International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Congestion Control Techniques in Programmable Computer Networks

Ankita[1], S. Vamshi Krishna[2]

[1]Ms., M-tech (EC) Student, MGM COET, Department of ECE, Noida, UP, India

[2]Head of the Department, MGM COET, Department of ECE, Noida, UP, India

Abstract: ***Software Defined Networking (SDN) is becoming an apparent network architecture where network control is separated from data plane which is directly programmable. It promises to remove the complexity of the network management and enable upheaval through network programmability. The POX is one of the open source Software Defined Network controller and a platform for the expeditious development and designing of network control software. This paper defines the difference between various parameter when calculated for traditional network and SDN network and proposes a set of rules which are defined in the POX controller as the controller component and based on which the open flow switch will take the decisions during transmission of data packets in the Open flow network. This method proposed to increase the performance of the network by avoiding congestion using spanning tree protocol which reroutes the packet in case of link failure/congestion. This paper also described the process of defining different set of rules in each switch in the network. The experimental results shows average round trip time(RTT) or average delay for network with and without congestion/link failure detection for POX controller. Along with the comparison between traditional and SDN network based on various parameters.***

Keywords: Software Defined Networking, POX Controller, Open flow, Mininet, Traditional Network, Python

1. Introduction

Traditional network architectures are futile to meet the requirements of today's enterprises, carriers, data centers and end users. Due to the major industry effort lead by the Open Networking Foundation (ONF), Software-Defined Networking (SDN) is reorganizing networking architecture. The SDN architecture is accomplished by separating the control plane from the data plane and by giving a programmable interface for that separated control plane, unlike the traditional architecture, the network intelligence and state are centralized analytically and the underlying network infrastructure is abstracted from the applications. As a result, the enterprises gain unprecedented network control, programmability and automation which enable them to build highly scalable, flexible and non-complex networks that readily adapt changing business requirements.

With this system in place for centralized command and control of the network through SDN and a programmable interface, more automated processes can be added to handle complex systems. Real-time decisions can be taken for traffic optimization, security, maintenance. Separate traffic types can be run side-by-side while receiving different paths and forwarding that can respond accurately to the network changes.SDN is currently attracting significant attention from both academic area and industry. A group of network operators, service providers, and vendors have recently created the Open Network Foundation, an industrial operatedorganization, to promote SDN and standardize the Open Flow protocol. On the educational side, the Open Flow Network Research Center [11] has been created with a focus on SDN research. There have also been standardization efforts on SDN at various industrial firms.
The main aim is to make software developers rely on the network components in an easy manner as they do on storage and computing resources. The SDN architecture is shown in figure 1. In SDN network intelligence is locally centralized in control plane (controller) whereas data plane consist of simple packet forwarding devices (switches,hosts) that can be programmed via open interface.

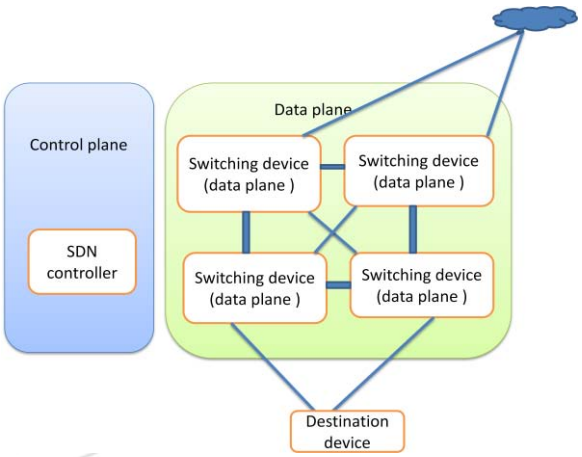

Figure 1: SDN architecture

POX is a open source SDN controller and a platform which allows to program the devices through controller and for prototyping the network control software using python programming language. POX is mainly used for research purpose in the field of SDN. POX can run anywhere, particularly it targets Linux, MAC OS and windows.The POX contains reusable sample components for shortest path selection, controlling the switch behavior, etc. It also supports GUI and visualization tools.

The field of SDN is quiet recent and growing very fast. There a lot of challenges to be addressed in this field now. In this paper we propose a set of components which are defined in the POX controller on which the Openflow switches will take appropriate action for forwarding the incoming packets to the destination. In this paper we are also defining difference between some parameters of traditional network

compared to Openflow network with the help of graphs based on experiments we have performed.

The method is presented to define different rules to the switch in the network. Then the method to reroute packets in case of a link failure/congestion is also defined. Also comparison between the Open flow network with congestion detection and the network without congestion detection in terms of average RTT is shown.

Open Flow [1] is the first standard communications link defined between the control and datalayers (consisting of forwarding components) of an SDN architecture. Open Flow permits the direct access and handling of the forwarding plane of network devices such as switches and hosts both physical and virtual (hypervisor based). It is the unavailability of an open link to the forwarding plane that has led tothe characterization of today's networking devices as inflexible, closed, and rigid like. Open Flow is one of the standard protocols available and a protocol like Open Flow is required to relocate the network control out of the switches to logically centralized control Software [9].

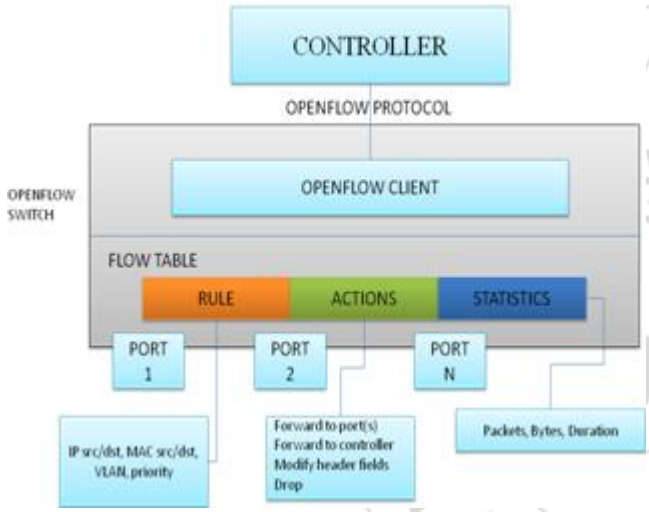

Figure2: The Open flow architecture

The OpenFlow protocol [4] is a key facilitator for software defined networks and currently is the one of the standardized SDN protocol that allows direct designing of the forwarding plane of network devices. It was initially applied to Ethernet-based networks and then the OpenFlow switching spread out to a much broader set of use cases. OpenFlow SDNs can be deployed on extant networks, both physical and virtual. Network devices support OpenFlow forwarding and also traditional forwarding, which makes it unchallenging for the enterprises andcarriers to progressively introduce OpenFlow-based SDN technologies, even in multi distributor network environments.

Mininet is a network emulator. It executes a collection of end-hosts, routers, legacy or Openflow switches and links on a Linux kernel. It utilizes the lightweight virtualization to form a singlesystem which looks like a complete network, operating onthe same system, kernel and user code. A Mininet hostacts like a real machine. The Ssh protocol is used andit is used to run whimsical programs (including the network services implemented on the underlying Linux system). In short, Mininet's virtual hosts, switches, links, and controllers are the real thing – they are just designed using software other than hardware – and for the most part their behavior is similar to discrete hardware elements. It is practically feasible to create a Mininet network that look like a hardware network, or a hardware network that looks like a Mininet network and to run the applications and the binary code on either platform.

This paper is arranged as follows. Section II discusses some of the related work. Section III describes the proposed method. Section IV discusses the evaluation procedure and performance results and section V discuss the conclusion and future work that can be done.

2. Related Work

Ethane [13], the antecedent of NOX and OpenFlow, is an early flow-based networking technology for creating reliable enterprise networks. Ethane shows that by restricting transmission in the network before an identity is verified by a central controller, strong security policies can be enforced in the network. Ethane does not considerusing parallelism in their designs.

NOX [7] is a platform for building network control applications which expands the Ethane work in two dimensions. First, it attempts to scale the centralized pattern to very large systems. The second extension is allowing general programmatic control of the network. The Ethane systems were created around a single application: identity-based access control. NOX provides a general programming link that makes it easier to sustain current management tasks and possible to provide more advanced management functionality.

Maestro [3] shows how the rudimentary problem of performance bottleneck in controller is resolved by parallelism. Maestro provides a basic programming model for programmers and exploits parallelism together with additional throughput increasing techniques. The throughput of Maestro can attain near linear scalability on an eight core server machine.

Hyper Flow [14] aims at upgrading the performance of the OpenFlow control plane. However, Hyper Flow takes a completely different outlook by expanding NOX to a distributed control plane. By synchronizing network-wide state among distributed controllers in the background through a administrated file system, HyperFlow ensures that the processing of a particular flow request can be localized to an individual controller machine. The techniques employed by Hyper Flow are orthogonal to the design of the controller and they can also strengthen Maestro to become fully distributed to attain even higher overallscalability.

DIFANE [15] provides a way to accomplish efficient rule based policy enforcement in a network by executing policy rules matching at the switches.DIFANE's network controller installs policy rules inswitches and does not need to be mixed up in matchingpackets against these rules as in OpenFlow. However,OpenFlow is more reliable since its control logic canrealize behaviors that cannot be easily

carried out by a set of static policy rules installed in switches. Ultimately, the techniques recommended by DIFANE to offload policy rules complementing onto switches and our techniques to increase the performance of the controller are extremely complementary: Functionalities that can be achieved by DIFANE can be off-loaded to switches, while tasks that require central controller processing can be handled systematically by Maestro.

Beacon [6] is a Java-based OpenFlow controller. Beacon reviewed new areas of the OpenFlow controller design space, with a focus on being favorable to the developer, high performance and having the capability to start and stop existing and new software and program at runtime. Beacon showed high performance and was able to scale linearly with various processing cores.

In the above mentioned papers, few of the issues have not been addressed such as the bandwidth utilization in POX controller, reducing the RTT time to increase the average RTT and avoiding the network congestion.

3. Proposed Method

In this section, the some of the methods are defined which are used to show the congestion management in SDN compared to traditional network in terms of comparison between various networks on different parameters. First algorithm is provided for discovering the network topology and to detect the link failure. Then certain components of POX are discussed which can be used to control the behavior of an Open flow switch. Finally some POX components responsible for rerouting the packets/load in case of link failure/congestion are discussed to avoid the congestion and increase the network performance.

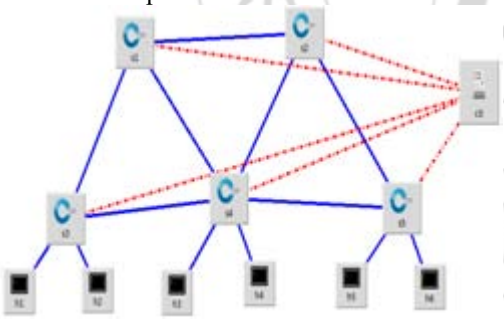

Figure 3: Experiment Tested Topology

A. Discovering Network Topology and Detecting Link Failure

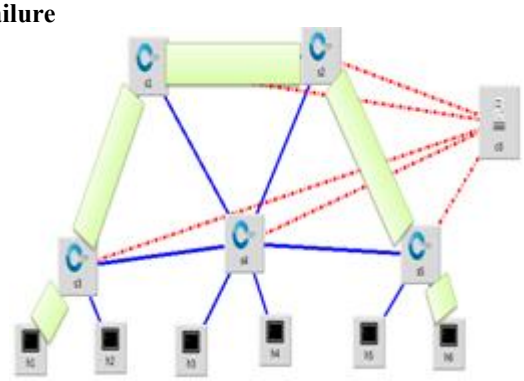

Figure 4: Path Selection for Transmission of Data Packets

The Open flow discovery component uses LLDP messages sent to and received from OpenFlow switches to determine the network topology. It also detects when network links go up or down. This information can be used by other components.

The component used is: **openflow.discovery**

The *Spanning Tree* component is required to eliminate the loops present in the network topology. It works with the OpenFlow Discovery component to establish a view of the network topology and constructs a spanning tree by disabling flooding on switch ports that aren't on the tree. The options***no-flood*** and ***hold-down*** are used to ensure no packets are flooded in the network before the component creates the spanning tree.

The component used is: **openflow.spanning_tree --no-flood --hold-down**

B. Switchasa Firewall and as a Dumb Hub

Switch as a hub forwards traffic out to all of its ports and no code is required to make a switch as a dumb hub. POX controller component is used to make the switch as a dumb hub.

Now to make the switch as a firewall a code is written in python programming language for the controller due to which the switch will block the flow from source to destination and vice versa.

Table 1: Hub Component

```
def make_dumb_hubs(self):
""" A command to convert OpenFlow
switches to dumb hubs"""
msg = of.ofp_flow_mod()
msg.priority = 1
msg.actions.append(of.ofp_action_out
put(port=of.ofp_port_rev_map['OFPP_F
LOOD']))
for switch in self.switches.keys():
connection = self.switches[switch]
connection.send(msg)
```

C. Avoiding LinkFailure/Congestion

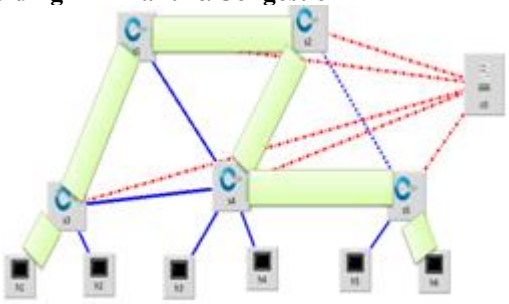

Figure 5: Updating the Transmission Path in case of Link failure/congestion

The network congestion is one of the common problems faced during transmission of data. The network congestion

occurs when a link carries large amount of data which results in deterioration of quality of services. This results in packet loss, queuing delay or the blocking of new connections. So to prevent this problem, an approach is proposed by defining different rules set in the switches. Whenever there is congestion in the network, the controller sends instructions to the switches based on the defined rules to find an alternative path to send the data which prevents network congestion.

The ***Spanning Tree*** component will respond to changes in the network topology. If a link failure is detected, and if an alternate link exists, it can maintain connectivity in a network by creating a new path that enables flooding on the ports connected to the alternate link.

The component used is: **openflow.spanning_tree --no-flood --hold-down**
The ***Host Tracker*** component attempts to keep track of hosts in the network. Host Tracker examines messages received by POX and learns MAC and IP address of hosts in the network. Host Tracker will work in our example but it relies on packets arriving at the controller. Packet forwarding in the network must be done reactively so we need to use a forwarding component like***forwarding.l2_learning***.
The component used is:**host_tracker**
The algorithm is described as follows:

Algorithm: Congestion Avoidance
1. Messages are sent to open flow switches to discover the network topology
2. Spanning tree is constructed by disabling flooding on switch ports that aren't on the tree
3. Open flow switches are made to act like Ethernet switches
4. It learns Ethernet MAC addresses, and matches all fields in the packet header so it may install multiple flows in the network for each pair of MAC addresses.
5. Packet enters the network from host 1 and is delivered to the destination host (h6)
6. Host tracker examines messages received by POX and learns MAC and IP of hosts in the network.
7. Packet dump will display on the log console information about data packets received by POX from switches.
8. If a link is broken or congested and if alternate link exists connectivity is maintained by creating
   a new tree that enables flooding on the ports connected to the alternate link within 45 sec.

4. Performance and Evaluation

To evaluate the open flow controller performance compared to the traditional network various parameters are calculated for the various network build on Miniedit using Mininet. Mininet is a network emulator used to create SDNs scenario in Linux environment. Every network device, hosts, switches and controller are virtualized and communicate via Mininet. A Python script is used to design the topology in Mininet and the trafficflows setup are received from a remote OpenFlow controller. Hence, the test environment applies and performs the actual protocol stacks that communicate with each other virtually. The Mininet environment authorizes the implementation of real protocols in a virtual network.

A. Evaluation Procedure

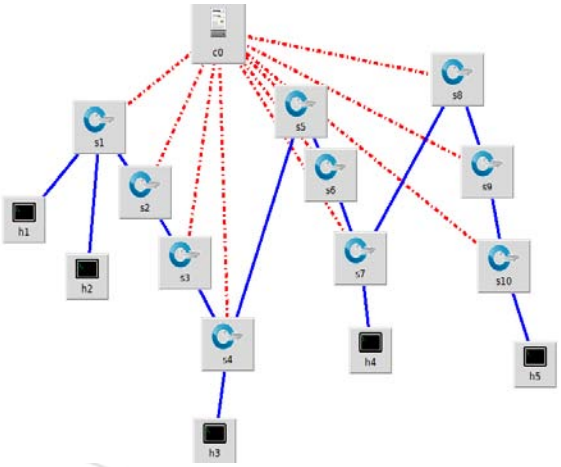

Figure 6: Experimental Tested Topology

To define the experiment, initially it is necessary to specify the hosts and switches that will be used. The OpenFlow controller has the task to define the best path to connect all hosts. To evaluate the network performance in each case (open flow and traditional) it was included into the experimental tested topology shown on figure 4, that creates and sends a large amount of OpenFlow messages to the controller in order to test its performance. The experiment execution results in obtaining the number of OpenFlow messages, the controller can support per second, besides the messages sent by actual switches or virtualized switches in Mininet.

The tests with the Mininet, simulated the presence of 30 switches, in the topology created on Mininet. In each round, 10 switches are used to test the performance and in these tests, the average RTT in milliseconds and bandwidth were calculated.

The graph of average RTT and bandwidth were plotted for different number of switches. Finally, the graph for average RTT for Openflow network with and without congestion detection was plotted.

B. Results

The performance test is shown is the following graphs. They show the performance in terms of various parameters like throughput and average RTT. The graph figure 8 shows the difference between average RTT for Open flow network and traditional network.

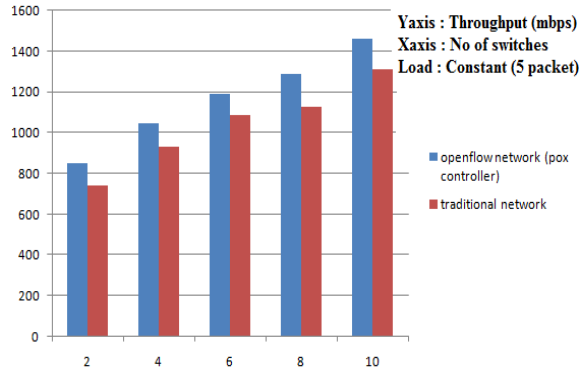

Figure 7: Throughput in POX controller

The above graph shows the throughput in traditional network and Openflow network using POX controller. As shown in the above graph, the throughput is higher in Open flow network than in traditional network.

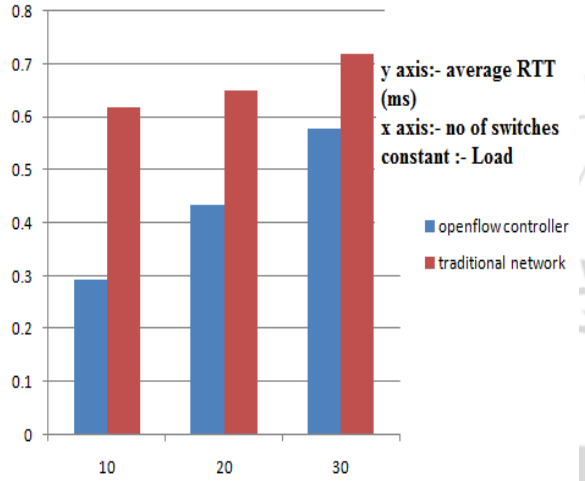

Figure 8: Average RTT in POX controller

The above graph shows the average RTT in milliseconds in traditional network and Openflow network using POX controller. The average RTT in Openflow network in much less than the traditional network showing better performance in Open flow network.

Table 2: Comparison between traditional network parameters and open flow parameters

parameter/ network → ↓	*RTT (before congestion)*	*RTT (after congestion)*
Traditional network	1.141ms	11.103ms
Open flow network	1.003ms	0.987ms

5. Final Result

As shown in the table 2 the performance of traditional network degrades as congestion occurs in the network whereas on the other hand congestion does not affect the performance of open flow network because of the pox controller controlling the open flow network. Hence this concludes the result of my experiment displaying the main difference between traditional network and open flow network.

6. Conclusion and Future Work

Software Defined Networking is a promising paradigm for future network management, and OpenFlow is becoming apparent as a successful industry-supported SDN building block. In this paper, a set of rules for a switch and the process of improving network performance compared to traditional network through a POX controller using mininet are discussed. The set of rules defined in the POX controller reduces the transmission time by about 25% and increases the performance of the network by about 20%. Also, performance of Open flow network increases when the link failure/congestion is detected compared to when it's not detected and how different rules are set for the switches to avoid the network congestion.

As future work, the discussed approach can be implemented in real-time network. In addition, a new approach can be designed to assign priority to the network packets dynamically and the implementation of different functionalities in multiple Openflow controllers.

References

[1] N.McKeown et al.; T. Anderson; H. Balakrishnan; G. Parulkar; L.Peterson; J. Rexford; S. Shenker and J. Turner (2008):"OpenFlow: Enabling Innovation in Campus Networks", ACMSIGCOMM Computer Communication Review, 38(2):69–74.

[2] "Software-Defined Networking: The New Norm for Networks",ONF White Paper, April 13, 2012

[3] Z. Cai et al.; A. Cox and T. Ng (2010): "Maestro: A system for scalable Open Flow control" Technical Report TR10-08, RiceUniversity.

[4] Marcial P Fernandez (2013):"Comparing OpenFlow ControllerParadigms Scalability: Reactive and Proactive" IEEE 27thInternational Conference on Advanced Information Networking andApplications, pp 1009-1016.

[5] Advait Dixit et al., Fang Hao, Sarit Mukherjee, T.V. Lakshman,Ramana Kompella (2013): "Towards an Elastic Distributed SDN Controller" HotSDN'13Hong Kong, China.

[6] David Erickson(2013):"The Beacon OpenFlow Controller", HotSDN'13, Hong Kong, China, pp. 13-18.

[7] N. Gude et al., T. Koponen, J. Pettit, B.Pfaff, M. Casado, N. McKeown, and S. Shenker (2008): "Nox:towards an operatingsystem for networks", ACMSIGCOMM Computer CommunicationReview, 38(3):105–110

[8] Bob Lantz et al., Brandon Heller, and Nick McKeown(2010):"Anetwork in a laptop: rapid prototyping for software-definednetworks", In Proceedings of the Ninth ACM SIGCOMMWorkshop on Hot Topics in Networks.

[9] Adrian Lara et al., Anisha Kolasani, and Byrav Ramamurthy(2014):"Network Innovation using OpenFlow: ASurvey", IEEE Communications Surveys & Tutorials, VOL. 16,No. 1,Pp 493-512

[10] HIDEyuki Shimonishi et al., Yasuhito Takamiya, Yasunobu Chiba, Kazushi Sugyo, Youichi Hatano,Kentaro Sonoda, Kazuya Suzuki, Daisuke

Kotani, and Ippei Akiyoshi(2012):"Programmable Network Using OpenFlow for Network Researches andExperiments", The Sixth International Conference on MobileComputing and Ubiquitous Networking.
[11] Open Networking Research Center (ONRC): http://onrc.net
[12] Pox:http://www.noxrepo.org/pox/about-pox/
[13] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo,Nick McKeown, and Scott Shenker. "Ethane: taking control of theenterprise", SIGCOMM '07: Proceedings of the 2007 conference onApplications, technologies, architectures, and protocols forcomputer communications, pages 1.12, New York, NY, USA, 2007.ACM.
[14] Amin Tootoonchian and Yashar Ganjali."Hyperflow: A distributedcontrol plane for openflow". INM/WREN, 2010.
[15] M. Yu, J. Rexford, M.J. Freedman, and J. Wang. "Scalable flowbased networking with DIFANE", Proc. ACM SIGCOMM, August2010.
[16] Iperf : http://iwl.com/white-papers/iperf