ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 7, July 2014

Towards an Ethernet Learning Switch and Bandwidth Optimization using POX Controller

Abhishek Bagewadi[1], Dr. K N Rama Mohan Babu[2]

M.Tech Student, Department Of ISE, Dayananda Sagar College of Engineering, Bangalore, Karnataka, India [1]

Professor, Deptartment of ISE, Dayananda Sagar College of Engineering, Bangalore, Karnataka, India [2]

Abstract: Software Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable. It promises to dramatically simplify the network management and enable innovation through network programmability. The POX is one of the open source SDN controller and a platform for the rapid development and prototyping of network control software. This paper proposes a set of rules which are defined in the POX controller and based on which the Ethernet switch will take the appropriate decisions during transmission of data packets in the network. The method is proposed to optimize the bandwidth of the network using the iPerf tool. Then the method of assigning priority to the network packets based on different fields in the open flow table is presented. Also described the process of defining different rules set in each switch in the network. The experimental results show that the utilization of bandwidth increases and the average Round-Trip Time (RTT) time decreases for the POX controller.

Keywords: Software Defined Networking, POX Controller, Openflow, Mininet, Ethernet Switch

I. INTRODUCTION

Traditional network architectures are ill-suited to meet the requirements of today's enterprises, carriers, and end users. Due to the broad industry effort spearheaded by the Open Networking Foundation (ONF), Software-Defined Networking (SDN) is transforming networking architecture. In SDN architecture, the control and data planes are decoupled, the network intelligence and state are centralized logically and the underlying network infrastructure is abstracted from the applications. As a result, the enterprises and carriers gain unprecedented network control, programmability and automation which enables them to build highly scalable and flexible networks that readily adapt to changing business requirements.

SDN is currently attracting significant attention from both academia and industry. A group of network operators, service providers, and vendors have recently created the Open Network Foundation, an industrial driven organization, to promote SDN and standardize the Open Flow protocol. On the academic 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 the IETF and IRTF.

The main idea is to allow software developers to rely on network resources in the same easy manner as they do on storage and computing resources. The architecture of SDN is shown in figure 1. In SDN [2], the network intelligence is logically centralized in software-based controllers (at the control plane), and the network devices become simple packet forwarding devices (the data plane) that can be programmed via an open interface.

In addition to abstracting the network, SDN architecture also supports a set of APIs [10] that make it possible to implement common network services including bandwidth management, traffic engineering, routing, multicast, storage optimization, access control, security, quality of service, processor and energy usage, and all forms of the policy management which will help to meet the business objectives.

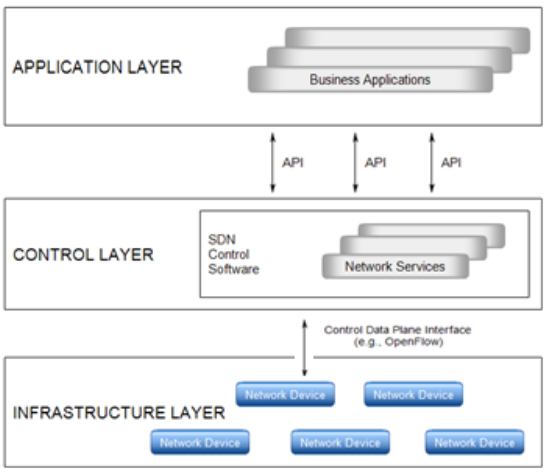

Fig. 1. The architecture of SDN [2]

POX [12] is a open source SDN controller and a platform for the rapid development and prototyping of network control software using Python. It is mainly used for research purposes in the field of SDN. POX can "Run anywhere". It can bundle with install-free PyPy runtime for easy deployment. It specifically targets Linux, Mac OS and windows. The POX contains reusable sample components for path selection, topology discovery, etc. It also supports GUI and visualization tools.

The field of software defined networking is quite recent and it is growing at a very fast pace. Still, there are lot of important research challenges need to be addressed. In this paper we proposes a set of rules which are defined in the

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 7, July 2014

POX controller and based on which the Ethernet switch will take the appropriate decisions during transmission of data packets in the network.

The method is presented to optimize the bandwidth of the network using the iPerf tool. Then the method of assigning priority to the network packets based on different fields in the open flow table is described. Also discussed the process of defining different rules set in each switch in the network.

Open Flow [1] is the first standard communications interface defined between the control and forwarding layers of an SDN architecture. Open Flow allows the direct access and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor based). It is the absence of an open interface to the forwarding plane that has led to the characterization of today's networking devices as monolithic, closed, and mainframe like. Open Flow is one of the standard protocols available and a protocol like Open Flow is required to move the network control out of the networking switches to logically centralized control software [9].

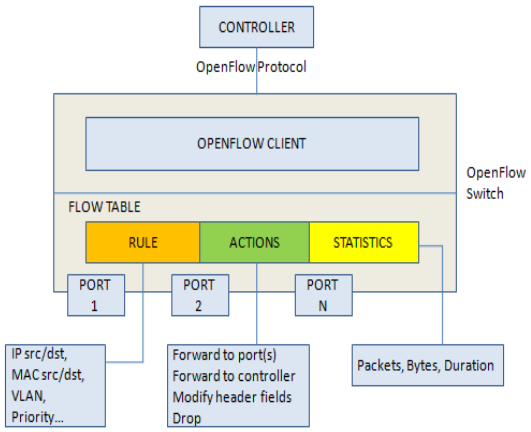

Fig. 2. The Openflow Architecture

The OpenFlow protocol [4] is a key enabler for software-defined networks and currently is the one of the standardized SDN protocol that allows direct manipulation of the forwarding plane of network devices. It was initially applied to Ethernet-based networks and then the OpenFlow switching extended to a much broader set of use cases. OpenFlow SDNs can be deployed on existing networks, both physical nand virtual. Network devices support OpenFlow forwarding and also traditional forwarding, which makes it easy for the enterprises and carriers to progressively introduce OpenFlow-based SDN technologies, even in multivendor network environments.

Mininet is a network emulator. It executes a collection of end-hosts, routers, switches and links on a Linux kernel. It utilizes the lightweight virtualization to form a single system which looks like a complete network, running on the same system, kernel and user code. A Mininet host behaves like a real machine. The ssh protocol is used and it is used to run arbitrary programs (including the network services implemented on the underlying Linux system). The programs execution will send packets through the Ethernet interface, with the specified link speed and delay. Packets get processed by the Ethernet switch, router, or the middle box with the given amount of queuing. When the two programs, like an iperf client and server communicates through the Mininet, the measured performance should match with the two native machines.
In short, Mininet's virtual hosts, switches, links, and controllers are the real thing – they are just created using software rather than hardware – and for the most part their behavior is similar to discrete hardware elements. It is possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network and to run the applications and the binary code on either platform.

The rest of this paper is organized as follows. Section II discuss some of the related work. Section III describes the proposed method. Section IV discuss the evaluation procedure and performance results and section V discuss the conclusion and future work.

II. RELATED WORK

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

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

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

HyperFlow [14] aims at improving the performance of the OpenFlow control plane. However, HyperFlow takes a completely different approach by extending NOX to a distributed control plane. By synchronizing network-wide state among distributed controller machines in the background through a distributed file system, HyperFlow ensures that the processing of a particular flow request is localizable to an individual controller machine. The techniques employed by HyperFlow are orthogonal to the design of the controller and they can also enhance Maestro to become fully distributed to attain even higher overall scalability.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 7, July 2014

DIFANE [15] provides a way to achieve efficient rule based policy enforcement in a network by performing policy rules matching at the switches themselves. DIFANE's network controller installs policy rules in switches and does not need to be involved in matching packets against these rules as in OpenFlow. However, OpenFlow is more flexible since its control logic can realize behaviors that cannot be easily achieved by a set of relatively static policy rules installed in switches. Ultimately, the techniques proposed by DIFANE to offload policy rules matching onto switches and our techniques to increase the performance of the controller are highly complementary: Functionalities that can be realized by DIFANE can be off-loaded to switches, while functionalities that require central controller processing can be handled efficiently by Maestro.

Beacon [6] is a Java-based open source OpenFlow controller. Beacon explored new areas of the OpenFlow controller design space, with a focus on being developer friendly, high performance and having the ability to start and stop existing and new applications at runtime. Beacon showed high performance and was able to scale linearly with multiple processing cores.

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

III. PROPOSED METHOD

In this section, the proposed methods are described to enhance the bandwidth utilization and to decrease the RTT time. First, the algorithm is provided for the Ethernet Switch that results in reduction of RTT time. Then the approach of bandwidth optimization using the iperf tool and the assignment of priority to the data packets is discussed. Finally, the method is discussed for avoiding the network congestion by defining a set of rules for the switches in the network .

A. Ethernet Learning Switch

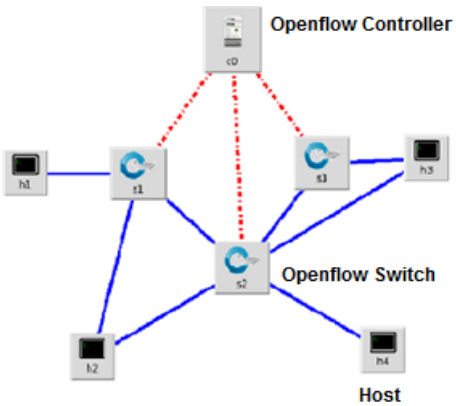

Fig. 3. Experiment testbed topology

The behavior of the OpenFlow switch to an intelligent (learning) Ethernet switch is changed and enhanced. Let us review the operation of a learning switch. When a packet arrives to any port of the learning switch, it can learn that the sending host is located on the port on which the packet has arrived. So, it can simply maintain a lookup table that associates the MAC address of the host with the port on which they are connected to the switch. So the switch stores the source MAC address of the packet, along with the incoming port in its lookup table. Upon receiving a packet, the switch looks up the destination MAC address of the packet and in case of a match, the switch figures out the output port and instead of flooding the packet, it simply sends the packet to its correct destination host (through the looked up port).The algorithm is described as follows :

Algorithm : Ethernet Learning Switch

For each packet from the switch,
(1) Use source address and switch port to update the lookup table.
(2) if Ethertype is LLDP(0X88cc)
(3) Drop the packet //Don't forward the link-local traffic
(4) else if destination address is multicast,
(5) Flood the packets.
(6) else if the output port is same as input port
(7) Drop the packets.
(8) else if lookup table contains port for the destination address,
(9) Send the packet to the destination address.
(10) else if lookup table does not contain destination address port
(11) Flood the packet
(12) else install the flow table entry in the switch.

A. Bandwidth Optimization

Iperf [16] is a commonly used network testing tool that can create Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) data streams and measure the throughput of a network that is carrying them. Iperf allows the user to set various parameters that can be used for testing a network, or for optimizing and tuning the network. The iPerf tool is used to optimize the bandwidth and the formula used for the calculation is :

A=((L+E+I+U)/L)*R

A=Actual link bandwidth in bits/second
L=Value given to iperf's "-l" parameter
E=Size of Ethernet framing
I=IPv4 header size
U=UDP header size
R=bits/second value reported by iPerf

The bandwidth optimization was done using the hierarchical token bucket concept. The Hierarchical Token Bucket (HTB) is a faster replacement for the class-based queueing (CBQ)queuing discipline in Linux.

HTBs help in controlling the use of the outbound bandwidth on a given link. HTB allows the usage of one single physical link to simulate multiple slower links and to send different kinds of traffic on different simulated links. HTB is very useful to limit the client's upload/download rate. Thus, the limited client cannot saturate the total bandwidth.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 7, July 2014

B. Priority assignment of packets

The priority is assigned to the network packets based on the mac address and port. Higher the number, higher will be the priority. The packet is matched against the flow table and the highest priority flow entry that matches the packet is selected. The network topology used in tree topology. All the packets are of IPv4(0x0800). Each switch in the network contains a hash table which contains the mac address and port number. The priority is assigned to the network packets based on destination address and port. Consider the following scenario with reference to figure 3: The network packets are assigned with highest priority that contains the destination address of h3. Let assume that video data is more critical than audio data. The source host say h1, sends audio data to the destination say h4, and meanwhile at the same time, h1 sends the video data to h3. Since h3 has highest priority, the audio data will be sent first to destination h3 and then video data is sent to h4.

Thus depending on the importance of the application, the priority is assigned to the network packets w.r.t particular destination and/or port. This functionality plays a crucial role in transmission of time-sensitive data such as online transactions.

C. Avoidance of network congestion

The network congestion is one of the common problem 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, queueing 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 defined rules contain different parameters like bandwidth, delay, timeout and priority. Depending on the network traffic, different values are assigned which will help to avoid the network congestion. The delay and bandwidth can be increased and the queue size can be decreased. After certain time, if an alternative network path is not found then all the packets will be dropped and the source once again has to send the data. The controller will send instructions to the switch to send the data to the destination. If there is network traffic in one path, an alternative path is selected for transmission of data based on the rules defined in the switches. If there is no such path available then the controller instructs the switch to increase the delay and bandwidth or to drop the packets based on the network traffic. The discussed approach will help in reducing the network congestion in mass call event situation in telephony networks (particularly mobile phones).

IV. PERFORMANCE AND EVALUATION

To evaluate the OpenFlow controller's performance, the scenario was built on a virtualized network using Mininet [8]. In this scenario, a host generates traffic to cross the entire network topology simulating a production network. The experiment was built over VMware Workstation. In the virtualized environment, Mininet was used. Mininet is a network emulator used to create SDNs scenario in Linux environment. The Mininet system permits the specification of a network interconnecting "virtualized" devices. Each network device, hosts, switches and controller are virtualized and communicate via Mininet. A Python script is used to create the topology in Mininet and the traffic flows setup are received from a remote OpenFlow controller. Therefore, the test environment implements and performs the actual protocol stacks that communicate with each other virtually. The Mininet environment allows the execution of real protocols in a virtual network.

A. Evaluation Procedure

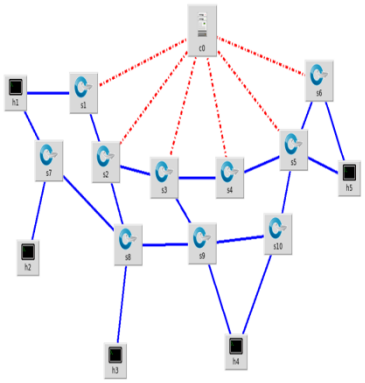

Fig. 4. Experiment testbed setup with 10 switches

To define the experiment, initially it is necessary to specify the hosts and network that will be used. The OpenFlow controller has the responsibility to define the best path to connect all hosts. To evaluate the controller performance it was included into the experimental testbed 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 60 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. Finally, the graph of average RTT and bandwidth were plotted for different number of switches.

In the OpenFlow network, when a host tries to send data packets to the destination address, and the switch has no such address in its flow table, the switch makes a query to the OpenFlow controller [5]. The controller will decide what action would be applied to this flow, e.g., choose the path from source to the destination to forward the packets to destination host.

To optimize the memory usage in the switches, a timeout (in our experiment 10 seconds) sets the removal of this entry on its flow table, forcing the path to rebuild whenever it is necessary.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 7, July 2014

B. Results

The performance test result is shown in the following Graphs (figure 5 and figure 6). They show the performance in terms of bandwidth utilization and average RTT using the POX controller in mininet environment.

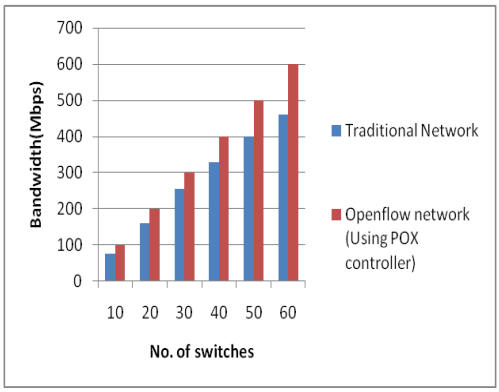

Fig. 5. Bandwidth utilization in POX controller

The above graph shows the bandwidth utilization in traditional network and openflow network using POX controller. As shown in the above graph, the bandwidth is efficiently utilized in openflow network than in traditional network. The average bandwidth utilization is increased by about 18%. As the number of switches in the network increases, the bandwidth in openflow network varies largely.

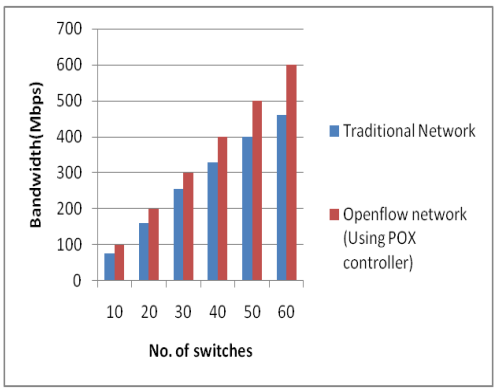

Fig. 6. Average RTT in ms using POX controller

The above graph shows the average RTT in milliseconds in traditional network and openflow network using POX controller. As shown in the above graph, the RTT is reduced in openflow network compared to the traditional network. The average RTT is reduced by about 23%. As the number of switches in the network increases, the RTT value in openflow network varies largely. The RTT is one of the important parameter that decides the transmission rate of data. Lower the RTT value, faster will be the transmission rate.

The above two figures, Figure 5 and Figure 6 shows the clear indication of effective bandwidth utilization and resource utilization during transmission of data packets. The effectivity of bandwidth while transmitting over the network results in better usage of CPU, heap memory and disk capacity utilization.

V. CONCLUSION AND FUTURE WORK

Software Defined Networking is a promising paradigm for future network management, and OpenFlow is emerging as a successful industry-supported SDN building block. In this paper, a set of rules for an Ethernet switch and the process of optimization of bandwidth in a POX controller using mininet are discussed. The set of rules defined in the POX controller reduces the transmission time by about 23% and increases the performance of the network by about 18%. Also, described about assigning the priority to the network packets 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 openflow control" Technical Report TR10-08, Rice University.

[4] Marcial P Fernandez (2013): "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive" IEEE 27th International Conference on Advanced Information Networking and Applications, 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 operating system for networks", ACMSIGCOMM Computer Communication Review, 38(3):105–110

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

[9] Adrian Lara et al., Anisha Kolasani, and Byrav Ramamurthy(2014):"Network Innovation using OpenFlow: A Survey", 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 and Experiments", The Sixth International Conference on Mobile Computing 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 the enterprise", SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, pages 1.12, New York, NY, USA, 2007. ACM.

[14] Amin Tootoonchian and Yashar Ganjali."Hyperflow: A distributed control plane for openflow". INM/WREN, 2010.

[15] M. Yu, J. Rexford, M.J. Freedman, and J. Wang. "Scalable flow-based networking with DIFANE", Proc. ACM SIGCOMM, August 2010.

[16] Iperf : http://iwl.com/white-papers/iperf