*I.J. Information Technology and Computer Science,* **2016, 9, 49-61**
Published Online September 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2016.09.07

# Design of Fuzzy-Based Traffic Provisioning in Software Defined Network

**Anju Bhandari and V.P. Singh**
Department of computer science and engineering, Thapar University, Patiala, Punjab
E-mail: er.anjugandhi@gmail.com

***Abstract***—This Software defined networks helps to realize extraordinary services that can be easily embedded in network operations of switch. It provokes the decomposition of the control and data planes. The control plane is more extensible, as it is unproblematic to change or introduce any new functionality into the network. It is studied that any new integration can be easily added up with a very low line of code (LOC). The work proposes a fuzzy based approach for traffic provisioning in SDN. Fuzzy Logic Control System (FLCS) is a controller comprising of two fuzzy systems- Label Switched Path setup System (LsS) and Traffic Splitting System (TSS). The computation of dynamic status of Load and Delay is utilized by LsS to arrange the paths in preference order. The attained Link Capacity and Utilization Rate are employing by TSS for maintaining congestion free path. Created three different topologies and performed *ping* reachability test and executed *iperf* testing tool for performance analysis on Mininet framework.The impact of this is to facilitate better decision making for splitting the traffic for different capable paths. Simulation setup is deployed using OpenFlow Switches and Controllers to study their performance. The packet delivery ratio remained above 98%, showing rare chances of congestion and delay was below than 2.6 seconds with TTL in range of 60-80 milliseconds.

***Index Terms***—OpenFlow, Software Defined Networking, Mininet, Fuzzy Controller, POX.

## I. INTRODUCTION

Many , independent research scholars and technicians have recommended that the promotion of packet based technologies is now an absolute necessary only then the reduction of Operational Expenses (OpEx) and Capital Expenses (CapEx) in their networks can be achieved for the time being. Therefore, it is overbearing on part of many research institutes to take up this work in this area implement reliable solution like Multi Protocol Label Switching (MPLS) based solutions as SDN.

In SDN, the network control is partitioned from the forwarding mechanism and is directly programmable. SDN is believed to be a new networking technology that adds potential benefits for the Next Generation Internet. The controller is a logically centralized having a broad view of network and controls multiple packet-forwarding devices (switches) that can be configured via an OpenFlow interface. An OpenFlow model consists of two key components *OpenFlow Switch and OpenFlow Controller*. These components communicate via the OpenFlow Protocol. The benefits include the simplified MPLS control plane architecture allows to introduce any new protocol to work. It reduces protocol load in router CPUs (LDP, IS-IS, MP-BGPLMP, RSVP-TE, I-BGP and OSPF). As, control plane is more extensible so it is new functionality can be effortlessly introduced in the network. It also enhances MPLS base recovery Fast Reroute Recovery (FRR) and Auto-bandwidth [1-3].

SDN provides dynamic network architecture that facilitates transformation from traditional network backbone into rich service delivery platforms. Software defined Networking is having following features: - (1) Highly Dynamic (2) Easily Manageable (3) Cost Effective (4) Adaptable and suitable for the high volume traffic. Researchers are working for developing more optimize and efficient SDN controllers [4], switches in different programming languages (pyretic, python 2.7, Perl).

The paper is prepared as follows. The section II, presented preliminaries on the subject of this work, i.e. Software Defined Networking, Multi Protocol Label Switching (MPLS), OpenFlow, Mininet and deployed approach FLCS. The section III, described the proposed fuzzy approach and its implementation. Section IV presented the simulation settings. Analysis is described in section V and finally, conclusions are discussed in section VI.

## II. PRELIMINARIES

An SDN instance consists of three major parts: application, control plane, and data plane. The application label indicates a part that exploits the decoupled control and data plane to achieve specific goals, such as a security mechanism or a network measurement solution.

Applications communicate with a controller at the control plane via the northbound interface of the control plane. The control plane manipulates forwarding devices through a controller to achieve the specific goal of the target application. The controller uses the southbound interface of the SDN-enabled switch to connect to the data plane as shown in Fig.1. The data plane supports a shared OpenFlow protocol with the controller and

handles the actual packets based on the configurations that are manipulated by the controller [5,6]. With the SDN, operators have the capability to make an offline optimization tool to online. The controller can access the results of tool and use OpenFlow to directly efficiently manipulate the forwarding tables of all LSRs [1-3].

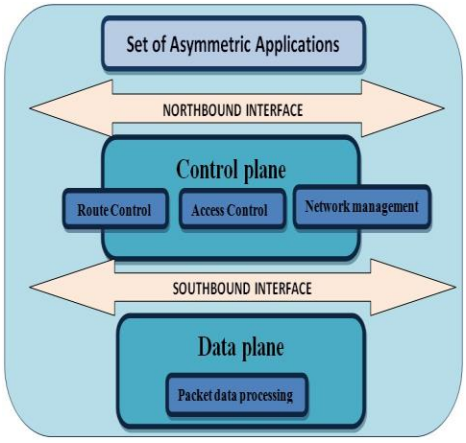



Fig.1. Functionality of data plane and control plane

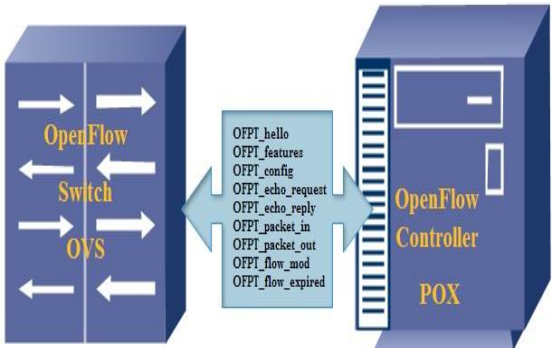



Fig.2. Communication between OpenFlow Switch and OpenFlow Controller

*A. Open Programming Extended Network (OpenFlow) Architecture*

OpenFlow provides an open programmable, virtualized platform for the switches and routers. An OpenFlow simulation model has two key components: *OpenFlow switch and OpenFlow controller*. These components channelized via the OpenFlow protocol [14, 15]. OpenFlow Switches consists of Flow Tables [FT1, FT2 ... FTn] and Group Table [GT] and perform packet lookups and forwarding [23]. Each *OpenFlow switch* has a chain of flow tables, and each table keeps a set of *flow entries*. A *flow* is defined as the set of packets that match a pair of source and destination MAC addresses [7, 8]. The forwarding/routing rules are defined by *flow entry*. It is composed of a bit pattern indicating the flow attributes a list of actions, and a collection of counters. Each flow entry states "execute the set of measures on all packets in the flow". As illustrated in algorithm 1, actions like forward the packet "a" from port A, drop the packet "b" from port B [23] and etc.

OpenFlow acts as a Switch-API that dynamically updates flow table state and therefore LSP state. It results in the emerging of multiple LSP in parallel that reduces network blend [1, 2, 11, 17, 23]. When a packet arrives at a switch, the switch searches for matched flow entries in the flow tables and executes the corresponding lists of actions. If no match is discovered for the packet, the packet is queued, and an inquiry event propels to the OpenFlow controller. The controller responds with a new flow entry for handling that queued packet. Subsequent packets in the same flow will be handled by the switch without contacting the controller, and will be forwarded at the switch's full line rate. The set of communication messages between switch and controller is shown in Fig 2.

An OpenFlow switch has one or more forwarding tables that are controlled by a centralized controller, thus realizing programmability in the control plane. Forwarding tables are used to control packets (forwarding or dropping). Therefore, according to the controller strategy that handles the forwarding tables, an Open Flow switch can take action as a router, switch, NAT, firewall, or exhibit similar functions that depend on packet-handling rules. The functionality of controller is studied using algorithm below.

***Algorithm 1: Functioning of OpenFlow Controller***

***begin***

*1. import python header files which includes topology.*

*/* openflow_controller.cc openflow_controller.h*
*openflow_session.cc openflow_session.h*
*openflow_interface.cc openflow_interface.h*
*openflow_messager.cc openflow_messager.h*
*openflow_switch_session.cc openflow_switch_session.h*
*openflowlib */*

*2. build network*
*3. test connectivity of all switch and host*
*/*successful connectivity of all links are tested*/*
*4. process*
*1. (modify_field) Check and update flow table entry.*
*2. (Drop) Drop packet if miss found.*
*3. (add_field) Add a new flow entry determining how to forward similar kind of*

*packet in future.*

***end***

### *B. MPLS-TE module*

The prevalence of smartphones and streamed audio and video services causes explosive increase in traffic which can be excellently managed by MPLS networks. MPLS is deployed by carrier due to its exceptional features like MPLS data plane possess basic mechanisms of pushing on, swapping and popping off MPLS labels in label switched path (LSP) [18]. These mechanisms are done by control driven protocols. In this paper, we are proposing fuzzy based control driven protocol providing traffic provisioning in the network. The work is accomplished using extensible OpenFlow & SDN. The significant advantage of doing so is this that control driven using fuzzy can be easily acceptable by control plane. This is so that MPLS would be able to provide more services than today. The aim is to universally optimize the resources, making them more and more vigorous by program networking applications [18]. New capabilities are 0=-not tied with the layers of protocols. The best part is this that in OpenFlow, there is no need to make changes in MPLS control plane algorithms. The results of using this novel approach have been a great significance that fits its well and makes control architecture ideal for next generation networks. The approach is proposed for MPLS *flow based usage Model* in which LSPs are set-up in the network by edge routers (Ingress-Egress) consists of attributes like:- (1) Packets are classified into forward equivalence class (FEC). (2) Resource Reservation Protocol (RSVP) forwards the paths. Following features are responsible for successful performance:-

1) Supplementary deterministic behavior in IP networks. (2) Superior efficiency in the utilization of network resources. (3) Ease of management, operations, utility of tunnels. (4) Most important features are Auto route, Auto bandwidth, Tunnel priorities, Differentiating services aware traffic engineering (DS-TE), Load Balancing, Explicit Routes and Re-Optimization times [1, 2, 11, 17, 18].

### *C. The implication of OpenFlow*

OpenFlow base SDN design supports map abstraction that allows the network to become programmable and manageable, scalable and agile. It validates that the proposed intelligent approach is able to allocate resources dynamically. The SDN concept reduces routing trouble by abolishing the need of many protocols (LDP, IS-IS, MP-BGPLMP, RSVP-TE, I-BGP and OSPF) [1, 2, 11, 12, 18]. So, hence lines of code (LOC) are also optimized. This is the big motivation for implementation in real network scenario, so that researchers can easily deploy new protocols [13]. The comprehensive study of SDN has been performed. Here, the practical details are listed below in Table 1.

Table 1. Practical Details

| Limitations of other technologies | Need for new architecture | Key Motivation Factors |
|---|---|---|
| • Complexity<br>• Incapability to scale<br>• Vendor dependence<br>• Inconsistent policy | • Changing utilization rates<br>• Consumerization of IT<br>• Argument of cloud application<br>• Link Quality /Data handling<br>• Numerical unstable metric computations | • Control plane- Data plane programmability<br>• Platform independency<br>• User-driven control<br>• Deployment is easy |

## III. Deploying Fuzzy Logic Controlled System (FLCS) in Control Plane

We leverage our previous work [19] on a Mininet emulation system. In this research work, we have exploited the behavior of OpenFlow-based SDN controller to deal with potential performance issues like (1) State Consistency (2) Scalability (3) Flexibility (4) Security (5) Availability. The techniques are not only improving the simulation performance, but are also valuable for designing scalable SDN controllers [23].

We have proposed fuzzy approach composed of two sets of fuzzy rule matrix or fuzzy mixed metric (FMM) – LsS and TSS. In LsS, rule base of 25 rules are processed on the dynamic input values of traffic statistics (*load, delay*). It generates the matrix after the execution of complete fuzzy inference system (FIS). The computed values from the lowest to highest fuzzy value of LSPs, generates the list of LSPs in preference order. It fulfills an appropriate QoS requirement of network. In TSS, rule base of 35 rules are processed on the updated value of frequency of using particular LSP

i.e. *utilization rate* and the *link capacity* of LSP. It performs traffic splitting among paths in order to realize TE in network. A prototype of design of fuzzy traffic provisioning approach for MPLS-TE shown in Fig. 3. The pseudo code of proposed fuzzy based methodology is described in next subsection A. The main algorithm and sub-algorithms (LsS and TSS) are followed by the description of data structure and variables used. The approach is strongly justifying the alarming need of load balancing in network [16, 17, 20]. Our approach is simple. So, it can be easily adaptable with the accessible design. The proposed effort assist researchers to set an appropriate and meaningful direction for future SDN

research [21, 22, 24]. The technique Ternary Content Addressable Memory (TCAM), in which each forwarding rule is added to TCAM memory as an OpenFlow entry is well suited and applied in this designed approach. As, the computed fuzzy predictions get saved in TCAM as shown in Fig. 3. The entry has the label value as the l2 destination and wildcard for other fields. Also the corresponding action of the rule is to send the packet to a specified port on the switch [25].

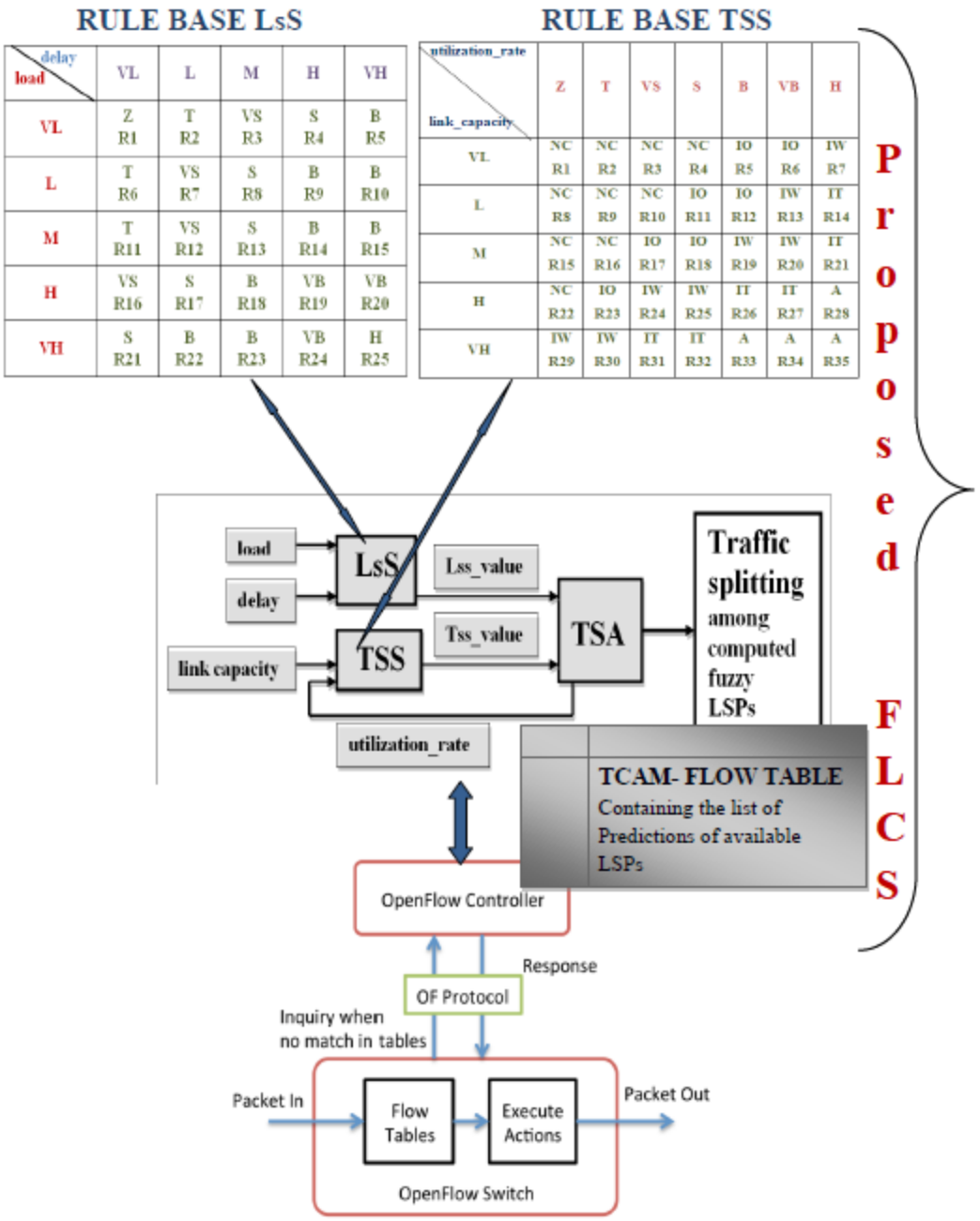



Fig.3. Prototype of proposed fuzzy based open flow controller

*A. Pseudo codes of Proposed fuzzy based routing: Fuzzy Logic Controlled System (FLCS):-*

**Description of Data Structures and Variables used in building algorithm are as follows:-**

1. *Link Matrix* describing connectivity between two nodes. *Link[i] [j]* is equal to 1 if connectivity exists between nodes *i* and *j* otherwise 0.
2. *Link capacity matrix LC[i][j]* giving bandwidth value of each link.
3. *Utilization rate matrix UR[i][j]* specifying utilization rate of each link.
4. *Delay matrix D[i][j]* specifying delay of each link.
5. *Load matrix L[i][j]* specifying number of packets waiting to be sent on each link.
6. *LsS Value Matrix LsS[i][j]* storing LsS value of

each link used to identify paths .

7. *TSS Value Matrix TSS[i][j]* storing TSS value of each link used to compute number of paths required for operation.
8. Variable *ingress* to store the identifier of source node.
9. Variable *egress* to store the identifier of destination node.
10. Variable *n* number of nodes in the network.
11. Variables *Lak, Lbk, Lck* for storing the limits of linguistic terms of linguistic input variable Load of LsS.
12. Variables *Dak, Dbk, Dck* for storing the limits of linguistic terms of linguistic input variable Delay of LsS.
13. Variables *LsSak, LsSbk, LsSck* for storing the limits of linguistic terms of linguistic output variable LsS_value of LsS.
14. Variables *URak, URbk, URck* for storing the limits of linguistic terms of linguistic input variable Utilization Rate of TSS.
15. Variables *LCak, LCbk, LCck* for storing the limits of linguistic terms of linguistic input variable Link Capacity of TSS.
16. *Freq[i][j]* storing the frequency of a particular link used.
17. Final decision *LSS FDlss[i][j]* and Final decision TSS *FDtss[i][j]* store the final defuzzified values of LsS_value with respect to each link and TSS_value with respect to each link respectively.
18. *Nlsps* gives the number of required LSPs.
19. *Tlsps* gives the total number of LSPs.
20. *Slsp[i]* give the selected LSP for operation according to optimal LsS_value.
21. $\mu_L$, $\mu_D$, $\mu_{UR}$, $\mu_{LC}$, $\mu_{LsS}$, $\mu_{TSS}$, $\mu_L$ are membership functions of load, delay, utilization rate, link capacity, LsS_value and TSS_value respectively.
22. $\mu_{Lk}$, $\mu_{Dh}$, $\mu_{URk}$, $\mu_{LCk}$ $\mu_{LsSk}$, $\mu_{TSSk}$ are membership functions of linguistic terms of load, delay, utilization rate, link capacity, LsS and TSS respectively.
23. RLsS[i]j[], RTSS[i][j] stores the rules of LsS and TSS respectively.
24. Variables i, j, k, m, n used for the loop control

Main algorithm

1. Read n
2. Repeat steps 3 to 10 for i, j =1 to 8
3. Read Link[i][j]
4. Read L[i][j], D[i][j] , UR[i][j]& LC[i][j]
5. LsS_value = Call LsS(D,L) /*call LsS() to compute LsS_value to identify links for the selection of LSPs.*/
6. Freq[i][j] =0.0 /* Initialization*/
7. Nlsps = Call TSS(UR, LC) /*call TSS() to compute TSS_value to obtain the number of LSPs required for operation. */
8. Repeat steps 9 & 10
   while(L[1][j]!=0) /*Transmission will be continue till load at node 1(ingress) is non zero*/
   for k= 1 to Nlsps /*Traffic splitting takes place among the Nlsps */
9. Slsp[k]=min(LSS[i][j] && Freq[i][j]) /*list of optimal LSPs(LSP having minimum LsS_value) in preference order is used for congestion free operation*/
10. Freq[i][j] = Freq[i][j] +1 /* respective frequency of LSP is updated */

**Sub- Algorithm LsS(L,D)**

1. Read limits of linguistic terms of input load and delay as

   $_{k=1}{}^{5}\{(Lak, Lbk, Lck), (Dak, Dbk, Dck)\}$.

2. Read limits of linguistic terms of output LsS_value as

   $_{k=1}{}^{7}\{(LsSak, LsSbk, LsSck)\}$.

3. Display the Degree of Strength (DS) for entered values of load and delay
   Repeat steps 3.1 to 3.4 for i, j =1 to 8 and steps 3.1 & 3.2 for p=1 to 5

   3.1 DS of L[i][j] and D[i][j] are obtained by identifying the area
   in which input value lies, to obtain linguistic terms on applying TMF

   $$\mu_L[p]=_{k=1}{}^{5}\{\mu_{Lk}\}$$

   $$\mu_D[p]=_{k=1}{}^{5}\{\mu_{Dk}\}$$

   3.2 Composition and aggregation

   $$RLsS[i][j] = \min(\mu_L[p], \mu_D[p])$$

   $$\mu_{LsS}[p]= {}_{i,j=1}{}^{5}\sum RLsS[i][j]$$

   3.3 Defuzzification using centroid method

   for h= 1 to 7

   $$LsS[i][j]= {}^{7}_{k=1}(\sum\mu_{LsS}[p]) * LsSbk/\mu_{LsS}[h])$$

   3.4 Apply TMF on LsS[i][j] for m=1 to 7, n=1 to 2, obtain
   Final decision as FDlss[m][n]
   Return(Slsps[i]=max(FDlss[m][n])) /*final decision is obtained
   by applying TMF on computed fuzzy value is returned to
   main algorithm for the selection of optimal path*/

**Sub-Algorithm TSS (UR, LC)**

1. Read limits of linguistic terms of input Utilization

rate and Link capacity as

$_{k=1}^{7}$\{(URak, URbk, URck)\}, $_{h=1}^{5}$ \{(LCah, LCbh, LCch)\}.

2. Read limits of linguistic terms of output TSS_value as

$_{k=1}^{5}$\{(TSSak, TSSbk, TSSck)\}.

3. Display the DS for entered values of utilization rate and link capacity
Repeat steps 3.1 to 3.4 for i,j =1 to 8,
Steps 3.1 & 3.2 for p=1 to 7 and steps 3.1 & 3.3 for l=1 to 5

3.1 DS of UR[i][j] and LC[i][j] are obtained by identifying the area in which input value lies,
to obtain linguistic terms on applying TMF

$$\mu_{UR}[p] =_{k=1}^{7}\{\mu_{URk}\}$$

$$\mu_{LC}[l] =_{h=1}^{5}\{\mu_{LCh}\}$$

3.2 Composition and aggregation

$$RTSS[i][j] = \min(\mu_{UR}[p], \mu_{LC}[l])$$

$$\mu_{TSS}[l] = {}_{i=1}^{5},{}_{j=1}^{7}\sum RTSS[i][j]$$

3.3 Defuzzification using centroid method

$$TSS[i][j] = {}_{k=1}^{7}(\sum\mu_{TSS}[l] * TSSbk/\mu_{TSS}[l])$$

3.4 Apply TMF on TSS[i][j] for m=1 to 5, n=1 to 2, obtain

Final decision as FDtss[m][n]

Return(Nlsps= max (FDtss[m][n]))

## IV. Experimental Setup

We have created network in Mininet environment. It is an open source network emulator that creates a network consists of Switches, Controller and Hosts. It works on Linux kernel. It easily creates virtual networks for swift prototyping of SDN designs using OpenFlow. This is the command line interface (CLI) which is highly flexible, scalable and realistic deployable. The basic installation procedure is discussed in [7,8, 11].

We proposed the use of MPLS data plane with an open control plane. An introducing fuzzy logic in control plane is demonstrated with this platform. We have assembled a network prototype to verify architecture constructs and authenticate the simplicity and extensibility of proposed approach. We have implemented the features of traffic engineering (TE) and Quality of Service (QoS) to optimize MPLS-TE. LOC is also shorter than using a traditional approach [1,2, 5,6, 12, 17]. In this paper, we have proposed the prototype of fuzzy based traffic management approach for OpenFlow controller. The design of Fuzzy Logic based System is significantly simple and providing elasticity to the system.

### A. Mininet Framework

It is a network simulation tool that runs a collection of hosts, switches, routers and links on a single Linux kernel. Mininet host runs standard Linux Network software and its switches support OpenFlow for versatile custom routing [21]. It allows emulating arbitrary OpenFlow network on machine. Mininet also enables *ping*, *iperf* softwares to study and generate traffic. It measures performance parameters like throughput, delay, packet drop.

### B. MPLS-TE

We have implemented this prototype of the algorithm in the POX controller and measure its performance in three different topologies. By adding changes in load balancing module and traffic type aware routing module [20, 21]. The traffic type aware routing module ensures that only those flows of a certain traffic type (voice, video, web) are permissible for routing activities. Table 2 describes the configurations of simulation setup. We used POX controller and implemented proposed algorithm. In this paper, a performance analysis is done by testing network connectivity between nodes. TCP and UDP throughput bandwidth monitoring is evaluated. Network connectivity is tested by ping, which correspond by ICMP echo request message and wait for reply indicating IP connectivity between defined nodes [9, 10, 18].

The simulation work has been done using the available Mininet commands [9, 10, 18, 20]. The snapshots of performed simulation tests are given in Appendix A. The details of snapshots are listed in below Table 3.

Table 2. Simulation Settings

| Parameter | Settings |
|---|---|
| Mininet Environment | 2.1.0 |
| OpenFlow software switch (ofdatapath, ofprotocol) | 1.3.0 |
| Open vSwitch | 2.3.90 (OF versions 0x1:0x1) |
| Reference controller (OVS-controller) | 2.0.1 |
| System | Ubuntu 14.04 (trusty) 64 bit |
| Controller | POX 0.2.0 |
| Language support | Python 2.7 |
| Processor | Intel® Core™2 Duo CPU T6670 @ 2.20GHz ×2 |
| Number of controllers | 1 |
| Number of Switches | 1 |
| Number of edges | 5,4,3(respective to topology) |
| Testing Tool | Ping, Iperf |
| Testing time | 30 sec |

The simulation is performed on three topologies as shown in Fig. 4. The topology with 4 host (host h1,host h2, host h3 and host h4) in Fig. 4 (a), 1 switch, 1 controller second topology consists of with 3 host (host h1,host h2 and host h3 ) in Fig. 4(b), 1 switch, 1 controller and third topology consists of 2 host (host h1 and host h2), 1 switch, 1 controller in Fig. 4(c).

The following Steps are performed on these network topologies:- (1) Generated hosts each one with dissimilar IP address as defined Algorithm 1. (2) Establish Connection of hosts (h1, h2, h3, h4) with Switch S1 with the Ethernet cable (eth0, eth1, eth2). (3) Set up of MAC address of host (h1, h2) to its IP address [18]. (4) Configures the s1 for the connection with the remote controller c0.

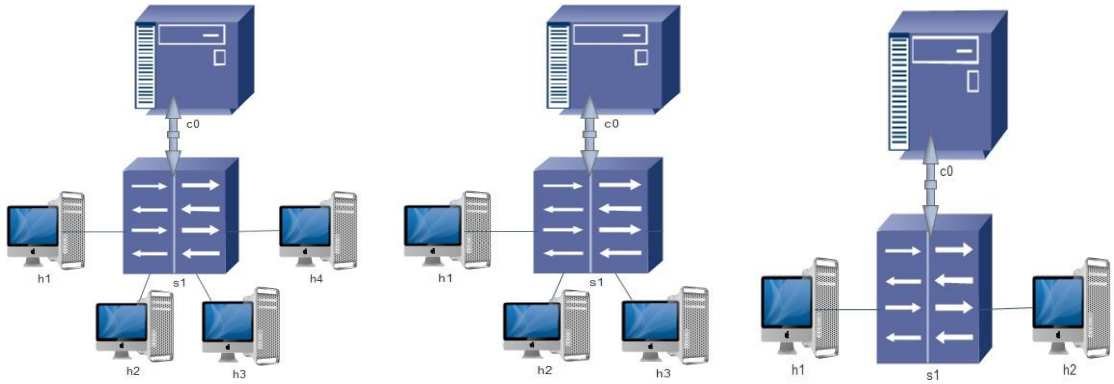



Fig.4. (a)Topology with 4 hosts created (b) Topology with 3 hosts created (c) Topology with 2 hosts created

Table 3. Snapshots of Linux machine executing Mininet Console

| Snapshot | Caption | Illustration |
|---|---|---|
| Fig.5. | 1controller, 1switch, 2hosts, IP routing table, ping, iperf | Creation of topology consists of one controller, one switch and two hosts. The IP routing table is also shown here. *Ping* and *iperf* are executed successfully. |
| Fig.6. | (a) Controller Remote Port-6633<br>(b) Internet connections OVS Controller | The port information of remote controller is shown. The details of the Internet connections between TCP servers and localhost are depicted. |
| Fig.7. | (a) Specifications of OVSSwitch, OVSController and creation of network (h1, h2;s1;c1)<br>(b) Design and connectivity of c0, s1 and h1,h2 | Specifications of OVSSwitch, OVSController are shown. Creation of network (h1, h2;s1;c1) and Specifications of other Components are shown. Design and connectivity of controller (c0), switch(s1) and hosts (h1,h2). |
| Fig.8. | (a) Topology (1 switches, three hosts, one controller) (b) POX controller | Processing of POX CONTROLLER for first topology to study the results. |
| Fig.9. | Execution of Xterms s1,c0,h1 and h2 | Execution of Xterms s, c0, h1 and h2 presenting the interaction of active nodes. |
| Fig.10. | . Internet performance test by *ping* giving ping statistics (RTT, LOAD, DELAY) | Clear execution of ping testing tool is visible. ping statistics giving values of Round Trip Time (max, min, avg, mdev ).One echo request packet is transmitted and successfully received by node h2. It has been verified that host can ping each other. |
| Fig.11. | Iperf testing TCP bandwidth with set of values of load and delay | Iperf testing TCP bandwidth with set of values of load and delay are obtained. |
| Fig.12. | Performance of *iperf* and *installation* of POX Controller | Testing of TCP bandwidth between h1-h2 is done by Performance testing tool *iperf* and *installation* of POX Controller is also shown. |
| Fig.13. | Internet performance test by ping | Ping statistics of h1 and h2 are depicted. |
| Fig.14 | Creating topology of 4 switches, 4 hosts and 1 controller | Creating topology of 4 switches, 4 hosts and 1 controller is depicted. |
| Fig.15. | (a) TCP traffic analysis (b) UDP traffic analysis | The communication of *iperf* h1 and *iperf* h2 for TCP Connection at default port 5001 with TCP Window Size 85.6 Kbyte is depicted. The communications of *iperf* h1 and *iperf* h2 for UDP Connection at default port 5001 with UDP Window Size 208 Kbyte along with the performance analysis results are depicted. |

## V. ANALYSIS

Empirical results show that, when the capacity of the high volume traffic lines is above 10 gigabits per second (Gb/s). The previous algorithms metric calculations become asymptote in nature and rate of change is become nearly constant while measuring some metrics. This situation becomes more unclear when we try to measure multiple variables influencing each other. Especially, when the work in mix bag of old and new switches and routers are also upgraded for scalability needs. The computation of the routing paths, priorities and delay becomes numerically unstable. The main reason for this the fuzzy nature of the metric values and overlapping

boundaries of the metrics. Our research work overcomes these numerical instability problems by separating the data plane and control plane. This way a specialization of task is introduced in the network and processing of control functions are separated from the data operations. Our work is based on the principles of open switch architecture. The main benefit is maximization of agility and choice with a full portfolio of from 1/10gbe and 10/40gbe. This implementation helps to act as a flexible building block for multiple data centres. The approach also forces us to think beyond the metrics like link speed (b/s), such as packets per second (p/s), connections per second (c/s), transactions per second (t/s), and maximum concurrent connections (mcc). As, the behaviour changes in case of stateful devices in high volume bandwidth switch networks. Our results show, even that higher rate the packet delivery ratio remain close to 99%, show that there is rare cases of congestion even if concurrent connections and packet volume increases exponentially and the delay remain below 2.6 milliseconds (end-to end). The values of the TTL also remained in good range(60-80 milliseconds) for all the experiments with proposed algorithm.

## VI. Conclusion and Future Recommendations

SDN is becoming popular due to its interesting features, design and organization of networks. However, there are still important challenges to be solved before realizing successful SDN. In this paper, existing SDN related technologies has been introduced and discussed several future directions to realize data plane and control plane programmability. In fact, current southbound Application Programming Interface (API) is not supple and mostly translated as Open Flow protocol. In traditional network architecture, the control plane and the data plane cooperate within a device via internal protocols. But, in SDN, the control plane and the data plane are separated.

Along with this the control logic is moved to an external controller. This external controller is answerable for monitoring and managing all of the states in the network. The controller release signals to the data plane using the *OpenFlow protocol*. It defines the communication between the controller and the data planes of all the forwarding elements. The *controller* sets rules about the data-forwarding behaviors of each forwarding device through the OpenFlow protocol, including rules including drop, forward, modify, or enqueue of packets. The proposed algorithm combined with the decision making based on fuzzy rules has exposed superior results. LOC also get reduced as well as the smooth integration of the various external modules and libraries have been done. Therefore, we believe that understanding of the design of SDN, functionality of OpenFlow and applying Fuzzy Logic maximizes the potential benefits of SDN. A *ping performance test* connected with assumed nodes has been executed. Echo request packets are channelized which have been fortunately received and ping stats have also been obtained. The verification that host can ping each other is depicted using Xterms. Internet Performance Test is done by *Iperf*, which is network testing tool and can generate TCP and UDP data streams and estimate the Max TCP Bandwidth and illustrates UDP specifications (Bandwidth, jitter, datagram loss, and speed). Thus to the highest degree, it clarifies the implementation of the algorithms and decision making system.

## Appendix A Snapshots of Simulation Results on Mininet Console

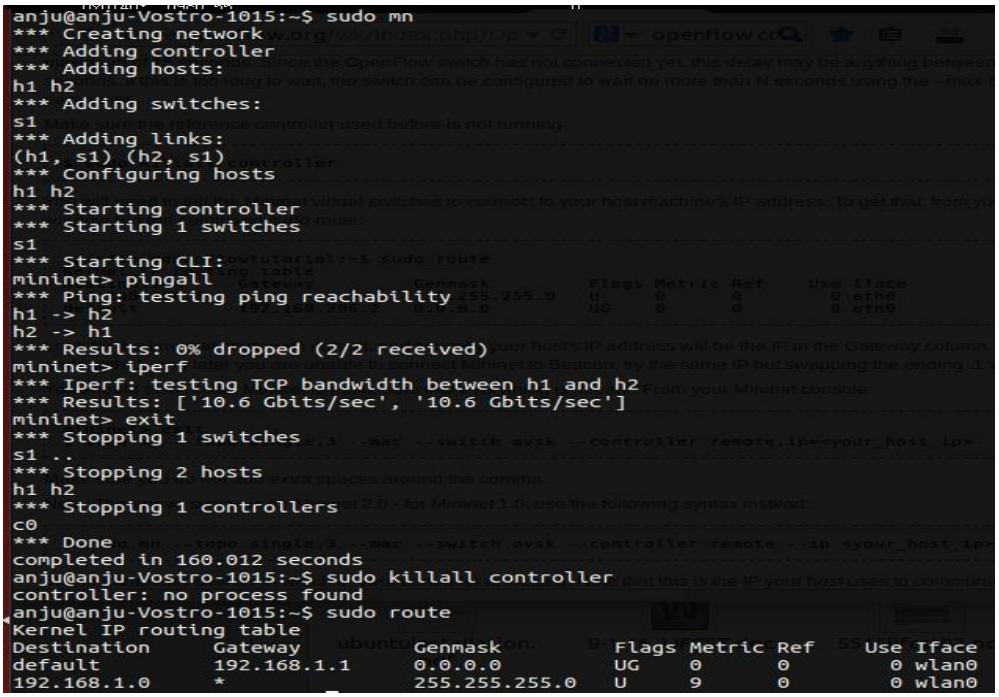



Fig.5. 1controller, 1switch, 2hosts, IP routing table, ping, iperf

```
Active Internet connections (servers and established)
tcp        0      0 *:6633                  *:*                     LISTEN
1099/ovs-controller
tcp        0      0 localhost:44107         localhost:6633          TIME_WAIT
```

(a)

```
  --listenport=LISTENPORT
                        base port for passive switch listening
  --nolistenport        don't use passive listening port
  --pre=PRE             CLI script to run before tests
  --post=POST           CLI script to run after tests
  --pin                 pin hosts to CPU cores (requires --host cfs or --host
                        rt)
  --version
anju@anju-Vostro-1015:~$ sudo mn -controller=remote -ip=127.0.0.1 -port=6633
Usage: mn [options]
(type mn -h for details)
```

(b)

Fig.6. (a) Internet connections OVS Controller (b) Controller Remote Port-6633

```
ovs-system Link encap:Ethernet  HWaddr 1e:dd:7f:f0:8a:9d
           BROADCAST MULTICAST  MTU:1500  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)

s1         Link encap:Ethernet  HWaddr 16:3b:ed:94:91:4e
           inet6 addr: fe80::70d8:87ff:fe43:9a2e/64 Scope:Link
           UP BROADCAST RUNNING  MTU:1500  Metric:1
           RX packets:13 errors:0 dropped:0 overruns:0 frame:0
           TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
           RX bytes:1038 (1.0 KB)  TX bytes:648 (648.0 B)

s1-eth1    Link encap:Ethernet  HWaddr 6a:41:53:13:05:11
           inet6 addr: fe80::6841:53ff:fe13:511/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:8 errors:0 dropped:0 overruns:0 frame:0
           TX packets:206 errors:0 dropped:2 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:648 (648.0 B)  TX bytes:38208 (38.2 KB)

s1-eth2    Link encap:Ethernet  HWaddr 52:c7:ba:3c:53:ce
           inet6 addr: fe80::50c7:baff:fe3c:53ce/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:8 errors:0 dropped:0 overruns:0 frame:0
           TX packets:203 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:648 (648.0 B)  TX bytes:37425 (37.4 KB)

wlan0      Link encap:Ethernet  HWaddr 20:7c:8f:33:10:8c
           inet addr:192.168.1.108  Bcast:192.168.1.255  Mask:255.255.255.0
           inet6 addr: fe80::227c:8fff:fe33:108c/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:74863 errors:0 dropped:0 overruns:0 frame:0
           TX packets:74923 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:87906985 (87.9 MB)  TX bytes:9279284 (9.2 MB)
```

(a)

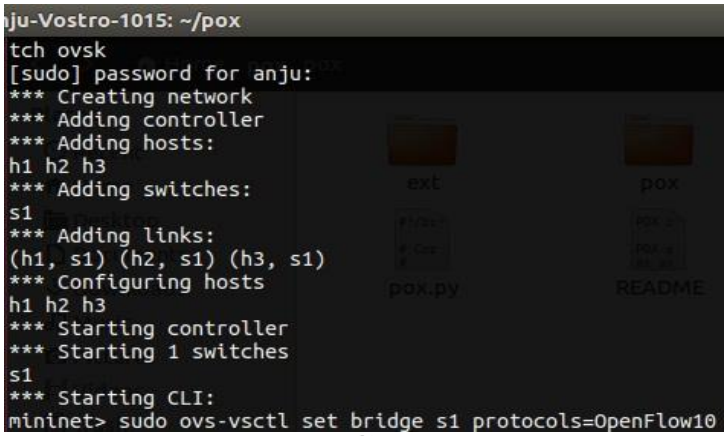
```
mininet> xterm h2

mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=6815>
<Host h2: h2-eth0:10.0.0.2 pid=6816>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=6821>
<OVSController c0: 127.0.0.1:6633 pid=6807>
mininet>
```

(b)

Fig.7. (a) Specifications of OVSSwitch, OVSController and creation of network (h1, h2;s1; c1)
(b) Design and connectivity of c0, s1 and h1,h2

```
anju@anju-Vostro-1015:~$ sudo mn --topo single,3 --mac --controller remote --swi
```

(a)




(b)

Fig.8. (a) Topology (one switch, three hosts, one controller) (b) POX controller

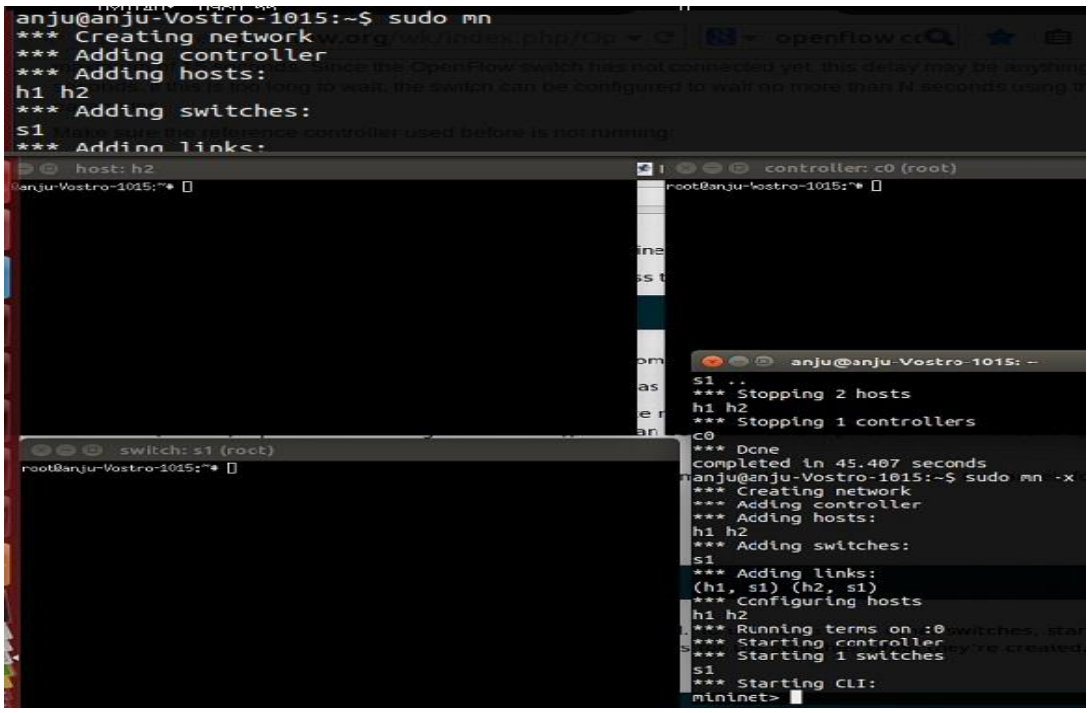



Fig.9. Execution of Xterm s1,c0, h1 and h2

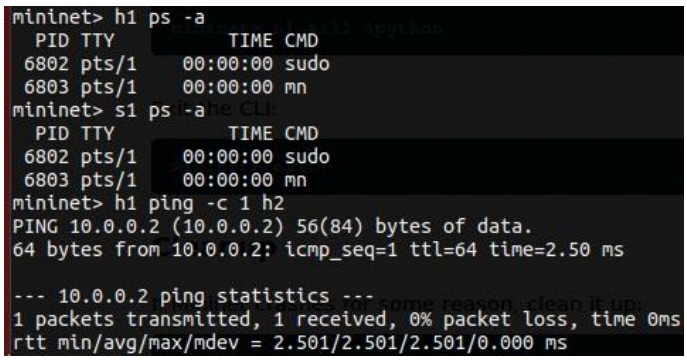



Fig.10. Internet performance test by *ping* giving ping statistics (RTT, LOAD, DELAY)

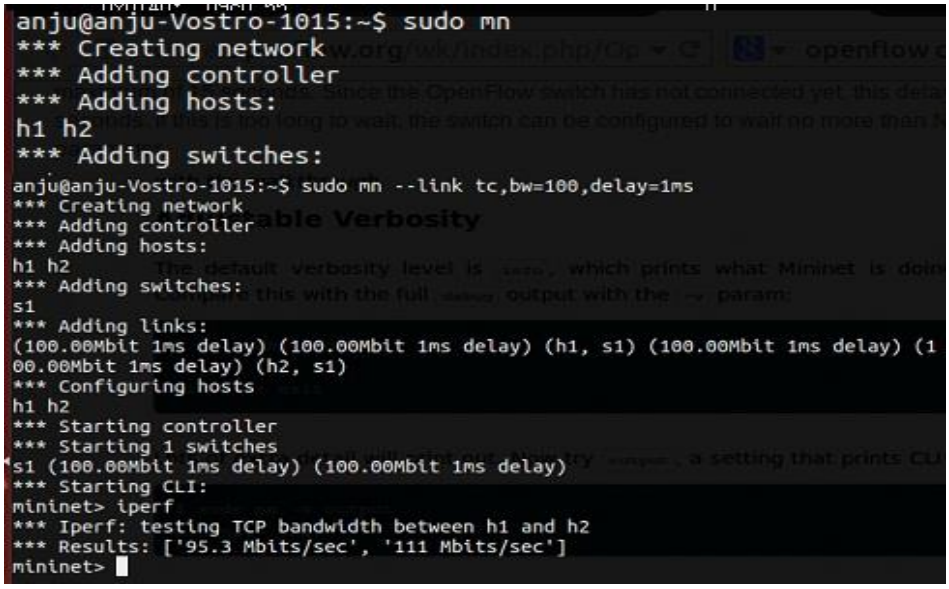



Fig.11. Iperf testing TCP bandwidth with set of values of load and delay

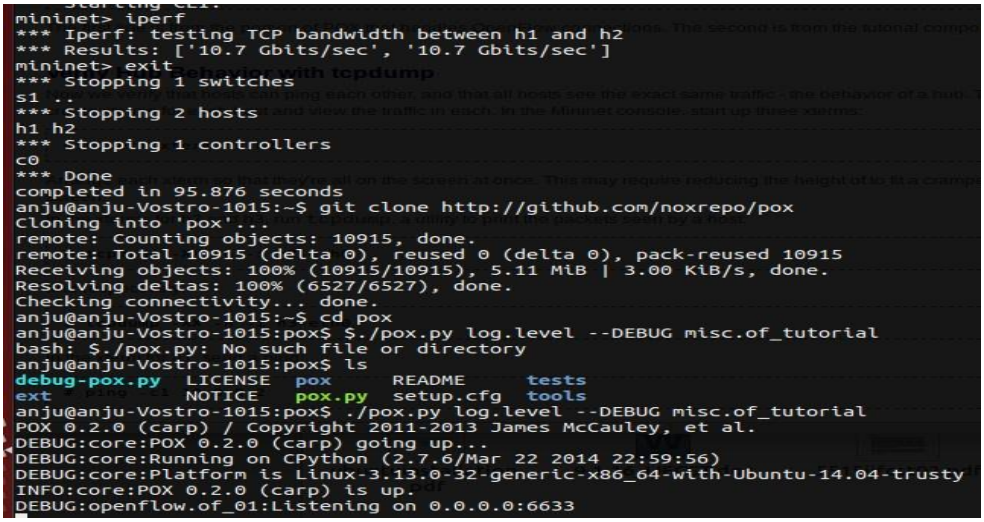



Fig.12. Performance of *iperf* and *installation* of POX Controller

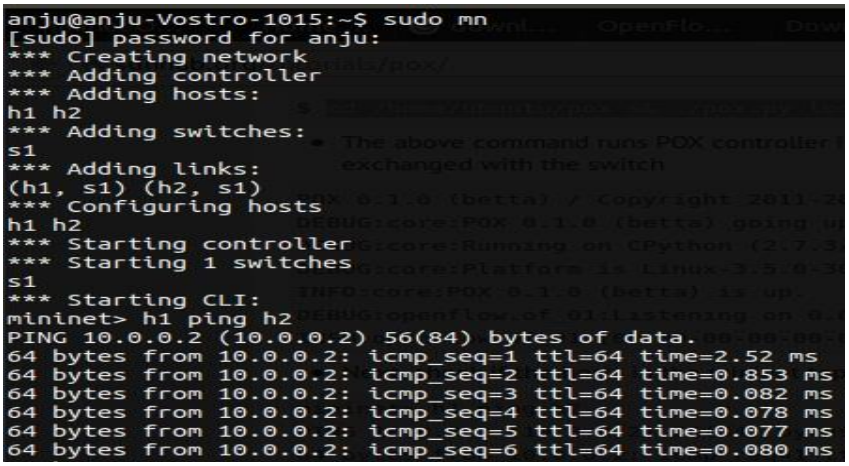



Fig.13. Internet performance test by ping

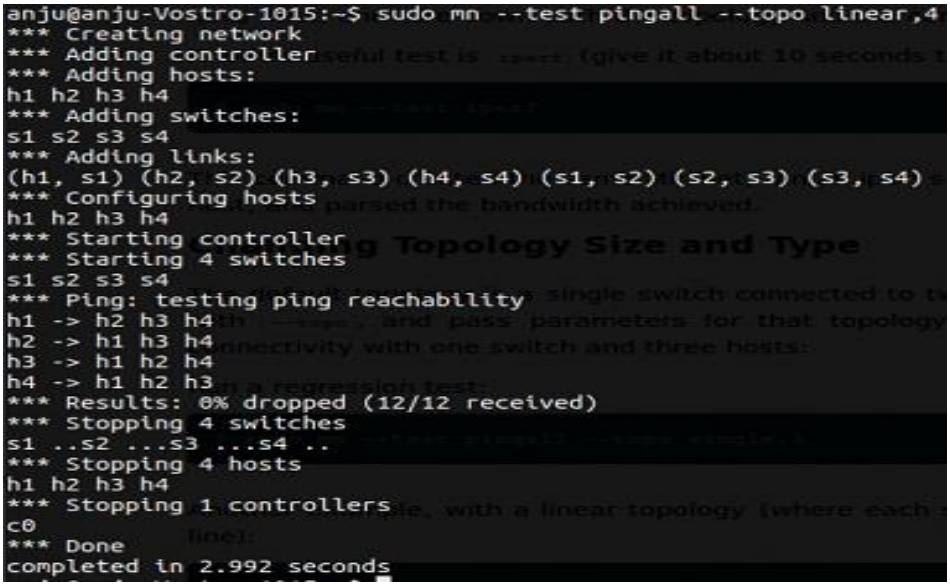
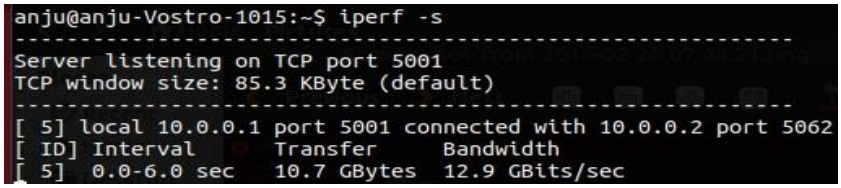
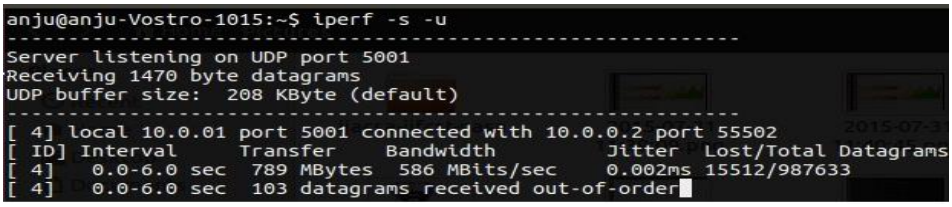



Fig.14. Creating topology of 4 switches, 4 hosts and 1 controller, ping reachability test




(a)




(b)

Fig.15. (a) TCP traffic analysis (b) UDP traffic analysis

## REFERENCES


[1] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, *38*(2), 69-74.

[2] Das, S., Sharafat, A., Parulkar, G., & McKeown, N. (2011, March). MPLS with a simple OPEN control plane. In *Optical Fiber Communication Conference* (p. OWP2). Optical Society of America.

[3] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks*, *71*, 1-30.

[4] Foster, N., Guha, A., Reitblatt, M., Story, A., Freedman, M. J., Katta, N. P., & Walker, D. (2013). Languages for software-defined networks. *Magazine, IEEE*, *51*(2), 128-134.

[5] Kreutz, D., Ramos, F. M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14-76.

[6] Jarraya, Y., Madi, T., & Debbabi, M. (2014). A survey and a layered taxonomy of software-defined networking. *Communications Surveys & Tutorials, IEEE*, *16*(4), 1955-1980.

[7] Mininet, An Instant Virtual Network on your Laptop (or other PC), http://mininet.org/, last accessed on June 2014.

[8] Team, T. M. (2012). Mininet: An Instant Virtual Network on Your Laptop (or Other PC).

[9] Ko3bayashi, Masayoshi, Et al., "Maturing of OpenFlow and Software – defined Networking through deployments," Science Direct Computer Networks, 2013.

[10] Kobayashi, M., Seetharaman, S., Parulkar, G., Appenzeller, G., Little, J., Van Reijendam, J., & McKeown, N. (2014). Maturing of OpenFlow and Software-defined Networking through deployments. *Computer Networks*, *61*, 151-175.

[11] Saurav Das, Et al.(2013), "Handbook of Fiber Optic Data Communication a Practical Guide to Optical Networking Chapter 17. 4th edition". Academic Press.

[12] Agarwal, S., Kodialam, M., & Lakshman, T. V. (2013, April). Traffic engineering in software defined networks. In *INFOCOM, 2013 Proceedings IEEE* (pp. 2211-2219). IEEE.

[13] Demestichas, P., Georgakopoulos, A., Karvounas, D., Tsagkaris, K., Stavroulaki, V., Lu, J., & Yao, J. (2013).

5G on the horizon: key challenges for the radio-access network. *Vehicular Technology Magazine, IEEE, 8*(3), 47-53.
[14] The Open Networking Foundation, "OpenFlow Switch Specification version 1.4.0." October 14. 2013.
[15] Molina, E., Jacob, E., Matias, J., Moreira, N., & Astarloa, A. (2015). Using software defined networking to manage and control IEC 61850-based systems. *Computers & Electrical Engineering*, *43*, 142 154.
[16] Kanai. K.. Muto. T.. Kisara. H.. Katto. J.. Tsuda. T.. Kameyama, W., & Sato, T. (2014, December). Proactive content caching utilizing transportation systems and its evaluation by field experiment. In *Global Communications Conference (GLOBECOM), 2014 IEEE* (pp. 1382-1387). IEEE.
[17] BELLESSA, J. (2015). *Implementing MPLS with label switching in software-defined networks* (Doctoral dissertation, University of Illinois at Urbana-Champaign).
[18] Bholebawa, I. Z., Jha, R. K., & Dalal, U. D. (2015). Performance Analysis of Proposed OpenFlow-Based Network Architecture Using Mininet. *Wireless Personal Communications*, *86*(2), 943-958.
[19] Bhandari, Anju, and V.P.Singh -. "Proposal and Implementation of MPLS Fuzzy Traffic Monitor", International Journal of Advanced Computer Science and Applications, 2016.
[20] Thamarakuzhi, Ajithkumar, and John A. Chandy. "Design and implementation of a nonblocking 2-dilated flattened butterfly switching network", 2010 IEEE Latin- American Conference on Communications, 2010.
[21] Farhady, Hamid Lee, HyunYong Nakao, Akihiro. "Software-Defined Networking: A survey.", Computer Networks The International Journal of Computer and Telecommunications Networking, April, 2015.
[22] Dotcenko, Sergei, Andrei Vladyko, and Ivan Letenko. "A fuzzy logic-based information security management for software-defined networks", 16th International Conference on Advanced Communication Technology, 2014.
[23] Jin, D., & Nicol, D. M. (2013, May). Parallel simulation of software defined networks. In Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation (pp. 91-102), ACM.
[24] Gagandeep Garg, Roopali Garg,"Accurate Anomaly Detection using Adaptive Monitoring and Fast Switching in SDN", IJITCS, vol.7, no.11, pp.34-42, 2015. DOI: 10.5815/ijitcs.2
[25] Qu, Y. R., & Prasanna, V. K. (2016). High-Performance and Dynamically Updatable Packet Classification Engine on FPGA. Parallel and Distributed Systems, IEEE Transactions on, 27(1), 197-209.

## Authors' Profiles

**Ms. Anju Bhandari** is pursuing doctoral program (PhD) at Computer Science and Engineering Department of Thapar University, Patiala, India. Her qualifications include B.Tech (CSE), M.Tech(CSE). She is member of ISTE. She has 9 years of teaching and research experience in softcomputing and computer networks.

**Dr. V. P. Singh** is PhD and ME in Computer Science from Thapar University, Patiala, India. He is presently serving as assistant professor in the Computer Science and Engineering Department of Thapar University, India. His research interests include soft computing, Computer networks, Computer forensics and Cyber Law. He has many research publications in reputed Journals and Conferences.

**How to cite this paper:** Anju Bhandari, V.P. Singh, "Design of Fuzzy-Based Traffic Provisioning in Software Defined Network", International Journal of Information Technology and Computer Science (IJITCS), Vol.8, No.9, pp.49-61, 2016. DOI: 10.5815/ijitcs.2016.09.07