Bachelor’s thesis

Information Technology

NINFOS11

2017

Olanrewaju Oladunjoye

SOFTWARE DEFINED NETWORKING

– The Emerging Paradigm To Computer Networking

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2017 | 39

Olanrewaju Oladunjoye

SOFTWARE DEFINED NETWORKING

- An Emerging Paradigm To Computer Networking

Software Defined Networking (SDN) is an emerging paradigm in networking technology that enables innovation on how network systems are managed and designed. SDN plays a huge role in the effort to make computer networks programmable.

This thesis discusses the history and efforts to programmable and active networks, the early practices towards separating the control plane and data plane. Highlighting the architectural concepts in networking that software defined networks emanated as well as the history and evolution of software defined networking. The thesis further discusses how SDN simplifies the complexity of managing large and distributed network system. It also examines the technologies that support software defined networking such as Network Virtualization and OpenFlow. The thesis reviews the components of software defined network architecture: the data plane layer, the controller plane layer, and application layer. Furthermore, it reports the practical implementation of SDN in network functions virtualization. Finally, it depicts the notion of software defined networking, whose Application Programming Interface may be implemented by OpenFlow.

The practical aspect of this thesis shows a simple emulation of software-defined network architecture using Mininet network emulation tool. Hopefully, this work aims to benefit those who intend to learn about the fundamental principles of Software Defined Networking.

KEYWORDS:

Software Defined Networking, Network Function Virtualization, Controller Plane, Data Plane, Application Plane, Computer Networks,mininet

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS 5

1. INTRODUCTION 6

2. WHAT IS COMPUTER NETWORKING? 7

2.1 Architectural theme recognition in computer networking where software defined networking emanated. 7

3. HISTORY AND EVOLUTION OF SOFTWARE DEFINED NETWORKING 9

3.1 Central Network Control. 9

3.2 Network Programmability 10

3.3 Network Virtualization 11

4. COMPONENTS OF SDN ARCHITECTURE 17

4.1 The SDN Data Plane 19

4.2 The SDN Controller Plane 19

4.3 The SDN Application Layer. 20

5. PRACTICAL IMPLEMENTATION OF SOFTWARE DEFINED NETWORKING. 22

5.1 Network Functions Virtualization 22

5.2 A simple software defined network architecture. 26

6. CONCLUSION 36

REFERENCES 37

FIGURES

Figure 1. Multiple instances of Forwarding Plane Elements (FE) and Control Plane Elements (CE) in FoRCES Network Element (NE). (Doria, et al., 2010). 11
Figure 2. Open Signalling in the tempest (Van Der Merwe, Rooney, Leslie, & Crosby, 1998). 13
Figure 3. The switch divider (Bavier, Feamster, Huang, Peterson, & Rexford, 2006). 14
Figure 4. Shows the internet in a slice architecture running on VINI (Bavier, Feamster, Huang, Peterson, & Rexford, 2006). 15
Figure 5. Basic SDN architecture components (Open Networking Foundation, 2012). 18
Figure 6. NFV Domains encompassed in the network function virtualization infrastructure (Chiosi, Clarke, Willis , Reid, Feger, & Ruhl , 2012). 23
Figure 7. Vision for network function virtualization (Nadeau, 2013). 24
Figure 8. Network Functions that could be virtualized (Nadeau, 2013). 25
Figure 9. SDN network topology consisting of three hosts with class A IP addressing, an OpenFlow Switch, and OpenFlow refrence controller. 28
Figure 10. Starting mininet. 29
Figure 11. IP address assigned to all hosts, and switch links are up. 30
Figure 12. H1 cannot reach H2 and vice versa. 31
Figure 13. Ping failure among all host. 32
Figure 14. Manual Installation of flows on open switch flow table. 33
Figure 15. Succesful pings between host H1 and H2, while H3 is unrecheable because the switch flow table includes entries for only H1 and H2. 34
Figure 16. Starting OVS reference controller. 35
Figure 17. All host are now recheable from anywhere on the network. 35

LIST OF ABBREVIATIONS (OR) SYMBOLS

AS	Autonomous System
API	Application Program Interface
BGP	Border Gateway Protocol
CABO	Concurrent Architecture are Better Than One
CPE	Customer Premise Equipment
DCPI	Data-Controller Plane Interface
FIB	Forwarding Information Base
ForCES	Forwarding and Control Element Separation
IETF	Internet Engineering Task Force
LAN	Local Area Network
LIB	Label Information Base
MAN	Metropolitan Area Network
NBI	North Bound Interface
NCP	Network Control Point
NE	Network Elements
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
OSPF	Open Shortest Path First
PCE	Path Computation Engine
RCP	Routing Control Platform
RIB	Routing Information Base
RIP	Routing Information Protocol
SS7	Signalling System 7
VNF	Virtual Network Functions

1. INTRODUCTION

Over the years of Computer Network existence and growth, the drastic increase in network complexity has brought about difficulties in computer network management. It tends to be difficult in configuring computer network systems by predefined policies, reconfiguring networks to respond to changes, faults, and loads. As it appears that current network systems are integrated vertically, the control plane and data plane are all tied up together. Software Defined Networking is an emerging standard that portrays changes in the current state of computer networks, by separating the network of equipment's control logic from fundamental switches and routers, logical promotion of centralized network control, introducing computer network programmability (Kreutz, Ramos, Verissimo, Tothenberg, Azodolmolky, & Uhlig, 2014). With Software Defined Networking, comes the ease of creation and introduction of new abstractions in computer networking, network management simplification and facilitation of computer network evolution.

This thesis will show the architectural theme recognition in computer networking where Software Defined Networking (SDN) emanated. Furthermore, the history on how networks start to become active and programmable to show the differences and improvement in traditional computer networking technologies compared to a software defined network.
And also showing the benefits of software defined networks are implemented with the aid of network virtualization. (Rostami, 2014)

2. WHAT IS COMPUTER NETWORKING?

Computer Networking is the practice of linking one or more computer systems together in other to share data. Computer systems comprise of computer hardware devices and software.Computer network is a group of one or more computer devices linked to each other. computer networks can be categorized into different types, including:
Local Area Networking(LANs): Computer systems linked together in one geographical area. For instance, in the same building or room.
Wide Area Networks(WANs): Connecting computer systems which are distanced, far apart, i.e., different geographical area.
Metropolitan Area Networks(MANs): A network implemented or designed for a city or town, for example, Spark net.
Additionally, computer networks can also be categorized based on the following characteristics:
Topologies: The logical or physical arrangement of computer systems. Types of topologies include: bus topology, star topology, ring topology, and mesh topology
Protocol: A protocol is the definition of the sets of rules and signals that network computers use for communication. For example, Ethernet is a popular network for LANs.
Architecture: It is the classification of network based on the model of connection, that is, either peer to peer or client/server architecture.
Computers on a network are often referred to as nodes and hosts. Also, a computer which provides resources in a network environment is referred to as servers.
In conclusion, two or more computers connected to each other with the ability to communicate is simply known as a computer network. (Florida Center for Instructional Technology, 2013)

2.1 Architectural theme recognition in computer networking where software defined networking emanated.

Software Defined Networking(SDN): A new computer networking paradigm where network behavior is controlled by a single high-level software program. A Software-Defined Network is a network architecture in which the control-plane (software programs which control the network system behavior) and data plane(devices that forward network traffic) are separated. An emerging network paradigm where network control is being decoupled from hardware such that the behavior a logically centralized software program controls the overall network behavior (Kreutz, Ramos, Verissimo, Tothenberg, Azodolmolky, & Uhlig, 2014).

Control Plane

In a typical network, the control plane is a software function that checks and manages the network behavior such as network paths and forwarding behavior. Also, It is known as a high-level software controller.

Data Plane

Network functions responsible for making decisions on forwarding traffic. The data plane is typically instantiated as forwarding tables on routers, firewalls, switches, and middleboxes.

Active Networks

In the 1990s, Active Networks are a collection of network architectures that shared the same goals as software defined networking and motivated by the widespread use of the internet (applications) and complexity in testing new ideas in real (Rostami, 2014). In active networks, communication patterns allow packets to flow through the network infrastructure to dynamically modify the network systems operation. Active networking is an attempt to make networks programmable. For instance, the programmable switches in an active network perform custom computation or processing functions on packets as it travels through those switches.

Network Virtualization

Representing many distinct logical networks on a single shared physical network infrastructure. For example, Virtual Network as implemented in The Tempest - practical framework for network programmability (Van Der Merwe, Rooney, Leslie, & Crosby, 1998) .

In today's technology, outburst of smart mobile devices, Internet of things (IOT), everything as a service, cloud computing and tremendous user's need for content, mobile broadband which all results to increasing necessity for data transfer over the trillions of connected networks which are practically hard-wired networks, brings about difficulties in network scalability and complexity in network management (Benson, Akella, & Maltz, 2009).

3. HISTORY AND EVOLUTION OF SOFTWARE DEFINED NETWORKING

The evolution of Software Defined Networking portrays the timeline of SDN from the 1980s till present, puts forward the principles behind SDN, and gain awareness about SDN ideas.
However, SDN principles are not entirely new, this evolution recognizes the architectural themes as seen in a packet switched network, initiatives on separation of the data and control plane signaling, for example, AT&T introduced the Network Control Point towards improving management and monitoring of its telephone network (Kreutz, Ramos, Verissimo, Tothenberg, Azodolmolky, & Uhlig, 2014).
This thesis will take a look at the concepts underlying the evolution of SDN - The Central Control (logical centralization of network control), Programming Networks (the ability to manage the computer network systems via software controller programs), Network Virtualization (possible forms of network virtualization that emerge to this paradigm shift in networking).

3.1 Central Network Control.

The origins of central network control date back to the 1980s in the form of AT&T's Network control point. In the early days, control and data planes operated together in the same channel, a paradigm known as in-band signaling where data and control planes are sent over the same channel, at certain frequencies in this channel, an example is the BLUE BOX which uses the in-band signaling mechanism to do things like resetting the phone trunk lines, pulses on the line could be used to route calls and set up sockets for calls. The resulting network happens to turn out to be brittle, insecure, etc. (Wikipedia, 2009).
The earliest initiatives in an attempt to decouple control and data plane signaling, AT&T introduced the Network Control Point, to improve the monitoring and management of its Telephone Network. This attempt brought about a faster innovation pace in the network and offered means for efficiency improvement (Sheinbein & Weber, 1982).
In exploring a solution to centralize the network control, other concepts and initiatives are traced back to the Routing Control Platform (RCP) (Feamster, Balakrishnan, Rexford, Shaikh, & Van der Merwe, 2004), which is a solution for interdomain routing control in IP-networks. RCP calculates Border Gateway Protocol (BGP) routes, for Autonomous System (AS), at a centralized server to give transit network operators greater control over BGP routing decision making (Caesar, Caldwell, Feamster, Rexford, Shaikh, & Van der Merwe, 2005). The RCP effectively used BGP as a control channel so that the forwarding elements thought that they were talking to another router but the network uses a single point centralization. Other concepts of centralized network control can be traced in the traditional telecommunications networks, which includes Signalling System 7 (SS7) and Intelligent Network.

Signalling System 7 (SS7)

An out-of-band signaling system that isolates the communication channel for management information from the user data's path, and a cleavage between user data and control into separate channel or links (Russell, 1998).

Intelligent Network (IN) architecture

On the other hand, is a network functionality support which enables call placements to invoke a query to the Detection Points that escape out of normal call handling and that cites per-service code (International Telecommunication Union, 1993).

3.2 Network Programmability

Network Programmability has a long history which provides a solution to the need for adaptation and flexibility in the network systems. This thesis will address the approaches and initiatives towards programmable network efforts, which laid a foundation to many of the SDN concepts that emerged.
Active networks, which is an early effort towards building new network architectures based on network programmability concepts. In such network architectures, switches perform custom computations on packets as it travels through the network, or modify the packet content if necessary. Active Networks portrays two methods: Programmable Switches approach and capsules approach.
The programmable switch approach does not make any changes to the existing packet format but uses a different mechanism by assuming that switching devices download and execute active networking programs with specific instructions on how packets are processed.
The capsule approach adds small programs to the present packet format, the packets are encapsulated in transmissions frames, and code embedded in the packets are prepared and executed at each active network node along the packet's' path.
Examples of an active network are routers on a network performing tracing program on a packet as the packets travel through the routers. MiddleBoxes are also examples of active networks, boxes in the network that delivers firewalling, proxying, application services. These are custom computation on traffic that is performed in the network or as they travel through the network.
The active network which proposes a similar motivation of accelerating innovation in existing network systems so that technology could be introduced more rapidly without consensus standardization. It is observed that active nodes allow routers to download new services into the network infrastructure which motivates user-driven innovation. Active Networks poses a legacy for SDN with the idea of providing programmable functions in the network to enable innovation. One of the motivations for active networks was the proliferation of different kind of middleboxes and a vision of unified

architecture, these kinds of architecture can also be seen in various SDN projects (Tennenhouse, Smith, Wetherall, & Minden, 1997).
The approach to Network programmability has not only be limited to Active Networking, but the Internet Engineering Task Force (IETF) also initiated Forwarding and Control Element Separation (ForCES) (Yang, Dantu, Anderson, & Gopal, 2004), in the early 2000s which proposed an approach to network programmability in a different manner from Active Networking. ForCES define an architectural framework and associated protocols to standardize information exchange and separates control plane and forwarding plane in a ForCES Network Elements.

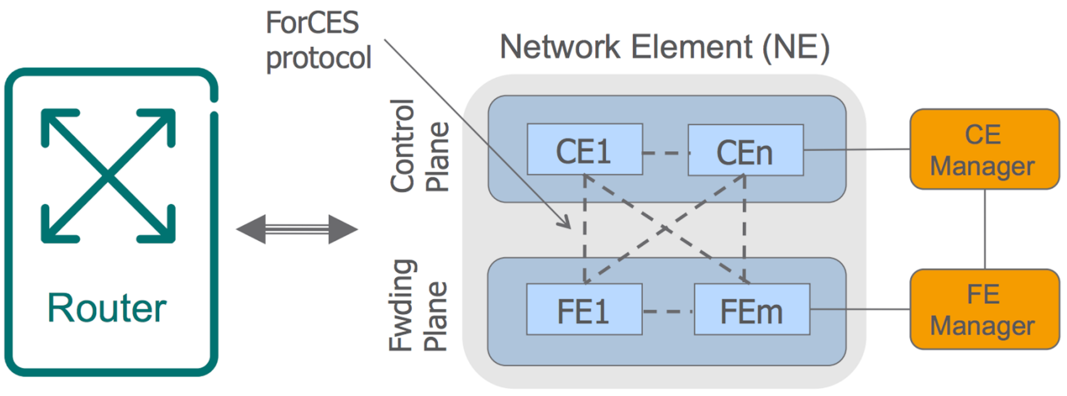


Figure 1. Multiple instances of Forwarding Plane Elements (FE) and Control Plane Elements (CE) in FoRCES Network Element (NE). (Doria, et al., 2010).

The standard essentially defines protocols that will allow multiple control elements to control forwarding elements which will essentially be responsible for forwarding packets, metering, shaping, performing traffic classification and so forth. The idea was that the switches and forwarding element could be controlled over a standard control channel called the ForCES interface and there might be multiple such controllers controlling the forwarding behavior of these forwarding elements. ForCES faced problems such as the requirement of standardization, adoption by vendors and deployment of new hardware (Doria, et al., 2010), (Yang, Dantu, Anderson, & Gopal, 2004).

3.3 Network Virtualization

What is Network Virtualization?

Network Virtualization is the representation of one or more logical network topologies on the same underlying physical infrastructure. There have been different instantiations of network virtualization such as virtual LANs(VLANs). This thesis will look into various technologies and different network testbeds that use and develop network virtualization that has mostly led to mature virtual network technology seen

in the form of companies and commercial products today. One of the benefits of network virtualization is sharing, one can instantiate multiple logical routers on top of a single physical node or a single platform, and one can instantiate multiple virtual networks on top of the same physical network infrastructure. This sharing requires the ability to isolate resources in terms of memory, CPU, bandwidth, forwarding tables and so forth. Also, Network virtualization offers a prospect of customizability in addition to sharing, users of a virtual network can get a view of their logical network and logical network topology that is separate from other logical network topology that may be running on the same underlying physical network infrastructure. The ability to see an independent logical network also allows capacity to run custom routing and forwarding software on that particular slice of the virtual network (Mosharaf Kabir Chowdhury & Raouf, 2010).
This thesis will explore three different examples of Virtual networks;

The Tempest architecture

Switchlets, separation of the control frameworks from the underlying switches itself and the capability to virtualize the underlying switch hardware to provide the appearance of multiple virtual switches (Van Der Merwe, Rooney, Leslie, & Crosby, 1998).

Virtual Network Infrastructure (VINI)

A Virtual Network Infrastructure provides a virtual network infrastructure so that experimenters could run experiments on their personal logical network system shared on the same underlying physical network topology (Bavier, Feamster, Huang, Peterson, & Rexford, 2006).

Concurrent Architecture are better than one (CABO)

A network architecture which use some of the vision of the emerging virtual network technologies to realize that virtual network could allow service providers to operate independently of the providers that make the underlying physical network infrastructure available (Feamster, Gao, & Rexford, How to lease the internet in your spare time, 2007).

The Tempest

Dated back in 1998, An initiative that introduces network virtualization by proposing the concepts of switchlets in ATM networks.Tempest adds the ability to control a given ATM switch with multiple controllers by partitioning the resources of that switch between those controllers, Switchlet is that partition, which is a small, but a complete switch to the controller (Van Der Merwe, Rooney, Leslie, & Crosby, 1998).

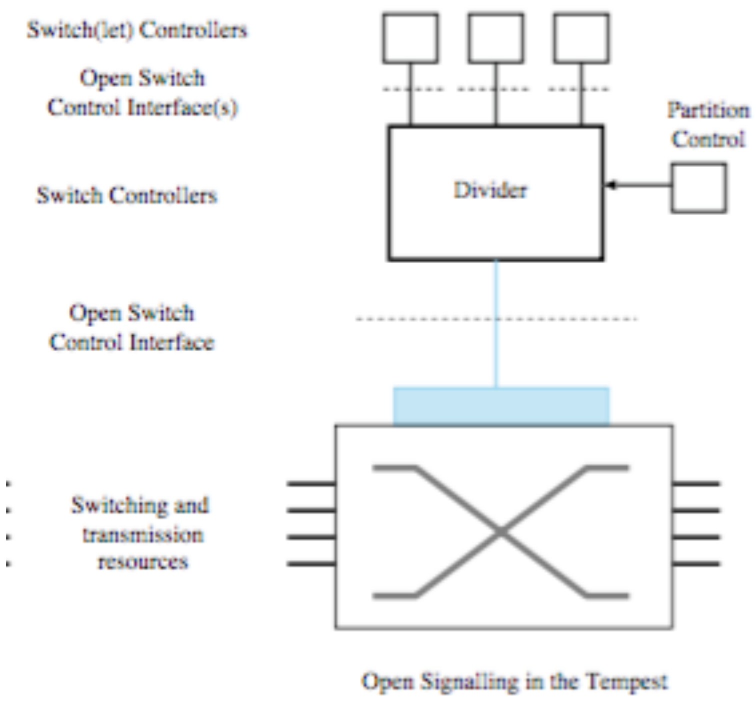

Switch(let) Controllers
Open Switch Control Interface(s)
Partition Control
Switch Controllers
Divider
Open Switch Control Interface
Switching and transmission resources
Open Signalling in the Tempest

Figure 2. Open Signalling in the tempest (Van Der Merwe, Rooney, Leslie, & Crosby, 1998).

Switchlets came out of the Tempest architecture where it has a single underlying switch with its resources and an open switch control interface that exposes those resources to their controllers. Switchlets allows multiple control architectures to operate over a single ATM network. The open control interface separated the switch controller and the fabric via an open signalling protocol, and the divider partitioned the switch resources to allow each multiple controllers to have their personal view of a logical switch (Van Der Merwe, Rooney, Leslie, & Crosby, 1998).

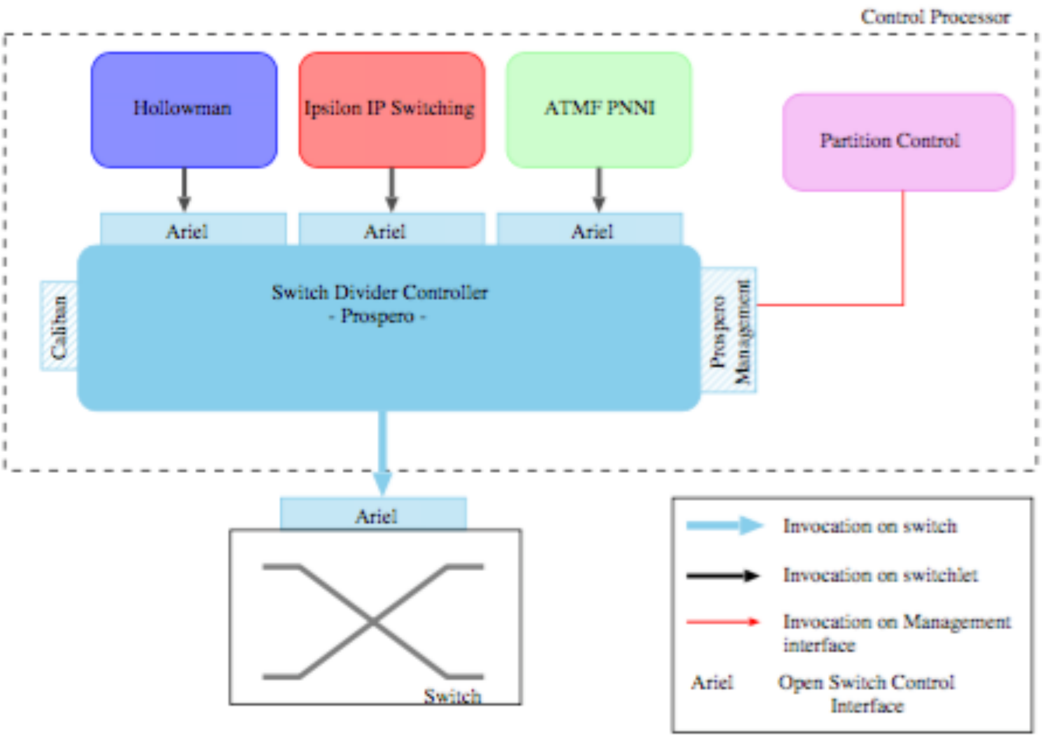


Figure 3. The switch divider (Bavier, Feamster, Huang, Peterson, & Rexford, 2006).

The switch divider, partition port space, bandwidth, and buffers, allow different controllers to control each switchlet. The tempest framework concluded that: by allowing multiple architectures to coexist, addresses the issues of migration and upgrading, that all network operators are familiar with (Van Der Merwe, Rooney, Leslie, & Crosby, 1998).

Virtual Network Infrastructure (VINI): Virtual network infrastructures can allow network experimenters and researchers to bridge the gap between small-scale simulations or experiments and real live deployments which were the motivation behind VINI. VINI runs real routing software and exposes realistic network conditions to the applications running on it. It gives control to the experimenter over different network events, such as failures, and also carries traffic on behalf of real users, it can also be shared among many different experimenters. However, VINI also uses the separation of the control and data plane to achieve some of its goals of network virtualization.

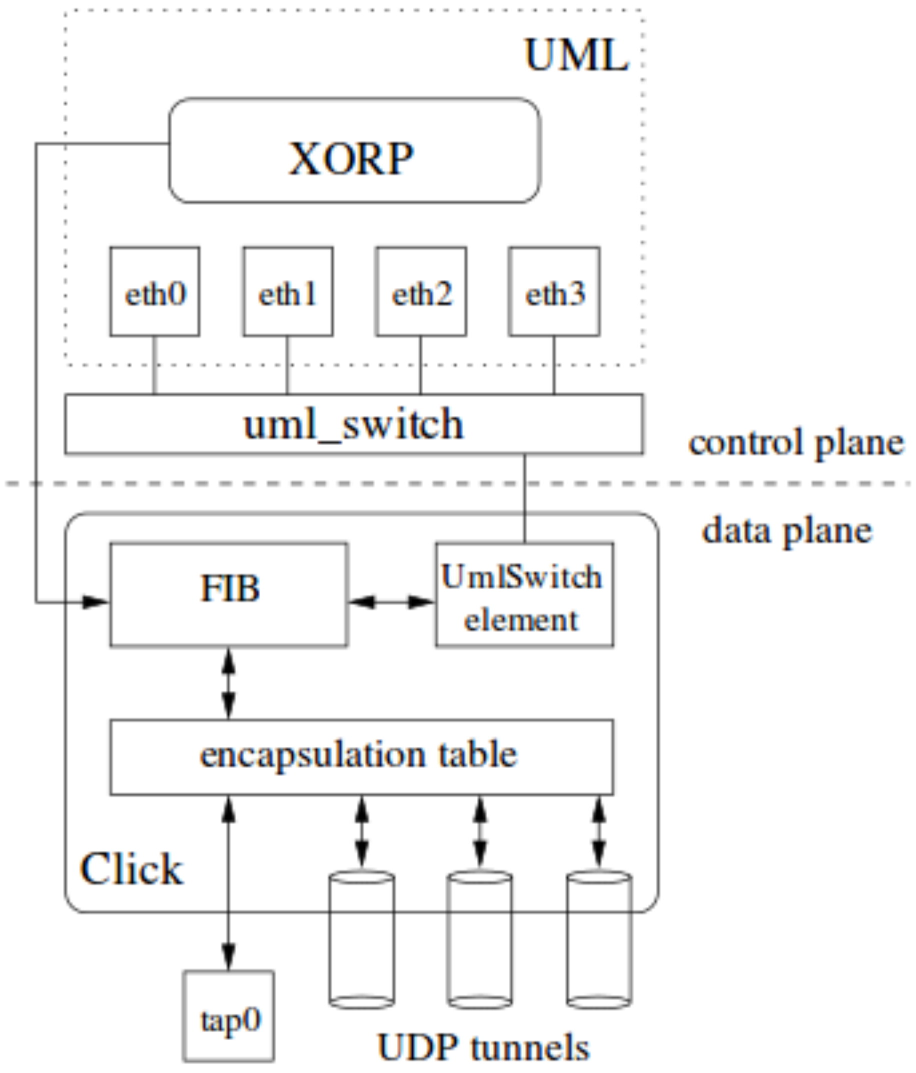


Figure 4. Shows the internet in a slice architecture running on VINI (Bavier, Feamster, Huang, Peterson, & Rexford, 2006).

The VINI control plane is software router called XORP, which runs a variety of different routing protocols with the goal of allowing experimenters to run real routing protocols on top of virtual network topologies. VINI's data plane provides the appearance of these virtual network topologies to experimenters, the data plane is implemented using a software router called Click, and the virtual interfaces were implemented using Tunnelling.
Tunnelling has also been used in many other virtual network technologies to create the appearance of virtual links. In VINI, experimenters could also deploy filters in front of these tunnels to create the illusion or appearance of a failing link. These filters essentially block packets on individual tunnels (Mosharaf Kabir Chowdhury & Raouf, 2010) (Bavier, Feamster, Huang, Peterson, & Rexford, 2006).

Concurrent Architecture are better than one (CABO)

CABO proposes an exploit in network virtualization which allows a service provider to simultaneously run multiple end to end services over infrastructure provider’s own equipment by decoupling infrastructure provider (those parties that maintain data centers, links routers, and other physical infrastructure) from services provider (that offers end to end services on top of that infrastructure) (Feamster, Gao, & Rexford, How to lease the internet in your spare time, 2007).

4. COMPONENTS OF SDN ARCHITECTURE

Separating Data Plane and Control Plane

The concept of separating the control plane from the data plane is not new in networking, for instance, the control plane or the brain of a multi-slot router or switch, built within the last ten years, executes on a dedicated processor or often two processors for redundancy. While the switching functions of the data plane perform independently on one or more line cards, which each of these cards has its individual dedicated processor (Stalling, 2016).
Separation of the data or forwarding plane and control plane is the fundamental characteristic of a software defined network. The coupling in conventional routers and switches which embodies a tight integration between the data plane and control plane portrays challenges in various management tasks such as monitoring or predicting routing behavior and debugging configuration problems. Software defined Networking addresses these challenges as efforts to separate the data plane, and the control plane emerges in this paradigm.
The trend in Separating the control plane and data plane catalyzed innovations such as;
An open interface between the forwarding or data plane and the control plane. For instance, ForCES (Forwarding and Control Element Separation) interface, an IETF standardization (Doria, et al., 2010).
Logically centralized programmatic control of the network, For example, Routing Control Platform (RCP) architecture (Feamster, Balakrishnan, Rexford, Shaikh, & Van der Merwe, 2004).
By definition, network management is a network system-wide activity, and it shows that segregating the control functionalities off of network equipment into different servers makes a lot of sense. The emergence of open source routing software, which reduced the barrier to designing prototype implementations and brought about possibilities for logically centralized routing controllers (Stalling, 2016).
The separation of the control plane from the data plane offers network operators advantages of centralized programmatic control and economic benefits such as the ability to consolidate complicated software to manage and configure commodity hardware.
Software Defined Networking aims towards the provision of open interfaces enabling software development that controls connectivity provided by an underlying set of network infrastructure or resources and network traffic that flow through them, as well as possible modification and inspection of traffic performed in the network (Chiosi, Clarke, Willis , Reid, Feger, & Ruhl , 2012).
The architecture of the SDN is based on Layer2 / Layer3 (L2/L3) switches architecture, which has the centralized controller system that controls the forwarding behavior for sets of distributed switches. According to the definition of SDN, its framework is treated as abstract and controlled by a programmable part with minimal interaction with the main network components (Nadeau, 2013).

A network architecture where network control (control plane) is logically centralized and decoupled from forwarding (data plane), and it is directly programmable. With software defined networking, the applications can be network aware, the SDN controller provides a view of the network state for applications and also translates application requirements to low-level rules (Feamster, Rexford, & Zegura, The Road to SDN: An intellectual history of programmable networks, 2014).

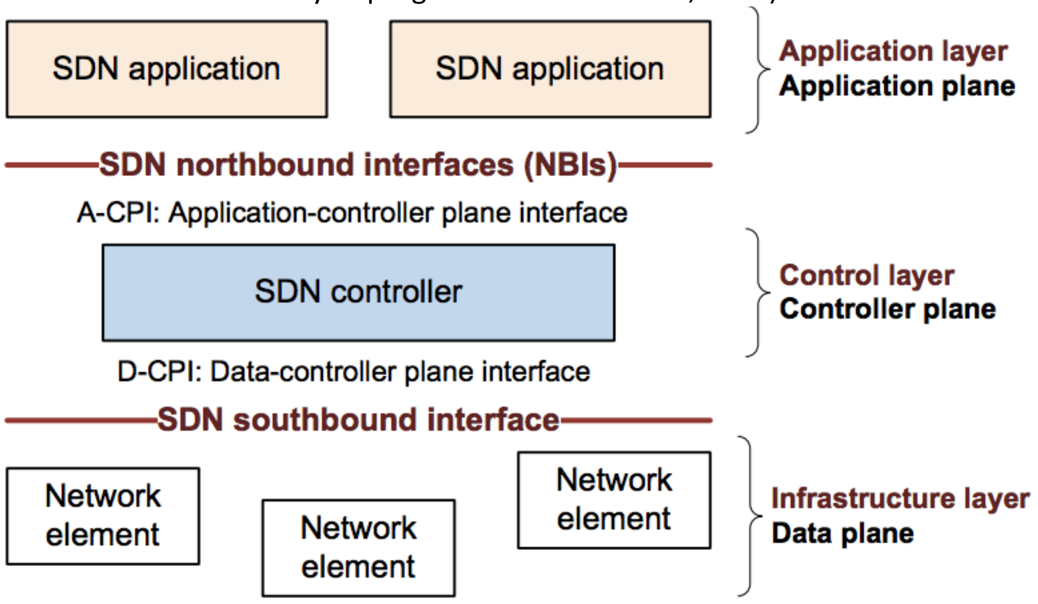


Figure 5. Basic SDN architecture components (Open Networking Foundation, 2012).

Figure 5 as similarly introduced in the ONF white paper, (Open Networking Foundation, 2012), depicts the basic SDN architecture components, which comprises of the infrastructure layer or data plane, control layer or controller plane and application layer or application plane. For the interaction of the three layers of the SDN, an open application program interface (API) is present and allows communication between them.

The data plane comprises of network elements, exposing their capabilities toward the controller plane through interfaces southbound from the controller, SouthBound Interface, also referred to as the data-control plane interface.

The application plane, where the SDN applications exist, communicates their network requirement to the control plane through northbound interfaces (NBIs) (Open Networking Foundation, 2014).

The SDN architecture offers a simplified and unified configuration for resources set, and its success can be significantly recognized if deployed within a preexisting, large and multilayer environment. While for the functionality of the SDN in a smaller environment, proper subsets can be profiled from its architecture (Nadeau, 2013).

4.1 The SDN Data Plane

The SDN data plane which is also referred in the ONF paper (Open Networking Foundation, 2014) as the infrastructure layer comprises of sets of network elements and or entities. At this layer, network elements perform the transport and processing of data packets by the decision made by the SDN controller plane; these decisions are forwarded to the data plane by the controller plane via the Data-controller plane interface (DCPI).
The data-controller plane interface (DCPI) is an application interface which defines the way the control plane interacts or take exclusive control over a set of resources exposed by network elements in the data or forwarding plane to make the necessary adjustment to meet business or networking needs. For instance, information exchanged through this interaction are such that include controlling information provided by the SDN controller plane to the data plane (e.g., policy provisioning or network resources configuration) (Open Networking Foundation, 2012).
The data plane resources are abstractions of the underlying physical network entities or capabilities, the data plane of an SDN is simply a system, a set of nodes with characteristics such as performing traffic forwarding and may also consume, produce, store or process traffic. Theses sets of nodes are network elements (NEs) and are interconnected by links. The Network Elements (NEs) provide external data plane ports to client equipment and other networks. As part of the SDN benefits are based on centralized control, an SDN controller will control more than one NE (Open Networking Foundation, 2012).
According to (Shin, Nam, & Kim, 2012), The data plane include various abstraction models such as

- Packet forwarding abstraction models (IPv4, IPv6, Ethernet, etc.).
- Circuit Switching abstraction Models(Optical, MPLS, etc.)
- Wireless Integration, characterization of wireless interfaces, flows, handover support
- Evolved packet core, LTE support.

4.2 The SDN Controller Plane

In a traditional network, i.e., a nonsoftware defined network in which the data plane and the control plane are not decoupled. The controller plane is a router component that determines how that particular individual box interacts with its neighbors with a state exchange. The Network OS in the router processes the Routing Information Base(RIB) and Label Information Base (LIB), which is used to populate the Forwarding Information Base (FIB) and Label Forwarding Information Base(LFIB). Router proprietors implement various ways of partitioning those tables between multiple routing instances.

The function of the control plane in a conventional networking device include; management, the system configuration, and exchange of routing table information and are infrequently performed. The route controller exchanges the topology information with other routers and constructs a routing table based on a routing protocol, for example, Routing Information Protocol (RIP), Open Shortest Path First (OSPF), or Border Gateway Protocol (BGP). It can also create a forwarding table for the forwarding plane. Since the control functions are not performed on each arriving packet, they do not have a strict speed constraint and are implemented in software (Chao & Liu, 2007).
On the other hand, Software Defined Networking (SDN) is a set of various technique and approach that allows users to program, orchestrate directly, control and manage network resources, which facilitates the design, delivery, and operation of network services in a dynamic and scalable way.
The SDN controller plane is a dedicated network entity. It comprises of controllers that are logically centralized, each of which has control over the resources exposed by the data plane The SDN controller plane, also referred to as SDN control layer, provides means to dynamically control the behavior of network resources (data plane), as required by the application plane or layer. The SDN applications define network system resources control and allocation by interacting with the SDN control layer via Application-Control Plane Interface (ACPI), this interface is also referred to as the NorthBound Interface(NBI). The northbound interface allows SDN applications to program application-specific network behavior and access network information. This helps for applications to be able to operate on an abstraction of the network and leverage network services and potentialities without being tied to the details of their implementations.
The orchestration functionality of the control plane provides automated control and management of the network resources on the data plane, requests coordination from application layer for network resources or data plane based on policy provision by the application layer or multi-layer management function. For instance, interacting with the multi-layer management services to provide management of SDN application-related operations such as service creation and provisioning, user management (Open Networking Foundation, 2012).

4.3 The SDN Application Layer.

The SDN applications are programs in the application layer or plane, which specifies business applications or network services and defines a service-aware behavior of network resources in a programmatic manner. These programs communicate their network requirements and response by interacting with the SDN controller plane via the Northbound Interface(NBI) so that controller plane can automatically customize the behavior of the network resources. These programs at the application plane (SDN applications) makes use of the global abstract network view of the network resources, for their internal decision-making purposes, provided by the SDN controller plane by

using information and data models exposed via the Northbound Interface (Telecommunication Standardization Sector of ITU, 2014).
SDN applications vary according to different kinds of services to achieve its objectives; it may invoke other external services, and orchestrate any number of SDN controllers. This application requires a certain amount of knowledge of their environment and roles. The ONF white paper (Open Networking Foundation, 2012) states that:
An application plane entity may act as an information model server by exposing an information model instances for use by other applications. Formally, the other applications are clients that communicate to the SDN application server agent.
An application plane entity may also act as an information model client by operating on an information model instance exposed by a server entity. This server entity may be either an SDN controller or a secondary application.
An application plane entity may act in both roles simultaneously. For example, a path computation engine (PCE) may rely on an SDN controller for virtual network topology information (maintained in a traffic engineering database), while offering the SDN controller a path computation service (Open Networking Foundation, 2012).

5. PRACTICAL IMPLEMENTATION OF SOFTWARE DEFINED NETWORKING.

Since the evolution of SDN, a network architecture that decouples the data plane from the control plane, and turning the control plane into a software-based centralized controller. Many vendors see values in SDN network architecture as enabling network programmability. Network programmability, on the other hand, is capable of harvesting information from network devices, evolving from the technology push and use pull which encouraged Active Networking as it grows into network programmability, and motivated pushing out new configurations, policies, profile definitions in response to dynamic network conditions or service provisioning requests (Shin, Nam, & Kim, 2012).
Over the evolving years, SDN has gained practical implementation in various aspects of networking technologies. This thesis will look into how SDN makes it easy to implement and manage Network Function Virtualization, Applications of Network Function Virtualization, Network Function Virtualization using Mininet, emulator for quick prototyping of Software Defined Network.

5.1 Network Functions Virtualization

Network virtualization environment supports the coexistence of multiple virtual networks on the same underlying physical substrate, where each of these virtual networks is a collection of virtual links and virtual nodes. However, these virtual networks can also be regarded as a subset of the underlying physical network infrastructure (Stalling, 2016).
Network Function Virtualization Infrastructure is a collection of resources and functions which encompass three main domain; the compute domain, hypervisor domain, Infrastructure network domain. In totality, NFVI consists of all software and hardware components which build up an environment where virtual network functions are deployed (Chiosi, Clarke, Willis , Reid, Feger, & Ruhl , 2012).

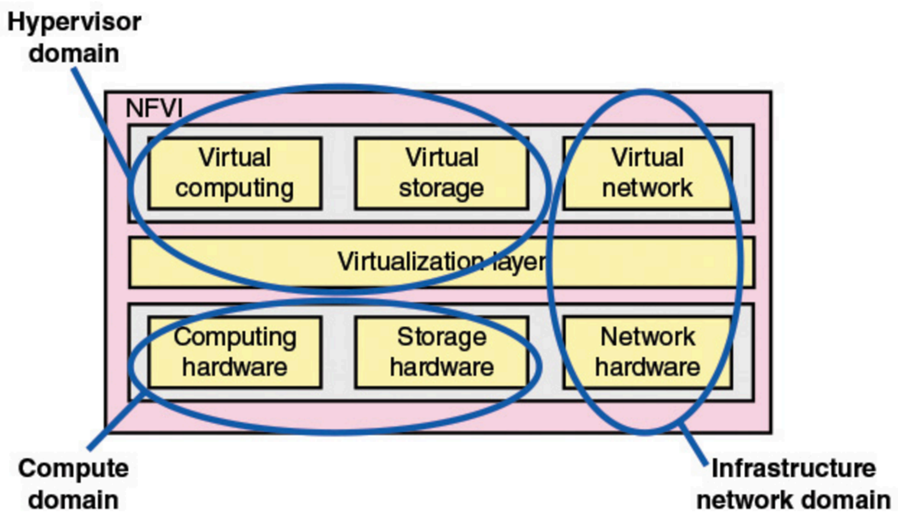


Figure 6. NFV Domains encompassed in the network function virtualization infrastructure (Chiosi, Clarke, Willis , Reid, Feger, & Ruhl , 2012).

The compute domain within the network functions virtualization infrastructure (NFVI) includes the servers and storage hardware which enable provision for commercial off-the-shelf (COTS) high-volume storage and servers.
The Hypervisor domain within NFVI provides abstraction layer of the hardware by mediating the compute domain resources to the virtual machines of the software appliances.
The Infrastructure network domain, this domain within the NFVI comprises of all generic and high volume switches which are interconnected together by a network, enabling configuration for infrastructure network services supply.
NFV implementation aims to transform how network operators and architects design network systems by evolving virtualization technologies to integrate various network equipment types onto industry standard high volume Ethernet switches, servers and storage, located in the end user premises, datacenters, and Network Nodes. NFV involves software implementation of network functions, which can run on various industry standard hardware, and moved to or instantiated in different network locations as required, without any need of new equipment installation (Chiosi, Clarke, Willis , Reid, Feger, & Ruhl , 2012).

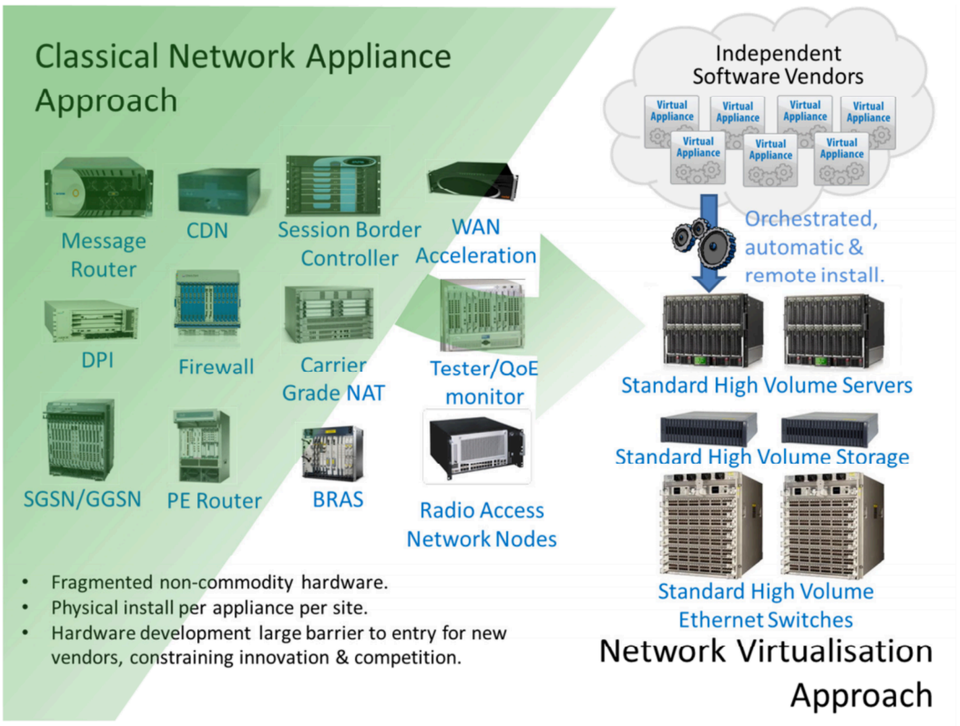
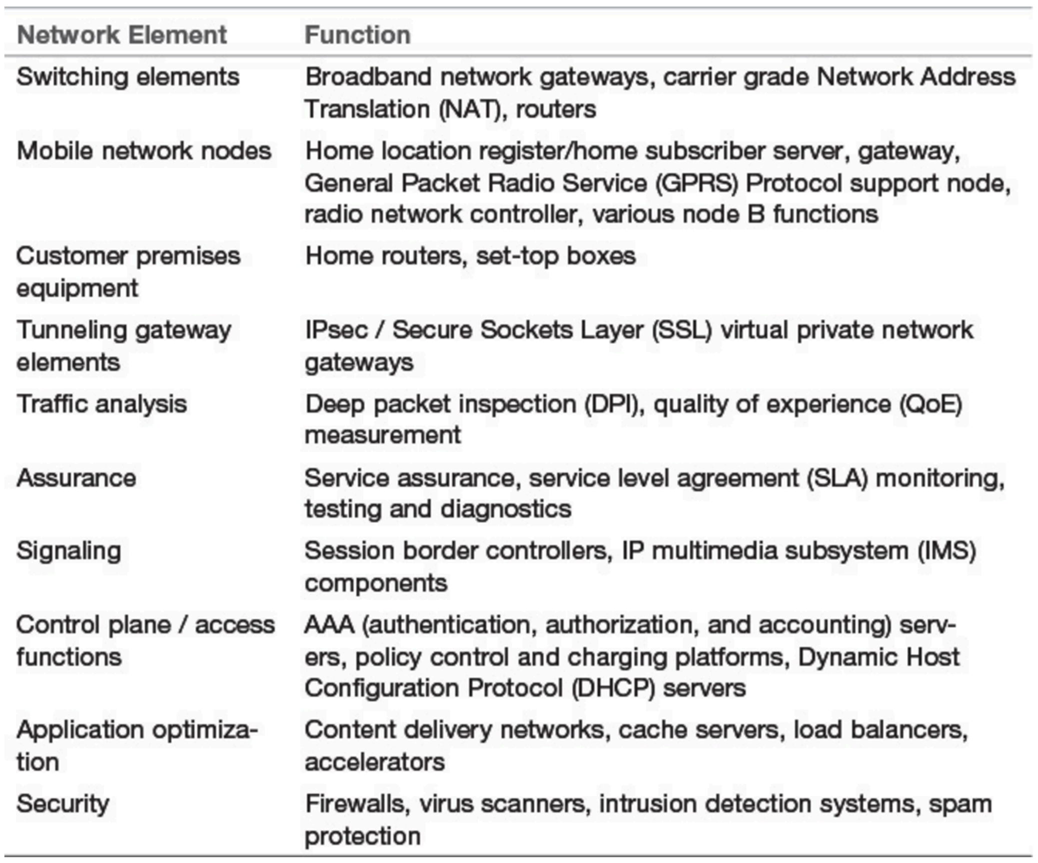

Classical Network Appliance Approach
Independent Software Vendors
Virtual Appliance
Message Router
CDN
Session Border Controller
WAN Acceleration
Orchestrated, automatic & remote install.
DPI
Firewall
Carrier Grade NAT
Tester/QoE monitor
Standard High Volume Servers
Standard High Volume Storage
SGSN/GGSN
PE Router
BRAS
Radio Access Network Nodes
Standard High Volume Ethernet Switches
• Fragmented non-commodity hardware.
• Physical install per appliance per site.
• Hardware development large barrier to entry for new vendors, constraining innovation & competition.
Network Virtualisation Approach

Figure 7. Vision for network function virtualization (Nadeau, 2013).

Application Of Network Function Virtualization

A virtualized implementation of traditional network function can also be regarded as a Virtual Network Function (VNF). Various network elements can be virtualized as shown in Figure 7.



Network Element	Function
Switching elements	Broadband network gateways, carrier grade Network Address Translation (NAT), routers
Mobile network nodes	Home location register/home subscriber server, gateway, General Packet Radio Service (GPRS) Protocol support node, radio network controller, various node B functions
Customer premises equipment	Home routers, set-top boxes
Tunneling gateway elements	IPsec / Secure Sockets Layer (SSL) virtual private network gateways
Traffic analysis	Deep packet inspection (DPI), quality of experience (QoE) measurement
Assurance	Service assurance, service level agreement (SLA) monitoring, testing and diagnostics
Signaling	Session border controllers, IP multimedia subsystem (IMS) components
Control plane / access functions	AAA (authentication, authorization, and accounting) servers, policy control and charging platforms, Dynamic Host Configuration Protocol (DHCP) servers
Application optimization	Content delivery networks, cache servers, load balancers, accelerators
Security	Firewalls, virus scanners, intrusion detection systems, spam protection

Figure 8. Network Functions that could be virtualized (Nadeau, 2013).

Furthermore, NFV could be applicable in a set of service models and high-level use cases which are intended to drive the development of products and standards for network-wide implementation. As addressed by ISG NFV, Applying NFV can be categorized into architectural use cases and Service-Oriented use cases. Architectural application of Network Function Virtualization focuses on the provision of general purpose services and applications which are based on the Network Function Virtualization Infrastructure(NFVI) architecture.

Network Function Virtualization Infrastructure as a Service (NFVIaaS): NFVI is provided as a service in a scenario which maps Network as a Service (NaaS) and cloud computing service model, Infrastructure as a Service (IaaS) as elements with NFVI. A service provider deploys and implements an NFVI that may be used to support Virtual Network Functions(VNFs) by both NFVIaaS provider and other network service providers. The Infrastructure is designed to support the vendor's requirements for deploying VNFs and more capacity that can be sold out to other providers (Nadeau, 2013).

5.2 A simple software defined network architecture.

This thesis work will show a practical implementation of necessary software defined network architecture by setting up a development environment on a single machine using various SDN tools along with general networking utilities on a virtualized environment.
Implementation will deploy SDN architecture, showing how the southbound API and Northbound API interact. The design features SDN/Open Flow architecture where the network will be designed on top of the standard interface, Open Flow, to ensure configuration and communication compatibility and interoperability among the data and control plane virtual devices. In this implementation, the two most important elements are the controller and forwarding devices. Utilizing a virtual network simulator, mininet, to create software defined network architecture for three hosts, a switch, and a controller.
SDN architecture with Open Flow controller using mininet.
Software defined Networking, an emerging paradigm and evolutionary approach to network design and functionality based on the capability to programmatically modify the behavior of network devices according to business and or traffic needs. SDN architecture makes it easier to program configurable and customizable software at the controller layer, which is independent of the underlying hardware on the forwarding plane to expand data flow control. This approach to networking will make networks more cost-effective, dynamic and flexible, in the long run, simplifying operational complexity.
In recent networking trends, change in network traffic pattern, big data, information technology consumerization, the rise of cloud computing and services, more bandwidth consumption, all brings about the need for a flexible architectural approach to networks (A Software Defined Network Architecture). The control plane in SDN allows the underlying network infrastructure to be abstracted and enabling network applications to view the network as a single, logical switch entity.
The application-programming interface (API) in SDN specifies how software components should interact with each other, making it possible to implement basic network functions such as; routing, security path computation, and other tasks. The southbound API allows the controller to define the behavior of the switches at the bottom of the architecture; a good example is the OpenFlow API as used in this thesis implementation. On the other hand, Northbound API provides a network abstraction interface to the applications and management systems at the top of the architecture.
The core of a software-defined network is the controller software at the control plane, which facilitates automated network management, making application programs integration and administration easier. The controller software uses protocols such as OpenFlow to configure network devices and manages flow control to enable intelligent networking.
OpenFlow, an approach as used in this thesis implementation, is a protocol that is used to define the communication interface between the control and forwarding layers. OpenFlow allows for manipulation and provides direct access to the forwarding plane of the network devices. It identifies network traffic using the concept of flows.

This approach helps to centralize control, increase network security and reliability, reduces complexity via automation.
With SDN, static networks could be transformed into scalable, flexible, programmable platforms that have the intelligence to allocate resources dynamically.

Mininet is an instant virtual network on a PC. Mininet creates a realistic virtual network, running real kernel, and switch and application code, on a single machine. Mininet is an excellent way to develop, share and experiment with OpenFlow and Software-Defined Networking systems (Mininet Team).
OpenFlow is the first standard communication interface defined between the control plane and the data plane of SDN architecture. Open Flow allows direct access to, and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based) (Open Networking Foundation) .
We will demostrate a simplified initial development and deployment process with mininet simulation tool which allows OpenFlow network to be emulated on a single machine.
To accomplish this demonstration, we will use a virtual environment for SDN development built by sdnhub.org, the virtual machine is a 64-bit ubuntu 14.04 image (3GB) that has a number of SDN software and tools installed. I will demonstrate a simple SDN/OpenFlow network with three hosts, an OpenVswitch and an OpenFlow reference controller.
From ubuntu terminal, Mininet will emulate a network topology consisting of three hosts, a switch, and an open flow controller and automatically assign Class A private IP adresses.

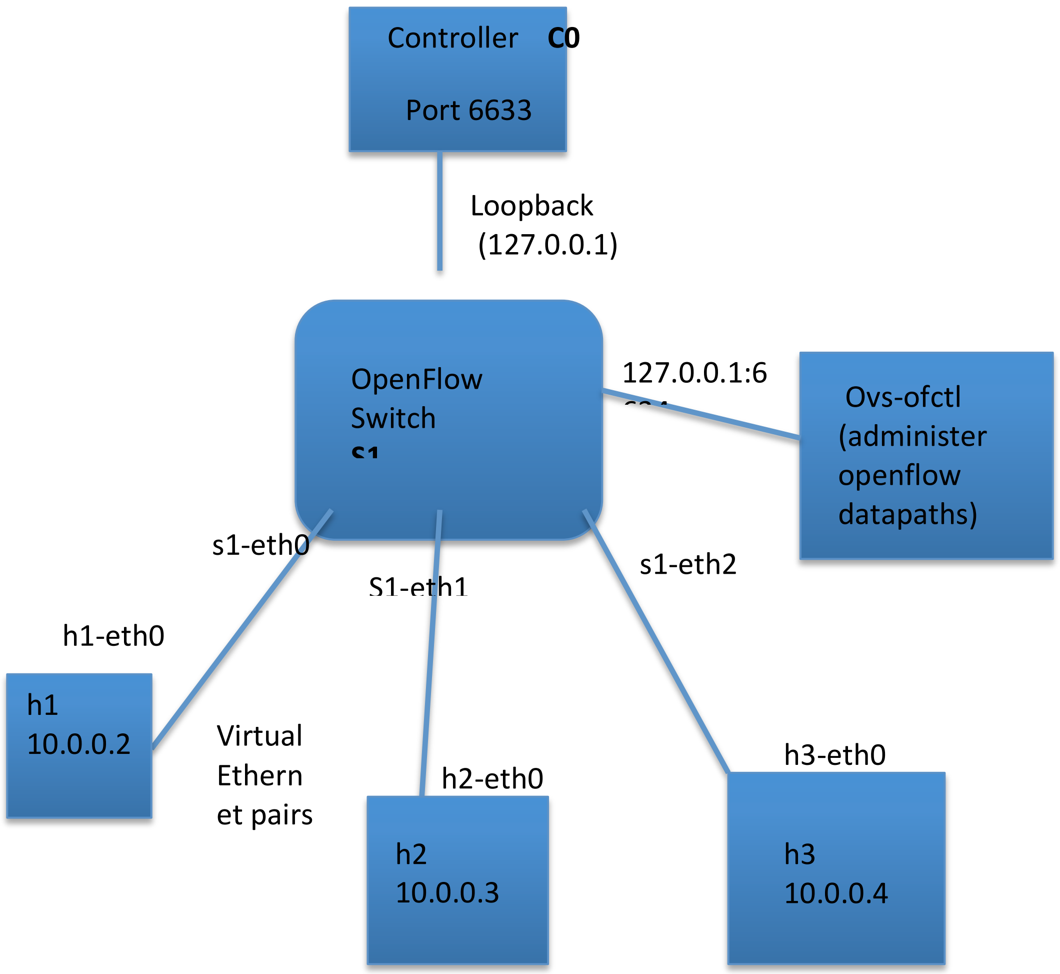
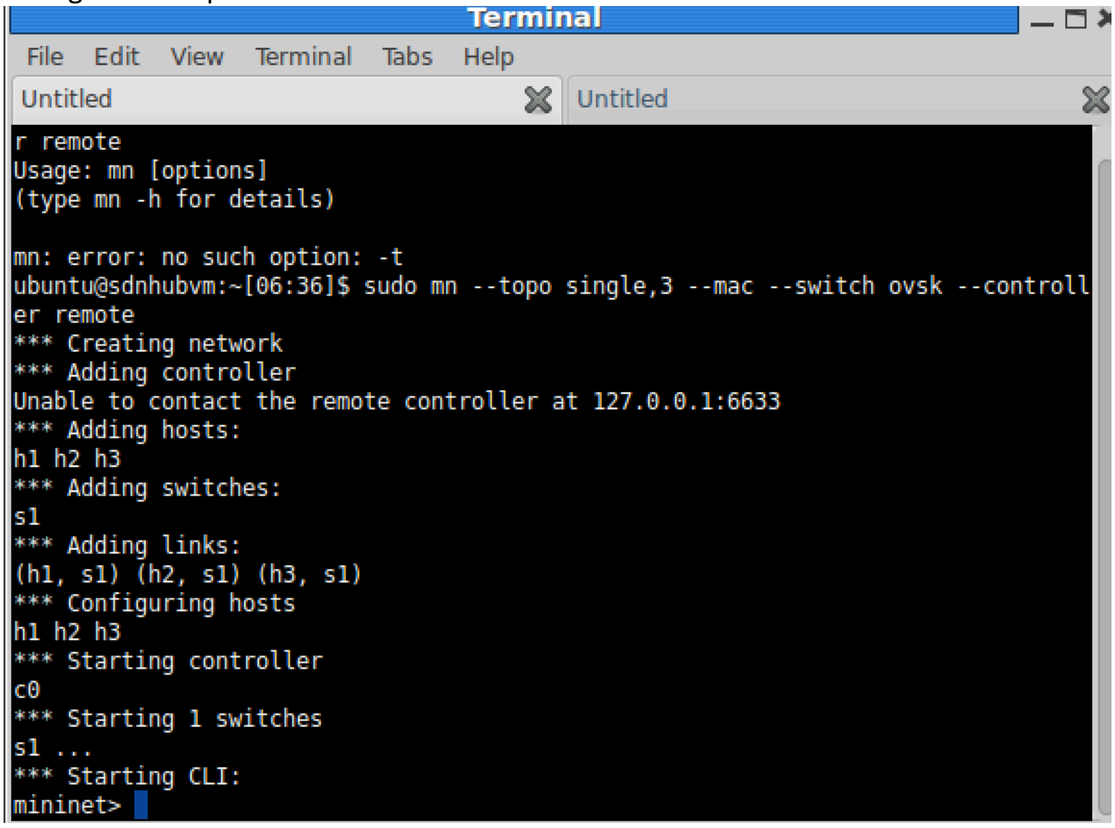


Figure 9. SDN network topology consisting of three hosts with class A IP addressing, an OpenFlow Switch, and OpenFlow refrence controller.

To run mininet and define network topology
Command: sudo mn –topo single, 3

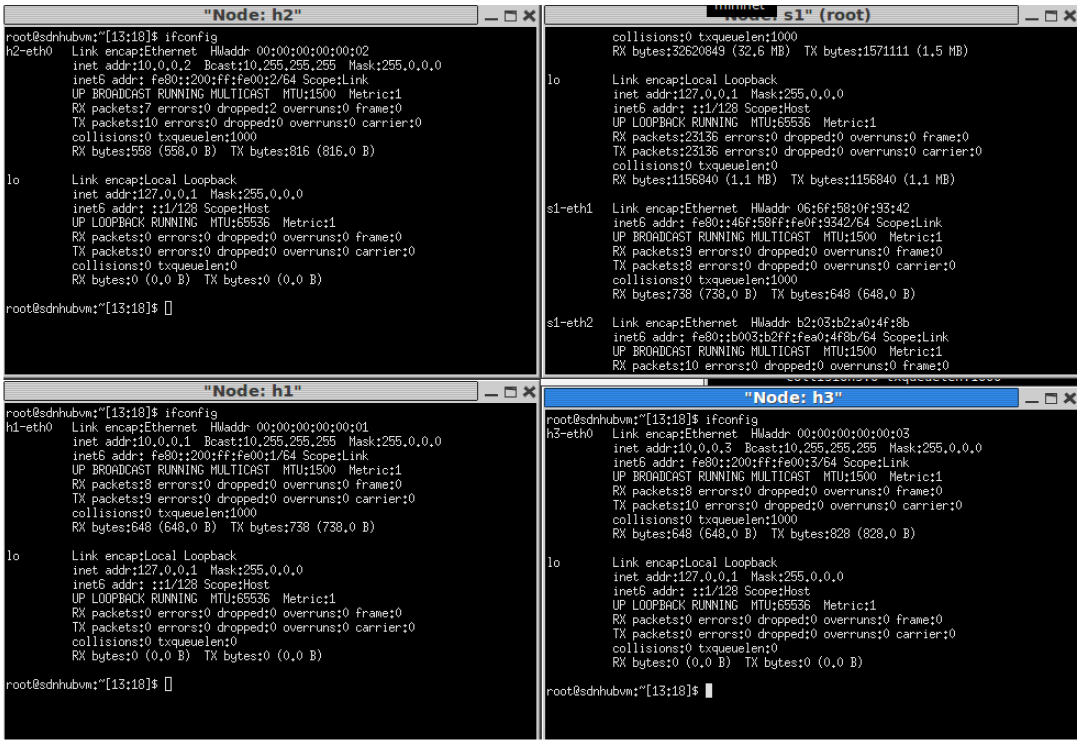
–mac –switch ovsk –controller remote
The above command tells Mininet to create three virtual hosts, and assign each with a separate ip address, create a single open Flow software switch in the kernel with three ports, connect each virtual host to the switch with a virtual Ethernet cable and configure the OpenFlow switch to connect to a remote controller



```
r remote
Usage: mn [options]
(type mn -h for details)

mn: error: no such option: -t
ubuntu@sdnhubvm:~[06:36]$ sudo mn --topo single,3 --mac --switch ovsk --controll
er remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figure 10. Starting mininet.

Mininet simply create a network topology, add a controller C0, add switch S1, add hosts h1, h2 and h3, adds links between h1 and s1, h2 and s2, h3 and s1. Furthermore mininet configures the three hosts, starts the controller and the switch.



```
root@sdnhubvm:~[13:18]$ ifconfig
h2-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:02
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:2 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:558 (558.0 B)  TX bytes:816 (816.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[13:18]$ 
```

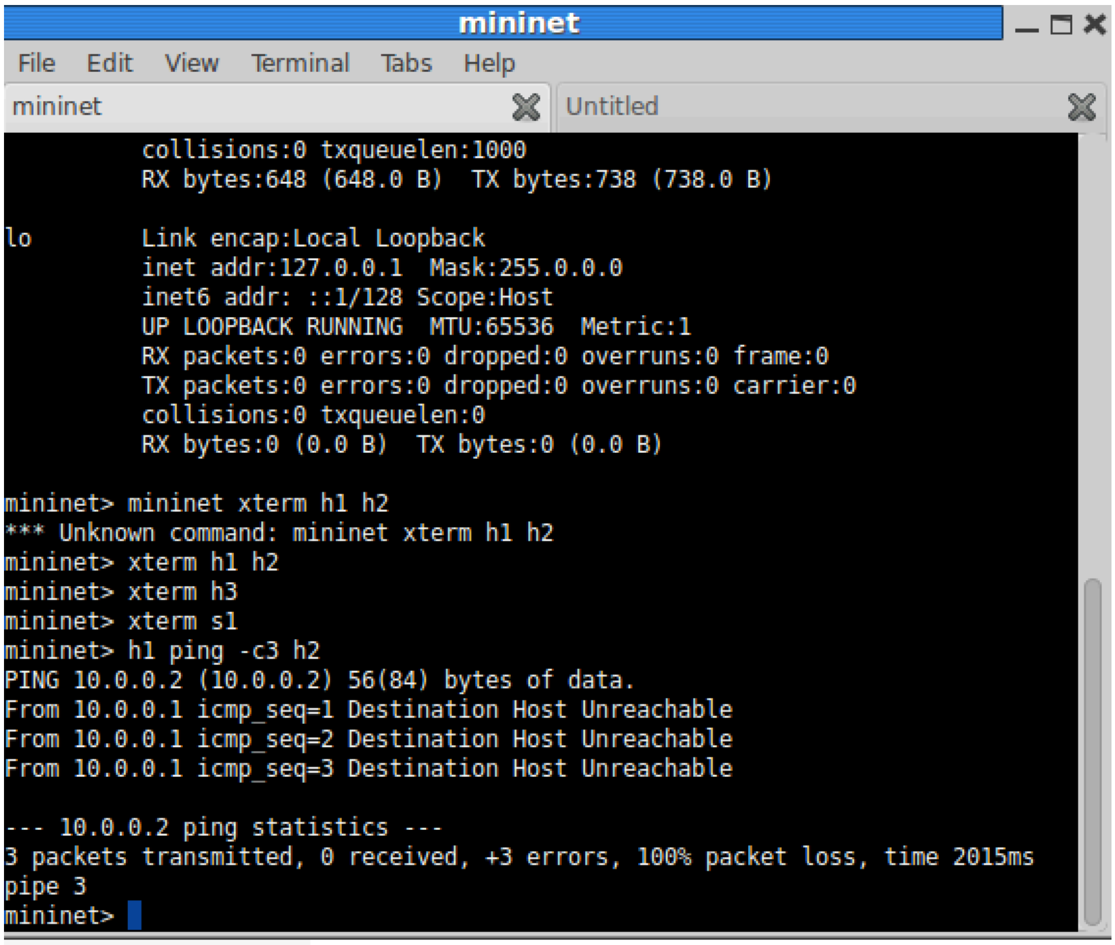
```
          collisions:0 txqueuelen:1000
          RX bytes:32620849 (32.6 MB)  TX bytes:1571111 (1.5 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:23136 errors:0 dropped:0 overruns:0 frame:0
          TX packets:23136 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1156840 (1.1 MB)  TX bytes:1156840 (1.1 MB)

s1-eth1   Link encap:Ethernet  HWaddr 06:6f:58:0f:93:42
          inet6 addr: fe80::46f:58ff:fe0f:9342/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:738 (738.0 B)  TX bytes:648 (648.0 B)

s1-eth2   Link encap:Ethernet  HWaddr b2:03:b2:a0:4f:8b
          inet6 addr: fe80::b003:b2ff:fea0:4f8b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
```

```
root@sdnhubvm:~[13:18]$ ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:738 (738.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[13:18]$ 
```

```
root@sdnhubvm:~[13:18]$ ifconfig
h3-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:03
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::200:ff:fe00:3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:828 (828.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[13:18]$ 
```

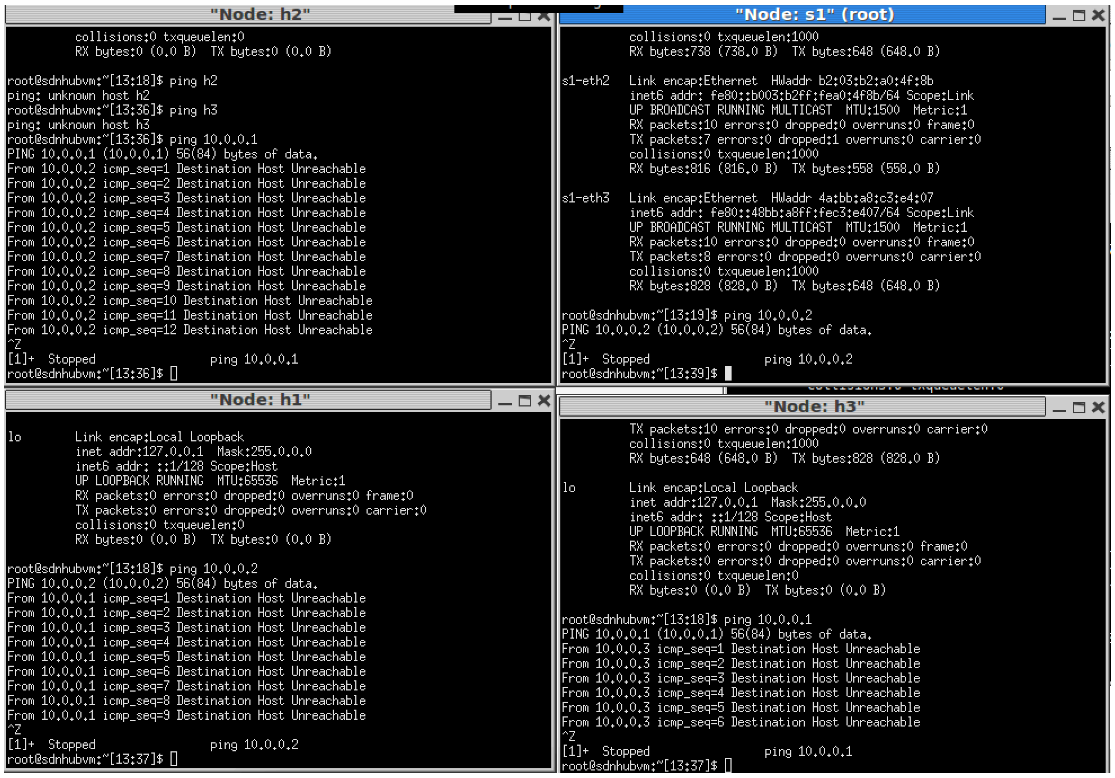
Figure 11. IP address assigned to all hosts, and switch links are up.

Ping testing among all host, to check if virtual ethernet pairs are recheable.
Command: h1 ping –c3 h2



```
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:738 (738.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> mininet xterm h1 h2
*** Unknown command: mininet xterm h1 h2
mininet> xterm h1 h2
mininet> xterm h3
mininet> xterm s1
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2015ms
pipe 3
mininet> 
```

Figure 12. H1 cannot reach H2 and vice versa.



```
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[13:18]$ ping h2
ping: unknown host h2
root@sdnhubvm:~[13:36]$ ping h3
ping: unknown host h3
root@sdnhubvm:~[13:36]$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable
From 10.0.0.2 icmp_seq=5 Destination Host Unreachable
From 10.0.0.2 icmp_seq=6 Destination Host Unreachable
From 10.0.0.2 icmp_seq=7 Destination Host Unreachable
From 10.0.0.2 icmp_seq=8 Destination Host Unreachable
From 10.0.0.2 icmp_seq=9 Destination Host Unreachable
From 10.0.0.2 icmp_seq=10 Destination Host Unreachable
From 10.0.0.2 icmp_seq=11 Destination Host Unreachable
From 10.0.0.2 icmp_seq=12 Destination Host Unreachable
^Z
[1]+  Stopped                 ping 10.0.0.1
root@sdnhubvm:~[13:36]$ 
```

```
          collisions:0 txqueuelen:1000
          RX bytes:738 (738.0 B)  TX bytes:648 (648.0 B)

s1-eth2   Link encap:Ethernet  HWaddr b2:03:b2:a0:4f:8b
          inet6 addr: fe80::b003:b2ff:fea0:4f8b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:816 (816.0 B)  TX bytes:558 (558.0 B)

s1-eth3   Link encap:Ethernet  HWaddr 4a:bb:a8:c3:e4:07
          inet6 addr: fe80::48bb:a8ff:fec3:e407/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:828 (828.0 B)  TX bytes:648 (648.0 B)

root@sdnhubvm:~[13:19]$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^Z
[1]+  Stopped                 ping 10.0.0.2
root@sdnhubvm:~[13:39]$ 
```

```

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[13:18]$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
^Z
[1]+  Stopped                 ping 10.0.0.2
root@sdnhubvm:~[13:37]$ 
```

```
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:828 (828.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@sdnhubvm:~[13:18]$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
From 10.0.0.3 icmp_seq=3 Destination Host Unreachable
From 10.0.0.3 icmp_seq=4 Destination Host Unreachable
From 10.0.0.3 icmp_seq=5 Destination Host Unreachable
From 10.0.0.3 icmp_seq=6 Destination Host Unreachable
^Z
[1]+  Stopped                 ping 10.0.0.1
root@sdnhubvm:~[13:37]$ 
```

Figure 13. Ping failure among all host.

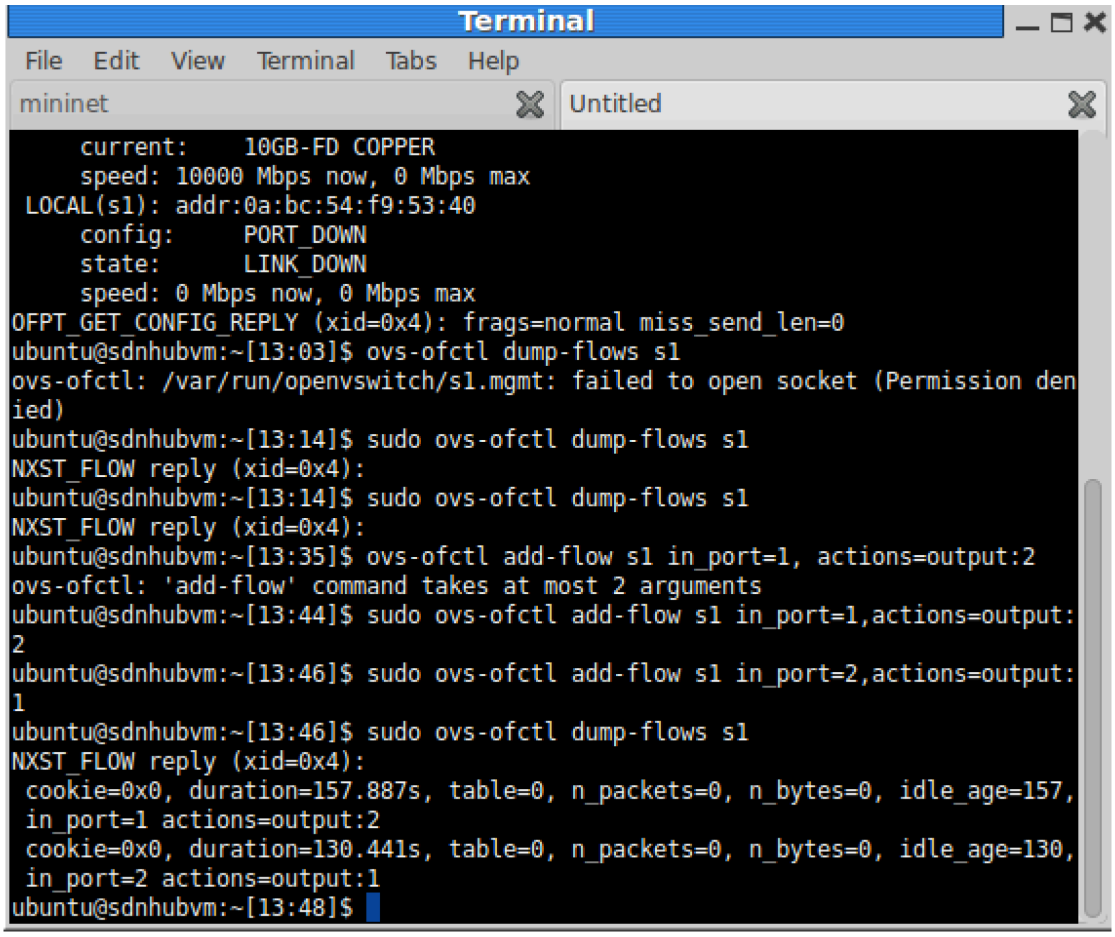
There were no ping replies among the hosts, the switch flow table is empty, flows can either be manually installed to forward necessary packets or get instructions from the controller. However, the controller is not yet connected to the open Vswitch; therefore the switch does not know what to do with the incoming traffic, and this leads to ping failure.

Note: At this point the switch flow table is empty.

To enable visibility and control over the switch flow table, which will be useful for debugging flow counter and flow state, we use the ovs-ofctl utility that comes with the open vswitch. Most open vswitch start up with a passive listening port.

Using ovs-ofctl, we will manually install the necessary flows on the open switch flow table to forward packets coming from port 1 to port 2 and vice-versa. Using the following command:

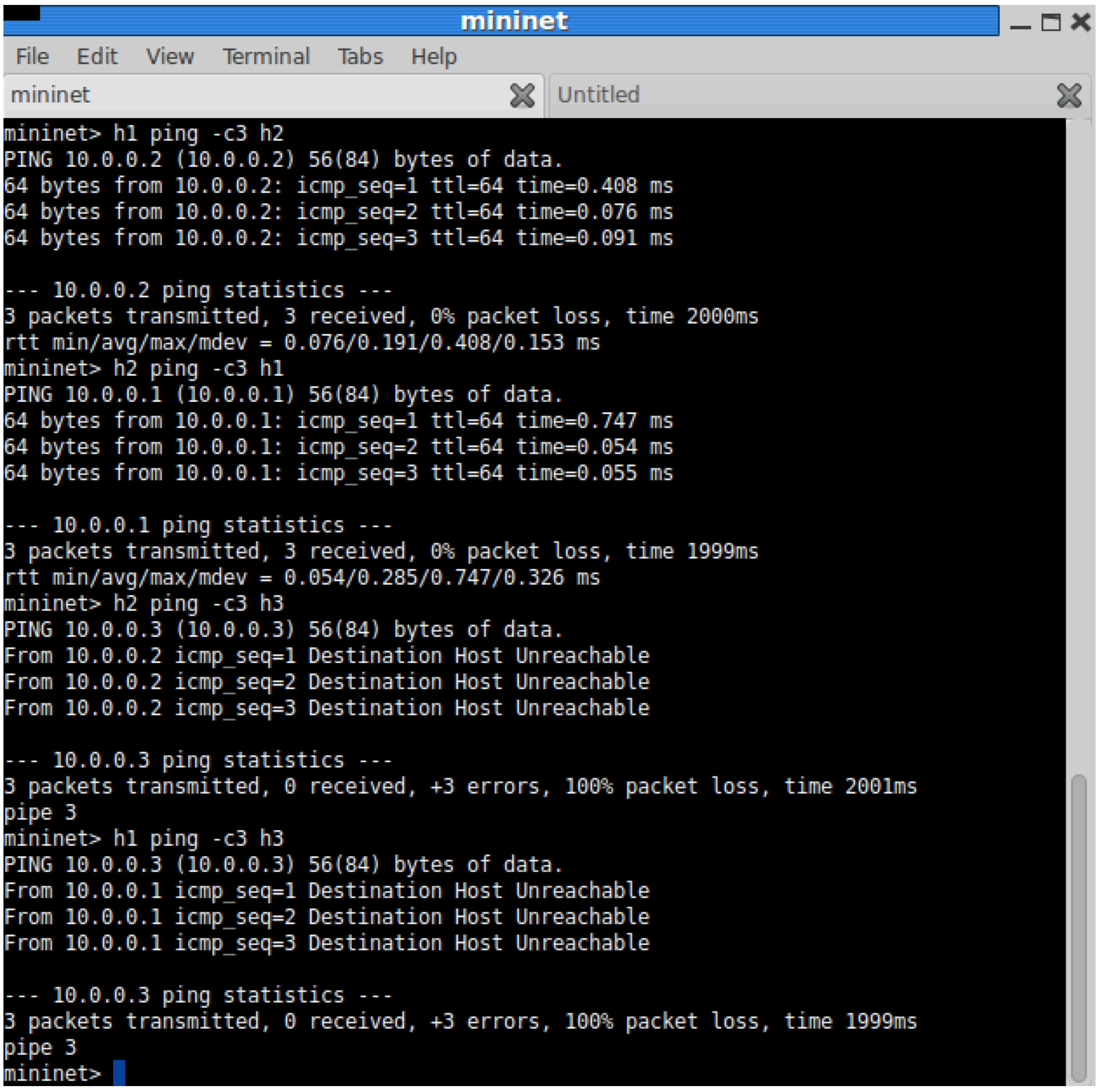
```
Command: # ovs-ofctl add-flow s1 in_port=1, actions=output:2
         # ovs-ofctl add-flow s1 in_port=2, actions=output:1
```



```
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:0a:bc:54:f9:53:40
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
ubuntu@sdnhubvm:~[13:03]$ ovs-ofctl dump-flows s1
ovs-ofctl: /var/run/openvswitch/s1.mgmt: failed to open socket (Permission den
ied)
ubuntu@sdnhubvm:~[13:14]$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
ubuntu@sdnhubvm:~[13:14]$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
ubuntu@sdnhubvm:~[13:35]$ ovs-ofctl add-flow s1 in_port=1, actions=output:2
ovs-ofctl: 'add-flow' command takes at most 2 arguments
ubuntu@sdnhubvm:~[13:44]$ sudo ovs-ofctl add-flow s1 in_port=1,actions=output:
2
ubuntu@sdnhubvm:~[13:46]$ sudo ovs-ofctl add-flow s1 in_port=2,actions=output:
1
ubuntu@sdnhubvm:~[13:46]$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=157.887s, table=0, n_packets=0, n_bytes=0, idle_age=157,
 in_port=1 actions=output:2
 cookie=0x0, duration=130.441s, table=0, n_packets=0, n_bytes=0, idle_age=130,
 in_port=2 actions=output:1
ubuntu@sdnhubvm:~[13:48]$
```

Figure 14. Manual Installation of flows on open switch flow table.

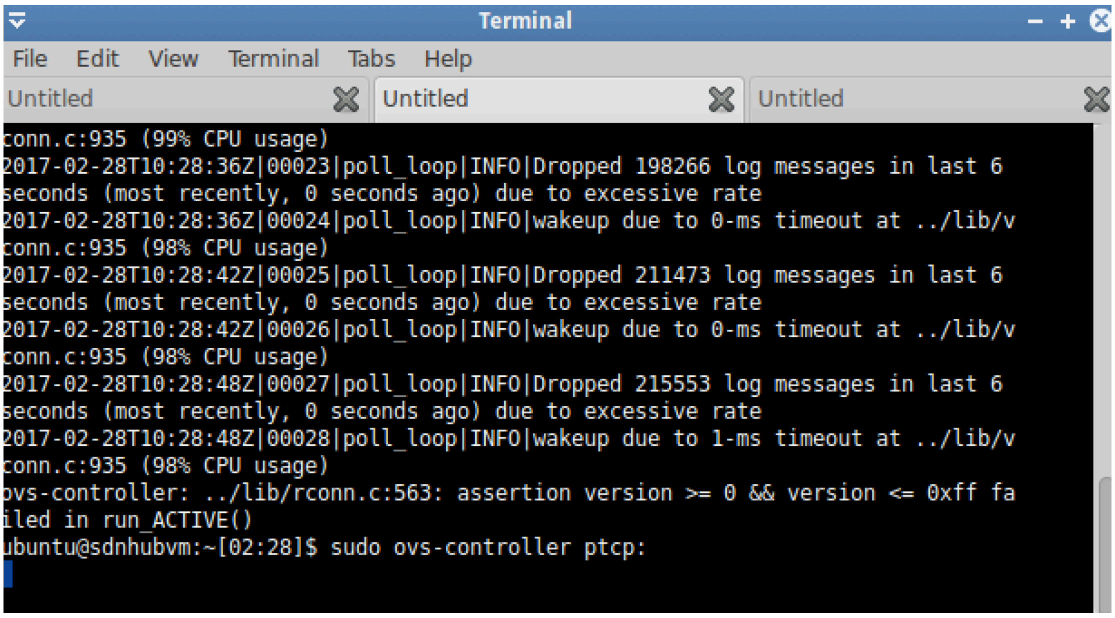
ovs-ofctl is talking to a local instance of open vswitch via unix domain socket which it is looking up by name (s1).
Now that we have configured the necessary flow on port 1 and port 2 of the open vswitch, this allows packets coming in from port 1 to be forwarded out on port 2 and vice versa. Therefore, host h1 and h2 connected to port 1 and port 2 of the switch can only be reachable, while host h3 cannot reach any other host on the network.



```
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.408 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.091 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.076/0.191/0.408/0.153 ms
mininet> h2 ping -c3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.747 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.055 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.054/0.285/0.747/0.326 ms
mininet> h2 ping -c3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2001ms
pipe 3
mininet> h1 ping -c3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 1999ms
pipe 3
mininet>
```

Figure 15. Succesful pings between host H1 and H2, while H3 is unrecheable because the switch flow table includes entries for only H1 and H2.

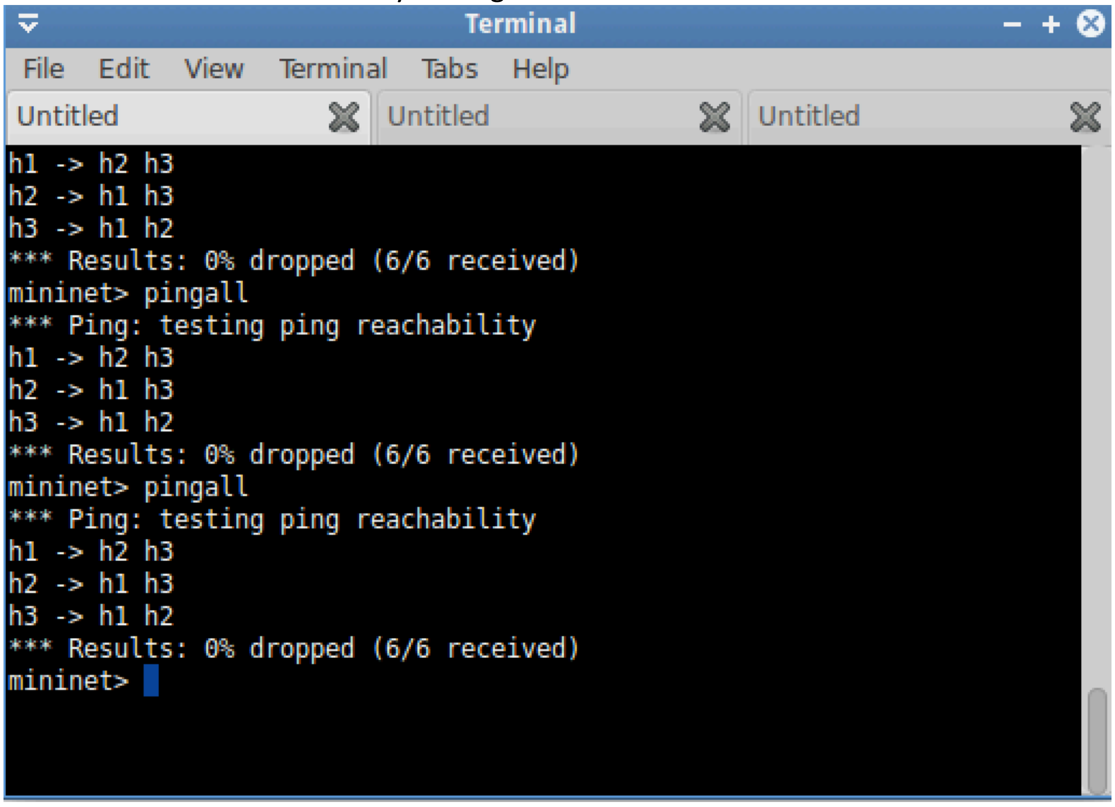
We will set up the controller to get all hosts to be reacheable
We will use Wireshark to capture and watch OpenFlow protocol messages. Having setup wireshark to listen to traffic, We will start the open flow reference controller, which starts a simple controller that acts as a learning switch without installing any flow-entries.
Command: #ovs-controller ptcp:



```
conn.c:935 (99% CPU usage)
2017-02-28T10:28:36Z|00023|poll_loop|INFO|Dropped 198266 log messages in last 6
seconds (most recently, 0 seconds ago) due to excessive rate
2017-02-28T10:28:36Z|00024|poll_loop|INFO|wakeup due to 0-ms timeout at ../lib/v
conn.c:935 (98% CPU usage)
2017-02-28T10:28:42Z|00025|poll_loop|INFO|Dropped 211473 log messages in last 6
seconds (most recently, 0 seconds ago) due to excessive rate
2017-02-28T10:28:42Z|00026|poll_loop|INFO|wakeup due to 0-ms timeout at ../lib/v
conn.c:935 (98% CPU usage)
2017-02-28T10:28:48Z|00027|poll_loop|INFO|Dropped 215553 log messages in last 6
seconds (most recently, 0 seconds ago) due to excessive rate
2017-02-28T10:28:48Z|00028|poll_loop|INFO|wakeup due to 1-ms timeout at ../lib/v
conn.c:935 (98% CPU usage)
ovs-controller: ../lib/rconn.c:563: assertion version >= 0 && version <= 0xff fa
iled in run_ACTIVE()
ubuntu@sdnhubvm:~[02:28]$ sudo ovs-controller ptcp:
```

Figure 16. Starting OVS reference controller.

After the controller is started, it acts as a learning switch and allows connectivity among all hosts, h3 will now be reacheable from other host on the network and vice versa.Now Test for reachability among all hosts.



```
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Figure 17. All host are now recheable from anywhere on the network.

6. CONCLUSION

Traditional Network systems became very complicated to manage for network operators, the network systems state changes continually, and it remains difficult to configure network devices across a broad network system. Complexity in today's network and lack of flexibility causes operators to master a variety of network protocols. However, in a traditional distributed network, operators can combine network devices from different vendors into a single large network system.
This work shows that SDN does not only addresses these problems but also simplifies it, by separating the control planes and the data planes of the network and centralizing the control and management of the overall network system with a uniform API. The SDN architecture makes it easier to have an overview of the network such that; network operators and administrators can tailor their network systems to meet their requirements and needs, as the network changes state. Also, it shows that the centralized control systems can have sufficient control and visibility over the data plane resources to automate and deploy a variety of network services such as middleboxes, network traffic profiling, load balancing, switching, access control, routing, etc. As a result, SDN has gained substantial interest and concern in both academic and commercial institutions.
This work also shows fundamental aspects of SDN architecture, such as the OpenFlow Protocol and OpenFlow API, that emerged and widely adopted. It indicates that SDN vendors and programmers can design programs that translate network requirements into OpenFlow tables and can maintain the flow tables as network system changes occur. Moreover, the quality of the controller plane programs is a critical factor in the overall performance of the network system.
The future of Software Defined Network is feasible and applicable in a various network environment, and this paradigm shift brings numerous benefits compared to traditional networking. This thesis has demonstrated that SDN is practical, efficient and implemented in various networking domain. It applies to use the simplest SDN programming model, where network topology can be simulated to implement various SDN techniques. However, It encourages networking student to adopt programming skills applicable in software defined networking domain; network programmers can develop a wide variety of SDN applications with a familiar, general-purpose programming language. SDN gives the flexibility of introducing new ideas, without limitations or restriction to proprietary network devices, through a software program. This practice makes it easier to manipulate and make changes, as compared to using a set of commands, on proprietary equipment. SDN benefits to network operators by tackling the complexity of managing large networks. With the centralized approach to network management and configuration, operators do not have to configure network devices individually when making changes in network behavior, but instead make traffic forwarding decisions at a logically centralized location, the controller layer, with a global view of the overall network state.

REFERENCES

Bavier, A., Feamster, N., Huang, M., Peterson, L., & Rexford, J. (2006). In VINI veritas: realistic and controlled network experimentation. *SIGCOMM '06 Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications* (pp. 3-14). New York: ACM.
Benson, T., Akella, A., & Maltz, D. (2009). *Unraveling the Complexity of Network Management.* Madison: Theophilus Benson.
Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., & Van der Merwe, J. (2005). Design and implementation of a routing control platform. *NSDI Networked Systems Design & Implementation. 2*, pp. 15-28. Berkeley, CA, USA: USENIX Association .
Chao, H., & Liu, B. (2007). *High Performance Switches and Routers.* Wiley-IEEE Press.
Chiosi, M., Clarke, D., Willis , P., Reid, A., Feger, J., & Ruhl , F. (2012). Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. *SDN and OpenFlow World Congress.* Darmstadt-Germany: Industry Specification Group (ISG).
Doria, A., Hadi Salim, J., Haas, R., Khosarvi, H., Wang, W., Dong, L., et al. (2010, 03). Forwarding and Control Element Separation (ForCES) .
Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., & Van der Merwe, J. (2004). The case for separating routing from routers. *ACM SiGCOMM workshop on Future Direction in Network Architecture.* (pp. 5-12). Oregon: ACM SiGCOMM.
Feamster, N., Gao, L., & Rexford, J. (2007). How to lease the internet in your spare time. *ACM SIGCOMM Computer Communication Review , 37* (1), 61-64.
Feamster, N., Rexford, J., & Zegura, E. (2014). The Road to SDN: An intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review , 44* (2), 87-98.
Florida Center for Instructional Technology. (2013). *What is a Network?* Retrieved 04 04, 2017, from An Educator's Guide to School Networks: https://fcit.usf.edu/network/
International Telecommunication Union. (1993, 03 01). Introduction to Intelligent Network Capability Set 1. HELSINKI.
Kreutz, D., Ramos, F. M., Verissimo, P., Tothenberg, E. C., Azodolmolky, S., & Uhlig, S. (2014). *Software-Defined Networking: A Comprehensive Survey.* Lisbon: IEEE.
Mininet Team. (n.d.). *Mininet*. Retrieved 02 04, 2017, from Mininet: http://mininet.org/
Mosharaf Kabir Chowdhury, N., & Raouf, B. (2010). A survey of network virtualization. *The International Journal of Computer and Telecommunications Networking , 54* (5), 862-876.
Nadeau, T. D. (2013). *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies.* Sebastopol, CA: O'Reilly Media.
Open Networking Foundation. (n.d.). *OpenFlow*. Retrieved 04 04, 2017, from Opennetworking.org: https://www.opennetworking.org/sdn-resources/openflow
Open Networking Foundation. (2014). *SDN architecture.* Open Networking Foundation. Palo Alto: ONF.

Open Networking Foundation. (2012). *Software-Defined Networking: The New Norm for Networks.* ONF White Paper. Open Networking Foundation.
Rostami, A. (2014). *The Evolution Of Programmable Networks: From Active Networks To Software Defined Network (SDN).* Ericsson Research,Stockholm. Stockholm: Ericsson.
Russell, T. (1998). *Signaling System 7 (Telecommunications).* Texas: Mcgraw-Hill .
Sheinbein, D., & Weber, R. (1982). Stored Program Controlled Network: 800 service using SPC network capability. *The Bell System Technical Journal , 61* (7).
Shin, M.-K., Nam, K.-H., & Kim, H.-J. (2012). Software-defined networking (SDN): A reference architecture and open APIs. *ICT Convergence (ICTC), 2012 International Conference.* IEEE.
Stalling, W. (2016). *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud.* Crawfordsville: Pearson Education, Inc.
Telecommunication Standardization Sector of ITU. (2014). *Framework of software-defined networking.* International Telecommunication Union. ITU-T.
Tennenhouse, D., Smith, J., Wetherall, D., & Minden, G. (1997, 01 01). A survey of active network research. *IEEE Communications Magazine , 35* (1), pp. 80-86.
Van Der Merwe, J., Rooney, S., Leslie, I., & Crosby, S. (1998). The Tempest-a practical framework for network programmability. *IEEE Network , 12* (3), 20-28.
Wikipedia. (2009, 12 20). *BlueBox*. Retrieved 04 04, 2017, from Wikipedia: https://en.wikipedia.org/wiki/Blue_box
Yang, L., Dantu, R., Anderson, T., & Gopal, R. (2004). *Forwarding and Control Element Separation (ForCES) Framework.* The Internet Society. Network Working Group.