International Journal of Computer Applications (0975 – 8887)
Volume 116 – No. 1, April 2015

Emulation of Shortest Path Algorithm in Software Defined Networking Environment

Arnav Shivendu
Department of Computer Science and Engineering
Sikkim Manipal Institute of Technology, East Sikkim, India

Dependra Dhakal
Department of Computer Science and Engineering
Sikkim Manipal Institute of Technology, East Sikkim, India

Diwas Sharma
Department of Computer Science and Engineering
Sikkim Manipal Institute of Technology, East Sikkim, India

ABSTRACT
The field of Computer network has evolved a lot since the last three decades from conventional static networks to dynamically programmed architecture. The main goal of Software Defined Networking (SDN) is for the network to be open and programmable. Traditional network devices like router and switches can take routing decision and forwarding of these packets, SDN separate these component into different planes by pulling different component apart called as Data plane and Control plane. Hence the switches only have packet forwarding capability and cannot make any routing decisions, while decision making is done by the controller. OpenFlow is the interface that helps the switches and the controller to communicate. It is a communication protocol that enables the controller to determine the route of the network packet via the switches. This project implements the Bellman – Ford algorithm to find the shortest path between two nodes in a network using SDN environment. POX API's has been used to implement the Bellman – Ford Algorithm.

Keywords
Bellman – Ford, Software Defined Networking (SDN), POX, Mininet.

1. INTRODUCTION
In traditional networks, the network devices have a control plane and data plane within the same device[1][16]. Control plane gives information to generate forwarding table while data plane forwards the packet based on the entry provided in the forwarding table. With the changes in network pattern and type of traffic there is a need to have a network architecture which is more dynamic and flexible [6]. (Software Defined Networking) SDN is a technology to separate the control plane and data plane of network devices. SDN uses a centralized controller to generate flow tables that configures the forwarding table responsible for forwarding the packets in the network[1][5][6]. Controller will typically have core services to aid in job of interfacing with network nodes and for providing a programmable interface to network application [6][7]. Data device or forwarding device receive packets, take action on those packets and update counters. Types of action include dropping of packets, modifying the header, sending packet to single or multiple ports. Instruction to how to handle the packets originates with SDN controller [9][10][16]. This device also caches this information for future use. Future packets of the same type can take a fast path with no need to contact the SDN controller. Application programs can be run on top of a controller to monitor and manage the network in a centralized manner[9] [11].

In a SDN, traffic can be shaped from controller without configuring the individual switches. The administrator can build application based on his or her organization requirement, thus giving flexibility and efficiently managing traffic. POX [3][12] is a Python based open source SDN Controller for developing SDN applications. POX [3][12] controller can be used to run different applications like switch, load balancer and firewall. Communication between the controller and the switches is carried by communication protocol such as OpenFlow[14]. Openflow[1][14] is the most famous standard protocol used in SDN. Switches are unable to function without being programmed by the controller. The figure 1 [13] below describes how the controller coordinates with the switch using open flow protocol along with the network application or controller application based on any organization need and policies.

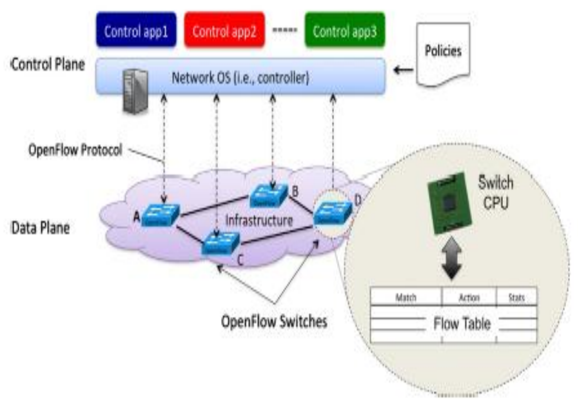


Fig 1: SDN reference design [13]

2. PRELIMINARIES
2.1 Mininet
Mininet is a network emulator tool that accurately emulates any type of forwarding element, in terms of function and performance. SDN network can be created as per required specifications and testing can be on different network configurations. Once the testing is done in the SDN solution on Mininet we can move it to a real physical network[15].

2.2 Bellman – Ford Algorithm
The Bellman – Ford Algorithm has been used as each node only needs to know its own number and be able to derive the number of neighbors it has. The calculation of the shortest path to a given destination is done by hoping neighbor by neighbor from each source to destination [2].

2.3. POX
POX [4][12] is a Python based open source SDN Controller for developing SDN applications. POX controller can be used to run different applications like switch, load balancer, and firewall.

3. THE BELLMAN – FORD ALGORITHM DESCRIPTION

Given a weighted, directed graph G=(V, E) and a single source node v, the Bellman - Ford algorithm can return a shortest path from the source node v to every other node, where V is the set of nodes and E is the set of edges. Table 1 shows Bellman - Ford algorithm [2] [3], whose input is a given graph G = (V, E), the edge weight setting cost, number of nodes n and the single source node v. The dist[u] to store the distance of the current shortest path from the source node s to the destination node u, and uses cost[u,i] to store the previous cost of u to another node i.

Table 1. The Bellman – Ford Algorithm [3]

Bellman – Ford Algorithm
Input : G(V,E), dist, cost, v,n
Output : dist[|V|]
for i := 1 to n do // Initialize dist. dist [i] : = cost [v , i] ; for k : = 2 to n-1 do for each u such that u ≠ v and u has at least one incoming edge do for each < i, u > in the graph do if dist [u] > dist [i] + cost [i,u] then dist [u] : = dist [i] + cost [i,u] ;

4. SIMULATION

4.1 Simulation Setting

Mininet is used to create a network topology as shown in Fig 2 with two hosts, eight switches and a controller.

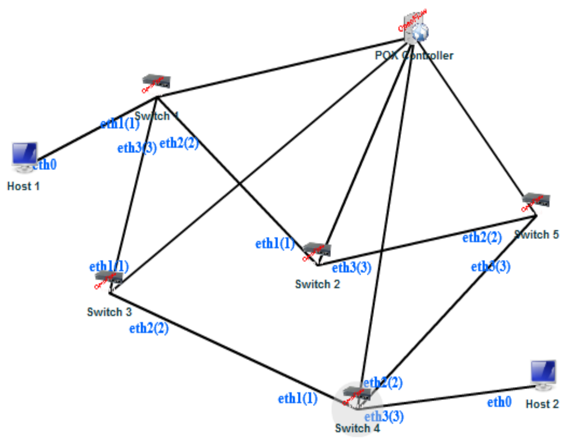
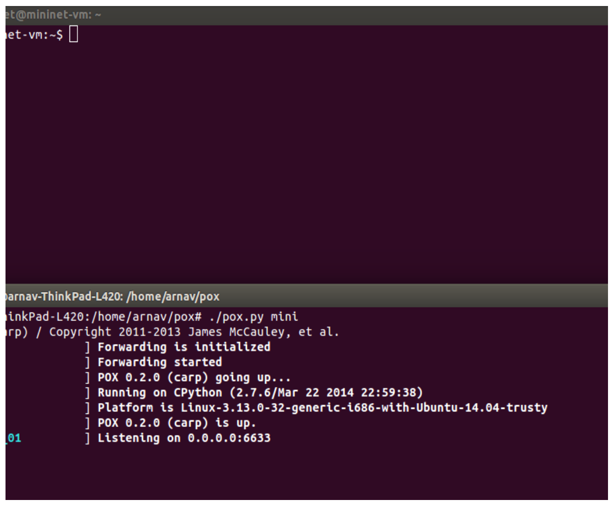


Fig 2. Network Topology

4.2 Simulation Results

In Fig 3. it can be seen that when the custom SDN application called "mini" is run it calls the launch function that gives details regarding the POX controller and whether it is running or not. It also listens to request on port no. 6633(can be changed) from a python script that creates the network topology.

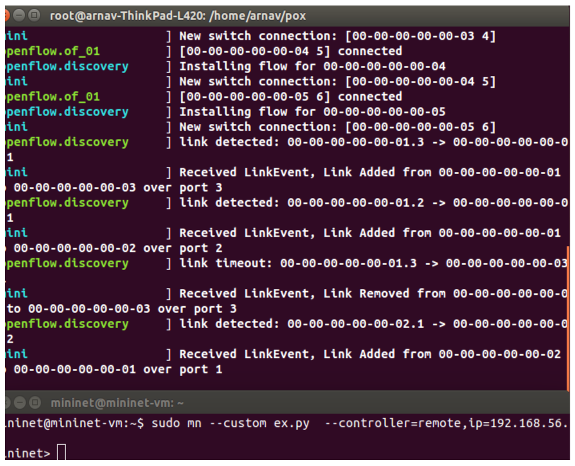


```
et@mininet-vm: ~
et-vm:~$ 

arnav-ThinkPad-L420: /home/arnav/pox
inkPad-L420:/home/arnav/pox# ./pox.py mini
rp) / Copyright 2011-2013 James McCauley, et al.
           ] Forwarding is initialized
           ] Forwarding started
           ] POX 0.2.0 (carp) going up...
           ] Running on CPython (2.7.6/Mar 22 2014 22:59:38)
           ] Platform is Linux-3.13.0-32-generic-i686-with-Ubuntu-14.04-trusty
           ] POX 0.2.0 (carp) is up.
01         ] Listening on 0.0.0.0:6633
```

Fig 3. Running Custom Application "mini"

In Fig 4. it can be observed that whenever a switch is added to the network after running a python script called "ex.py" containing the specification of the network it generates a "Switch connection event".

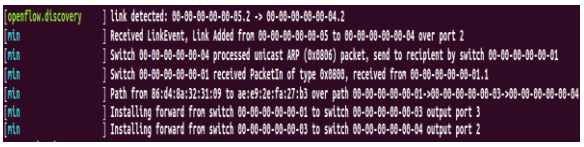
Also when a new link is added or removed it generates "Received LinkEvent" and "Remove LinkEvent" respectively.



```
root@arnav-ThinkPad-L420: /home/arnav/pox
ini                  ] New switch connection: [00-00-00-00-00-03 4]
penflow.of_01        ] [00-00-00-00-00-04 5] connected
penflow.discovery    ] Installing flow for 00-00-00-00-00-04
ini                  ] New switch connection: [00-00-00-00-00-04 5]
penflow.of_01        ] [00-00-00-00-00-05 6] connected
penflow.discovery    ] Installing flow for 00-00-00-00-00-05
ini                  ] New switch connection: [00-00-00-00-00-05 6]
penflow.discovery    ] link detected: 00-00-00-00-00-01.3 -> 00-00-00-00-00-0
1
ini                  ] Received LinkEvent, Link Added from 00-00-00-00-00-01
 00-00-00-00-00-03 over port 3
penflow.discovery    ] link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-0
1
ini                  ] Received LinkEvent, Link Added from 00-00-00-00-00-01
 00-00-00-00-00-02 over port 2
penflow.discovery    ] link timeout: 00-00-00-00-00-01.3 -> 00-00-00-00-00-03

ini                  ] Received LinkEvent, Link Removed from 00-00-00-00-00-0
to 00-00-00-00-00-03 over port 3
penflow.discovery    ] link detected: 00-00-00-00-00-02.1 -> 00-00-00-00-00-0
2
ini                  ] Received LinkEvent, Link Added from 00-00-00-00-00-02
 00-00-00-00-00-01 over port 1

mininet@mininet-vm: ~
ninet@mininet-vm:~$ sudo mn --custom ex.py  --controller=remote,ip=192.168.56.

ninet> 
```

Fig 4. Adding of switches and links dynamically

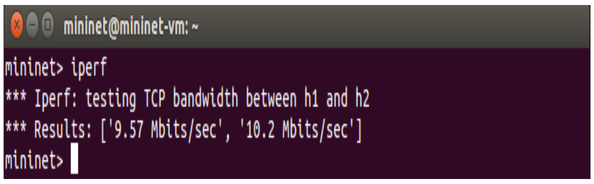
Once all the switches and links have been detected, transmission of packets is now possible. When host "h1" pings "h2", Bellman - Ford algorithm finds the shortest path for the transmission as shown in Fig 5. The packet starts getting transmitted one by one along the path. Initially ARP packet is sent to discover the IP addresses of each of the network devices involved and the flow table of the switches involved.



```
[openflow.discovery   ] link detected: 00-00-00-00-00-05.2 -> 00-00-00-00-00-04.2
[min                  ] Received LinkEvent, Link Added from 00-00-00-00-00-05 to 00-00-00-00-00-04 over port 2
[min                  ] Switch 00-00-00-00-00-04 processed unicast ARP (0x0806) packet, send to recipient by switch 00-00-00-00-00-01
[min                  ] Switch 00-00-00-00-00-01 received PacketIn of type 0x0800, received from 00-00-00-00-00-01.1
[min                  ] Path from 86:d4:8a:32:31:09 to ae:e9:2e:fa:27:b3 over path 00-00-00-00-00-01->00-00-00-00-00-03->00-00-00-00-00-04
[min                  ] Installing forward from switch 00-00-00-00-00-01 to switch 00-00-00-00-00-03 output port 3
[min                  ] Installing forward from switch 00-00-00-00-00-03 to switch 00-00-00-00-00-04 output port 2
```

Fig 5. Routing along shortest path

In fig 6, it can be observed that that the throughput of the network using a TCP stream comes out to be 38.6 Kbits/sec when the bandwidth is 10 Mbits/sec.



```
mininet@mininet-vm: ~
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.57 Mbits/sec', '10.2 Mbits/sec']
mininet>
```

Fig 6. Routing along shortest path

5. SUMMARY AND CONCLUSIONS

In this paper, the Bellman – Ford algorithm has been implemented in a Software Defined Networking environment using Mininet and POX to demonstrate the dynamic programmability using controller. The field of software defined networking is quite recent, but growing at a very fast pace. There are important research challenges to be addressed like security. If an organization requires a specific type of a network behavior it can develop or install application to do what it needs, this application may be common networking function for example traffic engineering, policy routing, firewall and security. SDN has the potential to facilitate the deployment and management of network applications and services with greater efficiency. In future load balancing and firewall can be emulated based on our campus requirements. The firewall policies could be different for any organization, considering that fact SDN gives the flexibility to develop application based on individual campus requirement like during a online examination in a university, the priority and higher bandwidth link can be given to html pages during that hour. Similarly load balancing can be implemented based on link cost and number of controllers.

6. REFERENCES

[1] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, Thierry Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", Vous Consultez L'Archive Hal, 19 Jan, 2014.

[2] David Walden," The Bellman-Ford Algorithm and Distributed Bellman-Ford".

[3] Shivani Sanan, Leena Jain, Bharti Kappor, "Shortest Path Algorithm", International Journal of Application or Innovation in Engineering & Management (IJAIEM).

[4] S.K. Kaur, Japinder Singh, Navtej Singh Ghumman, "Network Programmability Using POX Controller" International Conmference on Communication, Computing and Systems (ICCS 2014)

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson,J. Rexford, S. Shenker, and J Turer, "OpenFlow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] N. Mckeown, "How SDN will Shape Networking," October 2011. [Online]. Available: http://www.youtube.com/watch?v=c9-K5O qYgA

[7] ONF, "Open networking foundation," 2014. [Online]. Available: https://www.opennetworking.org

[8] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," IEEE Internet Computing, vol. 17, no. 2, 2013, pp. 20

[9] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "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 2014.

[10] K. Greene, "MIT Tech Review 10 Breakthrough Technologies: Software-defined Networking," http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/, 2009

[11] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin, "Anetwork-state management service," in Proceedings of the 2014 ACM Conference on SIGCOMM,

[12] M. McCauley, "POX," 2012. [Online]. Available: http://www.noxrepo.org/

[13] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller , Joseph Little , Johan van Reijendam , Paul Weissmann, Nick McKeown "Maturing of OpenFlow and Software-defined Networking through deployments".(2013), http://dx.doi.org/10.1016/j.bjp.2013.10.011

[14] OpenFlowTutorial. <http://www.openflow.org/wk/index.php/OpenFlow_Tutorial>.

[15] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, 2010.

[16] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg,Siamak Azodolmolky and Steve Uhlig," Software-Defined Networking: A Comprehensive Survey" Proceedings of the IEEE (Volume:103, Issue: 1) pp 14-76, Dec 2014