Topology Discovery, Loop Finding and Alternative Path Solution in POX Controller

Anish Kumar Saha, Koj Sambyo, and C.T. Bhunia

***Abstract*— Software defined networking (SDN) is a new networking paradigm which provides the separation between data plane and control plane. There is a centralized controller which controls the packet flows and instruct switches regarding its flow rules. The separation between data plane and control plane makes easy for the management of centralized network. It gives flexibility to the system administrator to control the different network functionality dynamically. Network topology is an important issue for any SDN controller, since it represents the central view of the network infrastructure. The topology helps in controlling the data path and provides flow entries for different switches to handle arrival packets. Link failure is a common issue for any network and it is needed to find alternative paths. This paper focuses on representation of network topology, loops finding and alternate path finding during link failures. The proposed approach designed using POX controller as SDN controller and Mininet emulator as network infrastructure in linux based Ubuntu 14.04 operating system platform.**

***Index Terms*— Software defined networking, Topology discovery, Link failure, Alternative path, POX controller.**

I. INTRODUCTION

Software defined networking (SDN) is a network model in which the data planes and control planes are decoupled from each other. There is a SDN controller, which controls the forwarding of packets and provides flow entries to every switch. The SDN switch has less functionality and these switches perform forwarding of packets based on the match in its flow tables. The instruction comes from SDN controller to every switch and these switches perform forwarding of packets accordingly. If there is no match found in switch's flow table, then these switches communicate to its controller regarding the arrived packet. The controller then instructs flow entries to every switches from source to destination and provide path for the communication. SDN is centralized, programmable, dynamic network architecture. SDN controllers performed as network operating system and different applications can run on its platform. There are three types application programming interface (API) in SDN. These interfaces are North Bound interface, South Bound interface and East-West interface. The interface between application programs & the controller, controller & network infrastructure and communication between multiple controllers named as North Bound API, South Bound API and East-West API respectively. The communications between a SDN controller to switches is done through secure channel communication. Transport Layer Security (TLS) is used for establish communication. Examples of well-known South Bound API are OpenFlow, Forwarding & Control Element Separation (ForCES). In [1-4], authors present surveys and futures on SDN.

Due to dynamic architecture and programmable functionality, SDN provides an easy implementation in different application areas of on-demand networking, energy efficient networking, secure networking from intruder, efficient traffic engineering, load balancing in server, smart grid etc [5-8].

Examples of different well-known controllers are NOX, POX, Beacon, OpenDay light, Floodlight, Ryu, Trema, ONOS, Junpier Contrail etc [9]. The objective of our paper is to emphasize on how to represent network topology, discover the shortest path possible and identify an alternative path during link down. The proposed approach designed using POX controller and mininet emulator as network infrastructure in linux based Ubuntu 14.04 operating system platform.

II. RELATED WORKS OF TOPOLOGY DISCOVERY IN SDN

Discovering the network topology is an important issue in software defines networking. One of the well-known loop free techniques is spanning tree topology. Jmal et al. [10] explained shortest path routing mechanism using POX controller. Pakzad et al. [11] proposed efficient topology discovery with minimum number of Packet-Out events from the controller to the switches. Link failure is a common phenomenon in any networks. In [12] [13], authors presented to handle link failure and provide failure recovery in OpenFlow protocol in SDN networks.

Here we proposed table driven based topology discovery, loops finding and alternate path finding in network. The path with least intermediate switches is selected for communication. When any link failure occurs, the algorithm finds alternate path for communication and update its adjacency matrix of network topology. Adjacency matrix of topology update accordingly with respect to changes occurs in topology.

Manuscript received July 20, 2015; revised October 7, 2015.

Anish Kumar Saha is with the National Institute of Technology, Arunachal Pradesh, District: Papumpare, State: Arunachal Pradesh, Pin: 791112, India. (phone number: +919862216460; fax: +91360-2284972; e-mail: anishkumarsaha@gmail.com).

Koj Sambyo is with the National Institute of Technology, Arunachal Pradesh, District: Papumpare, State: Arunachal Pradesh, Pin: 791112, India. (e-mail: sambyo.koj@gmail.com).

C.T. Bhunia is with the with the National Institute of Technology, Arunachal Pradesh, District: Papumpare, State: Arunachal Pradesh, Pin: 791112, India. (e-mail: ctbhunia@vsnl.com).

III. Proposed Approach of Topology Discovery

The proposed model has segmented into different algorithms. Each algorithm has functionality and in together performed topology discovery, loop finding and failure recovery. These algorithms are design to performed as applications in controller. Here we used POX controller as SDN controller. POX controller worked as publish-subscribe model. There are some objects which generate events and there are some subscribers which subscribe event through event handler. The communication between switch to controller is coordinated through events. There are collections of events and each events will fired under certain condition. POX controller uses OpenFlow protocol for South Bound API. OpenFlow protocol has different events and the events are named as Packet-In, Packet-Out, Port-Status, Flow-Removed, Connection-Up, Connection-Down, Error-In etc. The proposed algorithms usage different notations and these notations are listed in Table 01.

TABLE 1
LIST OF NOMENCLATURES

Different nomenclature
S : Set of all switches in the network
$S_i \in S$: A Switch in the network.
P_i : Set of all ports in Switch S_i
$P_{ij} \in P_i$: A port in the switch S_i
M : Set of all machines in the network
$M_{ij} \in M$: IPaddress of a machine attached to P_{ij} port of Switch S_i.
$P_i' S_i P_i$: P_i' is ingress port & P_i is egress port of S_i

On startup, all switches raise Connection-Up event and communicate to its controller through specified ip-address & port number. The controller gets details of data path identity (DPID) of different switches through Connection-Up event. DPID is a unique identity number to identify a switch. Controller can get other details of MAC address (network interface) of switch ports, attached machine's ip-address & MAC address etc through other events eg. Packet-in event.

Algorithm 1: SwitchPort-MachineIPaddress mapping module at controller
Output: Map_Table ($S_i \rightarrow P_{ij} M_{ij}$)
{ **For all** S_i in S, **do** records $S_i \rightarrow P_{ij} M_{ij}$ map }

The purpose of the Algorithm 1 is to map between switch S_i to machine M_{ij} through switch Port no P_{ij}. This mapping helps in finding where a particular machine M_{ij} attached to which switch S_i and its port number P_{ij}. Algorithm 1 is an event handler for Connection-Up & Packet-In events and will execute whenever events raise from a switch. The data structure of Map_Table($S_i \rightarrow P_{ij} M_{ij}$)is shown in Fig. 1.

Si	P_{i1} M_{i1}	P_{i2} M_{i2}	P_{i3} M_{i3}	P_{i4} M_{i4}	…	P_{in} M_{in}

Fig. 1.Data structure of Map_Table($S_i \rightarrow P_{ij} M_{ij}$)

There are two important events namely Packet-In, Packet-Out. Packet_In event raise from a switch and the controller subscribe the event through its event handler. The Packet-In event is raised when a switch gets a packet and do not have any match entry in its flow table and thus forward the packet to the controller. Conversely the event Packet-Out will rise when a controller wants to send messages to switches. Switch will receives message from controller through Packet-Out event. There is a link layer protocol called Link Layer Discovery Protocol (LLDP) for advertising identity, capabilities and neighbor of a network. Two events Packet-In, Packet-Out and Link Layer Discovery Protocol (LLDP) are needed to discover links in a network.

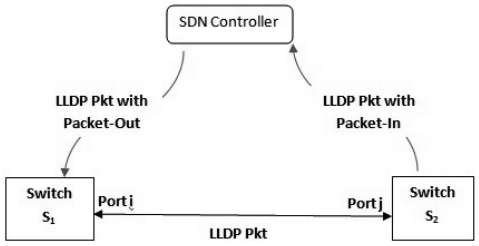

Fig. 2. Packet-In and Packet-Out Events generation

In Fig. 2, switch S_i gets a LLDP packet via Packet-Out event from the controller. Switch S_i then flood the received LLDP packet to all its ports except ingress port. Switch S_2 receives the LLDP packet from switch S_1 through its port j. Then switch S_2 forwards the LLDP packet to the controller via Packet-In event. From the contents of LLDP packet, controller come to know the link between (S_1, Port i) & (S_2, Port j). This process of link finding continues for all the switches exist in a network. Algorithm 2 is an application for the controller to send LLDP packets to every switch and receives all returned LLDP packets from switches. All discovered links are saved in adjacency table(A).

Algorithm 2: Generate and process LLDP-packet at controller
Output: Adjacency Table (A)
{ **For all** Switch S_i *in* S **do** Send LLDP-packet via Packet-Out event to switch S_i from Controller **For all** received LLDP-packet via Packet-in event **do** Insert Link of S_i & S_j in A with $A[i,j] = (Port_i, Port_j)$ where $Port_i \in P_i$ & $Port_j \in P_j$ }

Algorithm 3: Forward LLDP-Packet at switch.
{ **For all** incoming LLDP-Packet **do** **If** LLDP-Packet.inPort = Controller **then** Flood LLDP-Packet to all ports except ingress port **Else** Send LLDP-Packet to the Controller Via Packet-In event }

ISBN: 978-988-14047-6-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Algorithm 3 is a task for every switch. All switches will accept all LLDP packets and floods if the sender is controller (except ingress port) else forward it to the controller via Packet-In event as shown in Fig 2.

Let us take an example of a network in Fig 3. After discovering of all links using Algorithm 2 & Algorithm 3, the contents of adjacency table (A) shown in Table 2.

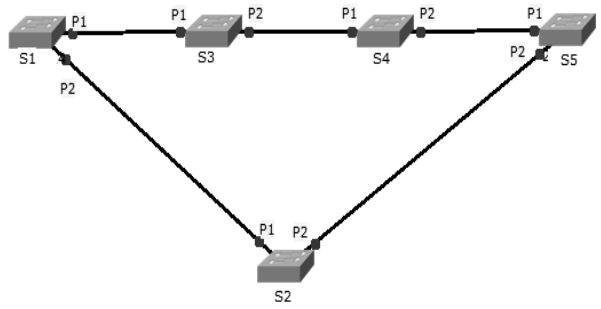

Fig. 3. Network contains five OpenvSwitch (S_1,S_2,S_3,S_4 & S_5)

A value $(S_i, S_j) = (Port_i, Port_j)$ means S_i is connected to S_j via the $Port_i$ and $Port_j$, where $Port_i \in P_i$ & $Port_j \in P_j$.

TABLE 2
ADJACENCY TABLE(A) OF SWITCH-TO-SWITCH CONNECTION

	S_1	S_2	S_3	S_4	S_5
S_1	-	P_2, P_1	P_1,P_1	-	-
S_2	P_1,P_2	-	-	-	P_2, P_2
S_3	P_1, P_1	-	-	P_2, P_1	-
S_4	-	-	P_1,P_2	-	P_2, P_1
S_5		P_2, P_2		P_1,P_2	-

Assume the structure of the network shown is in fig 3.

The next step is to find a path between every switch to every other switch in a network. Algorithm 4 finds path to every switch to every other switches and it also identify list of loops in a topology. This list of loops will help in finding alternative paths during link failure.

Algorithm 4: Path Finding at Controller

Data Structure: Input: Adjacency Table (A)
Output: Adjacency Table(A), Loop Table (L)

For all values of S_i in S **do**
  For all connection exist between (S_i, S_j) & $i \neq j$ **do**
    For all Connection exist in (S_j,S_k) & $i, j \neq k$ **do**
      $Temp = A[i,j] \cup S_j \cup A[j,k]$
    If A[i, k] = Null **then**
      $A[i,k] = Temp$
    Else
      Loop=$Port_i \cup S_i \cup Temp \cup S_k \cup \{reverse\_path(A[i,k]) - Port_i\}$
    If Loop **not exist in** L **then**
      Record Loop in L
    If no. of intermediate Switch in A[i,k] > No. of intermediate Switch(Temp) **then**
      $A[i,k] = Temp$

Let us take an example, how Algorithm 4 works for the network in Fig. 3.

Pass 1: For S_i=S_1, S_j=S_2, S_k=S_5,

$$A[1,5] = A[1,2] \cup S_2 \cup A[2,5]$$
$$= P_2, P_1 \cup S_2 \cup P_2 P_2$$
$$= P_2,\ P_1 S_2 P_2,\ P_2$$

New entry: **(S_1, S_5) = P_2, $P_1S_2P_2$, P_2**

After pass 1, the content of adjacency table (A) is shown in Table 3.

TABLE 3
ADJACENCY TABLE(A) AFTER PASS 1

	S_1	S_2	S_3	S_4	S_5
S_1	-	P_2,P_1	P_1,P_1	-	**P_2, $P_1S_2P_2$, P_2**
S_2	P_1,P_2	-	-	-	P_2, P_2
S_3	P_1, P_1	-	-	P_2, P_1	-
S_4	-	-	P_1,P_2	-	P_2, P_1
S_5		P_2, P_2		P_1, P_2	-

Pass 2: For S_i=S_1, S_j=S_3, S_k=S_4,

$$A[1,4] = A[1,3] \cup S_3 \cup A[3,4]$$
$$= P_1, P_3 \cup S_3 \cup P_2, P_1$$
$$= P_1, P_3 S_3 P_2, P_1$$

New entry **(S_1, S_4) = P_1, $P_1S_3P_2$, P_1** and the adjacency table's (A) content shown in Table 4.

TABLE 4
ADJACENCY TABLE(A) AFTER PASS2

	S_1	S_2	S_3	S_4	S_5
S_1	-	P_2,P_1	P_1,P_1	**P_1, $P_1S_3P_2$, P_1**	P_2,$P_1S_2P_2$,P_2
S_2	P_1,P_2	-	-	-	P_2, P_2
S_3	P_1,P_1	-	-	P_2, P_1	-
S_4	-	-	P_1,P_2	-	P_2, P_1
S_5		P_2,P_2		P_1, P_2	-

Pass 03: For S_i=S_1, S_j=S_4, S_k=S_5,

Similarly if we continue pass 03 steps for Algorithm 4, we get

$$Temp = A[1,4] \cup S_4 \cup A[4,5]$$
$$= P_1, P_1S_3P_2, P_1 \cup S_4 \cup P_2, P_1$$
$$= P_1, P_1S_3P_2, P_1S_4P_2, P_1$$

Since the content of A[1,5] is not null, means there is a presence of loop between S_1 to S_5.

$$Loop = Port_1 \cup S_1 \cup Temp \cup S_5 \cup \{reverse\_path\ (A[1,5]) - Port_1\}$$
$$= P_2 \cup S_1 \cup P_1, P_1S_3P_2, P_1S_4P_2, P_1 \cup S_5 \cup$$
$$\{reverse\_path(P_2, P_1S_2P_2, P_2) - P_2\}$$
$$= P_2S_1P_1, P_1S_3P_2, P_1S_4P_2, P_1S_5 \cup \{(P_2, P_2S_2P_1, P_2) - P_2\}$$
$$= P_2S_1P_1, P_1S_3P_2, P_1S_4P_2, P_1S_5P_2, P_2S_2P_1$$

The value of loop is identified and placed in loop table as shown in Table 6. Again path (P_1,$P_1S_3P_2$,$P_1S_4P_2$,P_1) contains two intermediate switch S_3 & S_4 and the path (P_2, $P_1S_2P_2$, P_2) contain one intermediate switch S_2. Here we use least intermediate node path, hence the value of (S_1,S_5) in adjacency table is (**P_2, $P_1S_2P_2$, P_2**). After Pass 03, the adjacency table(A) is shown in Table 5.

ISBN: 978-988-14047-6-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE 5
ADJACENCY TABLE(A) AFTER PASS 04

	S_1	S_2	S_3	S_4	S_5
S_1	-	P_2,P_1	P_1,P_1	**P_1, $P_1S_3P_2$, P_1**	**P_2, $P_1S_2P_2$, P_2**
S_2	P_1,P_2	-	-	-	P_2, P_2
S_3	P_1,P_1	-	-	P_2, P_1	-
S_4	-	-	P_1,P_2	-	P_2, P_1
S_5		P_2,P_2		P_1, P_2	-

TABLE 6
LOOP TABLE AFTER PASS 04

Index	Loop
1	$P_2S_1P_1, P_1S_3P_2, P_1S_4P_2, P_1S_5P_2, P_2S_2P_1$

If we continue our steps, the final adjacency table is shown in Table 7 and Loop table will be same as shown in Table 6.

TABLE 7
ADJACENCY TABLE(A) AFTER ALL PASS COMPLETED

	S_1	S_2	S_3	S_4	S_5
S_1	-	P_2, P_1	P_1,P_1	P_1,$P_1S_3P_2$,P_1	P_2,$P_1S_2P_2$,P_2
S_2	P_1,P_2	-	P_1, $P_2S_1P_1$, P_1	P_2,$P_2S_5P_1$,P_2	P_2, P_2
S_3	P_1,P_1	P_1,$P_1S_1P_2$,P_1	-	P_2, P_1	P_2,$P_1S_4P_2$,P_1
S_4	P_1,$P_2S_3P_1$,P_1	P_2,$P_1S_5P_2$, P_2	P_1,P_2	-	P_2, P_1
S_5	P_2,$P_2S_2P_1$,P_2	P_2, P_2	P_1, $P_2S_4P_1$,P_2	P_1, P_2	-

IV. ROUTING AND FAILURE RECOVERY USING ADJACENCY MATRIX TABLE & LOOP TABLE

Let us take a machine M_i attached to switch S_i wants to communicate with the machine M_j attached to switch S_j. When packets from M_i arrived at switch S_i, switch S_i searches the flow match for the destination address M_j in its flow table. At initial condition, there is no flow entry in its flow tables in switch S_i for the destination machine M_j and therefore forwarded to the controller via Packet-In event. From the IPaddress Map Table($S_i \rightarrow P_{ij} M_{ij}$), controller get the DPID and port number of destination switch for M_j. The communication path between source switch to destination switch can get from adjacency table (A). Finally controller instructs flow entry for every switch from S_i to S_j via Packet-Out event.

For instance, path between S_1 to S_5 is $S_1P_2 \rightarrow P_1S_2P_2 \rightarrow P_2S_5$. The meaning of S_1P_1 is P_2 is the egress port of S_1 switch and the meaning of P_2S_5 is P_2 ingress port of S_5. In addition, the intermediate switch S_2 has ingress port P_1 & egress port P_2. Controller give flow entry for switch S_1,S_2 and S_5 for the communication between M_i to M_j. All packets for the said communication will follow the path $S_1 \rightarrow S_2 \rightarrow S_5$.

Now assume, link $S_i \rightarrow S_j$ goes down. The controller comes to know that the link (S_i,S_i) is down via PortStatus event raised from the switches S_i & S_j. Henceforth the controller need to find alternate path for the link (S_i,S_i). For such case, the loop table (L) will helps for alternative path. Controller first searches a loop contained both switches S_i & S_j in loop table (L). If there is a loop available, then alternate path can get from loop by reading it in the reverse direction from S_i to S_j and converts all ingress port to egree port & vice versa for all switches S_i to S_j. The same is shown in Fig. 4. The changes are updated in its adjacency table (A) and in loop table (L).

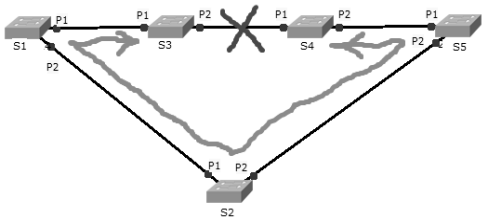

Fig. 4. Alternative path after down link (S_3 to S_4)

Algorithm 5**:** Alternate path at Controller

Data Structure:
Input: Down Link (S_i, S_j)
Loop Table (L)
Output: Alternate_Path of down Link (S_i, S_j)

For all loop in L **do**
if loop contained edge (S_i, S_j) **then**
$Temp = Reverse\_Read\_Path(loop\ from\ S_i\ to S_j)$
for all value S_i in S **do**
for all value S_j in S and $i \neq j$ **do**
if A[i,j] path contain edge (S_i, S_j) **then**
Replace (Si,Sj) in A[i,j] with Temp as below
$A[i,j] = Path\ (P_i \ldots Temp \ldots P_j)$
Delete Entry L in loop

Algorithm 5 shows to find an alternative path during any link down. Assume, $P_m^{'} S_m P_m, \ldots \ldots P_i^{'} S_i P_i, P_j^{'} S_j P_j, \ldots \ldots, P_n^{'} S_n P_n$ be a loop contain both (S_i,S_i). So, the Reverse_Read_Path will be

$$P_i S_i P_i^i \ldots \ldots P_m S_m P_m^{'}, P_n S_n P_n^{'}, \ldots \ldots P_j S_j P_j^{'}$$

For instance,
Assume down Link: (S_3,S_4).
Loop contained (S_3,S_4) link in L is,
$P_2S_1P_1$, $\mathbf{P_1S_3P_2}$, $\mathbf{P_1S_4P_2}$, $P_1S_5P_2$, $P_2S_2P_1$

Earlier value of A[1,4]= P_1, $P_1S_3P_2$, P_1

Temp= Reverse_Read_Path(L from S_3 to S_4)
$= P_2S_3P_1, P_1S_1P_2, P_1S_2P_2, P_2S_5P_1, P_2S_4P_1$
New Entry A[1,4]= Path(S_1P_1, Temp, P_1S_4)
=Path($S_1P_1, P_2S_3P_1, P_1S_1P_2, P_1S_2P_2, P_2S_5P_1, P_2S_4P_1, P_1S_4$)
= ($S_1P_2, P_1S_2P_2, P_2S_5P_1, P_2S_4$)
=($P_2, P_1S_2P_2, P_2S_5P_1, P_2$)

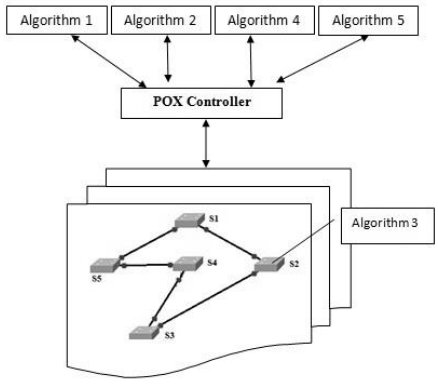

Fig.5. The network architecture with POX controller and its applications

We used here POX controller as SDN controller and mininet emulator as network infrastructure respectively. Network architecture and placement of different algorithms are shown in Fig. 5.

V. CONCLUSION

For any SDN architecture, topology discovery is an important issue. The proposed algorithms are work as event handler for the controller. We maintained adjacency table and loop table to find path and alternative paths. We used minimum intermediate node path for communication. Our next plan is to design north bound applications for energy efficient resource control, load balancing and traffic engineering in SDN paradigm.

REFERENCES

[1] Nunes, B.A.A.; Mendonca, M.; Xuan-Nam Nguyen; Obraczka, K.; Turletti, T., "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *Communications Surveys & Tutorials, IEEE* , vol.16, no.3, pp.1617,1634, Third Quarter 2014

[2] Kreutz, D.; Ramos, F.M.V.; Esteves Verissimo, P.; Esteve Rothenberg, C.; Azodolmolky, S.; Uhlig, S., "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE* , vol.103, no.1, pp.14,76, Jan. 2015

[3] Jarraya, Y.; Madi, T.; Debbabi, M., "A Survey and a Layered Taxonomy of Software-Defined Networking," *Communications Surveys & Tutorials, IEEE* , vol.16, no.4, pp.1955,1980, Fourthquarter 2014

[4] Fei Hu; Qi Hao; Ke Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *Communications Surveys & Tutorials, IEEE* , vol.16, no.4, pp.2181,2206, Fourthquarter 2014

[5] Msahli, M.; Pujolle, G.; Serhrouchni, A.; Fadlallah, A.; Guenane, F., "Openflow and on demand networks," *Network of the Future (NOF), 2012 Third International Conference on the* , vol., no., pp.1,5, 21-23 Nov. 2012

[6] Giroire, F.; Moulierac, J.; Phan, T.K., "Optimizing rule placement in software-defined networks for energy-aware routing," *Global Communications Conference (GLOBECOM), 2014 IEEE* , vol., no., pp.2523,2529, 8-12 Dec. 2014

[7] Carpa, Radu; Gluck, Olivier; Lefevre, Laurent, "Segment routing based traffic engineering for energy efficient backbone networks," *Advanced Networks and Telecommuncations Systems (ANTS), 2014 IEEE International Conference on* , vol., no., pp.1,6, 14-17 Dec. 2014

[8] Jianchao Zhang; Boon-Chong Seet; Tek-Tjing Lie; Chuan Heng Foh, "Opportunities for Software-Defined Networking in Smart Grid," *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on* , vol., no., pp.1,5, 10-13 Dec. 2013

[9] Khondoker, R.; Zaalouk, A.; Marx, R.; Bayarou, K., "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on* , vol., no., pp.1,7, 17-19 Jan. 2014

[10] Jmal, R.; Chaari Fourati, L., "Implementing shortest path routing mechanism using Openflow POX controller," Networks, Computers and Communications, The 2014 International Symposium on , vol., no., pp.1,6, 17-19 June 2014

[11] Pakzad, F.; Portmann, M.; Wee Lum Tan; Indulska, J., "Efficient topology discovery in software defined networks," Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on , vol., no., pp.1,8, 15-17 Dec. 2014

[12] Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P., "Fast failure recovery for in-band OpenFlow networks," *Design of Reliable Communication Networks (DRCN)*, 2013 9th International Conference on the , vol., no., pp.52,59, 4-7 March 2013

[13] Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P., "Enabling fast failure recovery in OpenFlow networks," *Design of Reliable Communication Networks (DRCN)*, 2011 8th International Workshop on the , vol., no., pp.164,171, 10-12 Oct. 2011