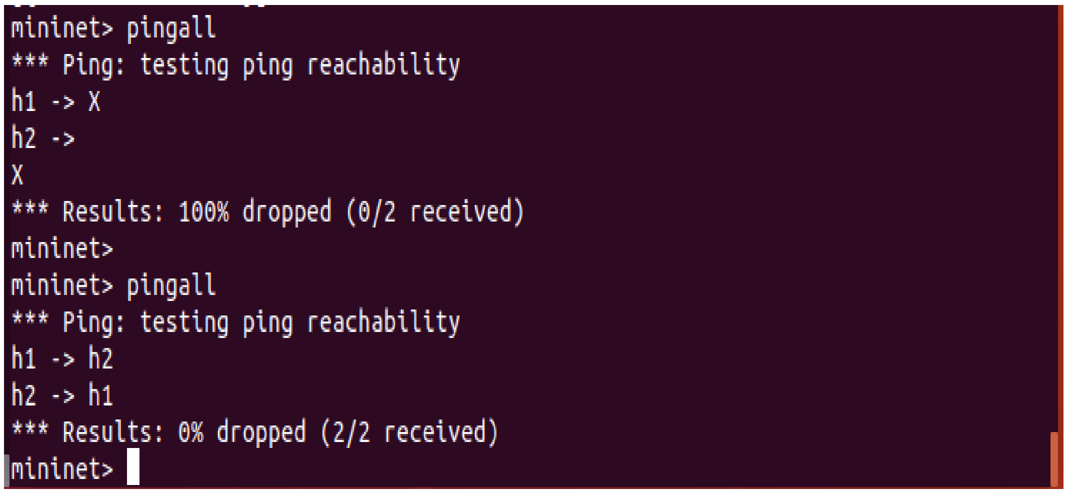
```
 class Simple( Topo ):
"Simple topology with 2 hosts and 3 switches connected with controlelr"

    def build( mySimple ):
    #add controller.
    c0=RemoteController('c0',ip=CONTROLLER_IP,port=6633)
    mysimple.addController(c0)

    # Create two hosts.
    h1 = mySimple.addHost( 'h1' )
    h2 = mySimple.addHost( 'h2' )

    # Create two switches
    s1 = mySimple.addSwitch( 's1' )
    s2 = mySimple.addSwitch( 's2' )
    s3 = mySimple.addSwitch( 's3' )

    # Add links between the switch and each host
    mySimple.addLink( s1, h1 )
    mySimple.addLink( s2, h2 )
    mysimple.addlink( s1,s2  )
    mysimple.addlink( s1,s3  )
    mysimple.addlink( s2,s3  )

    # Add links between the switch and controller
    mySimple.addLink( s1, c0 )
    mySimple.addLink( s2, c0 )
    mysimple.addlink( s3,c0  )
```

FIGURE 3.3: Sample Code for Creating Simple Network Topology using Python

### 3.2.2 Pingall Command and ICMP Packets

At next Pingall command is sent for checking connectivity between hosts. It shows 0% packet drop if hosts are reachable to each other. Internet Control Message Protocol (ICMP) is also used for generating error message if hosts are not reachable to each other.

Fig 3.4 shows the pingall result when hosts are reachable to each other.



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 ->
X
*** Results: 100% dropped (0/2 received)
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

FIGURE 3.4: Pingall Result Showing Connectivity

### 3.2.3 Flow Table Entries

After pingall, flow table entries are checked as the packet is received on switch. If match is found with entry then packet will be forwarded to the destination according to the rule installed. It shows that the rule for forwarding has already been installed on the OpenFlow switch. When no match found, packet will be forwarded to the controller where Controller instructs how to deal with the packet.

### 3.2.4 Controller

Floodlight controller is used for this work and to implement NAT program. It is pre-declared in the script which is launched in separate CLI when this script started running. As the packet is received on controller, it will take actions on packet. New functionalities can be added with ease on top of controller and these functionalities and rules can be installed on OpenFlow switches regarding such functionalities [52]. Figure 3.5 presents the working of controller.

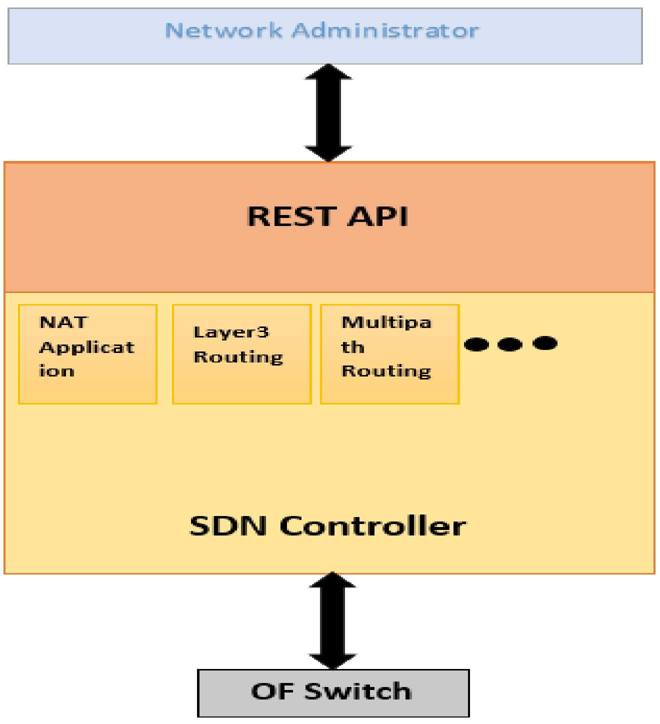


FIGURE 3.5: Working of Controller

The controller has several modules with each module implementing a particular network service. Every module is independent of other but depending on condition, can use internal API which is exposed by other module. AS an example, a packet must be routed through NAT device so NAT module can use Layer3 Routing module [53] or for converging packet Multipath Routing will also be used in conjugate with other modules . Developers are allowed to extend any existing functionality at software level without disturbing the hardware which depicts the advantage of using SDN.

### 3.2.5 NAT Module

The implemented NAT module has used Floodlight SDN controller's core services for dispatching actions and listening to subscribed events. Rules Regarding NAT entry running on OPenFlow switches are installed using openFlow provide of Controller. REST API and JAVA API is exposed by module to the controller and implemented module also interacts with Layer3 routing module.

Fig 3.6 represents the architecture of NAT module and interaction with Controller and its routing module.

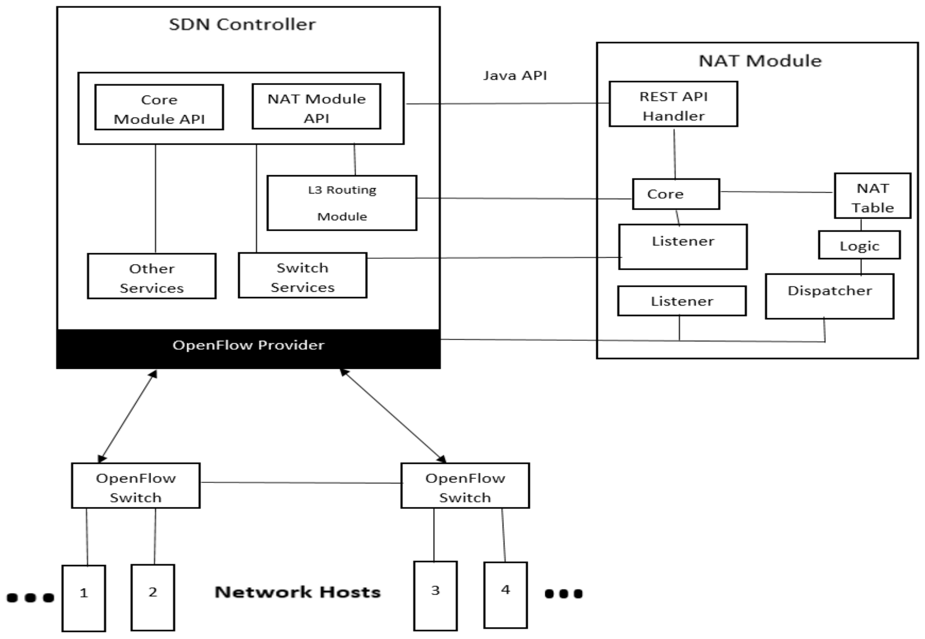


FIGURE 3.6: Architecture of NAT Module

The implemented NAT module has multiple components and it is compiled with controller. REST API and JAVA API are exposed to controller.

One of the listener components is used to listen changes at OPenFlow switch and other to listen messages that arrive at controller.

Dispatcher will build the OPenFlow messages queued on controller for sending to particular OpenFlow switch.

NAT logic and table change NAT entries on arriving packets.

#### 3.2.5.1 NAT Working

As controller will send packet to multiple applications, one such application is our implemented NAT application.

For implementing NAT service in SDN controller, it is not required that the forwarding devices modify the datagram and they must have NAT table. This task is done by the controller which is the main entity in SDN.

Controller has a global view of the network and will maintain NAT service state and new rules will be dispatched for OpenFlow switch and for every connection.

Steps performed for NAT are explained below.

**Steps:**

1. A rule about redirecting all ARP requests is installed in OpenFlow switches.
2. An ARP request is sent from private host asking for MAC address of default gateway.
3. ARP request is redirected to controller from OpenFlow switch.
4. An ARP reply is generated from controller containing the Medium Access Control (MAC) address of OpenFlow switch and is sent back to switch as shown in Figure 3.7.

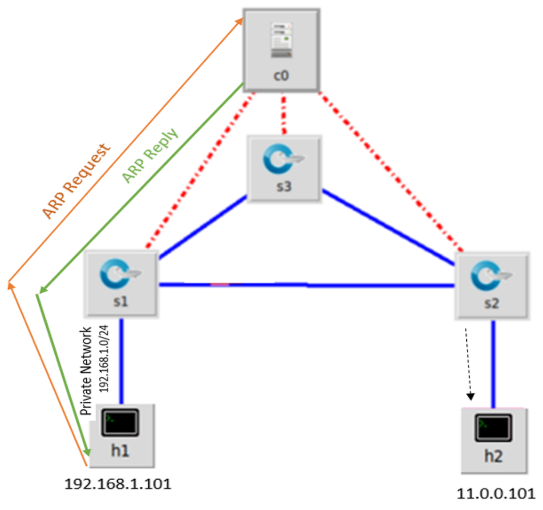


FIGURE 3.7: ARP Request and Reply Sequence

5. As soon as private host receives the ARP reply, the data packet is sent to MAC address which was received in ARP reply.

6. Packet is received on OpenFlow switch with no rule, so it is directed to controller.

7. Packet is analyzed in controller and NAT module fills a NAT entry and change the private IP and injects the packet on network. At the same time a rule is installed on OF switch to directly translate the packets destined for this destination and port through which this packet should be sent as shown in Figure 3.8.

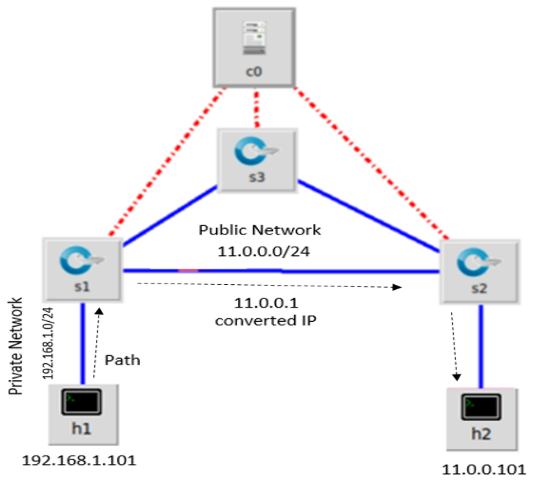


FIGURE 3.8: IP Conversion Through NAT Registry

8. The controller also tells the receiving switch that a packet with these particular credentials is received on this particular port and where to forward this packet.

9. Public host reply to the received packet.

10. As the reply data packet reaches the OpenFlow switch again, it is translated back because of installed rule and sent to host. Wireshark and Iperf are used for generating results and for confirming the conversion of IPs.

### 3.2.6 Link Convergence

Set of available and offered services differ from site to site. In case any particular site goes down then automatic connectivity can be assured using the service called as Multipath Routing. Another scenario is implemented for explaining this link connectivity as shown in Figure 3.9.

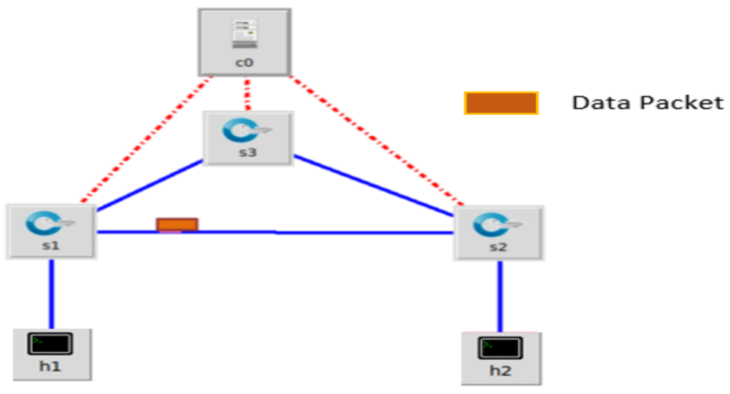


FIGURE 3.9: Organizations Connection When Link is Up

The above figure shows the connection of h1 to h2 with the shortest path Data is moving between h1 and h2 through path h1$\rightarrow s1 \rightarrow s2 \rightarrow h2$.

If this link goes down then service availability can be assured by calculating other path. Different services are offered at different sites (different Co-lo points). This service accessibility from different Co-Location points can be made available if link goes down by converging the traffic as shown in Figure 3.10.

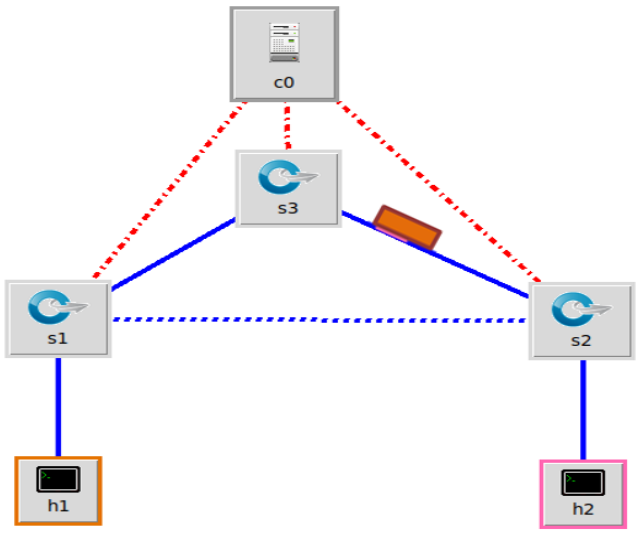


Figure 3.10: Organizations Connection When Link is Down

In the above figure the dashed line (——) shows that link is down and no traffic can pass through this link. Controller will recalculate the path and install new rules on OpenFlow switches and tell them that this path is no more available. The whole traffic will be converged to the path h1$\rightarrow s1 \rightarrow s3 \rightarrow s2 \rightarrow h2$.

# Chapter 4

# Results and Discussion

The main goal of this thesis is to present the application development on novel networking approach SDN for interconnecting organizations where they do not need to have unique IPs. Implementation detail is provided in Proposed Approach and Implementation chapter. This chapter contains details about testing environment, results and evaluations. Functional testing is done for checking the correctness of implemented network service and performance test is conducted for evaluating results.

## 4.1 Test Environment

Four our experiments, we have used a base machine running on Ubuntu version 18.04 LTS with the hardware containing Intel Core i7 8550U CPU with a clock speed of 2.00 Giga Hertz and RAM of 8 Gigabytes. The selected controller for this thesis is Floodlight controller. It is a java based controller and module is implemented and integrated with it using eclipse IDE. Apache ant is used for compiling controller's modules. Building and handling of big Java projects is done through this command-line tool [55]. Implemented module is tested through topology. Topology is emulated using Mininet and Floodlight controller is connected with it. Mininet is a tool for emulating network scenarios.

### 4.1.1 Floodlight Compilation

Floodlight has several modules where each module performs specific task. Source code contains almost 502,000 lines of codes with several external Java libraries. The routing module is already developed by Floodlight community and this module has been used in this thesis. The implemented module took 1-2 minutes for compiling with Floodlight controller.

## 4.2 Network Emulation

In order to check the implemented module topology is created with two hosts and three switches connected with controller. Mininet is used for creating network because it supports OpenFlow switches and several topologies can be created and initiated using python scripts.

### 4.2.1 Topologies

The topology for experimenting is presented in Figure 4.1.

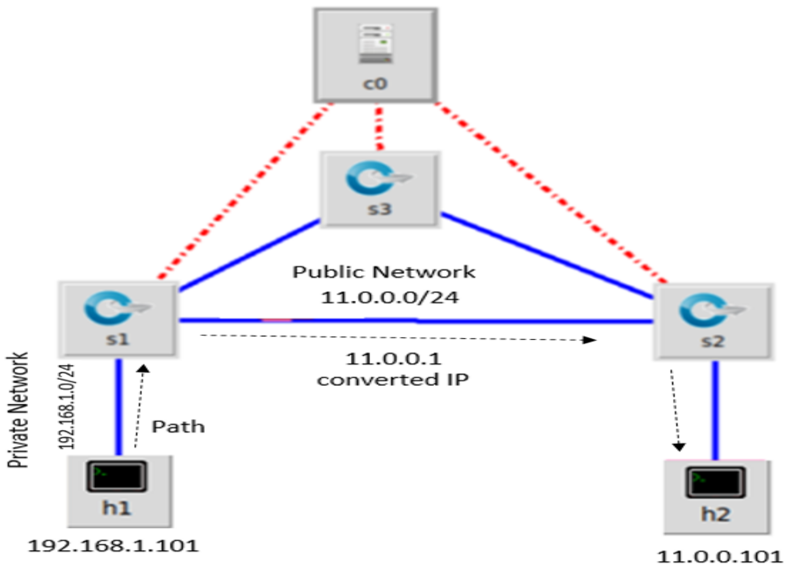


Figure 4.1: Topology for Experiment

The configuration of the hosts for topology of Figure 4.1 is shown in Table 4.1.

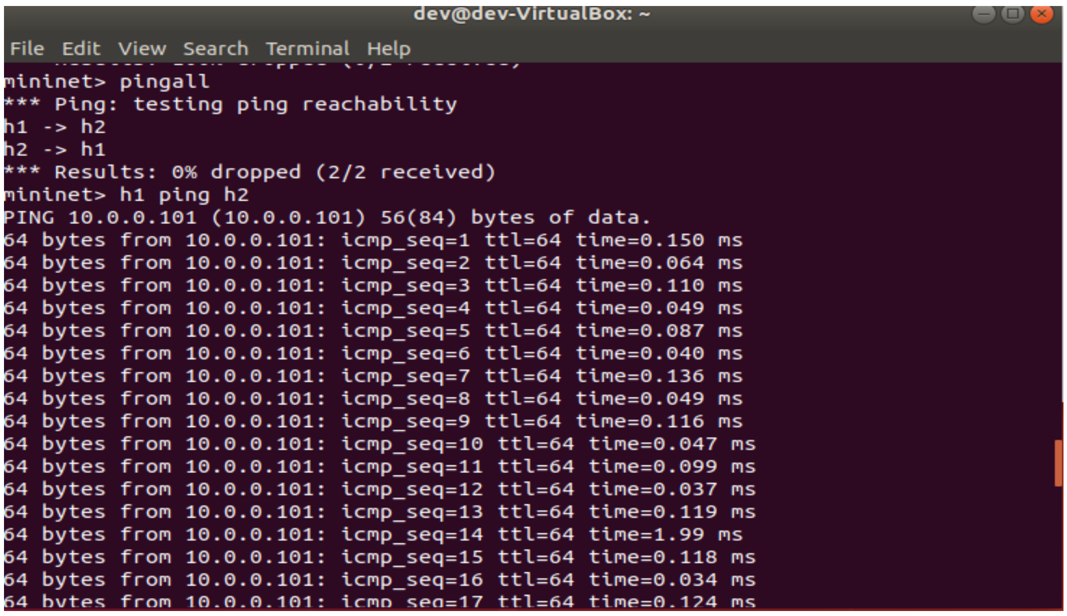
TABLE 4.1: Configuration of Hosts for Simple Topology

| **Host** | **IP** | **Default gateway** | **Network** | **Connect-ion** | **IP of h1 after con-version** |
|---|---|---|---|---|---|
| **h1** | 192.168.1.101 | 192.168.1.1 | 192.168.1.0/24 | h1—s1 | 11.0.0.2 |
| **h2** | 11.0.0.101 | 11.0.0.1 | 11.0.0.0/24 | h2—s2 | - |

TABLE 4.2: Configuration of Switches for Simple Topology

| **Switch** | **Datapath Id (dpid)** | **Connection** |
|---|---|---|
| **s1** | 00:00:00:00:01:00 | s1—h1<br>s1—s2 |
| **s2** | 00:00:00:00:01:01 | s2—s1<br>s2—h2 |

Host h1 is configured with private network having IP in range of 192.168.1.0/24 whereas h2 is configured with public IP 11.0.0.0/24. In order to confirm the connectivity between hosts Ping is performed between hosts which is shown in Figure 4.2.

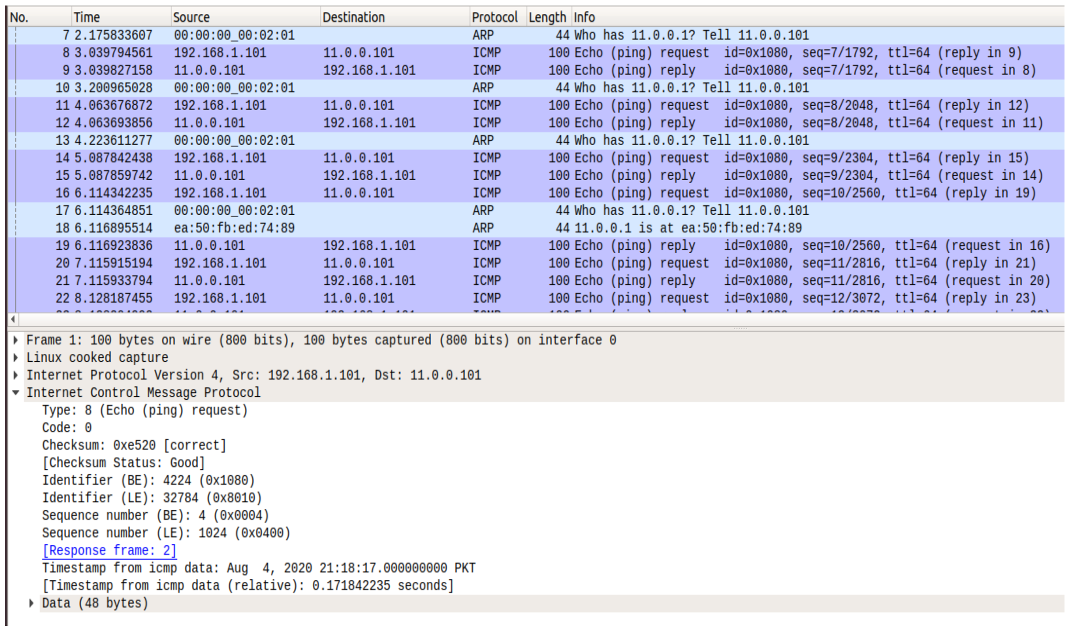


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> h1 ping h2
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=0.150 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.110 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.049 ms
64 bytes from 10.0.0.101: icmp_seq=5 ttl=64 time=0.087 ms
64 bytes from 10.0.0.101: icmp_seq=6 ttl=64 time=0.040 ms
64 bytes from 10.0.0.101: icmp_seq=7 ttl=64 time=0.136 ms
64 bytes from 10.0.0.101: icmp_seq=8 ttl=64 time=0.049 ms
64 bytes from 10.0.0.101: icmp_seq=9 ttl=64 time=0.116 ms
64 bytes from 10.0.0.101: icmp_seq=10 ttl=64 time=0.047 ms
64 bytes from 10.0.0.101: icmp_seq=11 ttl=64 time=0.099 ms
64 bytes from 10.0.0.101: icmp_seq=12 ttl=64 time=0.037 ms
64 bytes from 10.0.0.101: icmp_seq=13 ttl=64 time=0.119 ms
64 bytes from 10.0.0.101: icmp_seq=14 ttl=64 time=1.99 ms
64 bytes from 10.0.0.101: icmp_seq=15 ttl=64 time=0.118 ms
64 bytes from 10.0.0.101: icmp_seq=16 ttl=64 time=0.034 ms
64 bytes from 10.0.0.101: icmp_seq=17 ttl=64 time=0.124 ms
```

FIGURE 4.2: Hosts Pinging Each Other

By looking at the above figure it is confirmed that hosts are now connected and can communicate and send data to each other. After making sure that hosts can ping each other, the main task is to find whether packets are translated or not. It is checked by using the Wireshark during Ping request.

Fig 4.3 represents the Wireshark results for simple topology.

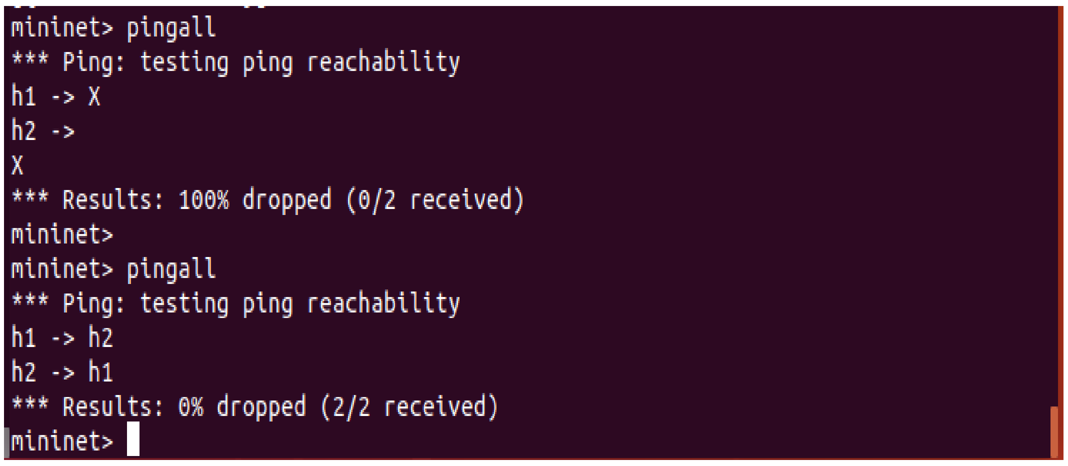


```
No.   Time          Source              Destination     Protocol Length Info
    7 2.175833607   00:00:00_00:02:01                   ARP        44 Who has 11.0.0.1? Tell 11.0.0.101
    8 3.039794561   192.168.1.101       11.0.0.101      ICMP      100 Echo (ping) request  id=0x1080, seq=7/1792, ttl=64 (reply in 9)
    9 3.039827158   11.0.0.101          192.168.1.101   ICMP      100 Echo (ping) reply    id=0x1080, seq=7/1792, ttl=64 (request in 8)
   10 3.200965028   00:00:00_00:02:01                   ARP        44 Who has 11.0.0.1? Tell 11.0.0.101
   11 4.063676872   192.168.1.101       11.0.0.101      ICMP      100 Echo (ping) request  id=0x1080, seq=8/2048, ttl=64 (reply in 12)
   12 4.063693856   11.0.0.101          192.168.1.101   ICMP      100 Echo (ping) reply    id=0x1080, seq=8/2048, ttl=64 (request in 11)
   13 4.223611277   00:00:00_00:02:01                   ARP        44 Who has 11.0.0.1? Tell 11.0.0.101
   14 5.087842438   192.168.1.101       11.0.0.101      ICMP      100 Echo (ping) request  id=0x1080, seq=9/2304, ttl=64 (reply in 15)
   15 5.087859742   11.0.0.101          192.168.1.101   ICMP      100 Echo (ping) reply    id=0x1080, seq=9/2304, ttl=64 (request in 14)
   16 6.114342235   192.168.1.101       11.0.0.101      ICMP      100 Echo (ping) request  id=0x1080, seq=10/2560, ttl=64 (reply in 19)
   17 6.114364851   00:00:00_00:02:01                   ARP        44 Who has 11.0.0.1? Tell 11.0.0.101
   18 6.116895514   ea:50:fb:ed:74:89                   ARP        44 11.0.0.1 is at ea:50:fb:ed:74:89
   19 6.116923836   11.0.0.101          192.168.1.101   ICMP      100 Echo (ping) reply    id=0x1080, seq=10/2560, ttl=64 (request in 16)
   20 7.115915194   192.168.1.101       11.0.0.101      ICMP      100 Echo (ping) request  id=0x1080, seq=11/2816, ttl=64 (reply in 21)
   21 7.115933794   11.0.0.101          192.168.1.101   ICMP      100 Echo (ping) reply    id=0x1080, seq=11/2816, ttl=64 (request in 20)
   22 8.128187455   192.168.1.101       11.0.0.101      ICMP      100 Echo (ping) request  id=0x1080, seq=12/3072, ttl=64 (reply in 23)

▸ Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 11.0.0.101
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xe520 [correct]
    [Checksum Status: Good]
    Identifier (BE): 4224 (0x1080)
    Identifier (LE): 32784 (0x8010)
    Sequence number (BE): 4 (0x0004)
    Sequence number (LE): 1024 (0x0400)
    [Response frame: 2]
    Timestamp from icmp data: Aug  4, 2020 21:18:17.000000000 PKT
    [Timestamp from icmp data (relative): 0.171842235 seconds]
  ▸ Data (48 bytes)
```

FIGURE 4.3: Wireshark Capture of NAT Result in Simple Topology

Host h1 is configured with private network having IP 192.168.1.101and wants to communicate with h2 which is residing out of this private network. As request is received on the switch s1 it is redirected to controller which is the ARP request. The controller replies to this ARP request and installs rules on OpenFlow switches and changes IP. Now this private IP is converted to public IP 11.0.0.2 and can communicate and access data.

The above figure confirms that implemented service is working fine in Floodlight controller and packets are translated correctly and checksum status is also correct which shows that data is reaching destination without corruption. Request always goes to controller which is the ARP request and then controller installs rules for further packets. Rules are installed on OpenFlow switches that tells them that where to forward the upcoming (if their destination is same as of previous packets) and there is no need to redirect packets to controller every time. Beside installation rules are refreshed after 10s for finding that topology is still same or not. Figure 4.4 represents the ping reachability when rules are installed in the OpenFlow switches.

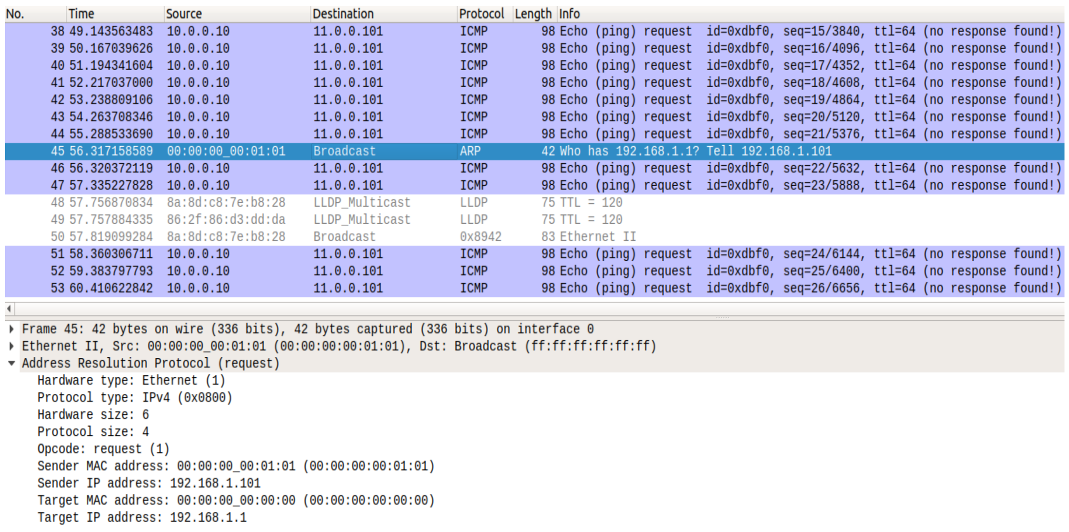


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 ->
X
*** Results: 100% dropped (0/2 received)
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

FIGURE 4.4: Ping Reachability Before and After Rules Installation

When ARP request is sent to controller, the controller takes decision about packet, and tells switches where to forward packet. The above diagram show the ping reachability when controller is installing rules in switches. During rules installation for first time, host are not reachable to each other that is why Ping results fail.

Fig 4.5 presents the Wireshark result for ARP.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 38 | 49.143563483 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=15/3840, ttl=64 (no response found!) |
| 39 | 50.167039626 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=16/4096, ttl=64 (no response found!) |
| 40 | 51.194341604 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=17/4352, ttl=64 (no response found!) |
| 41 | 52.217037000 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=18/4608, ttl=64 (no response found!) |
| 42 | 53.238809106 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=19/4864, ttl=64 (no response found!) |
| 43 | 54.263708346 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=20/5120, ttl=64 (no response found!) |
| 44 | 55.288533690 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=21/5376, ttl=64 (no response found!) |
| 45 | 56.317158589 | 00:00:00_00:01:01 | Broadcast | ARP | 42 | Who has 192.168.1.1? Tell 192.168.1.101 |
| 46 | 56.320372119 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=22/5632, ttl=64 (no response found!) |
| 47 | 57.335227828 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=23/5888, ttl=64 (no response found!) |
| 48 | 57.756870834 | 8a:8d:c8:7e:b8:28 | LLDP_Multicast | LLDP | 75 | TTL = 120 |
| 49 | 57.757884335 | 86:2f:86:d3:dd:da | LLDP_Multicast | LLDP | 75 | TTL = 120 |
| 50 | 57.819099284 | 8a:8d:c8:7e:b8:28 | Broadcast | 0x8942 | 83 | Ethernet II |
| 51 | 58.360306711 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=24/6144, ttl=64 (no response found!) |
| 52 | 59.383797793 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=25/6400, ttl=64 (no response found!) |
| 53 | 60.410622842 | 10.0.0.10 | 11.0.0.101 | ICMP | 98 | Echo (ping) request  id=0xdbf0, seq=26/6656, ttl=64 (no response found!) |

```
▸ Frame 45: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:01:01 (00:00:00:00:01:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 00:00:00_00:01:01 (00:00:00:00:01:01)
    Sender IP address: 192.168.1.101
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.1.1
```

FIGURE 4.5: Wireshark Results for ARP

The above figure shows that ARP request is sent from host to the default gateway so that it can be redirected to controller. As h1 is not connected to h2 at this time ping result is also as shown in figure 4.4. But as rules are installed then host can ping each other. Figure 4.6 present the graphical representation of RTT of 50 PING request further explaining the connectivity between hosts.

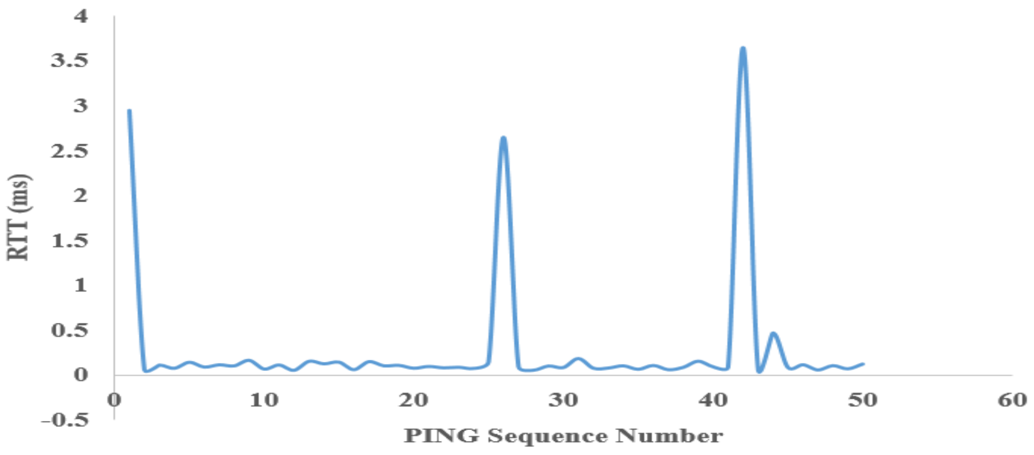


FIGURE 4.6: RTT by PING Sequence Number On Simple Topology

RTT is taken for 50 Ping request. At some points in the above graph, RTT is exceptionally very high. This is due to the installation of the OpenFlow rules in OpenFlow switches. When Ping is started then rules are installed in the OpenFlow switches which increases the round trip time. After the installation of rules, RTT is seen too be at low value. OpenFlow rules are refreshed periodically by controller due to which value of RTT is high. Figure 4.7 presents the graphical representation of RTT results for link up and down (Topology presented in Figures 3.9 and 3.10 in chapter 3).

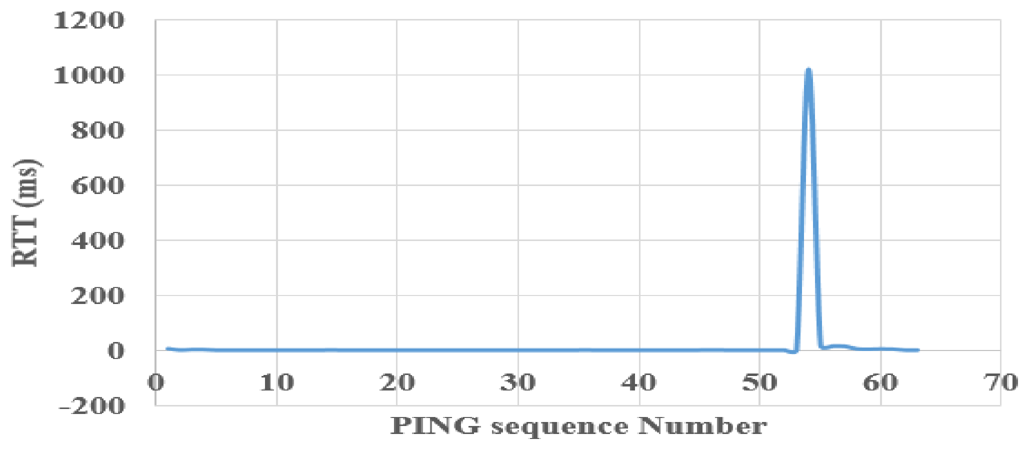


FIGURE 4.7: RTT by PING Sequence When Link is Up and Down

The value of RTT is high when rules are refreshed. As the link is up RTT's value is not increasing but as the link goes down the value of RTT becomes exceptionally high. This is because of the fact that controller is recomposing new topology to converge the traffic to some other path. It takes almost 10-15ms to converge the traffic to new path in order to assure service availability. For evaluation purpose we have further tested our implemented topology for parameters like Bandwidth vs Jitter and Bandwidth vs Out of Order Packet % and then Time Interval vs Jitter and Time Interval vs Out of Order Packet %. Firstly, we have kept the time Interval constant i.e. 10s and changed the data transfer rate from 5 Mbps to 40Mbps and calculated Jitter and Out of Order Packet %. The results are presented in the Table 4.3.

TABLE 4.3: UDP Value for Jitter and Out of Order Packet % (10s Time Interval)

| **Bandwidth (Mbps)** | **Jitter (ms)** | **Out of Order Packets %** |
|---|---|---|
| 5 | 0.031 | 0.2359 |
| 10 | 0.027 | 0.2796 |
| 20 | 0.032 | 0.1711 |
| 30 | 0.06 | 0.2218 |
| 40 | 0.035 | 0.1583 |
| 5 | 0.031 | 0.2359 |

Figure 4.8 and 4.9 shows the graphical representation of results.

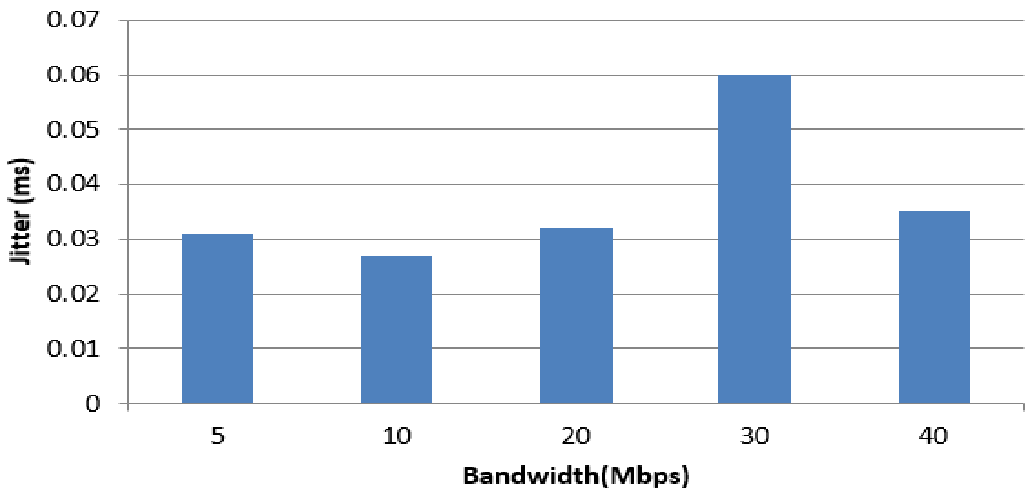


FIGURE 4.8: Bandwidth/Jitter

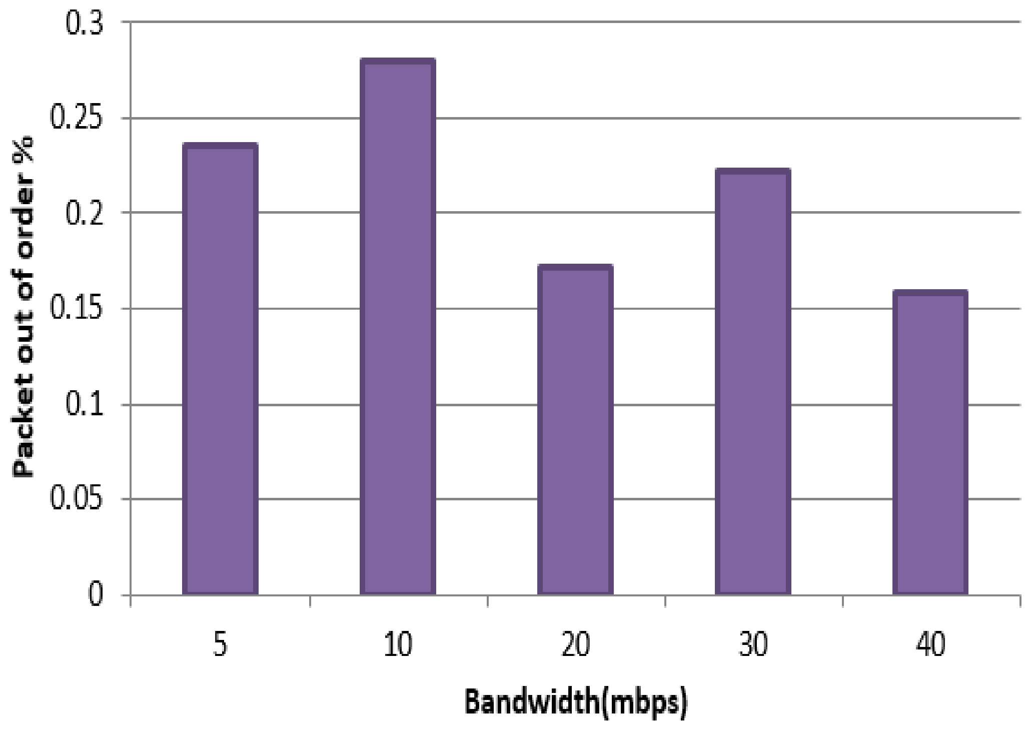


Figure 4.9: Bandwidth/Out of Order Packet %

The next result is taken by keeping the Bandwidth constant i.e. 20Mbps and varying the time Interval. The results are presented in Table 4.4.

Table 4.4: UDP Value for Jitter and Out of Order Packet % (20Mbps)

| **Time Interval (sec)** | **Jitter (ms)** | **Out of Order Packets %** |
|---|---|---|
| 10 | 0.003 | 0.21 |
| 20 | 0.001 | 0.052 |
| 30 | 0.006 | 0.012 |
| 40 | 0.041 | 0.081 |
| 50 | 0.001 | 0.077 |
| 60 | 0.007 | 0.038 |

Figure 4.10 and 4.11 represents graphical representation of Time Interval/ Jitter and Out of Order Packet %.

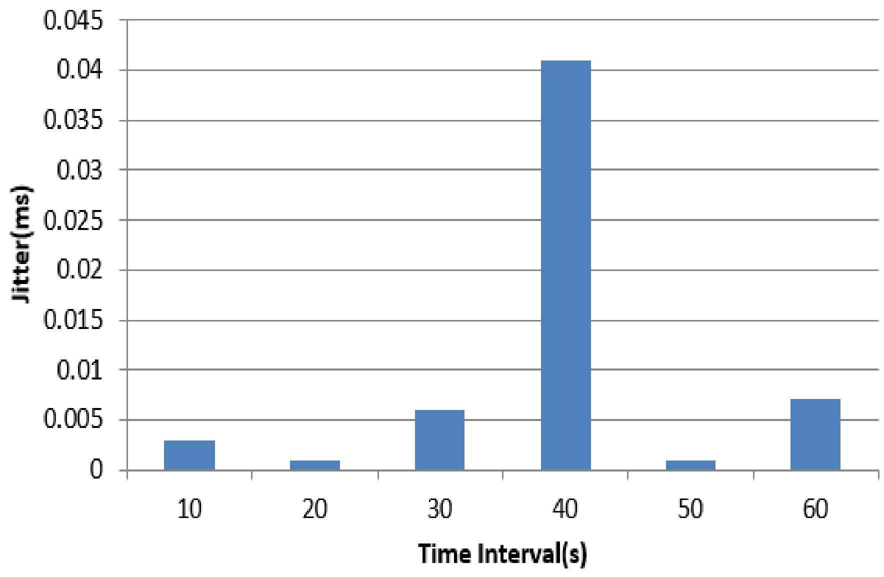


FIGURE 4.10: Time Interval/Jitter

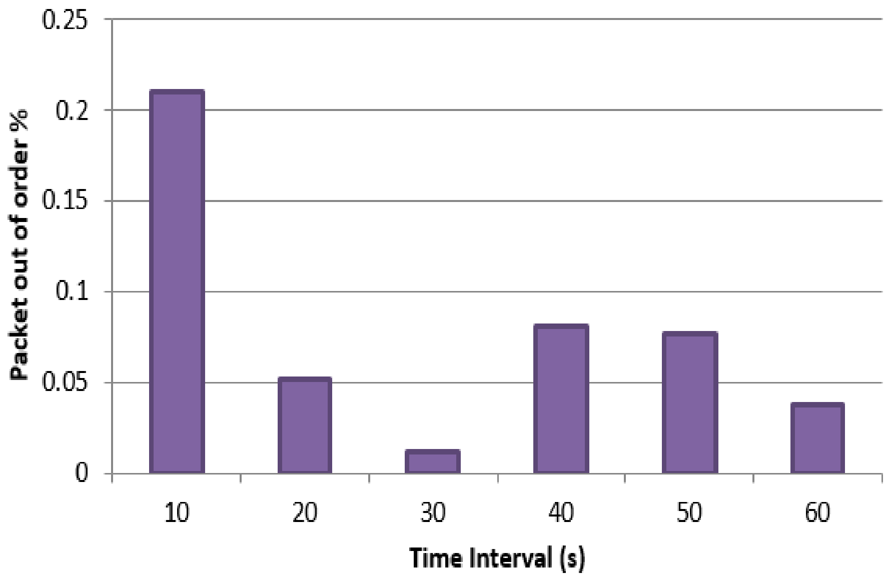


FIGURE 4.11: Time Interval/Out of Order Packet %

## 4.3 Comparison

We have compared our approach and results with dataset of existing approach proposed by Monir et al. [8]. They did the translation by introducing hardware and we implemented the same translation process on software level. Table 4.5 represents the comparison of results of our proposed approach and previous approach [8].

TABLE 4.5: Comparison of Results (w.r.t Bandwidth)

| **Previous Approach** | | | **Proposed Approach** | | |
|---|---|---|---|---|---|
| **Bandwidth (Mbps)** | **Jitter (ms)** | **Out of Order Packets %** | **Bandwidth (Mbps)** | **Jitter (ms)** | **Out of Order Packets %** |
| 5 | 0.1126 | 0.4001 | 5 | 0.031 | 0.2359 |
| 10 | 0.1502 | 1.0620 | 10 | 0.027 | 0.2796 |
| 20 | 0.1703 | 0.9701 | 20 | 0.032 | 0.1711 |
| 30 | 0.0612 | 1.8401 | 30 | 0.06 | 0.2218 |
| 40 | 0.3596 | 0.6228 | 40 | 0.035 | 0.1583 |

We can clearly see from the Table 4.5 that the jitter and out of order packet % encountered through our proposed approach is quite less than that of previous approach. As the implementation is at software level so packet doesn't have to wait on any extra hardware for processing. The graph in Figure 4.12 and Figure 4.13 represents in a better way.

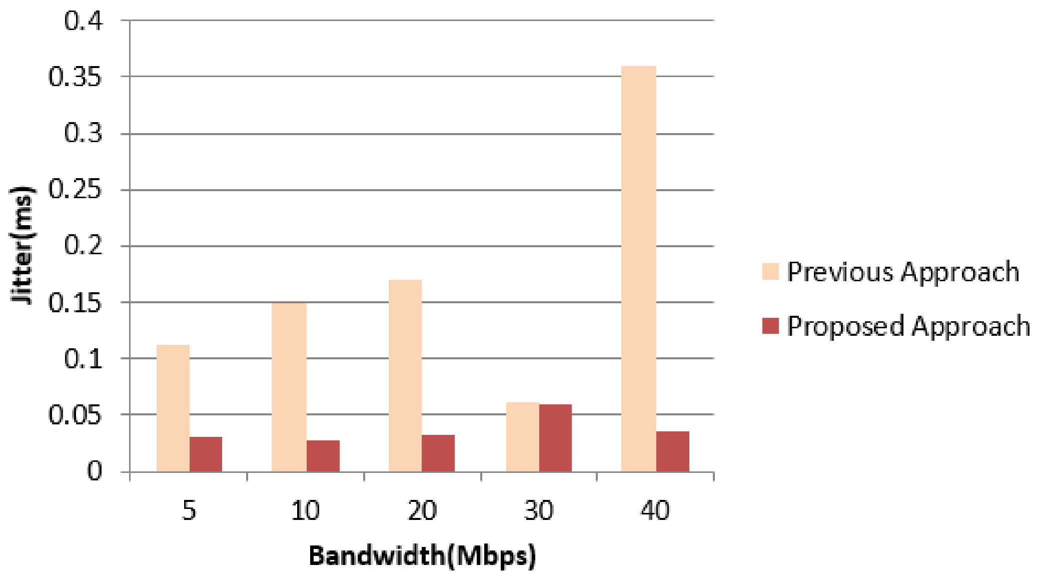


FIGURE 4.12: Comparison of Previous and Proposed Approach (Bandwidth/Jitter)

From graph represented in Figure 4.12, we can see that maximum jitter through our approach is approx. 0.06 ms which is less than the least jitter encountered through previous approach. Figure 4.13 shows the graphical representation of comparison for Bandwidth/ Out of Order Packet%.

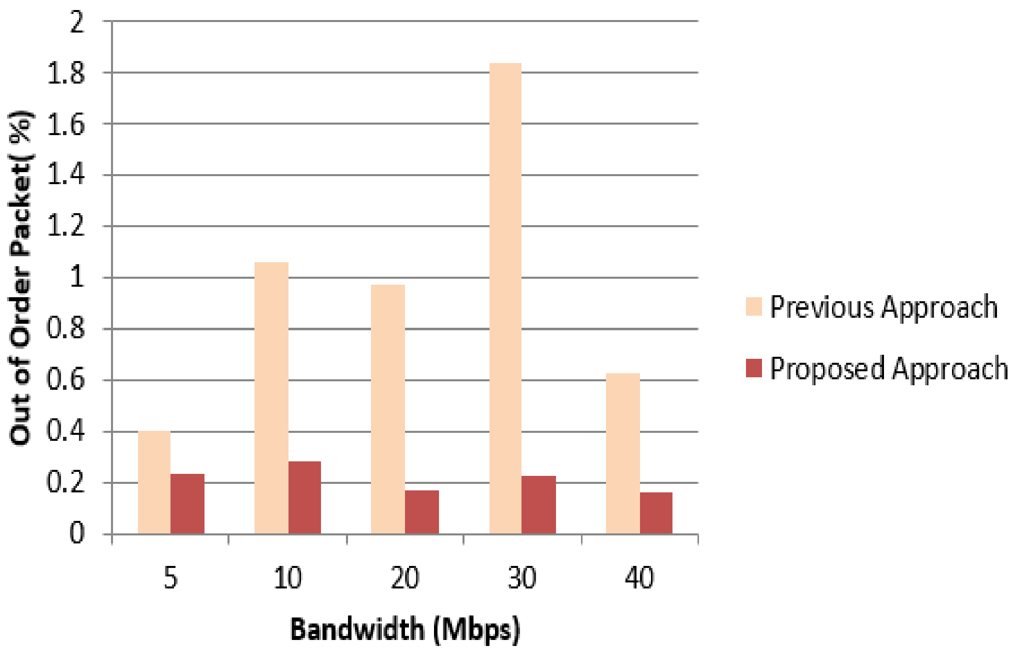


FIGURE 4.13: Comparison of Previous and Proposed Approach (Bandwidth/Out of Order Packet %)

It is clear from the Figure that Packet Out of Order% has decreased with our approach. The highest value here is 0.23% at 10 Mbps transfer rate while with previous approach it is 1.14% for the same Bandwidth.

Table 4.6 presents the comparison of difference for jitter and Out of Order Packet % for variable values of Time interval.

TABLE 4.6: Comparison of Results (w.r.t Time Interval)

| **Previous Approach** | | | **Proposed Approach** | | |
|---|---|---|---|---|---|
| **Time Interval (s)** | **Jitter (ms)** | **Out of Order Packets%** | **Time Interval (s)** | **Jitter (ms)** | **Out of Order Packets%** |
| 10 | 0.024 | 0.39 | 10 | 0.003 | 0.21 |
| 20 | 0.032 | 0.49 | 20 | 0.001 | 0.052 |
| 30 | 0.038 | 0.3 | 30 | 0.006 | 0.012 |
| 40 | 0.075 | 0.17 | 40 | 0.041 | 0.081 |
| 50 | 0.029 | 0.22 | 50 | 0.001 | 0.077 |

From the table we can see that performance difference is also depicted here. The graphs represent the difference in a better way.

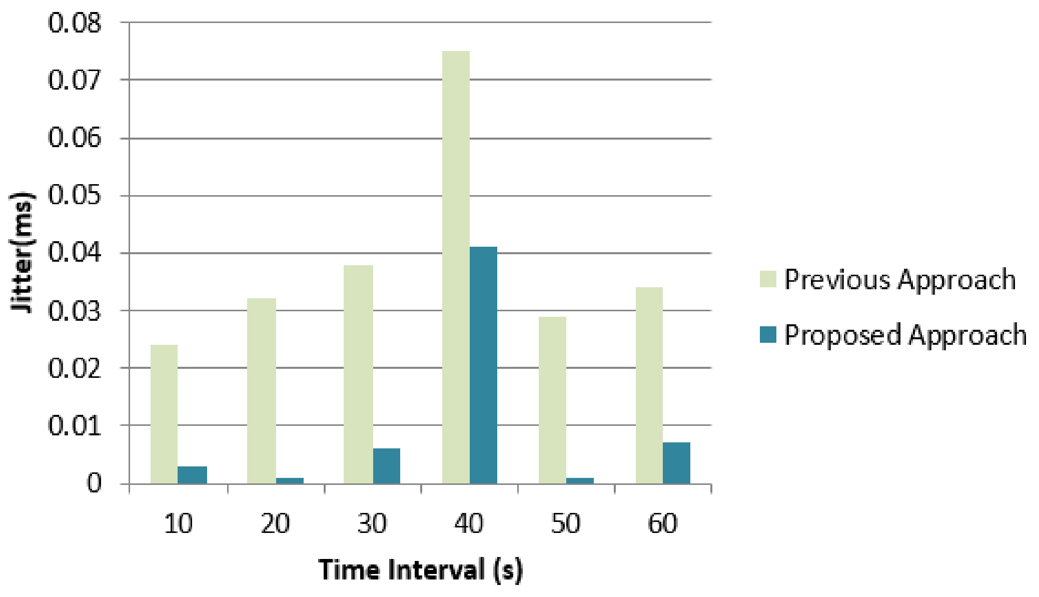


FIGURE 4.14: Comparison of Previous and Proposed Approach (Time Interval/Jitter)

From the Figure 4.14, we can see that the maximum value of jitter through pervious approach is 0.075ms for 40Mbps but through our approach it is fairly low.

Next Figure shows the graphical representation of Out of Order Packet

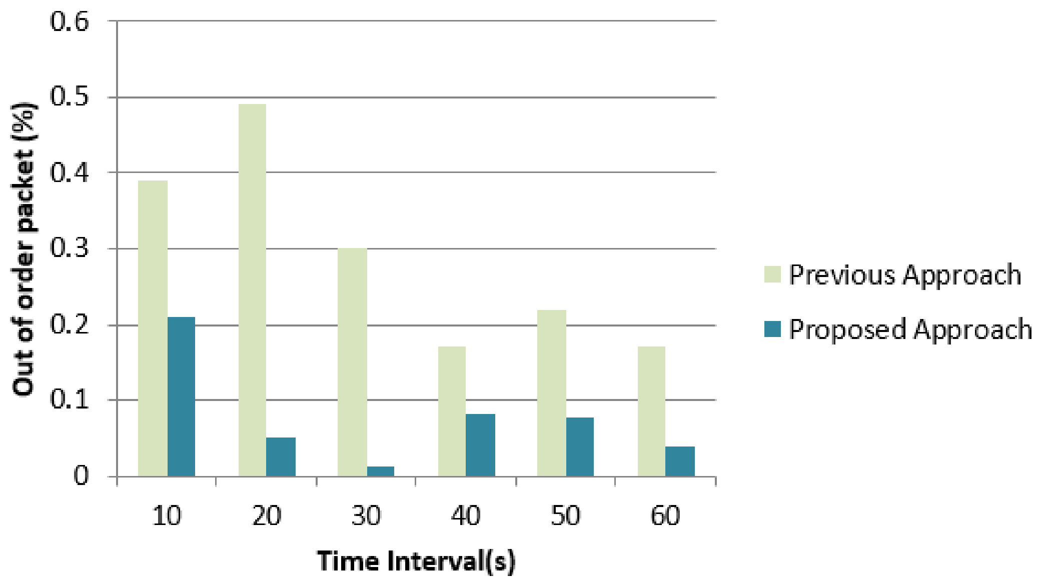


FIGURE 4.15: Comparison of Previous and Proposed Approach (Time Interval/Out of Order Packet %)

From the graph we can clearly see that the peak value for achieved through our approach is 0.21% but still it is much less than the previous approach.

Following are the answers of research questions mentioned in Chapter 1 which we have identified after conducting literature review and experiments.

**RQ1: How organizations can be interconnected using their private IPs and without collision due to same addresses?**
Organizations can connect using their private IPs if they agree on placing a registry on their routers which is not a feasible and scalable solution. Besides if any external hardware is used for IP translation purpose then processing on hardware cause workload and adds delay. Our solution is based on taking advantage of the underlying concept of Software Define Networking. NAT application is developed on SDN controller which aims to help enterprises by automating the connectivity with co-location points as per their needs. Rules are installed by controller on the OpenFlow switches and consumer can get resources from any provider. Consumer initiates request using private IP and the controller changes IP of requested packet and inject it on network. This way no two IPs remain same because all the OpenFlow switches work under controller and redirect their request to controller.

**RQ2: Which SDN controller can perform best when interconnecting organizations? What could be parameters for comparing and selecting controller?**
Different open source SDN controllers are available with different specifications. In depth study of different controllers is conducted with formulation of parameters for comparison. As controller controls whole network and keeps a central view of network so central architecture is required. Beside this the ease of implementing any new service according to the need and demand of new applications, low learning curve with a lot of developer's community support and documentation are the specifications of parameters for selecting controller. Based on these requirements Floodlight SDN controller is selected for this thesis.

**RQ3: How service availability and disaster management can be assured in case of particular service provider goes down?**
Set of available and offered services (from Colo-providers) differ from site to site. In case any particular site goes down then how those services can be accessed

and continuity of business (COB) can be assured. One way could be to get two connections one primary and other COB by paying double. If primary goes down then COB connection can be accessed as backup. Other way could be to automate the process of accessing same service by using intelligent system which connects to backup automatically when primary goes down. Our approach focuses on the second way for accessing service. If service provider goes down or have performance issue, then proposed solution dynamically connects the consumer to the service provide and hence assues service availability.

# Chapter 5

# Conclusion and Future work

The main goal of this thesis is to make interconnection process of organization easy and feasible especially when organizations are trying to migrate their businesses over cloud. In particular we have focused on interconnecting private organizations where they may communicate with each other using the novel paradigm of networking called as SDN. The growth of internet has increased tremendously in recent years. Networking requirements have changed with the emergence of new type of services and applications like social media, big data, cloud and mobile computing and internet of things. Changing such a sizeable network according to new requirements and assigning unique IPs to all devices is not easy. Software defined technologies are considered to be a valid solution for solving the difficulties in current network. After critically analyzing the literature, we have observed that traditional networking use NAT gateways for IP conversion. Even in SDN environment, IP conversion process is taking place at NAT gateways.

We argue that using NAT gateways or any extra hardware for IP conversion in SDN is not a feasible solution because it adds delay while processing the packet, degrading the performance of SDN. Another disadvantage of placing any hardware for IP conversion is that it adds extra cost and is not scalable.

Our proposed solution is based on implementing NAT service on SDN controller. By using the important feature of SDN that is about implementing services at software level, we can ease the process of IP conversion. Floodlight controller is used for implementing NAT service because it has centralized architecture with modular approach. Well documentation, developer's community support and low learning curve makes it suitable for implementing new service. Results have shown that by implementing NAT program on controller, less delay and out of order percentage has received.

Overall findings of our study are : implementation of NAT service on controller so changes can be made at software level without disturbing hardware, link convergence using multipath routing due to which services can be accessed from alternative paths if one link goes down and overall less delay and out of order packet %.

## 5.1 Future Work

Currently controller is doing routing and convergence based on reachability. If failure happens then it goes to other switch and ensures reachability. In future we will try to enhance the controller's capability to not only deal with link failure metric but also with other metrics like performance metric, business metric, delay and cost offered for same services available at different Co-Location points. The controller should be smart enough to connect to other point offering minimum price level for same service. More controllers can also be added in future for achieving quality of service.

In current scenario convergence of traffic as link goes down is happening after 10-15 sec. In future we will try to implement some other protocol for quicker detection of link down and helps in converging traffic quickly.

Not all the switches working in organizations are OpenFlow switches or OpenFlow compliant. We aim to extend our work and find a path where the solution can work with their existing switch product. Algorithm can be designed to expand the capabilities to work with the switches which may or may not be OpenFlow complaint.

# Bibliography

[1] O. Michel and E. Keller, "Sdn in wide-area networks: A survey," in *2017 Fourth International Conference on Software Defined Systems (SDS)*. IEEE, 2017, pp. 37–42.

[2] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–9.

[3] H.-C. Wang, C. Chen, and S.-H. Lu, "An sdn-based nat traversal mechanism for end-to-end iot networking," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2019, pp. 1–4.

[4] L. L. Zulu, K. A. Ogudo, and P. O. Umenne, "Emulating software defined network using mininet and opendaylight controller hosted on amazon web services cloud platform to demonstrate a realistic programmable network," in *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*. IEEE, 2018, pp. 1–7.

[5] G. Kim, J. Kim, and S. Lee, "An sdn based fully distributed nat traversal scheme for iot global connectivity," in *2015 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2015, pp. 807–809.

[6] J. W. J. L. C. M. Peng Wu, Yong Cui, "Transition from ipv4 to ipv6: A state-of-the-art survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1407–1424, 2012.

[7] C. Trois, M. D. Del Fabro, L. C. de Bona, and M. Martinello, “A survey on sdn programming languages: Toward a taxonomy,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2687–2712, 2016.

[8] M. F. Monir, R. Uddin, and D. Pan, “Behavior of napt middleware in an sdn environment,” in *2019 4th International Conference on Electrical Information and Communication Technology (EICT)*. IEEE, 2019, pp. 1–5.

[9] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, “Software defined networking: State of the art and research challenges,” *Computer Networks*, vol. 72, pp. 74–98, 2014.

[10] Y. Sinha, K. Haribabu *et al.*, “A survey: Hybrid sdn,” *Journal of Network and Computer Applications*, vol. 100, pp. 35–55, 2017.

[11] R. di Lallo, G. Lospoto, M. Rimondini, and G. Di Battista, “Supporting end-to-end connectivity in federated networks using sdn,” in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 759–762.

[12] M. Bano, U. A. Qureshi, R. N. B. Rais, M. Tufail, and A. Qayyum, “Miracle: An agile colocation platform for enabling xaas cloud architecture,” in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2019, pp. 604–610.

[13] P. Srisuresh, B. Ford, and D. Kegel, “State of peer-to-peer (p2p) communication across network address translators (nats),” *Internet Engineering Task Force, Request For Comments*, 2008, accessed: Aug. 04,2020. [Online]. Available: https://www.hjp.at/doc/rfc/rfc5128.html

[14] Definition - what does control plane mean? Accessed: Aug. 03, 2020. [Online]. Available: https://www.techopedia.com/definition/32317/control-plane

[15] Control and data planes. Accessed: Aug. 03, 2020. [Online]. Available: https://networkdirection.net/articles/network-theory/controlanddataplane/

[16] E. Conrad, S. Misenar, and J. Feldman, "Chapter 4 - domain 4: Communication and network security," in *Eleventh Hour CISSP® (Third Edition)*, third edition ed., E. Conrad, S. Misenar, and J. Feldman, Eds. Syngress, 2017, pp. 95 – 116. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128112489000048

[17] Sdn controller (software-defined networking controller). Accessed: Aug. 03, 2020. [Online]. Available: https://searchnetworking.techtarget.com/definition/SDN-controller-software-defined-networking-controller#:~:text=An%20SDN%20controller%20is%20an,switches%20where%20to%20send%20packets.

[18] What is openflow? definition and how it relates to sdn. Accessed: Aug. 03, 2020. [Online]. Available: https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/

[19] Ccna 7.7.c: Northbound and southbound apis. Accessed: Aug. 03, 2020. [Online]. Available: https://www.econfigs.com/ccna-7-7-c-northbound-and-southbound-apis/

[20] Restful api (rest api). Accessed: Aug. 03, 2020. [Online]. Available: https://searchapparchitecture.techtarget.com/definition/RESTful-API

[21] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, 2013, pp. 15–26.

[22] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[23] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[24] What is an sdn controller? definition. Accessed: Aug. 03, 2020. [Online]. Available: https://www.sdxcentral.com/networking/sdn/definitions/what-is-sdn-controller/

[25] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[26] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications surveys & tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[27] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: decoupling architecture from infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 43–48.

[28] R. Sahba, "A brief study of software defined networking for cloud computing," in *2018 World Automation Congress (WAC)*. IEEE, 2018, pp. 1–5.

[29] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for iot," *Ieee Access*, vol. 6, pp. 115–124, 2017.

[30] M. Manel, Y. Habib *et al.*, "An efficient mpls-based source routing scheme in software-defined wide area networks (sd-wan)," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2017, pp. 1205–1211.

[31] P. L. Gallegos-Segovia, J. F. Bravo-Torres, P. E. Vintimilla-Tapia, J. O. Ordonez-Ordonez, R. E. Mora-Huiracocha, and V. M. Larios-Rosillo, "Evaluation of an sdn-wan controller applied to services hosted in the cloud," in *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*. IEEE, 2017, pp. 1–6.

[32] R. E. Mora-Huiracocha, P. L. Gallegos-Segovia, P. E. Vintimilla-Tapia, J. F. Bravo-Torres, E. J. Cedillo-Elias, and V. M. Larios-Rosillo, "Implementation of a sd-wan for the interconnection of two software defined data centers," in *2019 IEEE Colombian Conference on Communications and Computing (COLCOM)*. IEEE, 2019, pp. 1–6.

[33] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[34] Odl: Platform overview. Accessed: Aug. 03, 2020. [Online]. Available: https://www.opendaylight.org/what-we-do/odl-platform-overview

[35] S. Badotra and J. Singh, "Open daylight as a controller for software defined networking." *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, 2017.

[36] M. Bjorklund *et al.*, "Yang-a data modeling language for the network configuration protocol (netconf)," 2010, accessed: Aug. 03, 2020. [Online]. Available: https://www.hjp.at/doc/rfc/rfc6020.html

[37] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 13–18.

[38] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*, 2012.

[39] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[40] M. Paliwal, D. Shrimankar, and O. Tembhurne, “Controllers in sdn: A review report,” *IEEE Access*, vol. 6, pp. 36 256–36 270, 2018.

[41] D. Erickson, “Home-beacon-confluence 2013,” accessed: Aug. 01, 2020. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home.html

[42] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, “Advanced study of sdn/openflow controllers,” in *Proceedings of the 9th central & eastern european software engineering conference in russia*, 2013, pp. 1–6.

[43] A. Giorgetti, A. Sgambelluri, R. Casellas, R. Morro, A. Campanella, and P. Castoldi, “Control of open and disaggregated transport networks using the open network operating system (onos),” *IEEE/OSA Journal of Optical Communications and Networking*, vol. 12, no. 2, pp. A171–A181, 2019.

[44] A. Giorgetti, R. Casellas, R. Morro, A. Campanella, and P. Castoldi, “Onos-controlled disaggregated optical networks,” in *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE, 2019, pp. 1–3.

[45] Onos: Distributed operation. Accessed: Aug. 03, 2020. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Distributed+Operation

[46] Floodlight controller. Accessed: Aug. 03, 2020. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview?homepageId=1343545

[47] L. V. Morales, A. F. Murillo, and S. J. Rueda, “Extending the floodlight controller,” in *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE, 2015, pp. 126–133.

[48] Sdn series part five: Floodlight, an openflow controller. Accessed: Aug. 03, 2020. [Online]. Available: https://thenewstack.io/sdn-series-part-v-floodlight/

[49] Floodlight architecture. Accessed: Aug. 03, 2020. [Online]. Available: https://thenewstack.io/sdn-series-part-v-floodlight/

[50] S. Asadollahi, B. Goswami, and M. Sameer, "Ryu controller's scalability experiment on software defined networks," in *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC).* IEEE, 2018, pp. 1–5.

[51] Comparison of software defined networking (sdn) controllers. part 5: Ryu. Accessed: Aug. 03, 2020. [Online]. Available: https://aptira.com/comparison-of-software-defined-networking-sdn-controllers-part-5-ryu/

[52] How to write a module. Accessed: Aug. 03, 2020. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343513/How+to+Write+a+Module

[53] B. G. Saleh Asadollahi, "Experimenting with scalability of floodlight controller in software defined networks," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT).* IEEE, 2017.

[54] Mininet overview. Accessed: Aug. 04, 2020. [Online]. Available: http://mininet.org/overview/

[55] Apache ant. Accessed: Aug. 05, 2020. [Online]. Available: https://ant.apache.org/

[56] C.-C. Tseng, C.-L. Lin, L.-H. Yen, J.-Y. Liu, and C.-Y. Ho, "Can: A context-aware nat traversal scheme," *Journal of network and computer applications*, vol. 36, no. 4, pp. 1164–1173, 2013.

[57] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, pp. 20–40, 2013.