SOFTWARE DEFINED NETWORK (SDN)

A Project Work Submitted according
to the requirements for the Degree
Of
BACHELOR OF TECHNOLOGY
In
INFORMATION TECHNOLOGY
By

Amarendra Boro (Roll No. Gau-C-11/L-224)

Maheswar Daimary (Roll No. Gau-C-11/132)

Arnab Roy (Roll No. Gau-C-11/145)

Divya Jyoti Kurmi (Roll No. Gau-C-11/128)

Under the Supervision and Guidance of

RANJAN PATOWARY
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY
केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार
CENTRAL INSTITUTE OF TECHNOLOGY KOKRAJHAR
(A Centrally Funded Institute under Ministry of HRD, Govt. of India)
BODOLAND TERRITORIAL AREAS DISTRICTS :: KOKRAJHAR :: ASSAM :: 783370
Website: www.cit.kokrajhar.in, www.cit.ac.in

JUNE 2015

SOFTWARE DEFINED NETWORK (SDN)

A Project Work Submitted according

to the requirements for the Degree

Of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Amarendra Boro (Roll No. Gau-C-11/L-224)

Maheswar Daimary (Roll No. Gau-C-11/132)

Arnab Roy (Roll No. Gau-C-11/145)

Divya Jyoti Kurmi (Roll No. Gau-C-11/128)

Under the Supervision and Guidance of

RANJAN PATOWARY
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार

CENTRAL INSTITUTE OF TECHNOLOGY KOKRAJHAR

(A Centrally Funded Institute under Ministry of HRD, Govt. of India)

BODOLAND TERRITORIAL AREAS DISTRICTS :: KOKRAJHAR :: ASSAM :: 783370

Website: www.cit.kokrajhar.in, www.cit.ac.in

JUNE 2015

DEPARTMENT OF INFORMATION TECHNOLOGY

केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार

CENTRAL INSTITUTE OF TECHNOLOGY, KOKRAJHAR

(An Autonomous Institute under MHRD)

Kokrajhar – 783370, BTAD, Assam, India

Certificate by Board of Examiners

This is to certify that the project work entitled "SOFTWARE DEFINED NETWORK" submitted by Amarendra Boro, Maheswar Daimary, Arnab Roy, Divya Jyoti Kurmi to the Department of Information Technology of Central Institute of Technology, Kokrajhar has been examined and evaluated.

The project work has been prepared as per the regulations of Central Institute of Technology and qualifies to be accepted in partial fulfillment of the requirement for the degree of B. Tech.

Ranjan Patowary 26/05/15

Project Coordinator

Ranjan Patowary

Assistant Professor, Dept. of IT

Assistant Professor
Dept. of Information Tech.
Central Institute of Tech.
Kokrajhar

[illegible] 26/5/15

Board of Examiner

EXRERNAL EXAMINER

Information Tech

DEPARTMENT OF INFORMATION TECHNOLOGY

केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार

CENTRAL INSTITUTE OF TECHNOLOGY, KOKRAJHAR

(An Autonomous Institute under MHRD)

Kokrajhar – 783370, BTAD, Assam, India

CERTIFICATE OF APPROVAL

This is to certify that the work embodied in this project entitled **"Software Defined Network"** mitted by Amarendra Boro, Maheswar Daimary, Arnab Roy, Divya Jyoti Kurmi to the Department of rmation Technology, is carried out under our direct supervisions and guidance.

The project work has been prepared as per the regulations of Central Institute of Technology and I strongly commend that this project work be accepted in partial fulfillment of the requirement for the degree of B.Tech.

Ranjan Patowary 26/05/15

Supervisor

(Mr. Ranjan Patowary)

Assistant Professor, Dept. of IT

Assistant Professor
Dept. of Information Technology
Central Institute of Technology
Kokrajhar

Countersigned by

[illegible] 26-5-2015

(Mr.Kongkon Kalita)

HOD (I/C)

Department of IT

Head
Dept. of Information Technology
CIT, Kokrajhar

ACKNOWLEDGEMENT

The satisfaction that accompanies from the successful completion of any task would be incomplete without e mention of people whose ceaseless cooperation made it possible, whose constant guidance and couragement crown all efforts with success.

We bestow our hearted appreciation and gratefulness to our project guides, Mr. Ranjan Patowary for the idance, inspiration and constructive suggestion that help us in the preparation of this project, without which r efforts would have remained astray.

ce: CIT, Kokrajhar

te: 26/05/2015

Amarendra Boro
Roll No. Gau-C-11/l-224
University Reg. No: 081779 (2012-13)
Amarendra Boro

Maheswar Daimary
Roll No. Gau-C-11/132
University Reg. No: 015172 (2011-12)
Maheswar Daimary

Arnab Roy
Roll No. Gau-C-11/145
University Reg. No: 015112 (2011-12)
Arnab Roy

Divya Jyoti Kurmi
Roll No. Gau-C-11/128
University Reg. No: 015107 (2011-12)
DivyaJyotiKurmi

ACKNOWLEDGEMENT

The satisfaction that accompanies from the successful completion of any task would be incomplete without he mention of people whose ceaseless cooperation made it possible, whose constant guidance and ncouragement crown all efforts with success.

We bestow our hearted appreciation and gratefulness to our project guides, Mr. Ranjan Patowary for the uidance, inspiration and constructive suggestion that help us in the preparation of this project, without which ur efforts would have remained astray.

lace: CIT, Kokrajhar

Date: 26/05/2015

Amarendra Boro
Roll No. Gau-C-11/l-224
University Reg. No: 081779 (2012-13)
Amarendra Boro

Maheswar Daimary
Roll No. Gau-C-11/132
University Reg. No: 015172 (2011-12)
Maheswar Daimary

Arnab Roy
Roll No. Gau-C-11/145
University Reg. No: 015112 (2011-12)
Arnab Roy

Divya Jyoti Kurmi
Roll No. Gau-C-11/128
University Reg. No: 015107 (2011-12)
DivyaJyotiKurmi

ABSTRACT

In today's Data center often employ SDN that are layered in most cases. As SDN provides huge wins in Data center, it is mostly used in this area. Data center consists of vast network topology, and due to this it faces problem in tracking out the shortest path. In our implementation we have considered the well-known shortest path algorithm, Bellmanford algorithm considering their nodes weights for a graph under SDN. We use this implementation to trace out the shortest path and transfer packets from source to destination with the help of mininet tools and POX controller. Thus we obtain good result in most of the topologies like FAT topology, Abilene topology, and simple tree topology by comparing their bandwidth , using Iperf tool.

Table of contents

List of figures

Chapter 1

1.1 Introduction:

Software-defined networking (SDN) is an approach to computer networking that allows network administrators to manage network services through abstraction of lower-level functionality. SDN providers offer a wide selection of competing architectures, but at its most simple, the SDN method centralizes control of the network by separating the control logic to off-device computer resources. All SDN models have some version of an SDN Controller, as well as southbound APIs and northbound APIs.

SDN (Software-Defined Networks) were designed for the purpose that smart open-source programmable controllers control dumb low-cost switches. However, controllability and flexibility of SDN are restricted by the lack of information organization and division. As a result, controller logics are often far more complex than they are supposed to be. In this poster, we discuss a new way of organizing topologies and information of OpenFlow networks in data center, and present a management model that divides network views and information orthogonally and orderly to reduce management complexity. In approach, regional networks on lower layers will be aggregated and viewed as single switches to upper layers. Information of management will be divided into three parts, which are, respectively, managed by network managers, regional controllers and tenants. To achieve this, established a mechanism determining which parts a massage should be sent to. In data centre networks, aggregation of networks means that a regional network will decide its own inside logic and be seen as a single switch whose ports are edge ports of the regional network to upper layers. SDN is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol is a foundational element for building.

1.2 Objectives:

SDN (Software-Defined Networks) were designed for the purpose that smart open-source programmable controllers control dumb low-cost switches. However, controllability and flexibility of SDN are restricted by the lack of information organization and division. As a result, controller logics are often far more complex than they are supposed to be. Data centers were originally designed to physically separate traditional computing elements (e.g. PC servers), their associated storage, and the networks that interconnected them with client users. The computing power that existed in these types of data centers became focused on specific server functionality running applications such as mail servers, database servers, or other such widely used functionality in order to clients. Previously, those functions which were executed on the often thousands (or more) of desktops within an enterprise organization were handled by departmental servers that provided services dedicated only to local use. As time went on, the departmental servers migrated into the data center for a variety of reasons first and foremost, to facilitate ease of management, and second, to enable sharing among the enterprise's users. Our main objective in this implementation is to trace out the shortest path of a topology.

Chapter 2

2.1 Overview:

In recent years, Data volume has been increasing at rapid speed which drives the creation of large Data Center hosting abroad range of services such as Web search, e-commerce, storage backup, large scientific applications, video streaming, data analytics and social networking. In such large Data Centers there are tens of thousands of servers spread over hundreds of racks with tens of peta bytes of storage interconnected by high capacity network infrastructure. Failures have severe impact in a large center with huge number of server- and network- elements and massive data storage. Therefore, it is critically important to develop solutions to ensure high availability of resources. Data availability is a major challenge faced by today's Data Centers. As data availability becomes a critically important requirement, many businesses use increased amount of resources to ensure continuous operations. Maintaining uninterruptible access to all Data Centre applications is highly desirable. As a result, Data Centers need a range of business continuance solutions, from simple tape backup and remote replication to synchronous mirroring and mirrored distributed Data Centers. Proactive replication is a key strategy coupled with mirroring for data protection. Data replication is an effective approach for achieving high data availability and durability in Data Centers. Data replication is a technique designed for replicating data at two or more storage nodes attached to different racks in a Data Centre. Such redundancy ensures at least one copy of data is available for continuous operation in the event of a rack switch failure or rack power failure. However, the design choice of data replication is complicated by keeping the copies as closely synchronized as possible and using as little network bandwidth as possible. Synchronous update of all copies provides high resilience to data loss but has poor write performance and results in high network cost. SDN is a framework that increases the flexibility of the network through separation of control and data planes. This separation makes the network switching and routing devices simpler and less expensive. The control plane could be implemented in a general purpose commodity server. This server as a centralized Controller takes care of routing and other policies according to which the network devices function.

Controllers: The brains of the network, SDN Controllers offer a centralized view of the overall network, and enable network administrators to dictate to the underlying systems (like switches and routers) how the forwarding plane should handle network traffic.

Southbound APIs: SDN uses southbound APIs to relay information to the switches and routers below. OpenFlow, considered the first standard in SDN, was the original southbound API and remains as one of the most common protocols. Despite some considering OpenFlow and SDN to be one in the same, OpenFlow is merely one piece of the bigger SDN landscape.

Northbound APIs: SDN uses northbound APIs to communicate with the applications and business logic above. These help network administrators to programmatically shape traffic and deploy services.[2]

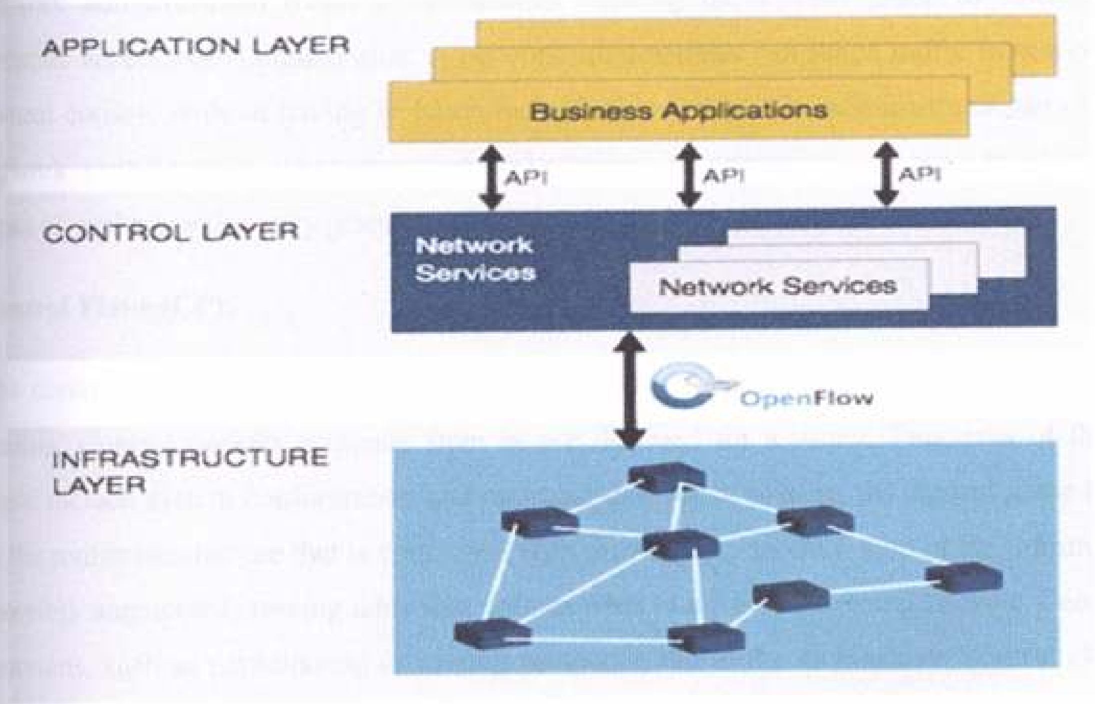


Fig.2.1.1SDN Architecture

2.2 Data Plane and Control Plane

Data Plane (DP):

The data plane (sometimes known as the user plane, forwarding plane, carrier plane or bearer plane) is the part of a network that carries user traffic. The data plane, the control plane and the management plane are the three basic components of telecommunications architecture. The control plane and management plane serve the data plane, which bears the traffic that the network exists to carry. The data plane enables data transfer to and from clients, handling multiple conversations through multiple protocols, and manages conversations with remote peers. Data plane traffic travels through routers, rather than to or from them.

In conventional networking, all three planes are implemented in the firmware of routers and switches. Software-defined networking (SDN) decouples the data and control planes and implements the control plane in software instead, which enables programmatic access to make network administration much more flexible. Moving the control plane to software allows dynamic access and administration. A network administrator can shape traffic from a centralized control console without having to touch individual switches. The administrator can change any network switch's rules when necessary prioritizing, de-prioritizing or even blocking specific types of packets with a very granular level of control. [5]

Control Plane (CP):

The control plane is the part of a network that carries signaling traffic and is responsible for routing. Control packets originate from or are destined for a router. Functions of the control plane include system configuration and management. [6] In routing, the control plane is the part of the router architecture that is concerned with drawing the network map or the information in a (possibly augmented) routing table that defines what to do with incoming packets. Control plane functions, such as participating in routing protocols, run in the architectural control element. In most cases, the routing table contains a list of destination addresses and the outgoing interface(s) associated with them. Control plane logic also can define certain packets to be discarded, as well as preferential treatment of certain packets for which a high quality of service is defined by such mechanisms as differentiated services. Depending on the specific router implementation, there may be a separate forwarding information base that is populated (i.e., loaded) by the control plane, but used by the forwarding plane to look up packets, at very high speed, and decide how to handle them.[7]

Control plane and data plane separation:

Control plane: Logic control forwarding plane.e.g. routing protocols, firewall configuration.

Data plane: forwards traffic according to configuration by control plane. e.g.: IP forwarding, Ethernet switching.

Why separate?

- Independent evolution and development. Especially software control of network.
- Control from a single high-level software program. Easier to reason and debug.

Why does it help?

- Routing
- Enterprise networks. Security
- Research networks. Coexistence with production networks.
- Data centers. VM migration. e.g.: Yahoo has 20,000 hosts, 400,000 VMs. Want sub-second VM migrations. Program switches from a central server, so that forwarding follows migration.
- e.g. AT&T filtering DoS attacks. IRSCP (commercialized RCP) will insert a null route to filter DoS at network edge.

Challenges for separation:

Scalability: Control element responsible for thousands of forwarding elements.

Reliability and security: What if a controller fails or is compromised.

Opportunities for control and data separation:

Two examples:

- New routing services in the wide area. Maintenance, egress selection, security.
- Data centers. Cost, management.

Examples. Wide area.

There are a few constrained ways to set inter domain routing policy: BGP.

Limited knobs, no external knowledge (time of day, reputation of route, etc.)

Instead of BGP route controller updates forwarding table.

Example 1: Maintenance dry-out. Planned maintenance of an edge router.Tell ingress routers to avoid the router with pending maintenance. Too difficult to do in existing networks, e.g. buy tuning OSPF.

Example 2: Customer controlled egress router. Customer selects data center they want to use. Difficult in existing networks, as routing uses destination prefix.

Example 3: Better BGP security. Offline we can determine reputation of a route. But this can't be incorporated into BGP route selection. Off-line anomaly detection. Prefer familiar routes over unfamiliar routes. RCP tells routers to avoid odd routes.

Example 4: Data Centre, addressing,Layer 2 addressing: less configuration, bad scaling. Layer 3 addressing: use existing routing protocols, good scaling, but high administration overhead. Use layer 2 addressing, but to make the addresses topology-specific rather than topology-independent.MAC addresses depend where they are in the topology.mHosts don't know they have MAC address re-assigned, so how is ARP done? Destination host won't respond.A fabric manager will intercept ARPs, it then replies with the topology-dependent Pseudo-MAC (pMAC).Switches re-write MAC addresses at network edge to hosts.

Other Examples:

(a) Dynamic access control

(b) Mobility and migration

(c) Server load balancing

(d) Network virtualization

(e) Energy-efficient networking

(f) Adaptive traffic monitoring

Challenges of separating control and data planes:

Scalability, reliability, consistency, Approaches in RCP and ONIX.

Scalability in RCP: RCP must stores routes and compute routing decisions for all routers in the AS. That's a lot to do at a single node. Strategies to reduce this are:

Eliminate redundancy: store a single copy of each route to avoid redundant computation.

Accelerate lookups: maintain indexes to identify affected routers. Then RCP computes routes only for routers affected by a change.

Punt: Only performs inter-domain (BGP) routing.

Scalability in ONIX: Partitioning. Keep track of subsets of the network state. Then apply consistency measures to ensure consistency between the partitions. Choice of strong and weak consistency models to select correctness versus computation tradeoff.

Aggregation: A hierarchy of controllers. ONIX controllers for departments or buildings, then a super-controller for the domain.

Reliability in RCP: Replicate. RCP has a hot spare. Each replica has its own feed of routes, receiving exactly same inputs, running exact same algorithms, so output should be the same. So no need for consistency protocol.

Consistency: But if different RCPs see difference routes then they will have different outputs. If the two replicas are inconsistent then they can install a routing loop.

Need to guarantee consistent inputs: for RCP that's easy as the IGP passes the full link-state to RCP. So RCP should compute next-hops only for routers it is connected to. .For example, one RCP in partitioned network. Only use candidate routes from partition 1 to set next-hops in partition 1.For example, two RCP in partitioned network. Since the two RCPs have the same data from each partition from the IGP then they give the same output for each partition.

Reliability in ONIX: Network failures. ONIX leaves it to applications to detect and recover.

Reachability to ONIX: Solve using typical network practices, such as multipath.

ONIX failure. Replication and distributed coordination protocol.

Three issues:

- Scalability. Making decisions for many network elements.
- Reliability. Correct operation under failure of the network or controller.
- Consistency. Ensure consistency between controller replicas, especially in partitioned networks.

Solutions:

- Hierarchy
- Aggregation
- State management and distribution

Each controller uses a set of tactics from those available. [8]

2.3 OpenFlow:

OpenFlow was originally imagined and implemented as part of network research at Stanford University. Its original focus was to allow the creation of experimental protocols on campus networks that could be used for research and experimentation. Prior to that, universities had to create their own experimentation platforms from scratch. What evolved from this initial kernel of an idea was a view that OpenFlow could replace the functionality of layer 2 and layer 3 protocols completely in commercial switches and routers. This approach is commonly referred to as the clean slate proposition. Later, in 2011a non-profit consortium called the Open Networking Foundation (ONF) was formed by a group of service providers to commercialize, Standardize, and promote the use of OpenFlow in production networks. The key components of the OpenFlow model, as shown in Figure 3-1, have become at least part of the common definition of SDN, mainly, Separation of the control and data planes (in the case of the ONF, the control plane is managed on a logically centralized controller system).Using a standardized protocol between controller and an agent on the network. Element for instantiating state (in the case of OpenFlow, forwarding state). Providing network programmability from a centralized view via a modern, extensible API.

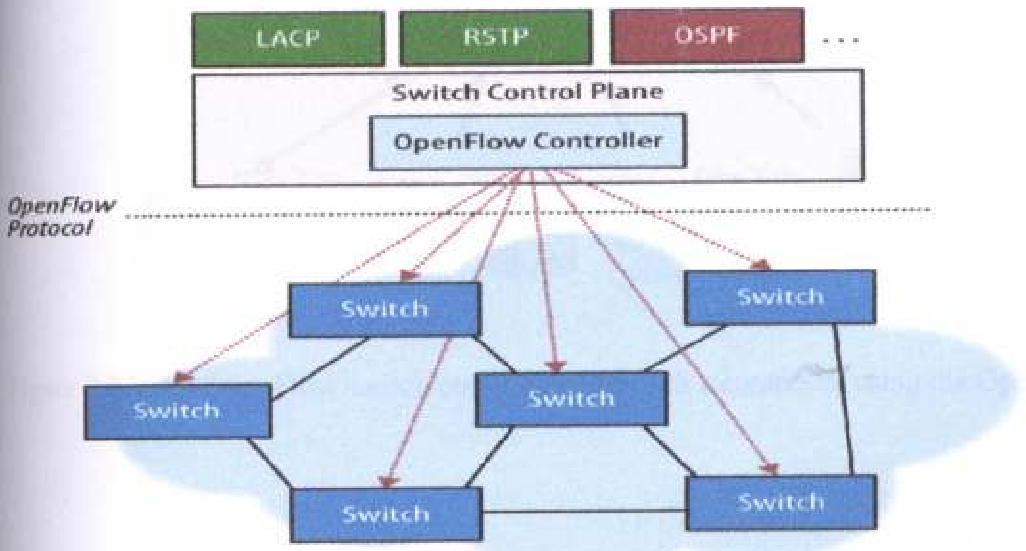


Figure 2.3-1. OpenFlow architecture (with the view that some of the control plane apps will ride on TOP of the controller–emulating the behavior of traditional control plane apps).

OpenFlow is a set of protocols and an API, not a product per se or even a single feature of a product.The OpenFlow protocol uses a standardized instruction set, through which any OpenFlow controller can send a common set of instructions to any OpenFlow Switch regardless to vendor.

OpenFlow Switching

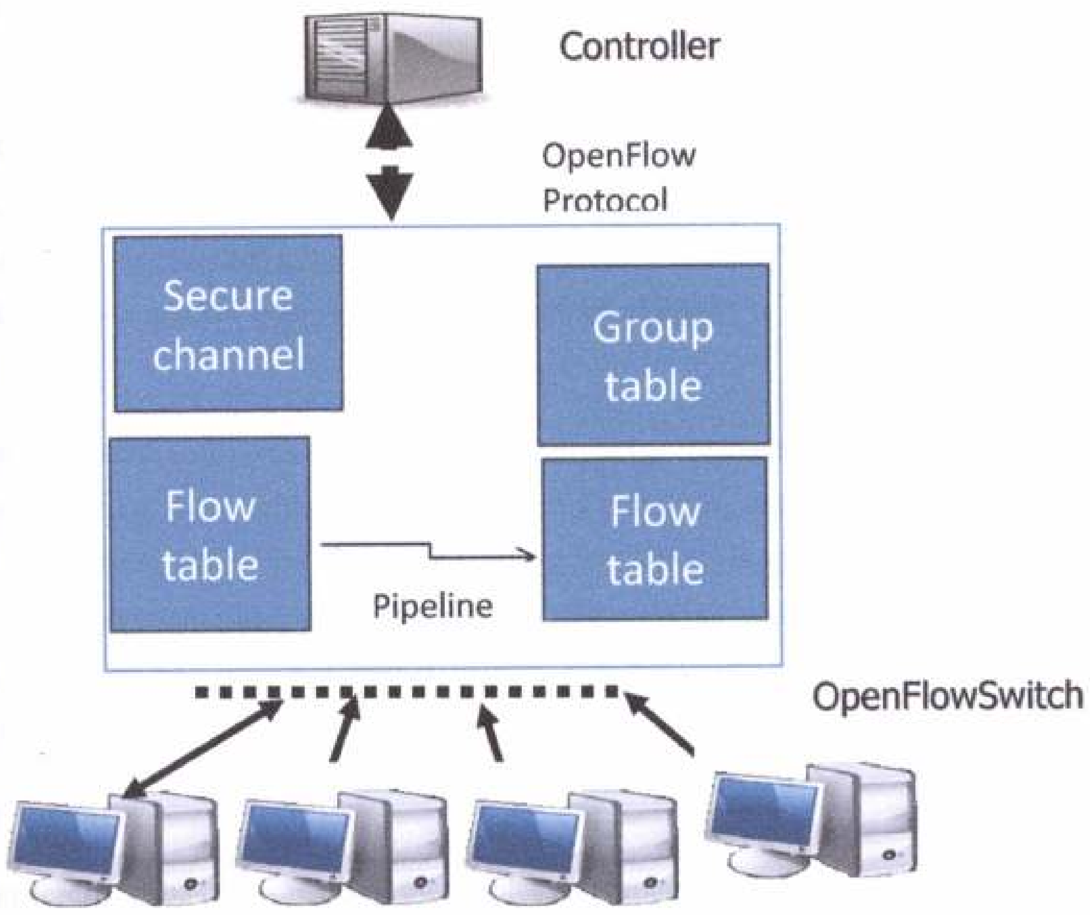


Figure 2.3-2:An OpenFlow switch communicates with a controller using the OpenFlow protocol.

Switch Components:

An OpenFlow Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller. The controller manages the switch via the OpenFlow protocol. Using this protocol, the controller can add, update, and delete flow entries, both reactively (in response to packets) and proactively. Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets.

Matching starts at the first flow table and may continue to additional flow tables. Flow entries match packets in priority order, with the first matching entry in each table being used. If a matching entry is found, the instructions associated with the specific flow entry are executed. If no match is found in a flow table, the outcome depends on switch configuration: the packet may be forwarded to the controller over the OpenFlow channel, dropped, or may continue to the next flow table.

Instructions associated with each flow entry describe packet forwarding, packet modification, group table processing, and pipeline processing. Pipeline processing instructions allow packets to be sent to subsequent tables for further processing and allow information, in the form of metadata, to be communicated between tables. Table pipeline processing stops when the instruction set associated with a matching flow entry does not specify a next table; at this point the packet is usually modified and forwarded.

OpenFlow Tables:

This section describes the components of flow tables and group tables, along with the mechanics of matching and action handling.

3.1 Flow Table:

A flow table consists of flow entries.

MATCH FIELDS	COUNTERS	INSTRUCTIONS

Table 2.3-1: Main components of a flow entry in a flow table.

Each flow table entry contains:

- **Match fields**: to match against packets. These consist of the ingress port and packet headers, and optionally metadata specified by a previous table.
- **Counters**: to update for matching packets
- **Instructions**:to modify the action set or pipeline processing.
- **Group Table**

A group table consists of group entries. The ability for a flow to point to a group enables OpenFlow to represent additional methods of forwarding.

Each group entry contains:

GROUP IDENTIFIER	GROUP TYPE	COUNTERS	ACTION BUCKETS

Table 2.3-2: A group entry consists of a group identifier, a group type, counters, action buckets and a list of.

- **Group identifier**: a 32 bit unsigned integer uniquely identifying the group
- **Group type**: to determine group semantics.
- **Counters**: updated when packets are processed by a group
- **Action buckets**: an ordered list of action buckets, where each action bucket contains a set of actions to execute and associated parameters.

Counters:

Counters may be maintained for each table, flow, port, queue, group, and bucket. OpenFlow-compliant counters may be implemented in software and maintained by polling hardware counters with more limited ranges. Duration refers to the amount of time the flow has been installed in the switch. The Receive Errors field is the total of all receive and collision errors as well as any others not called out in the table. Counters wrap around with no overflow indicator. If a specific numeric counter is not available in the switch, its value should be set to -1.

Instructions:

Each flow entry contains a set of instructions that are executed when a packet matches the entry. These instructions result in changes to the packet, action set and/or pipeline processing. Supported instructions include:

- **Apply-Actions action(s):** Applies the specific action(s) immediately, without any change to the Action Set. This instruction may be used to modify the packet between two tables or to execute multiple actions of the same type.
- **Clear-Actions**: Clears all the actions in the action set immediately.
- **Write-Actions action(s):** Merges the specified action(s) into the current action set. If an action of the given type exists in the current set, overwrite it, otherwise add it.
- **Write-Metadata metadata / mask**: Writes the masked metadata value into the metadata field. The mask specifies which bits of the metadata register should be modified.
- **Goto-Table next-table-id**: Indicates the next table in the processing pipeline. The table-id must be greater than the current table-id. The flows of last table of the pipeline cannot include this instruction.

Action Set:

An action set is associated with each packet. This set is empty by default. A flow entry can modify the action set using a Write-Action instruction or a Clear-Action instruction associated with a particular match. The action set is carried between flow tables. When an instruction set doesn't contain a Goto-Table instruction, pipeline processing stops and the actions in the action set are executed. The actions in an action set are applied in the order specified below, regardless of the order that they were added to the set. If an action set contains a group action, the actions in the appropriate action bucket of the group are also applied in the order specified below. The switch may support arbitrary action execution order through the action list of the Apply-Actions instruction.

1. **Copy** TTL inwards: apply copy TTL inward actions to the packet.
2. **Pop**: apply all tag pop actions to the packet.
3. **Push**: apply all tag push actions to the packet.
4. **Copy TTL outwards**: apply copy TTL outwards action to the packet.
5. **Decrement TTL**: apply decrement TTL action to the packet.
6. **Set**: apply all set-field actions to the packet.
7. **Qos**: apply all QoS actions, such as set queue to the packet.
8. **Group**: if a group action is specified, apply the actions of the relevant group bucket(s) in the order specified by this list.
9. **Output**: if no group action is specified, forward the packet on the port specified by the output action.

OpenFlow Channel:

The OpenFlow channel is the interface that connects each OpenFlow switch to a controller. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out the switch. Between the data path and the OpenFlow channel, the interface is implementation-specific, however all OpenFlow channel messages must be formatted according to the OpenFlow protocol. The OpenFlow channel is usually encrypted using TLS, but may be run directly over TCP.

OpenFlow Protocol:

The OpenFlow protocol is a standardized protocol for interacting with the forwarding behaviors of switches from multiple vendors. This provide us a way to control the behavior of switches throughout our network dynamically and programmatically. OpenFlow is a key protocol in many SDN solutions. The OpenFlow protocol supports three message types, controller-to-switch, asynchronous, and symmetric, each with multiple sub-types. Controller-to-

Switch messages are initiated by the controller and used to directly manage or inspect the state of the switch. Asynchronous messages are initiated by the switch and used to update the controller of network events and changes to the switch state. Symmetric messages are initiated by either the switch or the controller and sent without solicitation. The message types used by OpenFlow are described below.

1. Controller-to-Switch:

Controller/switch messages are initiated by the controller and may or may not require a response from the switch.

Features: The controller may request the capabilities of a switch by sending a features request; the switch must respond with a features reply that specifies the capabilities of the switch. This is commonly performed upon establishment of the OpenFlow channel.

Configuration: The controller is able to set and query configuration parameters in the switch. The switch only responds to a query from the controller.

Modify-modification: Modify-modification messages are sent by the controller to manage state on the switches. Their primary purpose is to add/delete and modify flows/groups in the OpenFlow tables and to set switch port properties.

Read-State: Read-State messages are used by the controller to collect statistics from the switch.

Packet-out: These are used by the controller to send packets out of a specified port on the switch, and to forward packets received via Packet-in messages. Packet-out messages must contain a full packet or a buffer ID referencing a packet stored in the switch. The message must also contain a list of actions to be applied in the order they are specified; an empty action list drops the packet.

Barrier: Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

2. Asynchronous: Asynchronous messages are sent without the controller soliciting them from a switch. Switches send asynchronous messages to the controller to denote a packet arrival, switch state change, or error. The four main asynchronous message types are described below.

Packet-in: For all packets that do not have a matching flow entry, a packet-in event may be sent to the controller (depending on the table configuration). For all packets forwarded to the port, a packet-in event is always sent to the controller. If the switch has sufficient memory to buffer packets that are sent to the controller, the packet-in events contain some fraction of the packet header (by default 128 bytes) and a buffer ID to be used by the controller when it is ready for the switch to forward the packet. Switches that do not support internal buffering (or have run out of internal buffering) must send the full packet to the controller as part of the event. Buffered packets will usually be processed via a Packet-out message from the controller, or automatically expired after some time.

Flow-Removed: When a flow entry is added to the switch by a flow modify message, an idle timeout value indicates when the entry should be removed due to a lack of activity, as well as a hard timeout value that indicates when the entry should be removed, regardless of activity.

The flow modify message also specifies whether the switch should send a flow removed message to the controller when the flow expires. Flow delete requests should generate flow removed messages for any flows with the OFPFF_SEND_FLOW_REM flag set.

Port-status: The switch is expected to send port-status messages to the controller as port configuration state changes. These events include change in port status events (for example, if it was brought down directly by a user).

Error: The switch is able to notify the controller of problems using error messages.

3. Symmetric: Symmetric messages are sent without solicitation, in either direction.

Hello: Hello messages are exchanged between the switch and controller upon connection startup.

Echo: Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They can be used to measure the latency or bandwidth of a controller-switch connection, as well as verify its aliveness.

Experimenter: Experimenter messages provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions.

A simple demonstration of an OpenFlow network:

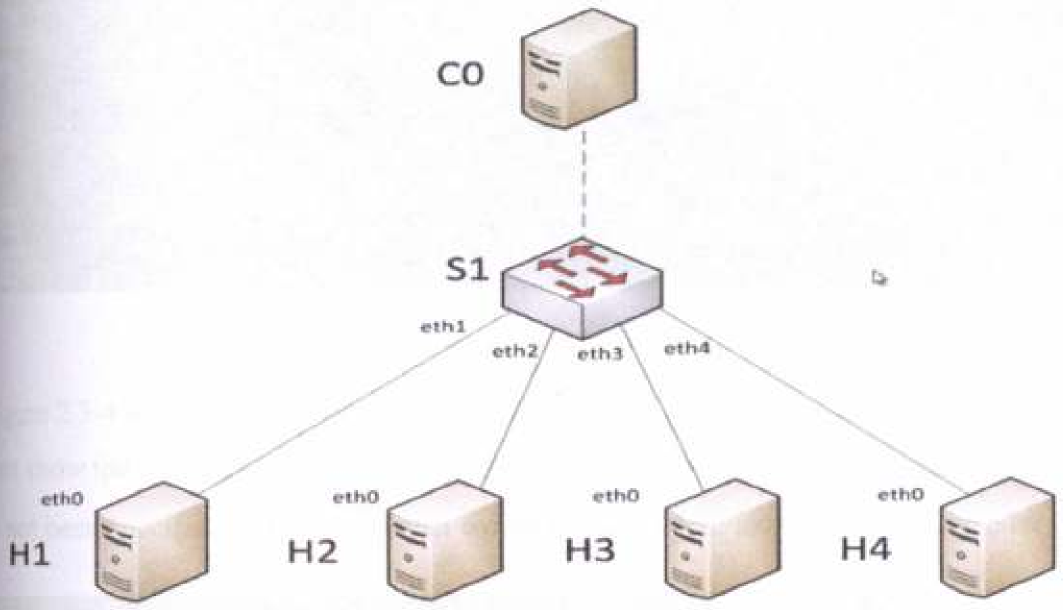
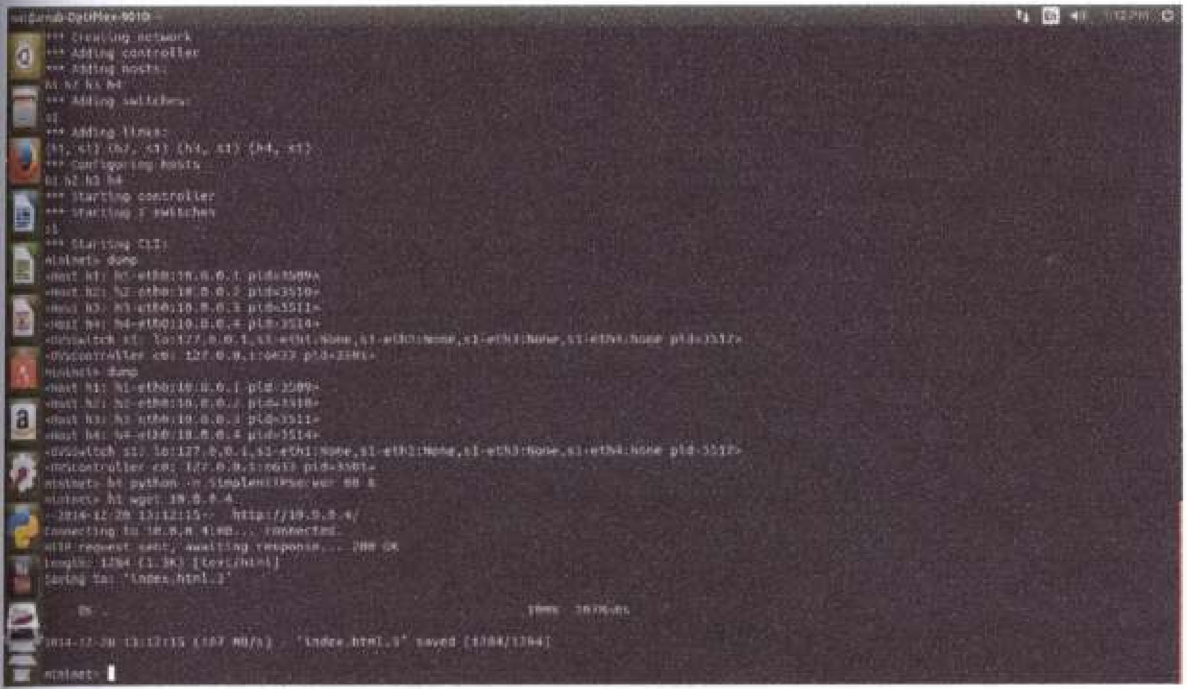


Figure 2.3-3: A simple OpenFlow set-up

A network is launched which would look like the above figure 2.3-3. It would have an Open Flow kernel switch S1, A OpenFlow reference controller C0 and four emulated host H1, H2, H3 and H4.A simple web server is launched in H4 and generate an http request from H1.

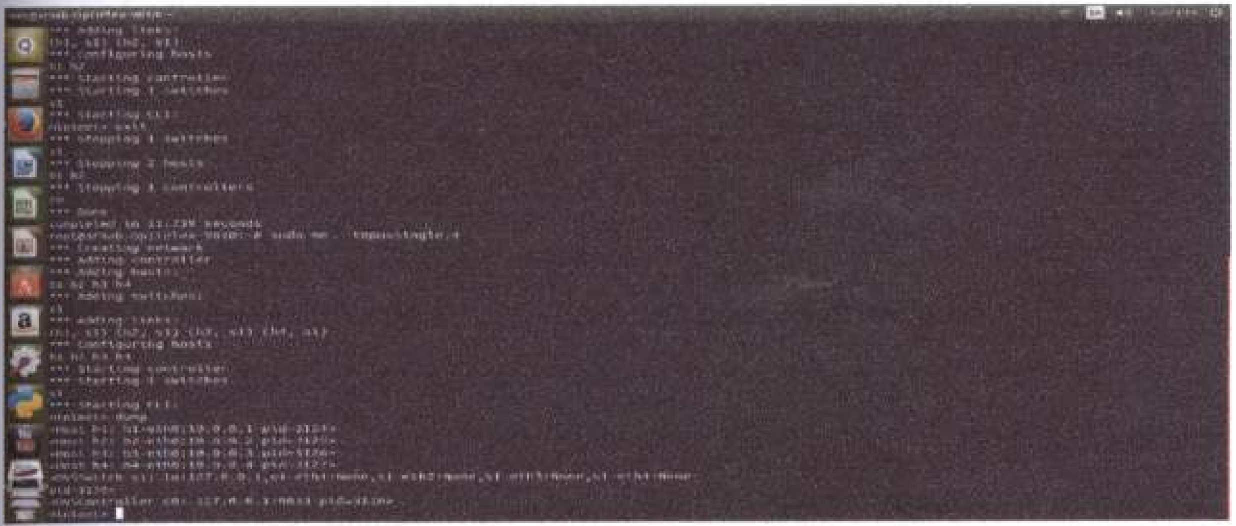
Mininet is used to emulate this network



```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3509>
<Host h2: h2-eth0:10.0.0.2 pid=3510>
<Host h3: h3-eth0:10.0.0.3 pid=3511>
<Host h4: h4-eth0:10.0.0.4 pid=3514>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None pid=3517>
<OVSController c0: 127.0.0.1:6633 pid=3501>
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3509>
<Host h2: h2-eth0:10.0.0.2 pid=3510>
<Host h3: h3-eth0:10.0.0.3 pid=3511>
<Host h4: h4-eth0:10.0.0.4 pid=3514>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None pid=3517>
<OVSController c0: 127.0.0.1:6633 pid=3501>
mininet> h4 python -m SimpleHTTPServer 80 &
mininet> h1 wget 10.0.0.4
--2014-12-28 13:12:15--  http://10.0.0.4/
Connecting to 10.0.0.4:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1284 (1.3K) [text/html]
Saving to: 'index.html.3'

    0% [illegible]                                          100% [illegible]

2014-12-28 13:12:15 (107 MB/s) - 'index.html.3' saved [1284/1284]

mininet> 
```

Figure 2.3-4: Console showing the topology

Figure 2.3-4 shows the mininet console which consist of sudo mn –topo=single, 4 commands that show the topology. The mininet dump Command would show the various nodes. Controller is not been specified therefore mininet would run the OpenFlow reference controller C0.



```
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> exit
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Stopping 1 controllers
c0
*** Done
completed in 11.739 seconds
root@[illegible]-OptiPlex-9010:~# sudo mn --topo=single,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=[illegible]>
<Host h2: h2-eth0:10.0.0.2 pid=[illegible]>
<Host h3: h3-eth0:10.0.0.3 pid=[illegible]>
<Host h4: h4-eth0:10.0.0.4 pid=[illegible]>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None
pid=[illegible]>
<OVSController c0: 127.0.0.1:6633 pid=[illegible]>
mininet> 
```

Figure 2.3-5: Console showing the launching of http request from H1 to H4.

In the figure 2.3- 5 A simple python web server is launched in H4 by h4 python –m SimpleHTTPServer 80 &command .Now a simple http request is generated from H1 to H4 by h1 wget 10.0.0.4 command.

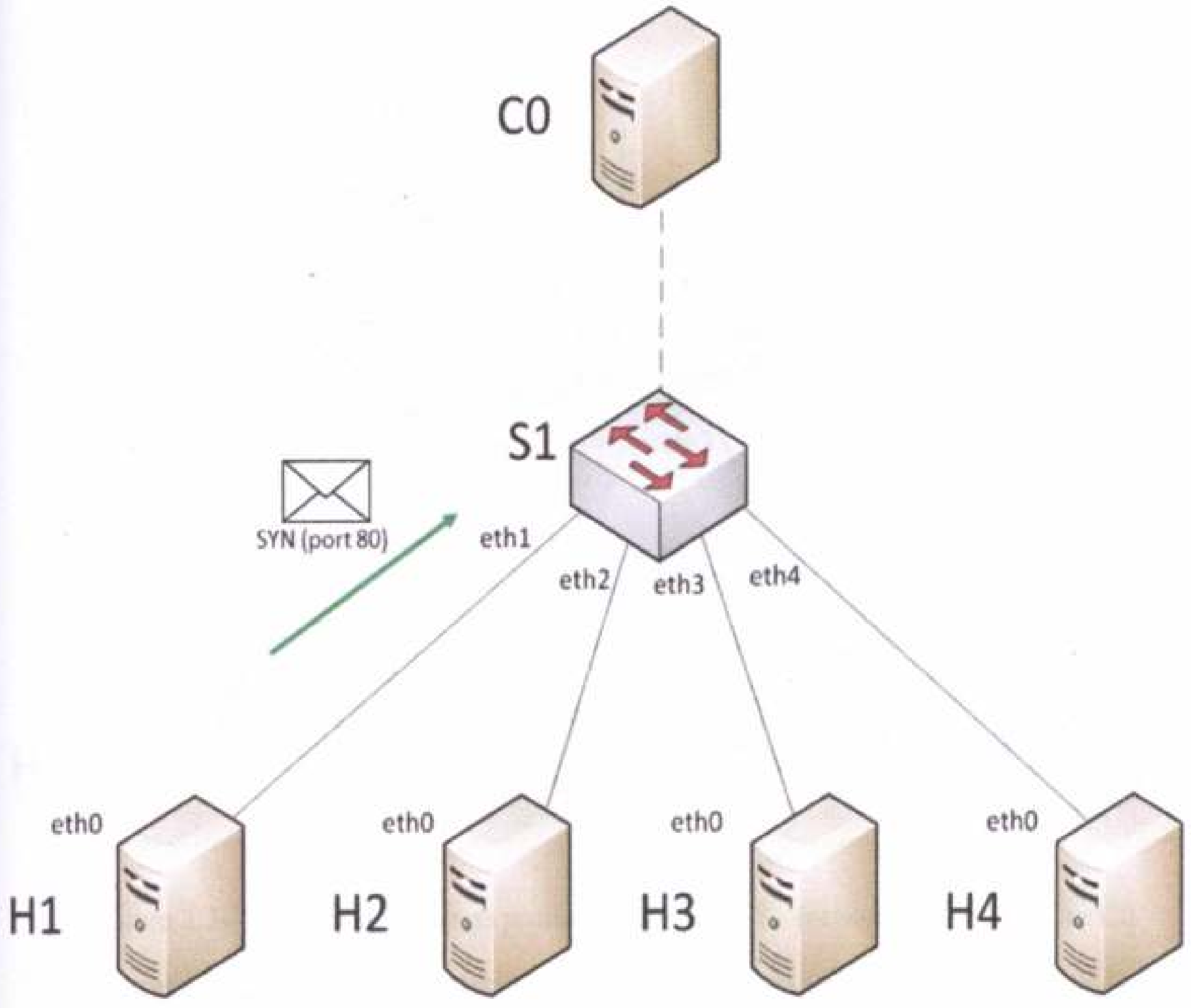


Figure 2.3-6: SYN packet passed from H1 to switch S1

Since it is a TCP conversation therefore it always start with a SYN message as shown in figure 2.3-6.As it is an OpenFlow switch,S1 would check its own local OpenFlow tables. Since it is the first packet for OpenFlow, it probably do not have any flow entry matching this packet, this is called the table-miss. Usually there are no matching flows the default action is to send this packet to the controller as shown in figure 2.3- 7.

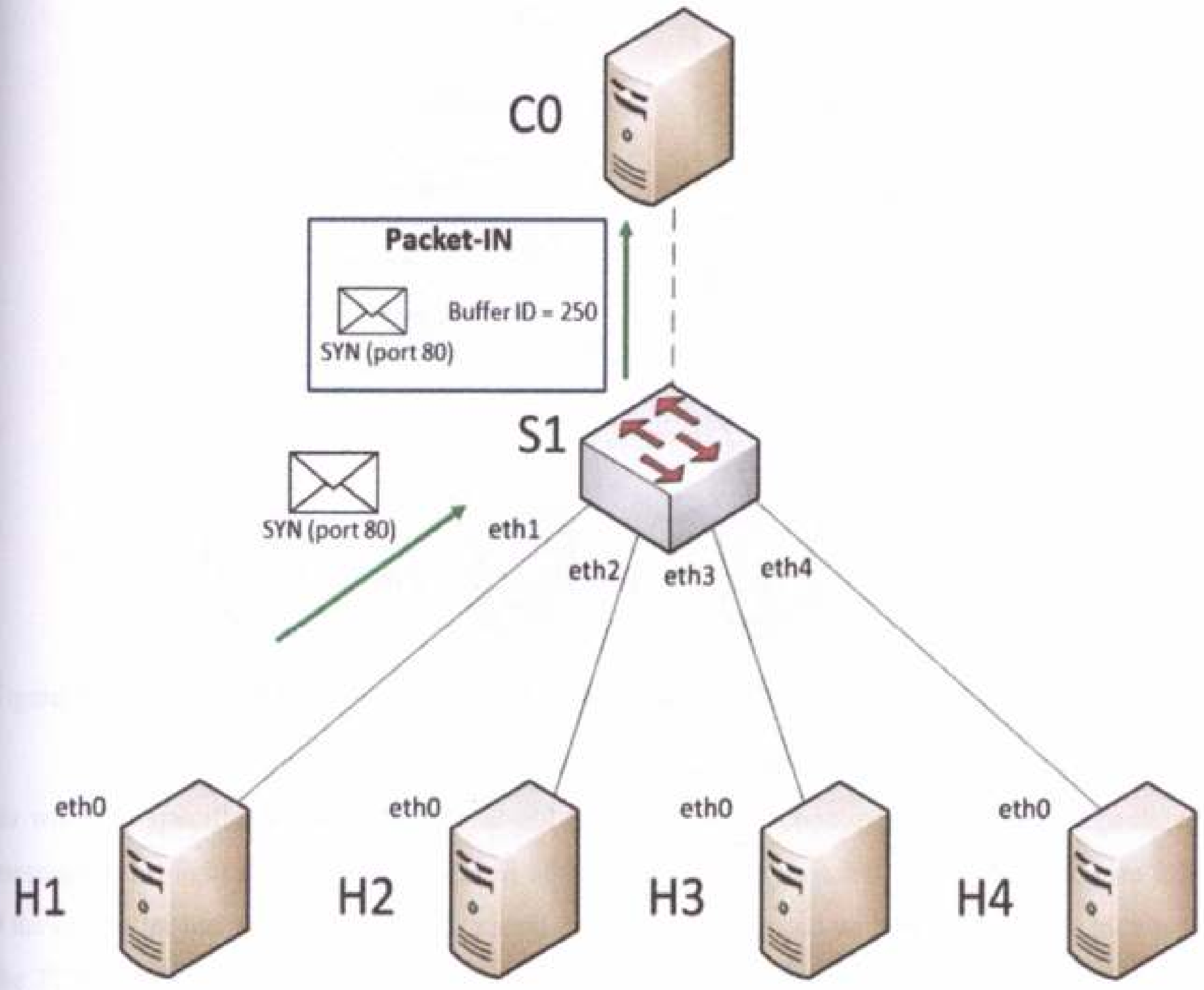


Figure 2.3-7: Packet-in message produced by switch S1 forwarded to Controller C0

S1 would send a Packet-in message to C0.This Packet-in message would encapsulate the original TCP SYN message. It may include the entire packet or might include the entire packet headers and references a buffer id. When using a buffer id, the switch buffers the entire packet so that the controller later instruct the switch what to do with the stored packet by referencing the buffer id. Now the controller get the Packet-in message. Typically a couple of things can happen here. The controller can send out a Packet-out message or Flow-modification message back to the switch.

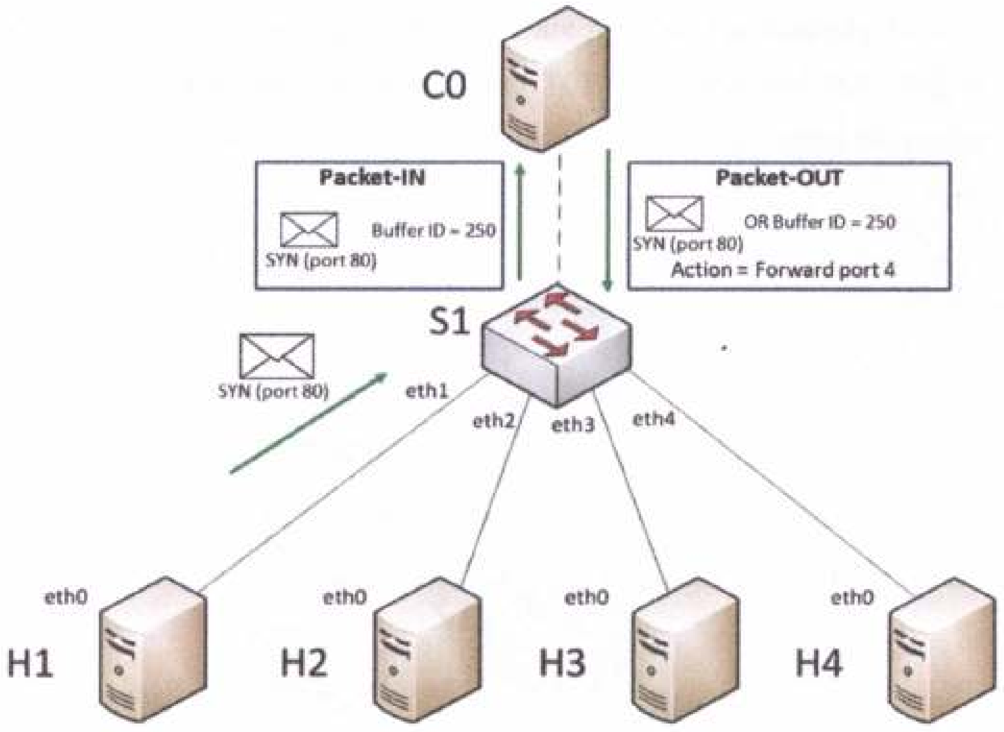


Figure 2.3-8: Packet-out message generated from Controller C0 forwarded to switch S1

In case of Packet-out message, it is the instruction from controller to the switch what to do with the specific packet. The Packet-out message would contain a complete encapsulated message or it might reference the buffer id of the packet to the switch is drawn. In the figure 2.3-8 the controller instruct the switch S1 to send the packet referenced with buffer id-250 which was the TCP SYN message from H1 out of its port 4 toward H4.Alternatively, the controller might also send the Flow-modification message which instruct the switch to install a new flow-entry in its flow table. Flow entries let the switch know what to with future similar packet arrives at the switch based on matching fields and masks. In the figure (2.3-9), the controller says effectively that any TCP port 80 request from IP address and MAC address of H1 to the IP address and MAC address of H4, send all of those out of port 4. The flow modification message also contain a buffer id, this will tell the switch that the first packet it had buffered i.e. number-250, release that packet from the buffer and apply the Action to the message. In the figure 2.3-9 consist of a

Simple and single Action but many set of Action are possible. For example Action can include changing multiple headers like Ips, MACs, and TCP ports. We can POP, PUSH or SWAP ampules labels. We also take Actions like flooding out all ports, dropping the packets or telling the switch to send the

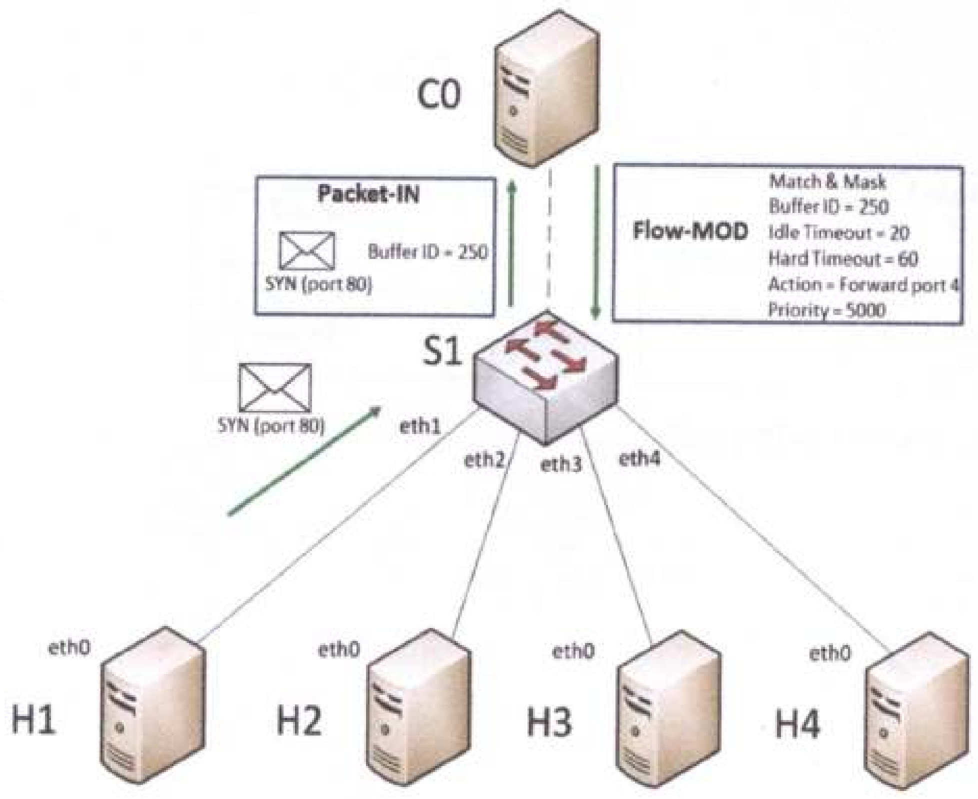


Figure 2.3-9: Flow-modification message generated from switch S1 forwarded to controller C0.

Matching packet to the controller. We can also tell the switch to use its non-OpenFlow packet processing. The Flow-modification message also shows two kind of timeouts. The timeout says how long to catch the flow entries. The Flow-modification message shows two timeouts:

Idle Timeouts: The Idle timeout is 20 which means if there won't be any matching http packets request for 20sec, remove this flow entry.

Hard Timeouts: The Hard timeout of 60 means, no matter if there are alike matching packets or not, just remove this flow entry. Another important component is the priority. Priority is required to sort the various flow entries.

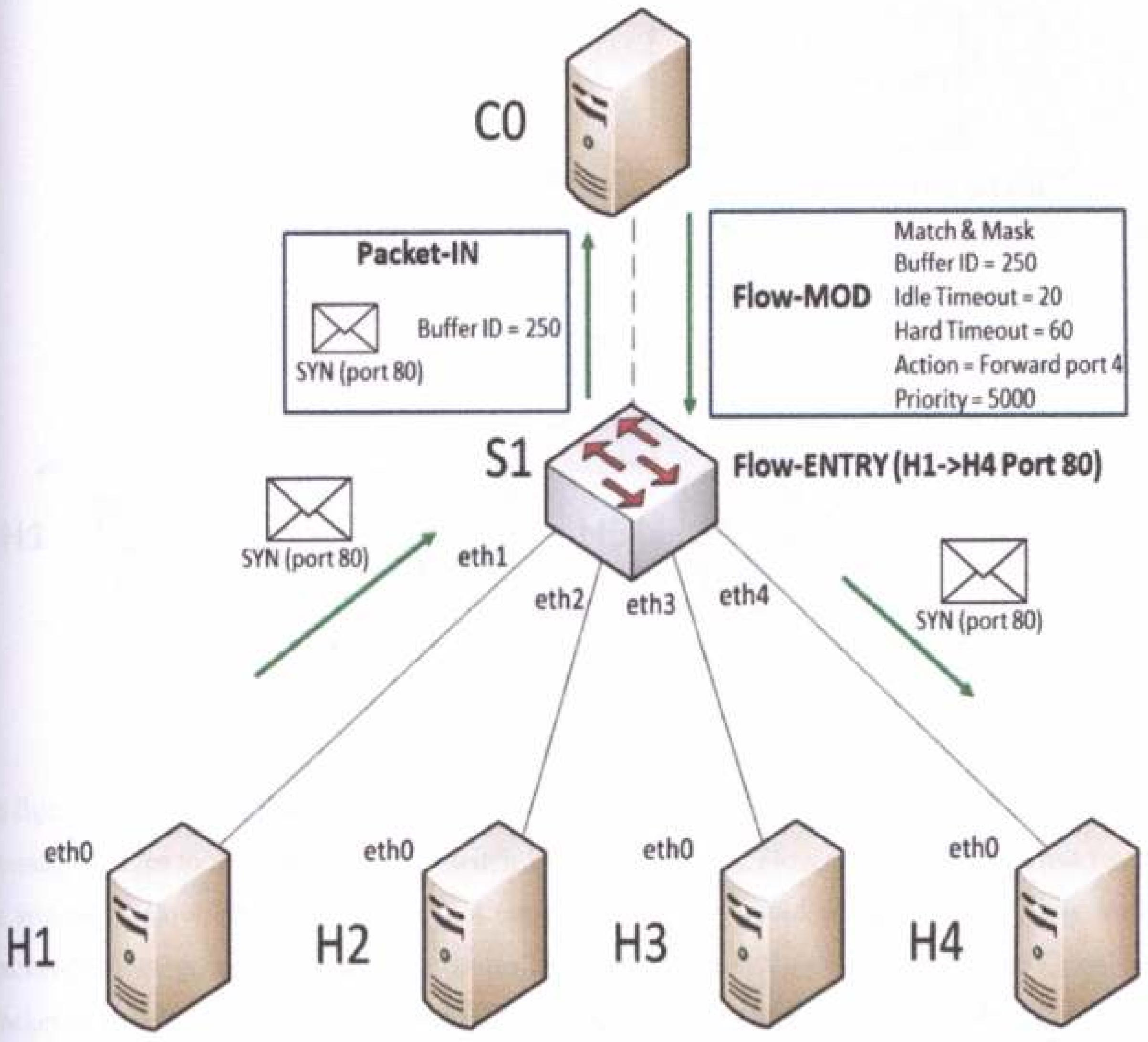


Figure 2.3-10: SYN message received by H4

The Flow-modification message resultant a new flow entry in switch S1 shown in figure 2.3-10 and since it references a buffer id, it also results in original TCP SYN message packet can be forwarded out of port 4 just like the action says to do. H4 receives the packet from H1.

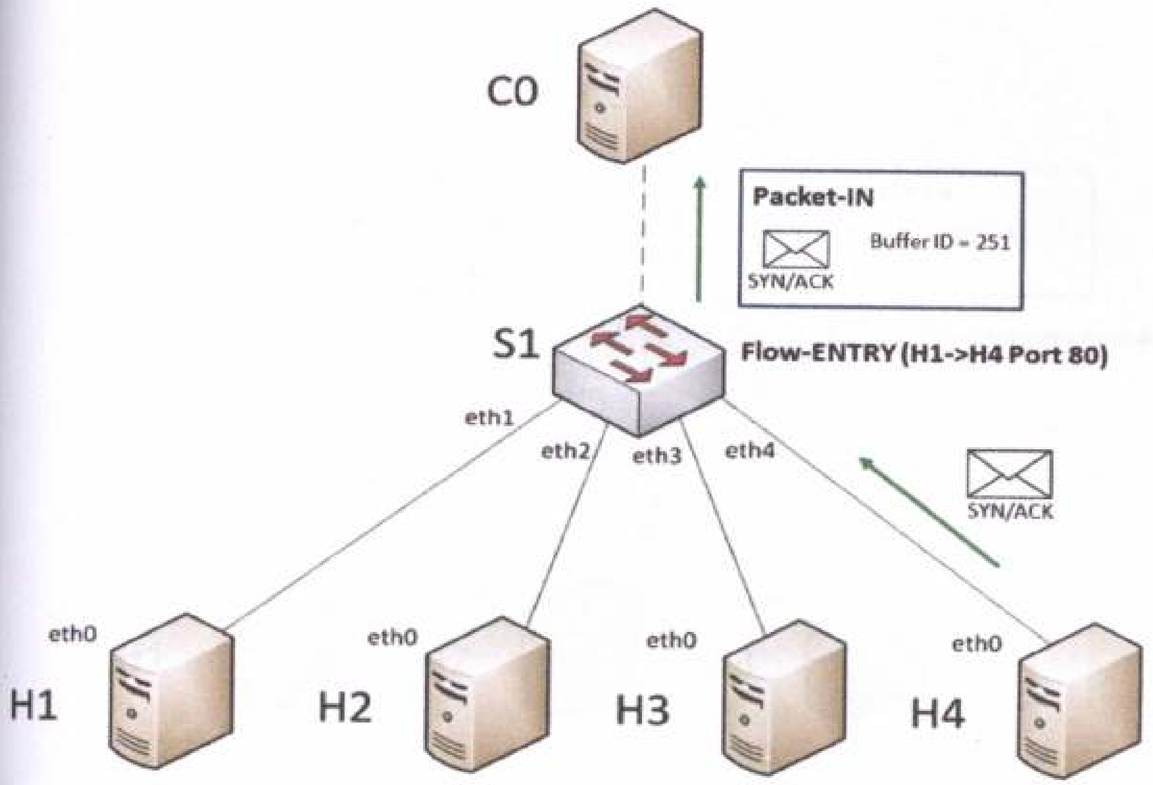


Figure 2.3 -11: H4 generates SYN/ACK message, received by switch S1 and S1 generates Packet-in message.

In figure 2.3-11 H4 replies the TCP SYN message through SYN/ACK message. The SYN/ACK message arrives to the switch S1.The switch S1 has no specific Flow entries matching this packet at any of its flow table. Again this is a table-miss. So S1 would encapsulate the reply into a Packet-in message and sends it to the controller. It may contain the entire packet in an OpenFlow Packet-in message or might be some headers from the packet along with the buffer id reference number .In this case it is 251.The SDN controller C0 would typically send a Packet-out or a Flow-modification message. The Packet-out message would tell the switch what to do with the specific packet. Again the controller might send the entire packet in one sent encapsulating within the Packet-out message or might references a buffer id stored in the switch saying to release the packet stored in buffer id 251 though proceeding the Actions specified in the Packet-out message shown in figure 2.3-12.

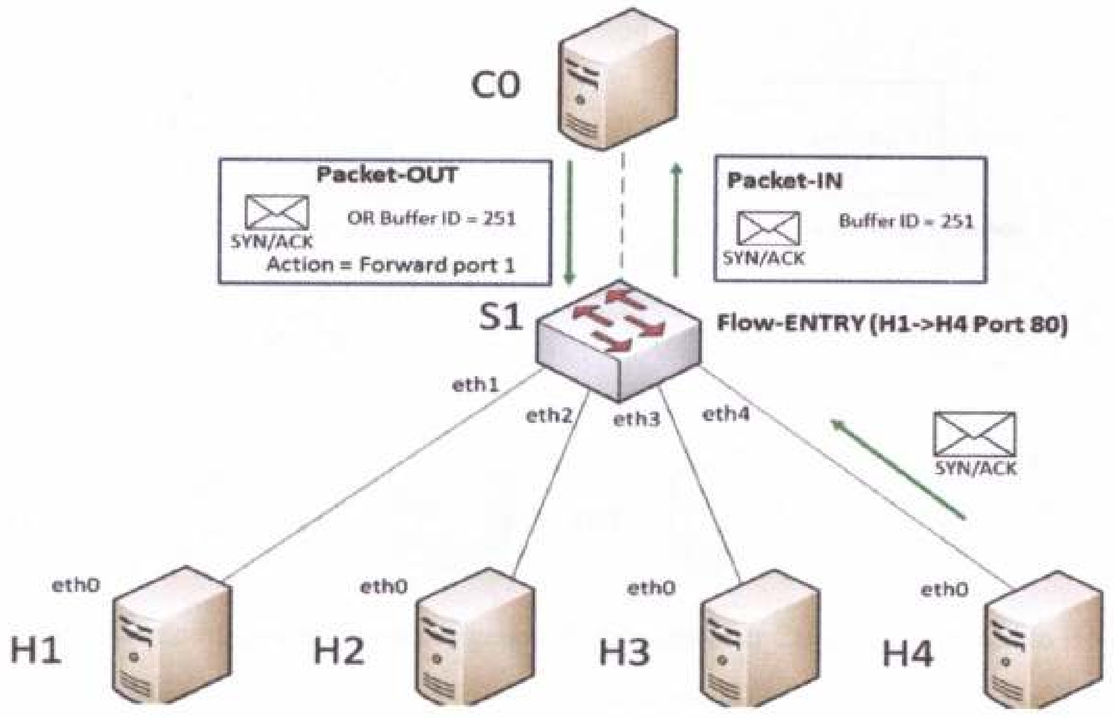


Figure 2.3-12: Controller C0 generates Packet-out message and send it to switch S1

Again the controller might send a Flow-modification message specifying what to do with the future packets matching specified fields. It may also have a buffer id to reference the currently buffer packet on the switch. It also have the timeouts, Actions and priority fields shown in figure 2.3-13.

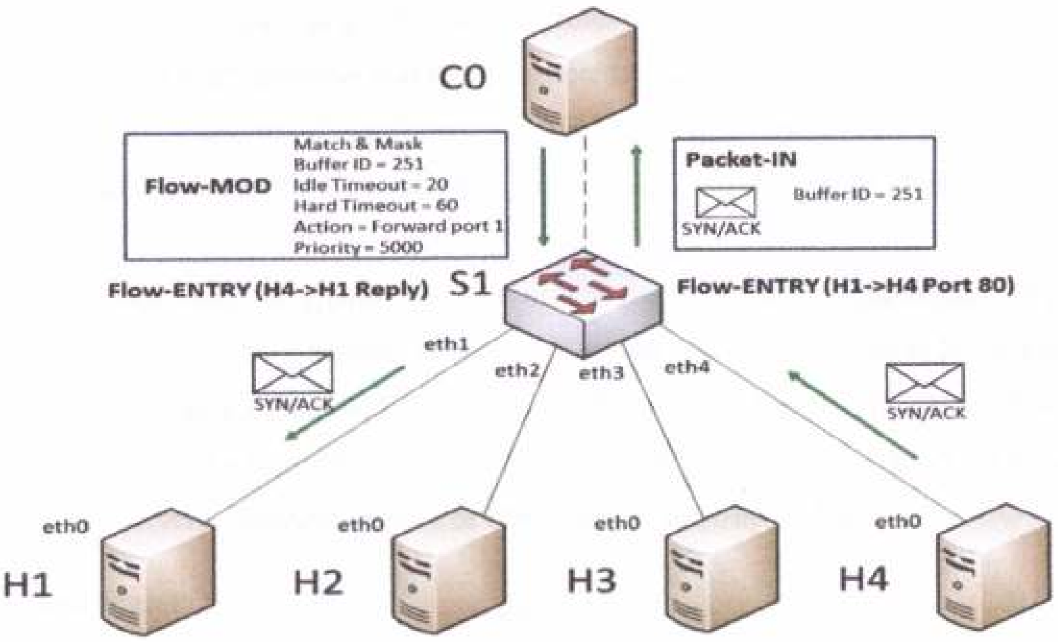


Figure 2.3-13: Flow-modification message is generated by controller C0 which is passed to H1

Thus through port1 the H1 receives the SYK/ACK message. The end results is simply some flow entries had cast on the OpenFlow switch S1 as shown in figure 2.3-14.This means that the rest of the conversation between H1 and H4 won't have to go to the controller since the switch have the flow entry and flow tables what to with the similar kind of packets. Thus it's the completion of the TCP handshake go through as well as the http request and the http reply go through the switch S1without any need to talk to the controller C0 .

2.4 Data Center Concept and Constructs:

Prior to the existence of data centers, computing, storage, and the networks that interconnected them existed on the desktop PCs of enterprise users. As data storage grew, along with the need for collaboration, departmental servers were installed and served this purpose. However, they provided services that were dedicated only to local or limited use. As time went on, the departmental servers could not handle the growing load or the widespread collaborative needs of users and were migrated into a more centralized data center. Data centers facilitated in case of hardware and software management and maintenance and could be more easily shared by all of the enterprise's users. Modern data center were originally created to physically separate traditional computing elements (e.g., PC servers), their associated storage (i.e., storage area networks or SANs) and the networks that interconnected them with client users. The computing power that existed in these types of data centers became focused on specific server functionality, such as running applications that included mail servers, database servers, or other enterprise IT applications.

Modern data centers were originally created to physically separate traditional computing elements (e.g., PC servers), their associated storage (i.e., storage area networks or SANs) and the networks that interconnected them with client users. The computing power that existed in these types of data center became focused on specific server functionality, such as running applications that included mail servers, database servers, or other enterprise IT applications.[10]

A Modern data center:

With further advances and increases in memory, computing, and storage, data center servers were increasingly capable of executing a variety of operating systems simultaneously in a virtual environment. Operating systems such as Windows Server that previously occupied an entire bare metal machine were now executed as virtual machines, each running whatever applications client users demanded. Moreover, network administrators now had the option to locate that computing power based not on physical machine availability. They could instead dynamically grow and shrink it as resource demands changed. Thus began the age of elastic computing. Within the elastic computing environment, operations departments were able to move servers to any physical data center location simply by pausing a virtual machine and copying a file across their network to a new physical computing location (i.e. server).They could even spin up new virtual machines simply by cloning the same file and telling the hypervisor, either locally or on some distant machine, to execute it as a new instance of the same service, thus expanding that resource. If the resource was no longer needed or demand waned, server instances could be shut down or even just deleted. This flexibility allowed network operators to start optimizing the data center resource location and thus utilization based on metrics such as power and cooling. By using bin packing techniques, virtual machines could be tightly mapped onto physical machines, thus optimizing for different characteristics such as locality of network between these servers, or as a means of even shutting down unused physical machines to save on power or cooling. In fact, this is how many modern data centers optimize for virtual machine placement because their dominating cost factors are power and cooling. In these cases, an operator can turn down (or off) cooling an entire portion of a data center. Similarly, an operator could move or dynamically expand computing, storage, or network resources by geographical demand. Figure 2.4-1 shows a modern data center.

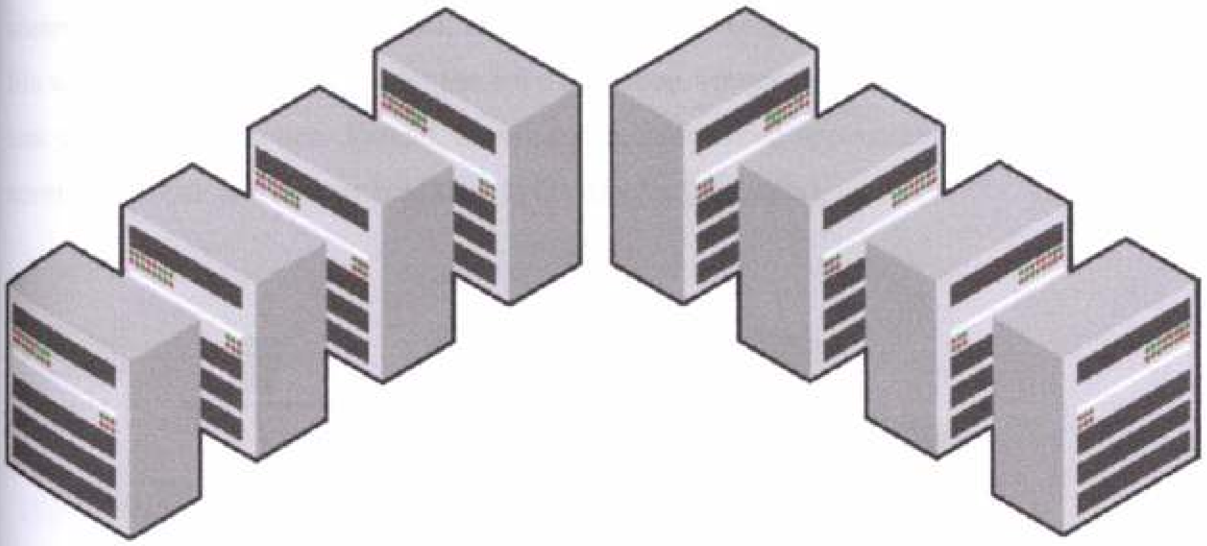


Figure 2.4-1: A modern data center comprised of compute, storage, and network resources

As with all advances in technology, this newly discovered flexibility in operational deployment of computing, storage, and networking resources brought about a new problem: one of operational efficiency both in terms of maximizing the utilization of storage and computing power and in terms of power and cooling.

As mentioned earlier, network operators began to realize that computing power demand in general increased over time. To keep up with this demand, IT departments would order all the equipment they predicted would be needed for the following year. However, once this equipment arrived and was placed in racks, it would consume power, cooling and space resources—even if it was not used for many months.

The three-tiered architecture:

The three-tiered architecture is shown in Figure 2.4-2. In the three-tiered architecture, a single VM is required to implement each of the three layers of the system, but more component VMs can be spun-up at any time to execute either locally on the same compute server or remotely in order to expand or contract computing resources. There is still some debate about how frequently live migration might occur between data centers as a DCI use case. The reasoning is that in order for any migration to take place, a file copy of the active VM to a new compute server must be performed, and while a VM is being copied, it cannot be running, or the file would change out from under the copy operation. Hence this operation is becoming less and less common in practice. Instead, moving to a three-tiered application architecture where the norm is to create and destroy machines is far simpler (and safer).

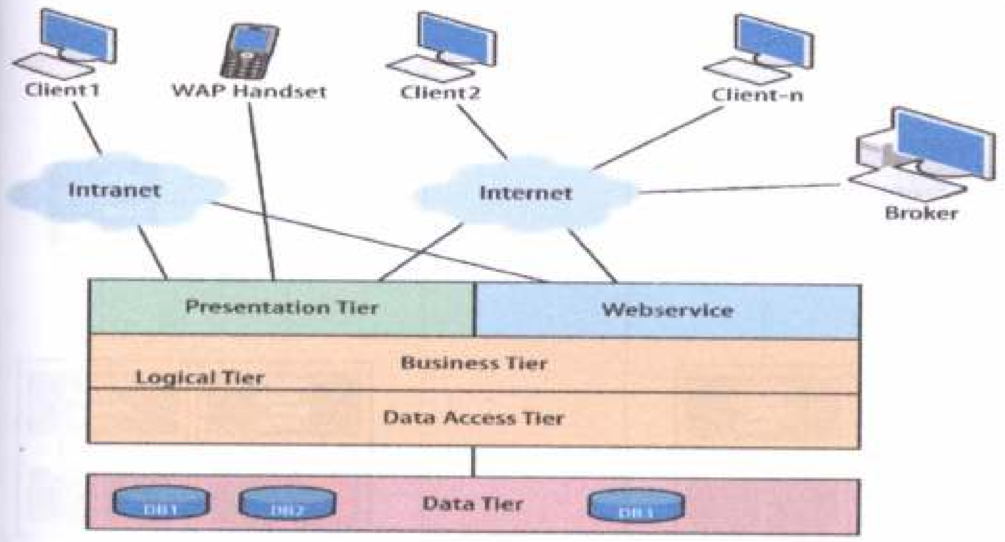


Figure 2.4-2:The three-tiered architecture

Data Centre Interconnect (DCI):

Now that we have introduced the basic concepts of what a data center is and how one can be built, let's discuss how one or more data centers can be connected. In particular, for configurations where multiple data centers are required either for geographic diversity, disaster recovery, service, or cloud bursting, data centers are interconnected over some form of Wide Area Network (WAN). This is the case even if data centers are geographically down the street from one another; even in these cases, some metro access network is typically used to interconnect them. A variety of technological options exist that can achieve these interconnections.

In cases where two or more data centers exist, then you must consider how to connect these data centers. For example, a tenant may have arbitrary numbers of virtual machines residing in each of these different data centers yet desires that they be at least logically connected. The Data Centre Interconnect (DCI) (see Figure 2.4-3) puts all VMs of a given tenant across all data centers on the same L2 or L3 underlying tenant network.

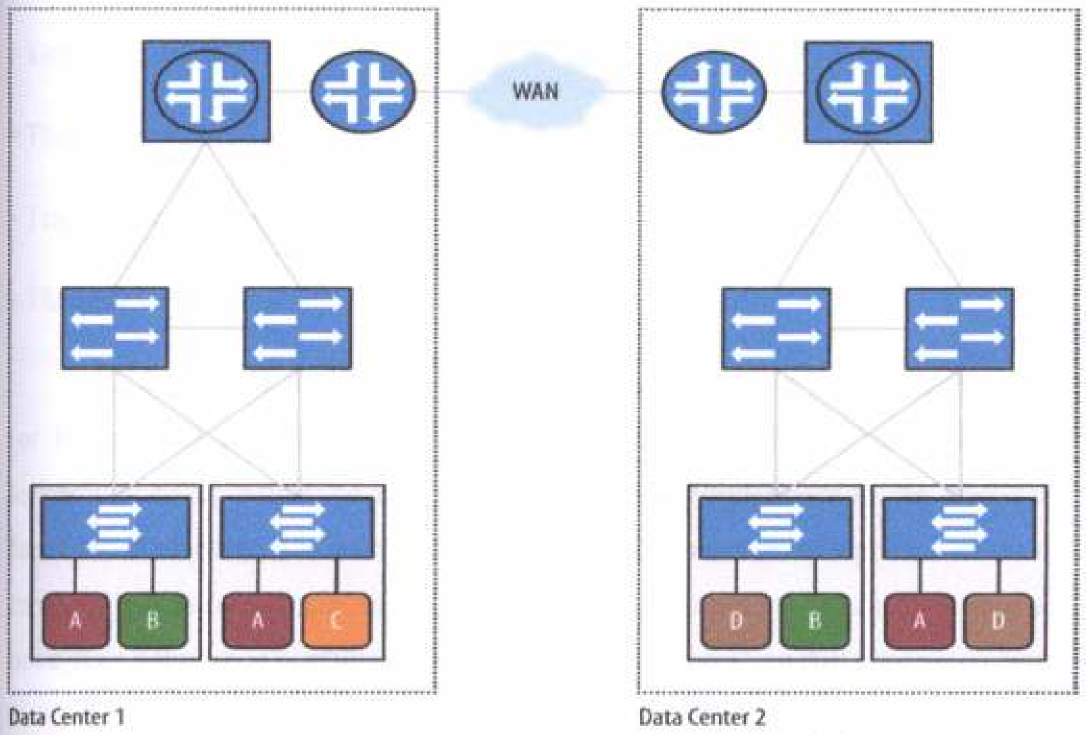


Figure 2.4-3: Data Centre Interconnect (DCI)

As it turns out, interconnecting data centers is not necessarily a simple thing because there are a variety of concerns to keep in mind. But before jumping feet first into all the various ways in which DCI can be implemented, let's first examine some of the requirements of any good DCI solution, and more importantly, some of its fallacies.

Fallacies of Data Centre Distributed Computing:

When designing a data center and an architecture, or strategy, for interconnecting two or more data centers, one inevitably needs to list the requirements of the interconnection. These often start with a variety of assumptions, and we have found that in practice, many of these fall into a category of things that seem to make sense in theory, but in practice are impossible to guarantee or assume. These assumptions include the following:

- The network is reliable.
- Latency is zero.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

At first, these notions may seem quite reasonable, but in practice they are quite difficult (or impossible) to achieve. For example, the first four points make assumptions about the technology being employed and the equipment that implements it. No equipment is perfect or functions flawlessly, and so it is often a safer bet to assume the opposite of these first points. In terms of points four to six, these fall under administrative or personnel issues. All things equal, once your network is configured, it should continue operating that way until changed.[11]

2.5 Area of Implementation:

One example, where SDN can provide huge wins, is in the data centre. A data centre, typically consists of many racks of servers. And any particular cluster might have, as many as 20,000 servers. Assuming that each one of these servers can run about 200 virtual machines. That's 400,000 virtual machines in a cluster. A significant problem is provisioning or migrating these virtual machines in response to varying traffic loads. SDN solves this problem by programming the switch state from a central database. So supposing, I have two virtual machines within the data centre that needs to communicate with one another. The forwarding state in the switches, in the data centre ensures that traffic is forwarded correctly. If we need to provision additional virtual machines. Or migrate a virtual machine from one server to another in the data centre, the state in these switches must be updated. Updating the state in this fashion is much easier to do, from a central controller or a central database, facilitating. This type of Virtual Machine Migration in the data centre is one of the early killer apps of software-defined networking. This type of migration is also made easier by the fact that the servers are addressed with Layer two Addressing. And the entire data centre Looks like a flat layer two topology. What this means, is that a server can be migrated from one portion of the data centre to another without requiring the virtual machine to obtain new addresses. All that needs to happen for forwarding to work,is the state of these switches. Needs to be updated. The task of updating switch date in this fashion is very easy to do, when the control data plans are and separate.

2.5.1 Data Centre:

In today's data centers often employ SDNs that are layered in most cases. However, information overwhelming has been a limitation to SDN deployment. A management model for data center networks has been presented in which, regional networks on lower layers will be aggregated and viewed as single switches to upper layers. Management information will be divided into three parts, which can be seen by network managers, regional controllers and tenants, respectively. Multi-tier data center networks (DCN) deploy rigid control and management platforms, which are not able to accommodate the ever-growing workload driven by the increasing demand of high-performance data center (DC) and cloud applications. In response to this, the EC FP7 project LIGHTNESS (Low Latency and High Throughput Dynamic Network Infrastructures for High Performance Datacenter Interconnects) is proposing a new flattened optical DCN architecture capable of providing dynamic, programmable, and highly available DCN connectivity services while meeting the requirements of new and emerging DC and cloud applications.

Chapter 3

3.1 Workdone:

We have created topologies using miniedit. The POX controller is implemented to control the network between different switches and carry out the bandwidth and the shortest path.In our implementation we have considered various topologies that is based on data center whish are FAT tree, Abilene topology etc.In this emulator which is mininet firstly we created a topology and assign some IP address to different Hosts, controller and switches, after that we have tried to send packets from one host to another and determine the shortest path between them using some algorithm.

3.1.1Testbed setup:

Using mininet software in Ubuntu environment we have perform our implementation, taking various topologies.In our implementation our main objective is to trace out the shortest path between two nodes for a particular topology. In this implementation we have created a topology in mininet environment and then use Bellmanford algorithm to find a shortest path between hosts.After creating the mininet topology we have moved to another terminal, kill the default controller, and run the Bellmanford algorithm of pox controller. Ping any two hosts from the mininet and we can see the shortest path that is created through Bellmanford algorithm.

3.1.2Mininet:

Mininet is a network emulator. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system.) The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middle box, with a given amount of queuing. When two programs, like an iperf client and server, communicate through Mininet, the measured performance should match that of two (slower) native machines. In short, Mininet's virtual hosts, switches, links, and controllers are the real thing – they are just created using software rather than hardware – and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform.

Some features of mininet:

(1) Its fast - starting up a simple network takes just a few seconds. This means that our run-edit-debug loop can be very quick.

(2) We can create custom topologies: a single switch, larger Internet-like topologies, the Stanford backbone, a data center, or anything else.

(3) We can run real programs: anything that runs on Linux is available for us to run, from web servers to TCP window monitoring tools to Wireshark.

(4) We can customize packet forwarding: Mininet's switches are programmable using the OpenFlow protocol. Custom Software-Defined Network designs that run in Mininet can easily be transferred to hardware OpenFlow switches for line-rate packet forwarding.

(5) We can share and replicate results: anyone with a computer can run our code once we have packaged it up.

(6) We can use it easily: we can create and run Mininet experiments by writing simple (or complex if necessary) Python scripts.

(7) Mininet is an open source project, so we are encouraged to examine its source code.

Working with Mininet:

Creating Topologies:

Mininet supports parameterized topologies. With a few lines of Python code, you can create a flexible topology which can be configured based on the parameters you pass into it, and reused for multiple experiments.

Running Programs in Hosts:

One of the most important things you will need to do in your experiments is run programs in hosts, so that you can run more tests than the simple pingAll() and iperf() tests which are provided by Mininet itself.

Each Mininet host is essentially a bash shell process attached to one or more network interfaces, so the easiest way to interact with it is to send input to the shell using the cmd() method.[3]

POX Controller:

POX is NOX's younger sibling. At its core, it's a platform for the rapid development and prototyping of network control software using Python. It's one of a growing number of frameworks for helping us write an Open Flow controller. As well as being a framework for interacting with OpenFlow switches, we're using it as the basis for some of our ongoing work to help build the emerging discipline of Software Defined Networking. We're using it to explore and prototype distribution, SDN debugging, network virtualization, controller design, and programming models. Our ultimate goal is to develop an archetypal, modern SDN controller. POX's is under active development, and we hope it stays that way. Is primary target is research, and many research projects are fairly short-lived. [4]

Goals of POX Controller:

(1) One of the goals for POX is to have it be easy to get up and running.

(2) Our ultimate goal is to develop an archetype for a modern SDN controller.

3.2 Result:

Using Bellmanford to find a shortest path:

In this implementation we have created a topology in mininet environment and then use Bellmanford algorithm to find a shortest path between hosts. After creating the mininet topology we have moved to another terminal, kill the default controller, and run the Bellmanford algorithm of pox controller. Ping any two hosts from the mininet and we can see the shortest path that is created through bellman ford algorithm.

A Simple Topology:

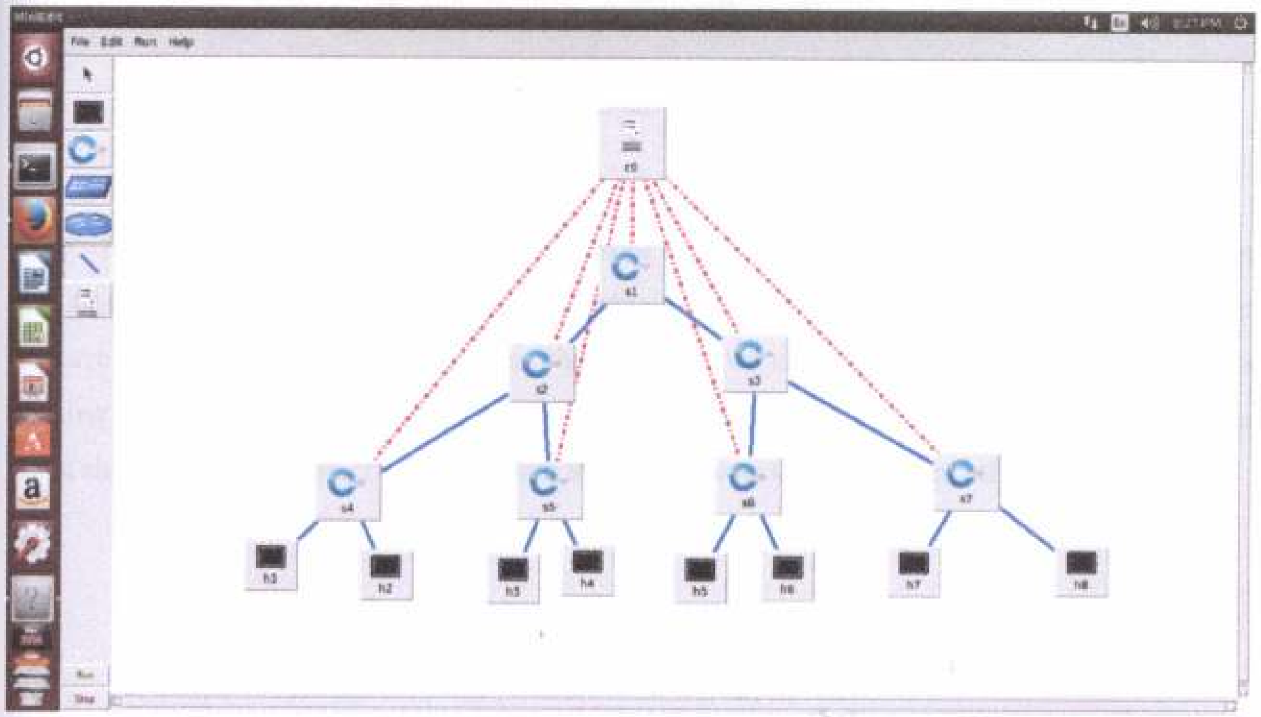
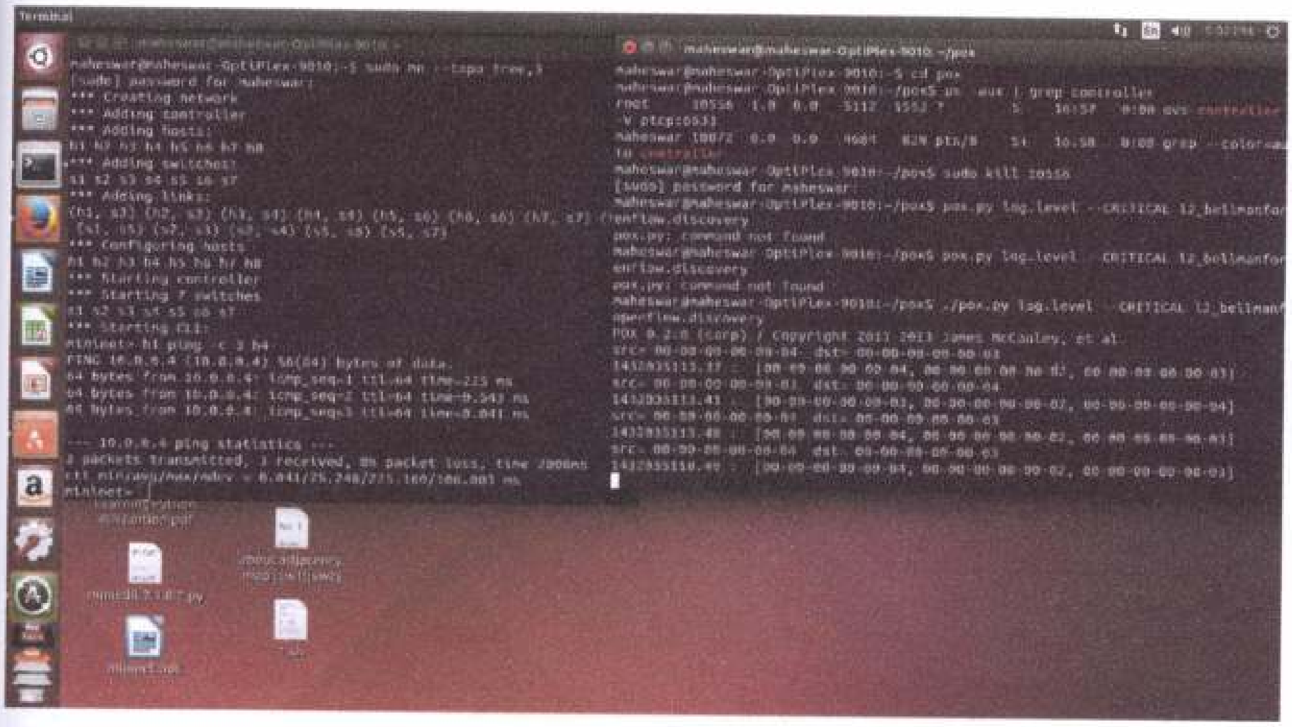


Fig 3.2.1: A Simple Topology in miniedit



```
maheswar@maheswar-OptiPlex-9010:~$ sudo mn --topo tree,3
[sudo] password for maheswar:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s6) (h6, s6) (h7, s7) (
 (s1, s5) (s2, s3) (s2, s4) (s5, s6) (s5, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
mininet> h1 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=225 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.543 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.041 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 0.041/75.248/225.160/106.001 ms
mininet>
```

```
maheswar@maheswar-OptiPlex-9010: ~/pox
maheswar@maheswar-OptiPlex-9010:~$ cd pox
maheswar@maheswar-OptiPlex-9010:~/pox$ ps -aux | grep controller
root     10556  1.0  0.0  5112  1552 ?        S    16:57   0:00 ovs-controller
 -v ptcp:6633
maheswar 10072  0.0  0.0  4684   828 pts/8    S+   16:58   0:00 grep --colour=a
 to controller
maheswar@maheswar-OptiPlex-9010:~/pox$ sudo kill 10556
[sudo] password for maheswar:
maheswar@maheswar-OptiPlex-9010:~/pox$ pox.py log.level --CRITICAL l2_bellmanfor
openflow.discovery
pox.py: command not found
maheswar@maheswar-OptiPlex-9010:~/pox$ pox.py log_level --CRITICAL l2_bellmanfor
enflow.discovery
pox.py: command not found
maheswar@maheswar-OptiPlex-9010:~/pox$ ./pox.py log.level --CRITICAL l2_bellmanf
openflow.discovery
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-01
1432035113.37 :  [00-00-00-00-00-04, 00-00-00-00-00-02, 00-00-00-00-00-03]
src= 00-00-00-00-00-01  dst= 00-00-00-00-00-04
1432035113.41 :  [00-00-00-00-00-03, 00-00-00-00-00-02, 00-00-00-00-00-04]
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-03
1432035113.48 :  [00-00-00-00-00-04, 00-00-00-00-00-02, 00-00-00-00-00-03]
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-03
1432035118.49 :  [00-00-00-00-00-04, 00-00-00-00-00-02, 00-00-00-00-00-03]
```

Fig 3.2.2: A snapshot of implementation on Bellmanford algorithm.

In the figure shown above is the snapshot of how a shortest is been tracked by bellman ford algorithm for a particular topology. For obtaining these we need to use the terminal where we have to create a topology and in the other terminal we need to run the pox controller. When we ping any two hosts of the topology and if it pings, we could observe from the pox controller that a shortest path is been created between these two hosts through Bellmanford algorithm.

A FAT Tree Topology:

In this implementation we have considered a FAT Tree topology, where we can determine the shortest path. Using a tool miniedit we have prepare this FAT Tree topology and try to ping any two hosts to determine the shortest path between these two hosts through which data can be send. In this topology we have taken many switches, hosts and one controller which is connected to each switches.

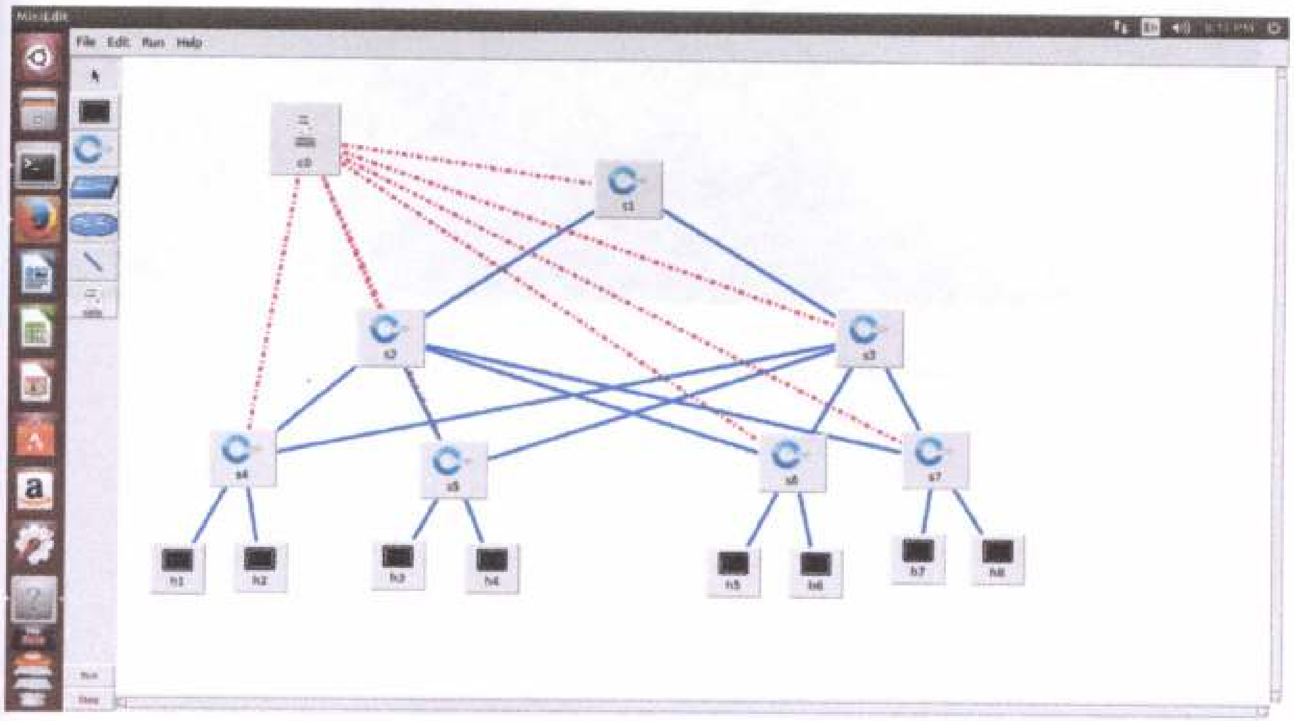
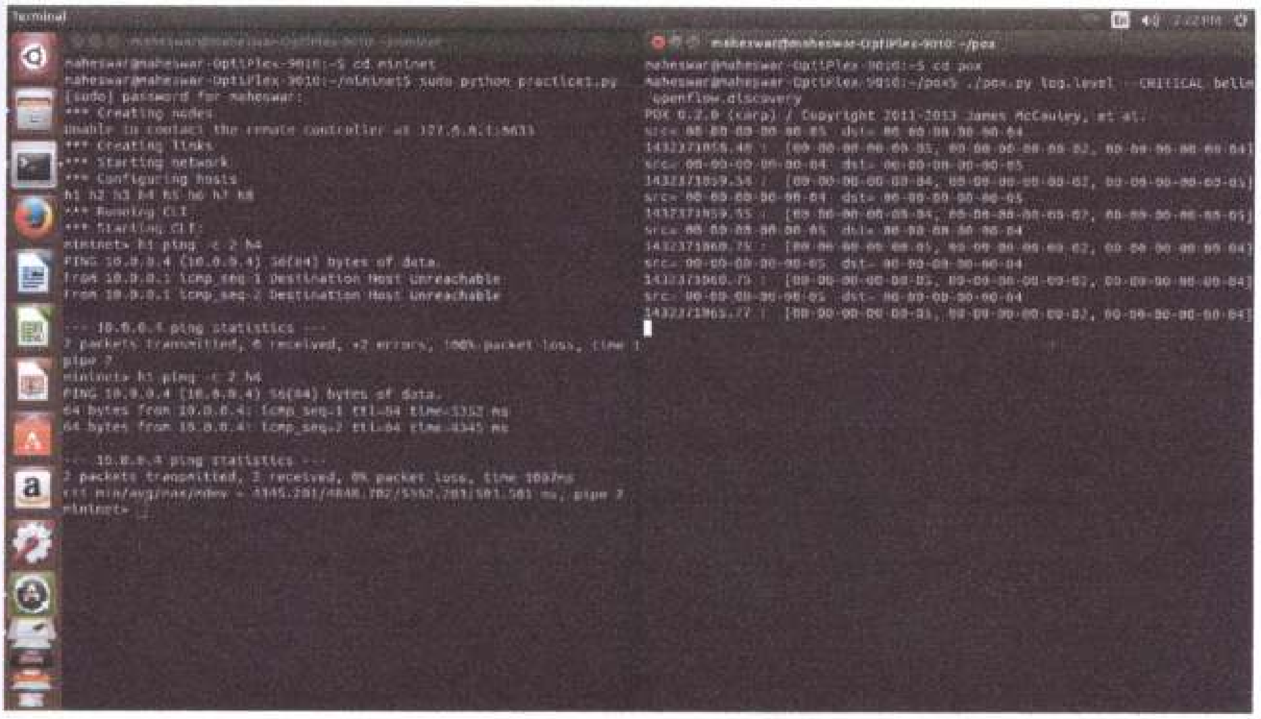


Fig 3.2.3: FAT Tree topology

In this figure shown above shows how hosts connect to each other and for pinging they trace out the shortest path. We have consider a FAT Tree topology where switches, hosts and controller are connected to each other. Using POX controller we run the shortest path algorithm which is shown above in the figure. Now when we ping any two hosts from the topology from the controller side we would observe the shortest path between two hosts.



```
maheswar@maheswar-OptiPlex-9010:~$ cd mininet
maheswar@maheswar-OptiPlex-9010:~/mininet$ sudo python practice1.py
[sudo] password for maheswar:
*** Creating nodes
Unable to contact the remote controller at 127.0.0.1:6633
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Running CLI
*** Starting CLI:
mininet> h1 ping -c 2 h4
PING 10.0.0.4 (10.0.0.4) 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

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1
pipe 2
mininet> h1 ping -c 2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=5352 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=4345 ms

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 4345.201/4848.702/5352.203/501.501 ms, pipe 2
mininet>
```

```
maheswar@maheswar-OptiPlex-9010:~$ cd pox
maheswar@maheswar-OptiPlex-9010:~/pox$ ./pox.py log.level --CRITICAL bellm
 openflow.discovery
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
src= 00-00-00-00-00-05  dst= 00-00-00-00-00-04
1432371858.48 :  [00-00-00-00-00-05, 00-00-00-00-00-02, 00-00-00-00-00-04]
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-05
1432371859.34 :  [00-00-00-00-00-04, 00-00-00-00-00-02, 00-00-00-00-00-05]
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-05
1432371859.55 :  [00-00-00-00-00-04, 00-00-00-00-00-02, 00-00-00-00-00-05]
src= 00-00-00-00-00-05  dst= 00-00-00-00-00-04
1432371860.75 :  [00-00-00-00-00-05, 00-00-00-00-00-02, 00-00-00-00-00-04]
src= 00-00-00-00-00-05  dst= 00-00-00-00-00-04
1432371860.75 :  [00-00-00-00-00-05, 00-00-00-00-00-02, 00-00-00-00-00-04]
src= 00-00-00-00-00-05  dst= 00-00-00-00-00-04
1432371865.77 :  [00-00-00-00-00-05, 00-00-00-00-00-02, 00-00-00-00-00-04]
```

Fig 3.2.4: Snapshot of implementation of FAT Tree topology

ABILENE Topology:

A high-performance backbone network suggested by INTERNET 2 is ABELINE Network. ABELINE Topology is prepared for a big geographical region such as a particular large country. An ABELINE network has more than 10Gbps networking between neighboring hosts. This ABELINE Topology have been prepared by us in mininet and try to determine the shortest path.

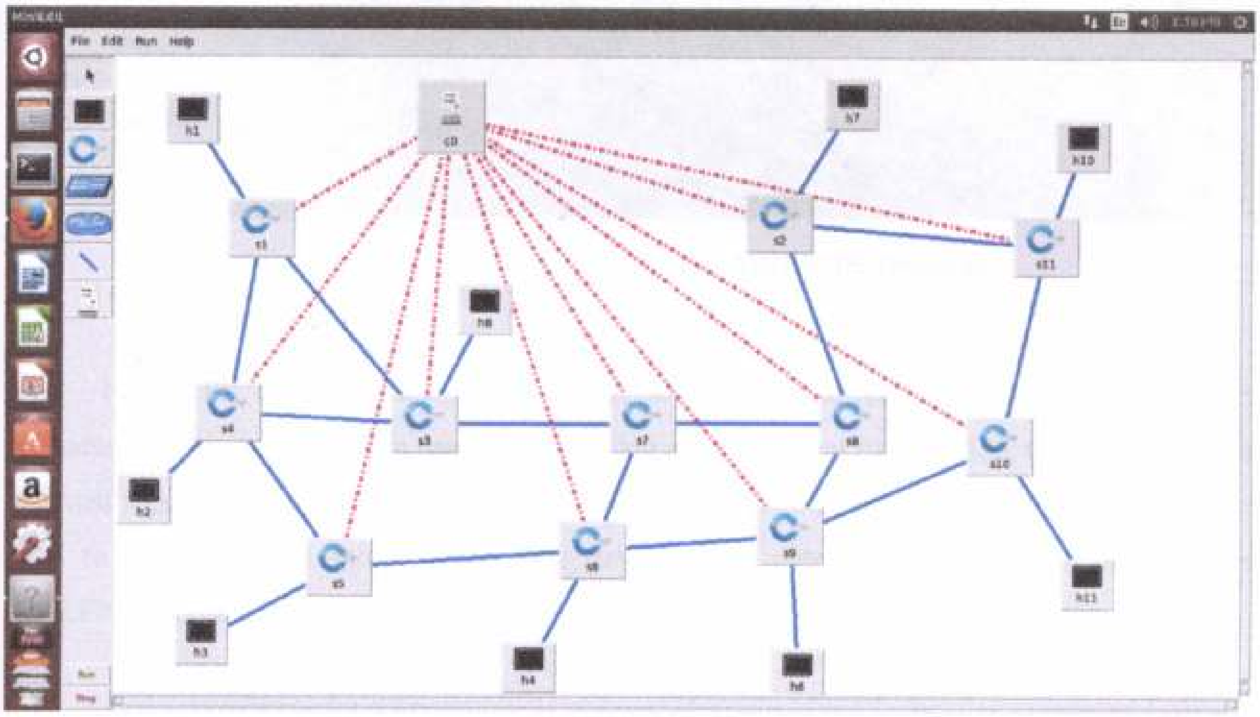
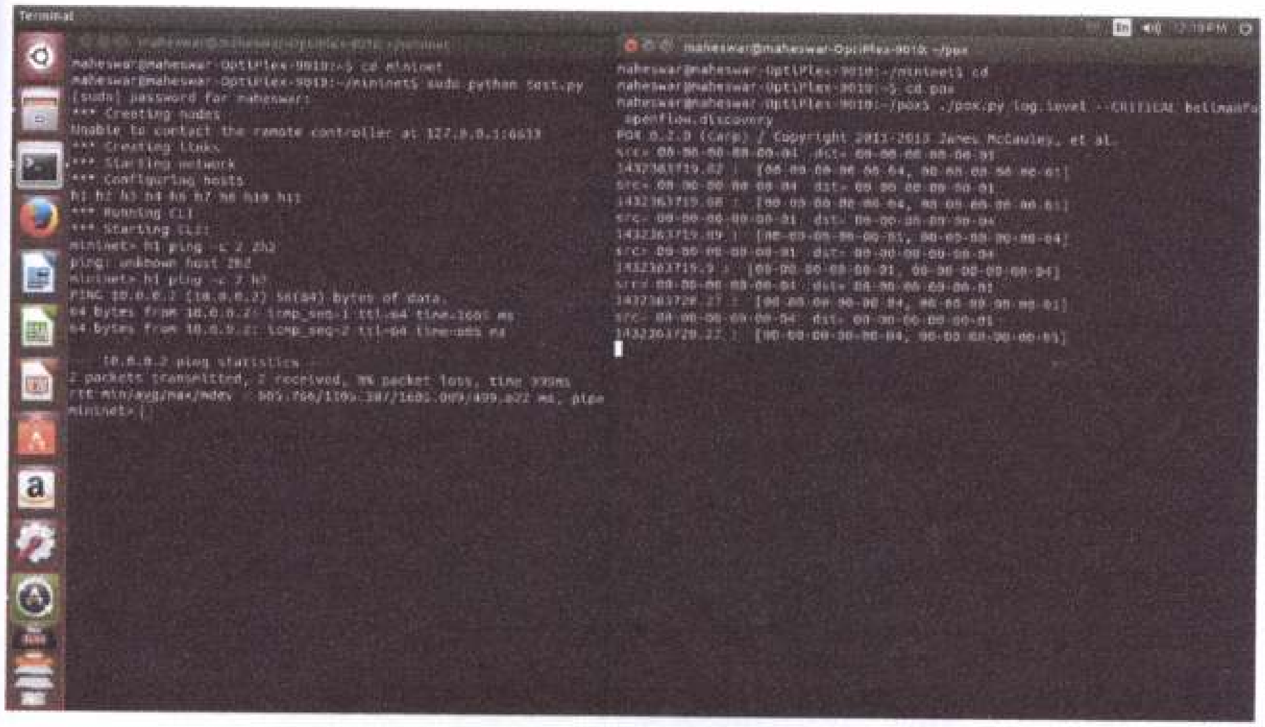


Fig 3.2.5: ABILENE topology



```
maheswar@maheswar-OptiPlex-9010:~$ cd mininet
maheswar@maheswar-OptiPlex-9010:~/mininet$ sudo python test.py
[sudo] password for maheswar:
*** Creating nodes
Unable to contact the remote controller at 127.0.0.1:6633
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h6 h7 h8 h10 h11
*** Running CLI
*** Starting CLI:
mininet> h1 ping -c 2 2h2
ping: unknown host 2h2
mininet> h1 ping -c 2 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=1605 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=605 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 605.766/1105.387/1605.009/499.622 ms, pipe
mininet> []
```

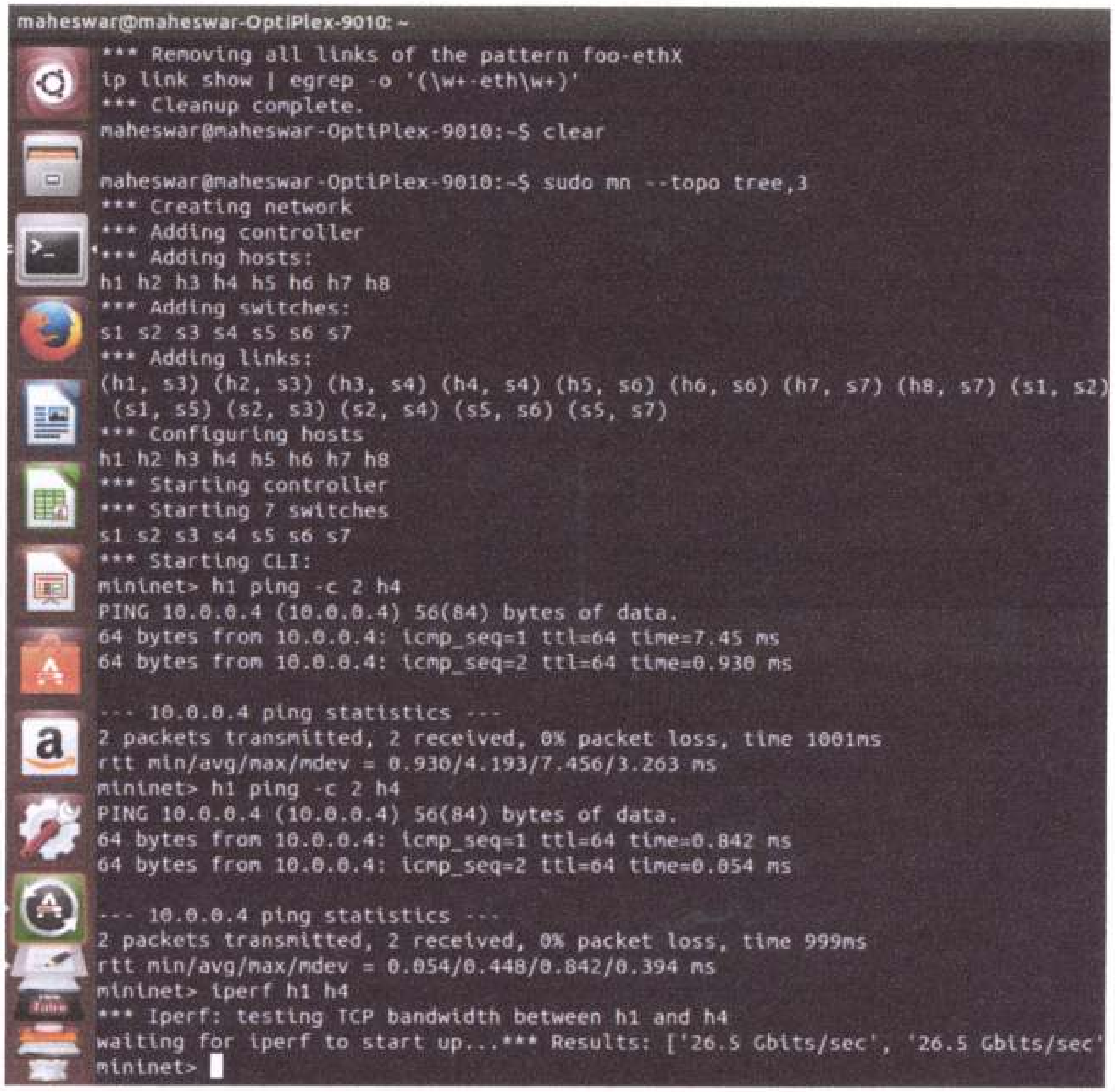
```
maheswar@maheswar-OptiPlex-9010:~/mininet$ cd
maheswar@maheswar-OptiPlex-9010:~$ cd pox
maheswar@maheswar-OptiPlex-9010:~/pox$ ./pox.py log.level --CRITICAL bellmanfo
 openflow.discovery
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
src= 00-00-00-00-00-01  dst= 00-00-00-00-00-01
1432363719.02 :  [00-00-00-00-00-04, 00-00-00-00-00-01]
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-01
1432363719.08 :  [00-00-00-00-00-04, 00-00-00-00-00-01]
src= 00-00-00-00-00-01  dst= 00-00-00-00-00-04
1432363719.09 :  [00-00-00-00-00-01, 00-00-00-00-00-04]
src= 00-00-00-00-00-01  dst= 00-00-00-00-00-04
1432363719.9 :  [00-00-00-00-00-01, 00-00-00-00-00-04]
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-01
1432363720.27 :  [00-00-00-00-00-04, 00-00-00-00-00-01]
src= 00-00-00-00-00-04  dst= 00-00-00-00-00-01
1432363720.27 :  [00-00-00-00-00-04, 00-00-00-00-00-01]
```

Fig 3.2.6: Snapshot of implementation of ABILENE Topology

Here is the implementation procedure of how we run the ABILENE topology in mininet and determine the shortest path using shortest path algorithm that runs in POX controller. As this is a vast topology there consists of various path to connect one host to another. From them we need to determine the shortest path. We have try we determine the shortest path of this ABILENE topology which is shown in the figure above.

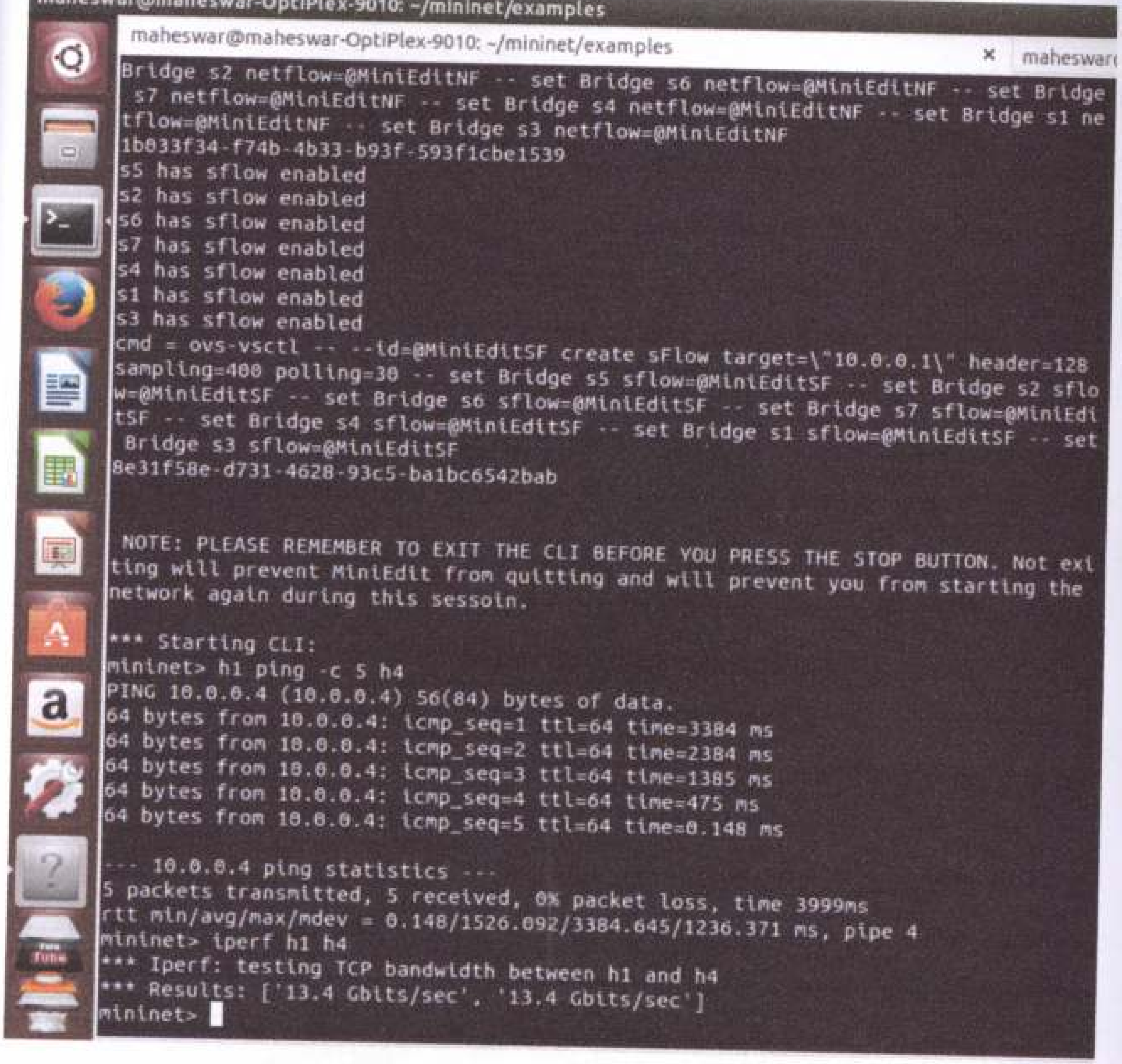
Iperf:

Though we need to determine the shortest path between hosts of a topology, along with it we need to determine the throughput of these two hosts of a topology. We can determine this bandwidth by using command Iperf. An Iperfis a tool to measure the throughput of a network that is carrying them.



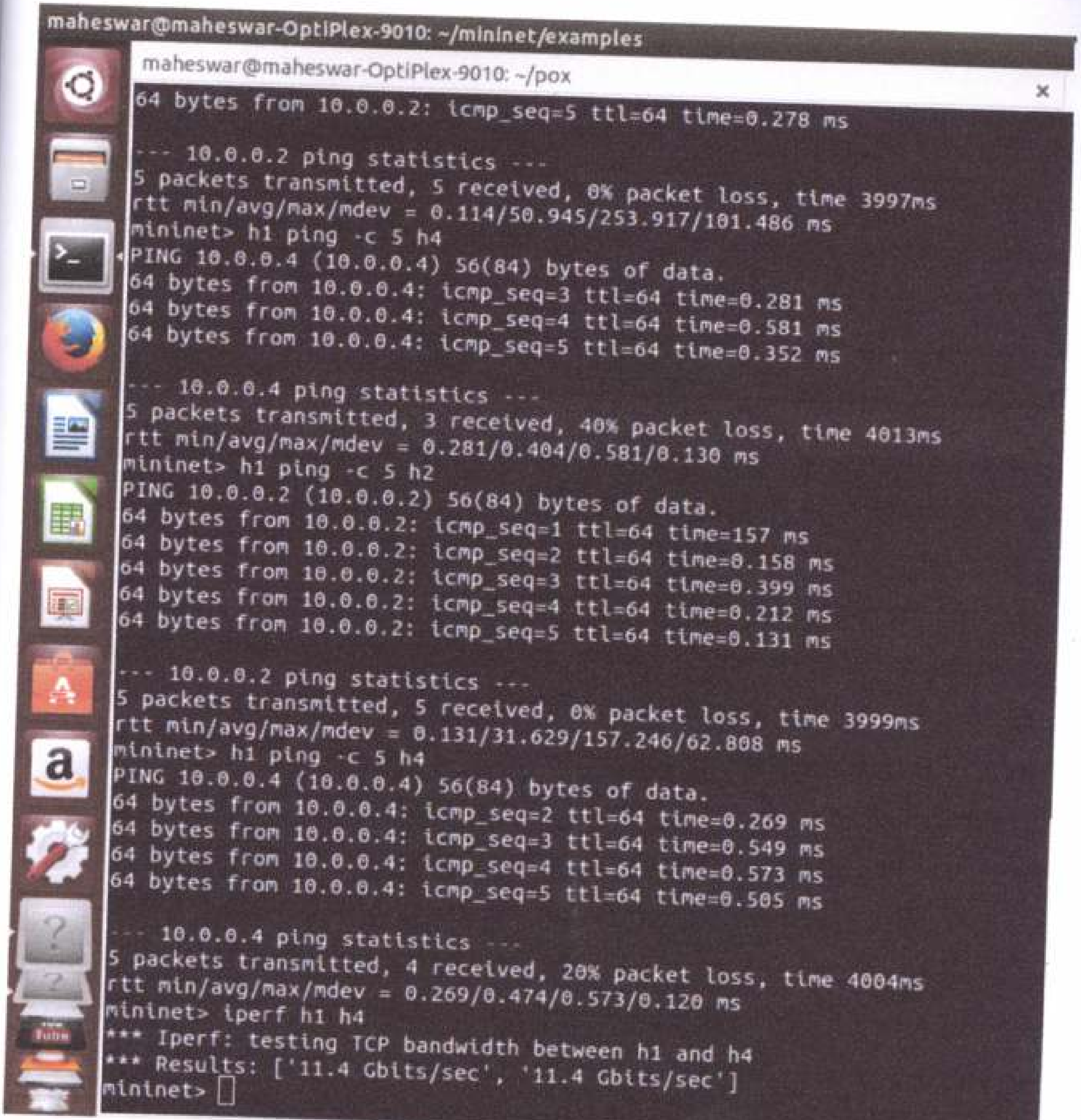
```
maheswar@maheswar-OptiPlex-9010: ~
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '(\w+-eth\w+)'
*** Cleanup complete.
maheswar@maheswar-OptiPlex-9010:~$ clear

maheswar@maheswar-OptiPlex-9010:~$ sudo mn --topo tree,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (h5, s6) (h6, s6) (h7, s7) (h8, s7) (s1, s2)
 (s1, s5) (s2, s3) (s2, s4) (s5, s6) (s5, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
mininet> h1 ping -c 2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=7.45 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.930 ms

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.930/4.193/7.456/3.263 ms
mininet> h1 ping -c 2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.842 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.054 ms

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.054/0.448/0.842/0.394 ms
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
waiting for iperf to start up...*** Results: ['26.5 Gbits/sec', '26.5 Gbits/sec'
mininet> 
```

Fig 3.2.7: Snapshot of bandwidth value in a simple topology



```
maheswar@maheswar-OptiPlex-9010: ~/mininet/examples
maheswar@maheswar-OptiPlex-9010: ~/mininet/examples
Bridge s2 netflow=@MiniEditNF -- set Bridge s6 netflow=@MiniEditNF -- set Bridge
 s7 netflow=@MiniEditNF -- set Bridge s4 netflow=@MiniEditNF -- set Bridge s1 ne
tflow=@MiniEditNF -- set Bridge s3 netflow=@MiniEditNF
1b033f34-f74b-4b33-b93f-593f1cbe1539
s5 has sflow enabled
s2 has sflow enabled
s6 has sflow enabled
s7 has sflow enabled
s4 has sflow enabled
s1 has sflow enabled
s3 has sflow enabled
cmd = ovs-vsctl -- --id=@MiniEditSF create sFlow target=\"10.0.0.1\" header=128
sampling=400 polling=30 -- set Bridge s5 sflow=@MiniEditSF -- set Bridge s2 sflo
w=@MiniEditSF -- set Bridge s6 sflow=@MiniEditSF -- set Bridge s7 sflow=@MiniEdi
tSF -- set Bridge s4 sflow=@MiniEditSF -- set Bridge s1 sflow=@MiniEditSF -- set
 Bridge s3 sflow=@MiniEditSF
8e31f58e-d731-4628-93c5-ba1bc6542bab


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this sessoin.

*** Starting CLI:
mininet> h1 ping -c 5 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=3384 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=2384 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=1385 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=475 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.148 ms

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.148/1526.092/3384.645/1236.371 ms, pipe 4
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['13.4 Gbits/sec', '13.4 Gbits/sec']
mininet>
```

Fig 3.2.8: Snapshot of bandwidth value in FAT Tree topology



maheswar@maheswar-OptiPlex-9010: ~/mininet/examples
maheswar@maheswar-OptiPlex-9010: ~/pox

```
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.278 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3997ms
rtt min/avg/max/mdev = 0.114/50.945/253.917/101.486 ms
mininet> h1 ping -c 5 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.281 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.581 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.352 ms

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 3 received, 40% packet loss, time 4013ms
rtt min/avg/max/mdev = 0.281/0.404/0.581/0.130 ms
mininet> h1 ping -c 5 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=157 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.158 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.399 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.212 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.131 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.131/31.629/157.246/62.808 ms
mininet> h1 ping -c 5 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.269 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.549 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.573 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.505 ms

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4004ms
rtt min/avg/max/mdev = 0.269/0.474/0.573/0.120 ms
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['11.4 Gbits/sec', '11.4 Gbits/sec']
mininet> 
```

Fig 3.2.9: Snapshot of bandwidth value in Abilene topology

Chapter 4

4.1:Conclusion:

In our implementation we have extended the Bellmanford shortest path algorithm considering nodes weights under SDN. In this implementation we have considered a particular complex topology and determine the shortest path through which a data can be send from source to destination. We have consider FAT Topology, Abilene Topology, and simple tree with mininet in which the bandwidth comparison for each topology varies as follows 13.4 Gbits/sec, 11.4 Gbits/sec, 26.5 Gbits/sec and for the measurement of this all topology we used iperf .

Reference:

1. http://en.wikipedia.org/wiki/Software-defined_networking
2. https://www.sdncentral.com/resources/sdn/what-the-definition-of-software-defined-networking-sdn/
3. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
4. http://www.noxrepo.org/pox/about-pox/
5. http://searchsdn.techtarget.com/definition/data-plane-DP
6. http://searchsdn.techtarget.com/definition/control-plane-CP
7. http://en.wikipedia.org/wiki/Routing_control_plane
8. http://vk5tu.livejournal.com/37824.html
9. http://standards.ieee.org/getieee802/download/802.112012.pdf
10. https://www.safaribooksonline.com/library/view/sdn-software defined/9781449342425/ch06.html
11. A book on Software defined network by Thomas D.Nadeau and Ken Gray
12. http://csie.nqu.edu.tw/smallko/sdn

ENERGY

"UNDERWATER SENS

A Project Work Submitted according
to the requirements for the Degree
Of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Dhanjit Singh Boro (Roll No. Gau-C-11/146)

Manwendra Tiwari (Roll No. Gau-C-11/129)

Rituraj Borkakoty (Roll No. Gau-C-11/121)

Sharmistha Kundu (Roll No. Gau-C-11/122)

Under the supervision of

Mr. Kongkon Kalita

&

Mr. Bikramjit Choudhury

Central Institute Of Technology
Kokrajhar ... Bodoland
ESTD : 2006

DEPARTMENT OF INFORMATION TECHNOLOGY

केन्द्रीय प्रौद्योगिकी संस्थान कोकराझार

CENTRAL INSTITUTE OF TECHNOLOGY KOKRAJHAR

Centrally Funded Institute under Ministry of HRD, Govt. of India

RRITORIAL AREAS DISTRICTS :: KOKRAJHAR :: ASSAM :: 78

w.cit.kokraihar. In. www.cit.ac.in