Analysis of Software Defined Network Switch

Rajvardhan Somraj Deshmukh
Spire Id: 30384584
ECE Department,
University Of Massachusetts Amherst

***Abstract*—Software defined networks is an emerging field in communication networks and has become a promising area for research and intelligent implementation of Controller logic. Switch is the basic element in communication and analyzing its behavior for various conditions helps us use appropriate approaches necessary for smooth communication.**

Switch is a basic component of this topology and its memory management is the topic in this project. We analyze switch in Mininet, and observe flows created for various traffic like UDP and TCP generated by iperf. We implement different hard timeouts and compare the flows programmed. We use the emulation results show that hard timeout at T=8 is the most optimum.

Keywords- SDN,TCP,UDP

I. INTRODUCTION

Software-defined networking (SDN) is the new developing approach to computer networks that allows us to manage the network services to fine granular level of functionality. We have separate data plane and control plane. The switch talks to the controller via the control. The controller is the main brain and calculates the rules. The data plane uses the same concept of FIB (Forwarding Information base) and routes the packets according to the flows it has in its table. As fine granularity rules can be set up in SDN, correspondingly the flows stored in the switch table also have a lot of match parameters and this can result in sluggish performance.

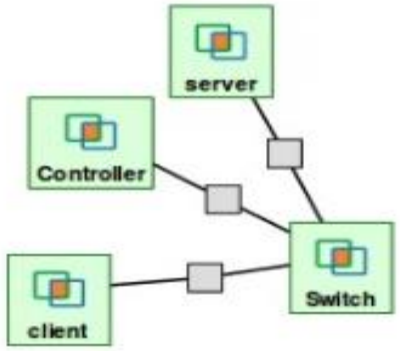

Fig 1. Topology

The packet received by the switch is checked with the current flows in the table, and if the flows do not match with the packet details then this packet is sent to the controller. The controller creates a rule for this packet and sends the flow along with the packet to the switch. This flow is programmed in the switch and appropriate action is taken for the packet. To make sure that the switch does not collapse due to data flow the Controller can set idle timeout (certain time after which the unused flow is removed) or/and hard timeout (fixed interval after which a packet is removed). [7] Deals with the analysis of packet arrival as Possion and Exponential random variable and derives E[x] expectation /average and Blocking probability B for different hard timeouts. This project focusses on analyzing different timeouts to find the optimum hard timeout. The flow statistics for different hard timeouts are compared and the results displayed.
The emulation is run with the topology in Fig 1 for 4 timeouts for a full cycle. The corresponding timeouts are T=3, 8, 10 and 15.

II. CONTROLLER LOGIC

We have used the pox controller. It maintains a table which maps MAC addresses to the port from which they would be accessible. Whenever a packet with MAC address that is not present in the table (or multicast) arrives, its source MAC address is stored with the port number it came from, and the packet is flooded across all ports other than the one it came from.
Further when a packet with the destination address that is present in the switch flow table arrives, the switch checks if the corresponding port is the same from which it came, if yes then it drops the packet, else it sends it out the corresponding port.

III. EMULATION SYSTEM DESIGN

The parameters considered for the experiment are as follows:

Table 3.1 Emulation Parameters

Network Emulator	Mininet
Traffic Generator	iperf
Traffic Type	TCP,UDP
Simulation Time	51s
Number of Nodes	4
Number of Controller	1
Controller used	pox
Number of switch	1

Number of nodes connected to Switch	2

SYSTEM DESCRIPTION: The simulation time is 51s.
The observed statistics at the server and the client during full run is described below.

Table 3.2 Traffic Statistics for Timeouts (T)

1.T=3
1.1 iperf tcp:(fpersec y<3:y=0)
TCP window size:85.3 KBytes
Time 51sec
Transfer 13GB
Bandwidth 2.24 Gbits/sec
1.2 iperf udp:(fpersec y<=1:y=0)
UDP buffer size 208 KBytes
Time 51 Sec
Transfer 8.33 Mbytes
Bandwidth 1.05 Mbits/sec
Jitter 0.117 ms
Lost/Datagrams 30/5940
30 datagrams recieved out of order

2. T=10
2.1 iperf tcp:(fpersec y<3:y=0)
TCP window size:85.3 Kbytes
Time 51sec
Transfer 42GB
Bandwidth 4.68 Gbits/sec
2.2 iperf udp:(fpersec y<=1:y=0)
UDP buffer size 208 Kbytes
Time 51 Sec
Transfer 8.27 Mbytes
Bandwidth 1.05 Mbits/sec
Jitter 0.031 ms
Lost/Datagrams 27/5902
27 datagrams recieved out of order

3. T=8
3.1 iperf tcp:(fpersec y<3:y=0)
TCP window size:85.3 Kbytes
Time 51sec
Transfer 31.6GB
Bandwidth 3.36 Gbits/sec
3.2 iperf udp:(fpersec y<=1:y=0)
UDP buffer size 208 Kbytes
Time 51 Sec
Transfer 7.92 Mbytes
Bandwidth 1.05 Mbits/sec
Jitter 0.138 ms
Lost/Datagrams 12/5650
12 datagrams recieved out of order

4. T=15
4.1 iperf tcp:(fpersec y<3:y=0)
TCP window size:85.3 Kbytes
Time 51sec
Transfer 46.9 GB
Bandwidth 5.43 Gbits/sec
4.2 iperf udp:(fpersec y<=1:y=0)
UDP buffer size 208 KBytes
Time 51 Sec
Transfer 8.24 MBytes
Bandwidth 1.05 Mbits/sec
Jitter 0.021 ms
Lost/Datagrams 4/5881
4 datagrams recieved out of order

The flows programmed by the controller into the switch, as observed by the switch have been displayed in the following figures 3.1 and 3.2. The flow dump of the switch is taken every second and difference of flows observed is stored and displayed.

Nomenclature:
tcpt3 = timeout T=3 sec for TCP traffic
tcpt8 = timeout T=8 sec for TCP traffic
tcpt10 = timeout T=10 sec for TCP traffic
tcpt15 = timeout T=15 sec for TCP traffic
udpt3 = timeout T=3 sec for UDP traffic
udpt8 = timeout T=8 sec for UDP traffic
udpt10 = timeout T=10 sec for UDP traffic
udpt15 = timeout T=15 sec for UDP traffic

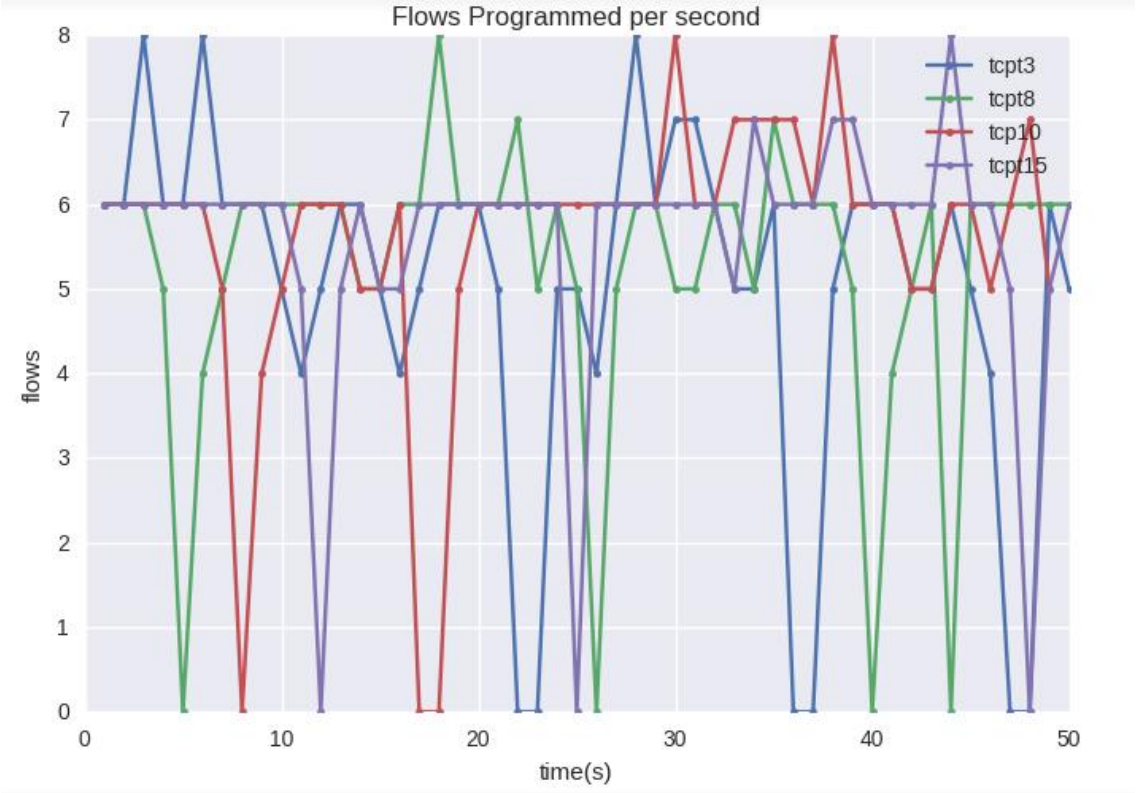

Fig 3.1 Flows programmed per second in mininet topology for TCP traffic

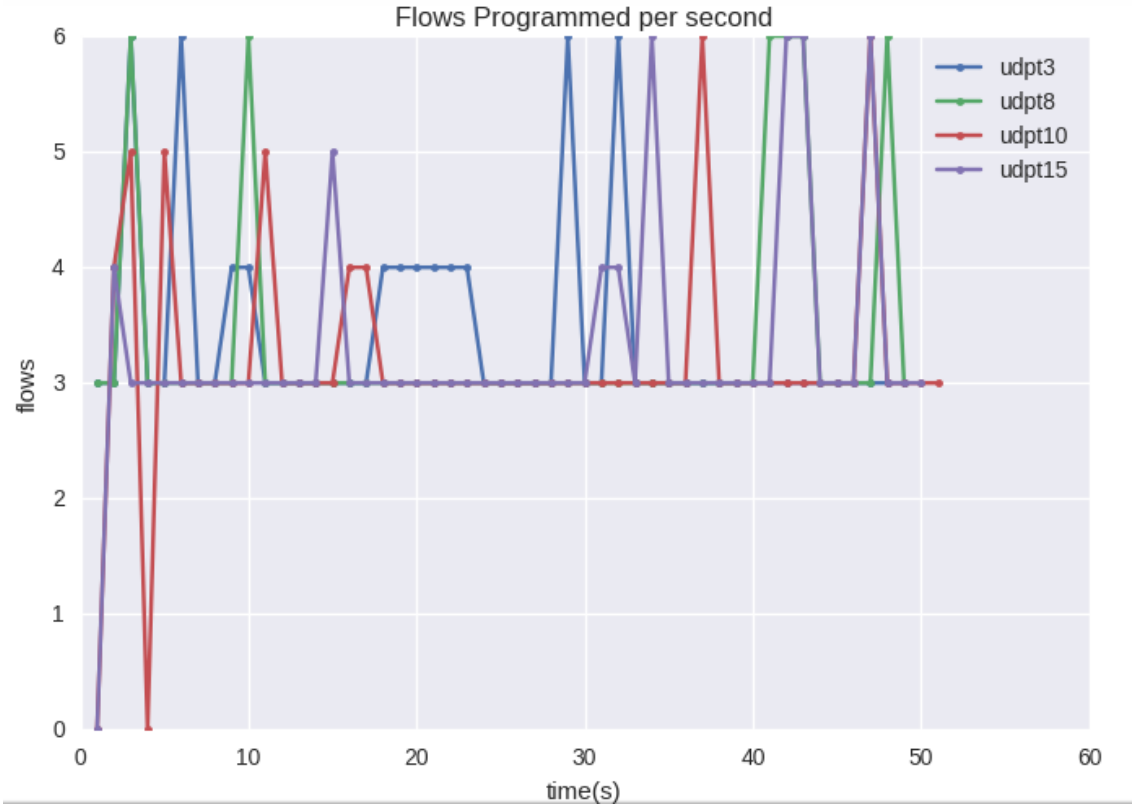

Fig 3.2 Flows programmed per second in mininet topology for UDP traffic

IV. SIMULATION RESULTS

We evaluate the simulation results Hard timeouts= 3, 8, 10 and 15 seconds. For performance evaluation, we use 3 metrics: (FNFPS) Frequency of no (zero) flows programmed per second, (AFPS) Average Flows programmed per second and (SFP) Sum of flows programmed during the full run.
Corresponding statistics are displayed. These are calculated as:

FNFPS = Count the number of times when flow=0

$$AFPS = \frac{\text{Total number of flows}}{\text{Total time of the experiment}}$$

SFP = Σ (Flows per second)

FNFPS: Is the same for all types of traffic (TCP and UDP) across all timeouts (T=3, 8, 10 and 15).

AFPS: 1. For TCP traffic
1.1 AFPS is high (above 5.6 flows/second) for timeout T=10 and 15.
1.2 AFPS high is relatively low for T=3.
1.3 For T=8 AFPS is towards lower side, approx. 0.08 less than the median (mid-point between highest and lowest value).
2. For UDP traffic
2.1 AFPS is high (approx. 3.4 flows/second) for timeout T=3.
2.2 AFPS Is low (approx. 3.16 to 3.27 flows/second) for T=10 and 15.
2.3 For T=8 AFPS is towards higher side but is still 0.1825 close to the median (mid-point between highest and lowest value).

SFP: 1. For TCP traffic
1.1 Least for T=3 (230 flows programmed)
1.2 SFP for T=10 deviated by 10 flows more in the almost linear graph. Tends towards the SFP for T=15.
1.3 1 flow less than the median for T=8.
2. For UDP traffic
2.1 Most for T=3 (167 flows programmed)
2.2 SFP for T=10 deviated by 10 flows more in the almost linear graph. Tends towards the SFP for T=15.
2.3 1 flow less than the median for T=8.

FNFPS is the same for all timeouts in both UDP and TCP traffic so we should look at the other parameters. In both UDP and TCP we get approx. median AFPS and SFP for timeout T=8. This observation is consistent with the observations of AHTM [7].

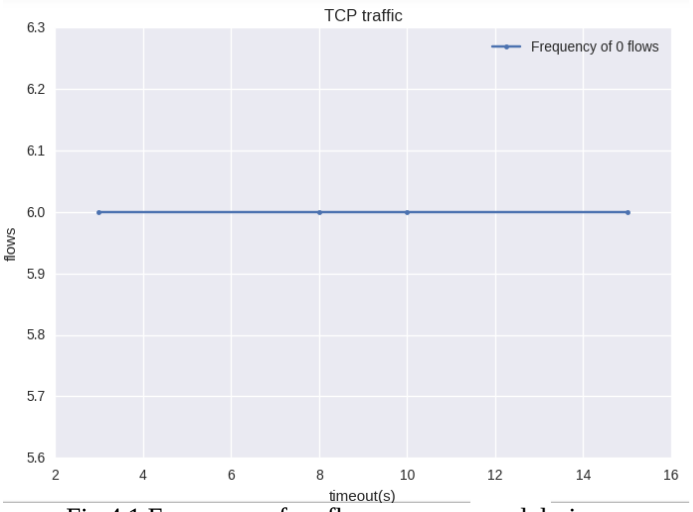

Fig 4.1 Frequency of no flows programmed during TCP traffic full run

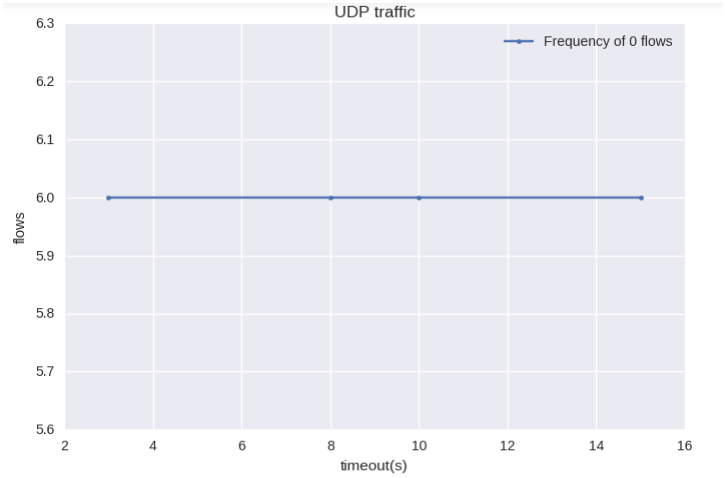

Fig 4.2 Frequency of no flows programmed during UDP traffic full run

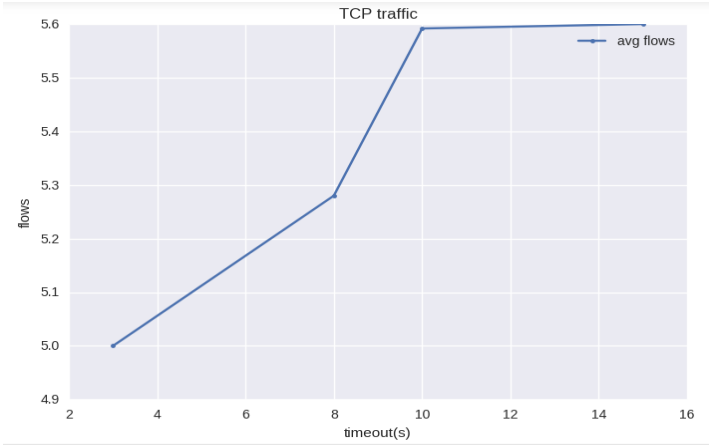

Fig 4.3 Average flows programmed during TCP traffic full run

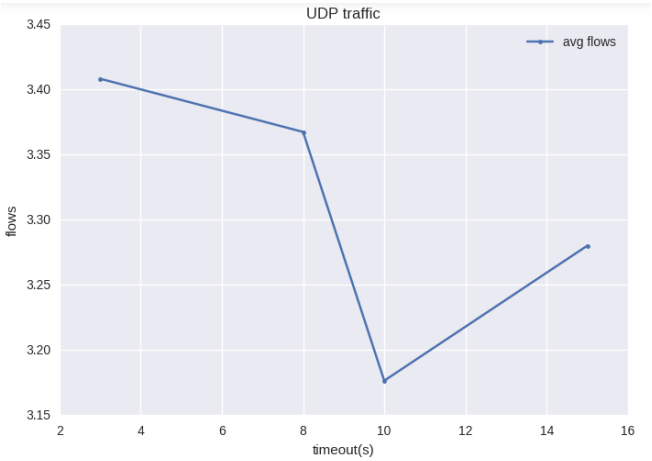

Fig. 4.4 Average flows programmed during UDP traffic full run

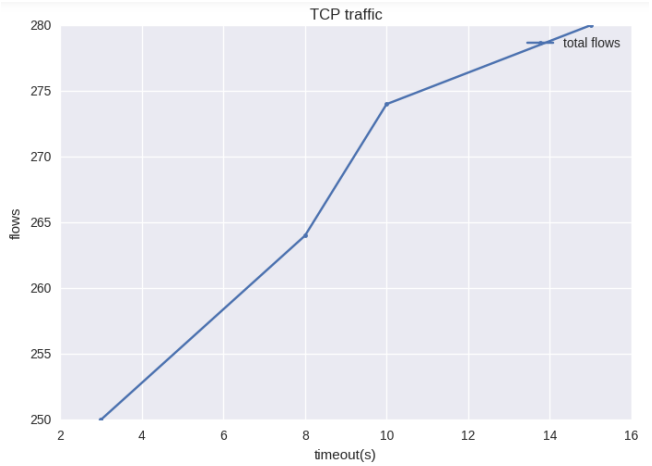

Fig. 4.5 Total flows programmed during TCP traffic full run

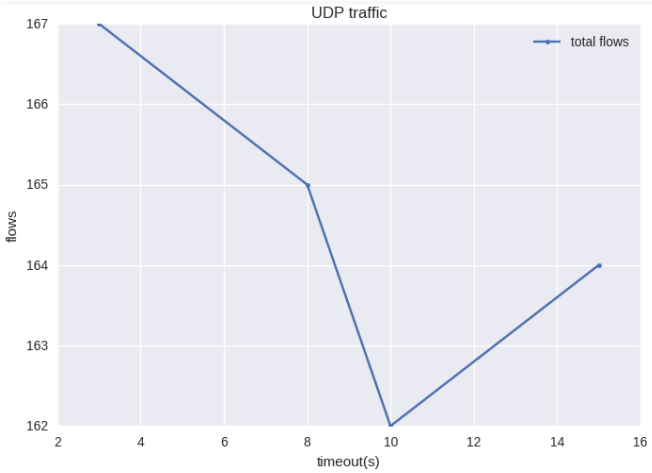

Fig. 4.6 Total flows programmed during UDP traffic full run

V. CONCLUSION

We compare different timeouts, after implementing them in the model depicted in Figure 1 [4], we pass iperf udp, and tcp traffic and establish that T=8 among the timeouts proves to be the best in terms of balance between higher flow retention and lower rate of packet programmed per second. The communication is faster and more efficient as per table 2. This observation is consistent with the observations of [7]. We used mininet for emulation, Openflow dump commands and customized bash scripts to sniff packets in order to carry out the research. The analysis of all the parameters was carried out on the controller and switch and the results were viewed using Openflow commands, customized scripts and python matplot.lib. These observations help us analyze and compare the outputs. The observations proved that when we consider udp or tcp the best performance in the scenario is demonstrated by T=8 hard timeout.
This timeout is the best out of all the timeouts considered, which minimizes the resource consumption of the device and provides satisfactory results.

VI. ACKNOWLEDGEMENT

We would like to thank Prof. Mike Zink and Divyashri Bhat for their instrumental comments and suggestions got this project.

VII. REFERENCES

[1] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for highperformance networks," in ACM SIGCOMM Computer Communication Review, vol. 41, no. 4. ACM, 2011, pp. 254-2651

[2] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," IEEE Commun. Mag., vol. 51, no. 2, pp. 136-141, 2013pp.108,113, 12-14 Nov. 2012

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, 2008

[4] K. Kannan and S. Banerjee, "Flowmaster: Early eviction of dead flow on sdn switches," in Distributed Computing and Networking. Springer, 2014, pp. 484-498

[5] B. Ryu, D. Cheney, and H.-W. Braun, "Internet flow characterization: Adaptive timeout strategy and statistical

modeling," in in Proc. Passive and Active Measurement workshop. Citeseer, 2001

[6] Linlian Zhang, Sheng Wang, Shizhong Xu, Rongping Lin, Hongfang Yu, "TimeoutX: An Adaptive Flow Table Management Method in Software Defined Networks", *Global Communications Conference (GLOBECOM) 2015 IEEE*, pp. 1-6, 2015.

[7] L. Zhang, R. Lin, S. Xu, S. Wang, "AHTM: Achieving efficient flow table utilization in software defined networks", *Global Communications Conference (GLOBECOM) 2014 IEEE. IEEE*, pp. 1897-1902, 2014

[8] https://github.com/TUDelftNAS/SDN-OpenNetMon.

[9]https://openflow.stanford.edu/display/ONL/POX+Wiki