Experimenting With Scalability of Beacon Controller in Software Defined Network

Tony Manuel, Bhargavi H Goswami

Abstract*: *In traditional network, a developer cannot develop software programs to control the behavior of the network switches due to closed vendor specific configuration scripts. In order to bring out innovations and to make the switches programmable a new network architecture must be developed. This led to a new concept of Software Defined Networking(SDN). In Software defined networking architecture, the control plane is detached from the data plane of a switch. The controller is implemented using the control plane which takes the heavy lift of all the requests of the network. Few of the controllers used in SDN are Floodlight, Ryu, Beacon, Open Daylight etc. In this paper, authors are evaluating the performance of Beacon controller using scalability parameter on network emulation tool Mininet and IPERF. The experiments are performed on multiple scenarios of topology size range from 50 to 1000 nodes and further analyzing the controller performance.

Index Terms*: *SDN, Beacon, Mininet, Controller.

I. INTRODUCTION

The network is expanding day by day and the current devices that are used for networking are becoming incapable of managing the flow of data from source to destination in an optimal and simplified way. The low level language programmed devices which are close to vendor specific configuration not only causes limitations for innovation and implementations of new ideas for improved switching, but also does not provide mechanism for automatically responding to wide ranges of events that may occur in the network [1]. In order to respond to certain event in the network, the network operator must have to manually make the necessary changes in the network configuration by adding ad hoc scripts. The manual configuration of device frequently leads to misconfiguration of other events. To overcome the problem of manual configuration of networking devices the researchers came up with a new model in networking-Software defined networking (SDN) [2]. Section II explains SDN in brief. Section III of this paper provides the details about Beacon controller. Section IV deals with the installation and loading the controller, Section V describes the experimental scenarios, Section VI with performance analysis followed by conclusion and references..

II. SOFTWARE DEFINED NETWORKING

SDN provides platform for innovative techniques [3]. It changed the approach of managing and designing the networks [1], [3]. The characteristics of SDN are 1) Control layer is separated from the data layer. 2) A control software is implemented on the centralized controller device. The task of control plane is to decide how to manage the incoming requests in the network. The data plane then forwards the request as per the logic received from the control plane. With the help of SDN an operator can program the network according to the situation instead of configuring the vendor specific devices. The centralized control plane allows network operators to have a control over the entire networks. The separated data and control plane from the traditional device removes the barrier of vendor specific low level configuration and thus allows a network operator to program the device in high level languages which will be easier to change and modify as and when required. The control plane provides complete control over data plane devices like switches and routers in the network.

A. SDN Architecture

Figure 1, describes the SDN architecture. The architecture is divided into three parts. 1).Infrastructure. 2).Controller. 3). Application. Infrastructure consists of networking devices like Router, switches, and Wireless access point. These parts form a data plane of the network. The controller part consists of different programs that determine the data flow path among the network. Applications are services that an operator uses to determine various statuses like Traffic and security monitoring,

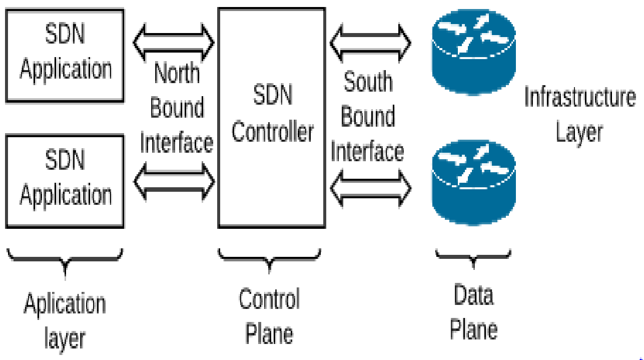


Fig. 1. Architecture of SDN

access control, energy-efficient networking etc. The centralized controller is linked with the data plane and the application. The link between controller and data plan is accomplished using an API generally referred to as Southbound API. Controller and application is linked using Northbound API. Southbound and northbound api's are important objectives in software defined networking [4]. All the dynamic changes made by the controller according to the needs and demands in the network is accomplished with the help of Southbound API.

Revised Manuscript Received on January 25, 2019.

Tony Manuel, Computer Science, Christ (Deemed To Be University), Bangalore, India.

Dr.Bhargavi H Goswami, Computer Science, Christ (Deemed To Be University), Bangalore, India.

Open flow is one of the most common southbound interfaces between data plane and controller used in SDN. It is an industry standard that characterizes the way the SDN Controller ought to communicate with the forwarding plane to make acclimations to the dynamic network, so it can better adjust to changing business prerequisites [5]. Northbound API consists of frameworks like Procera, Frenetic to communicate to the top layer of the controller [6]. To communicate business logic, policies enforced over top layers at application layer to the controllers, Northbound APIs are used by Software Defined Networks. It is the critical part where the automation configuration takes place. The scalability of a network is studied based on performance metrics [7].

III. BEACON CONTROLLER

The development of the Beacon started in early 2010. It is an open source SDN controller which is developed using java. Same as other SDN controllers [8], [16], [17], [18], Beacon is also generating traffic OpenFlow Packets and manages incoming and outgoing packets. The controller can run on several platforms, ranging from lofty dual-core Linux servers to smart phones. The cross platform support is an added advantage of the controller.

The controller can control a 100 vswitch, 20 physical switch provisional data center [9]. Code packages in Beacon can be begun/halted/invigorated/introduced at runtime, without hindering other non-dependent groups (ie supplant your running Learning Switch application without disengaging switches). The architecture of Beacon controller is shown in figure 2.

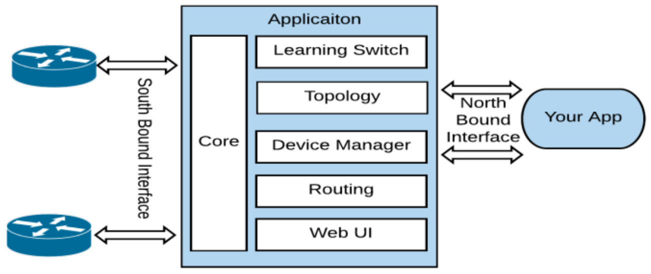


Fig. 2. Architecture of Beacon Controller

The architecture of the controller contains applications like, Learning switch, topology, device manager, routing, Web UI. The functionalities of each application are as follows.

Learning switch: This application helps the controller to communicate with the openflow switches during the packet-in and packet-out process.

Topology: When the network is up and running this application helps the controller to discover the existing links in the network.

Routing: When the packet arrives the controller the routing application helps the controller to enforce the routing details and policies about the packet to the flow table of the openflow switches.

Web UI: The controller allows the network administrators to gain access to Beacon controller using a simple and easy to use web gui.

Your App: The Beacon controller can work with the custom application of users with the help of northbound api. User can implement their own custom applications in the controller when they need to make any changes in the behavior of the network.

Core: It is the central part of the controller. All the openflow switches in the SDN network is connected in this layer.

Device manager: This application helps to learn the devices in the network and figure out the source and destination of the packets.

IV. INSTALLATION

The installation of the tools and the experimental scenarios discussed in this paper are carried out on a virtual machine running UBUNTU operating system. The configuration of the system is shown in table I.

TABLE I
System Configuration Of Virtual Machine

Tools	Configuration
Operating System	Ubuntu 14.04 LTS
Processor	Intel Core I3
Memory	8GB
Virtual Machine	VM Ware 12
Mininet	2.2.0
IPERF	Version 3

A. Installation and configuration of prerequisite

Beacon is a java based application. In order to run the controller, a jdk of version 6 or 7 must be present in the computer. Any version above 7 is not supported by the controller. The following steps will help the readers to install jdk 6 on his / her system.

Step 1: Type “sudo apt-get install openjdk-6-jdk” on the terminal.

Step 2: Once the installation is complete the researcher must make jdk 6 as the default jdk by typing the command “sudo update-alternatives --config java” on the terminal. This will display all the jdk version installed on the system.

Step 3: Choose jdk 6 by entering its corresponding serial number.

B. Installation of Beacon Controller

Once the researcher has verified the jdk version the Beacon controller can be installed by the following approach.

Step 1: For installing beacon you must download its binary package file from the mentioned URL. "https://openflow.stanford.edu/display/Beacon/Releases".

Step 2: Scroll down to ‘Binary Packages’ section and choose the appropriate version of your Operating System. Once you have selected your version, download will start automatically.

Step 3: Save the file in Desktop or to any other directory. In this paper the author has used desktop as installation directory.

Step 4: The downloaded file is an archive type and must be extracted before use. Simply follow the below procedure for extracting the contents of the file. Right click on the file and select extract here.

Step 5: Rename the extracted folder to ‘Beacon’.

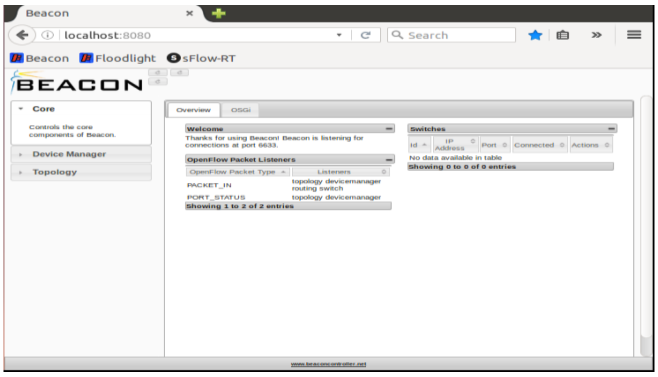
C. Loading the controller

This section deals with the steps that will help the researcher to load the Beacon controller.

Step 1: Open a new terminal by typing : “Ctrl + Alt + t”.

Step 2: Change the present working directory to Beacon by typing the command: “cd Desktop/Beacon” from the terminal.

Step 3: Once the path is changed to “Beacon” type the command: “./beacon” to start the controller. The terminal will display "controller listening on *:6633" once the controller has started successfully. The GUI of the controller can be accessed from a web browser by entering http://localhost:8080”. If beacon controller is active and running, the browser will display the GUI of Beacon Controller as shown in figure 3.



Controls the core components of Beacon.

Thanks for using Beacon! Beacon is listening for connections at port 6633.

OpenFlow Packet Type	Listeners
PACKET_IN	topology devicemanager routing switch
PORT_STATUS	topology devicemanager

Showing 1 to 2 of 2 entries

Id	IP Address	Port	Connected	Actions

No data available in table

Showing 0 to 0 of 0 entries

Fig. 3. Beacon Web GUI

V. EXPERIMENTAL SCENARIO

Scalability of Beacon controller is experimented in this section. For emulating the network environment, mininet network emulator is used [10]. In this experiment authors start the performance evaluation with a topology consisting of 6 switches and 50 hosts as shown in figure 4. The switch used in this experiment is OpenFlow kernel switch [11].

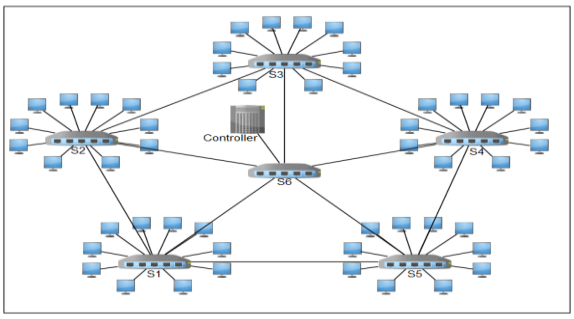


Fig. 4. Experimental Mesh Topology

10 hosts are added to each of 5 switches ranging from S1 to S5 and each switch is linked to s6 switch in a star topology manner. The central switch S6 is connected to Beacon Controller.

The topology is designed using python language [12]. Figure 5 depicts the python script used by the authors for the experimental scenario. The authors of this research paper have followed the below steps for executing the python script and testing the throughput.

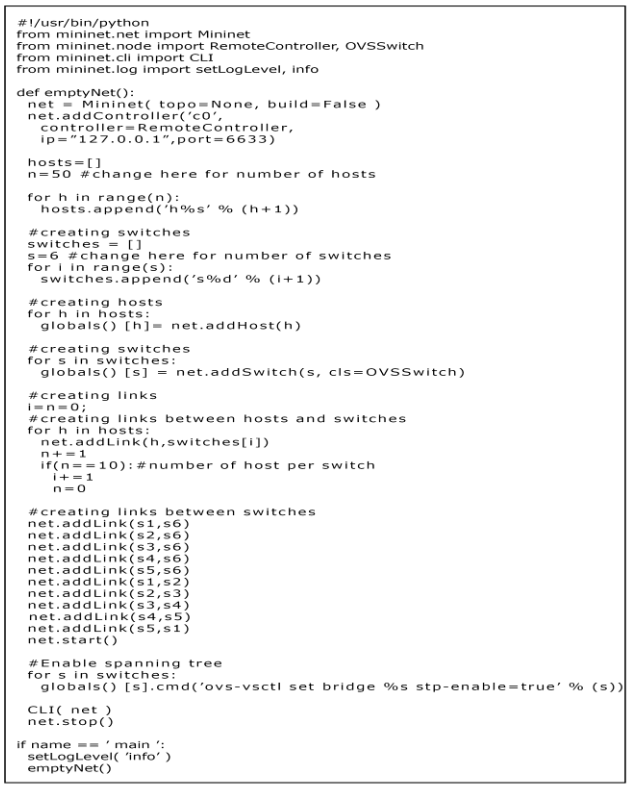
Step 1: Execute the python script by typing “sudo ./topo.py~” in the terminal. Here “topo.py” refers to the name of the python script file and the file resides in the home directory.

Step 2: Once the topology is up and running, a client-server connection is established between the first host and the last host [13]. In order to establish the client-server connection, the consoles of the first host and the last host must be accessed from mininet by typing the command "xterm h1 h50". Here h1 refers to host number 1 and h50 refers to host number 50 as per scenario 1 shown in table II.

Step 3: Configure host 1 as the server by typing the command "iperf3 -s -p 5566 -i 1 > result.txt" on the console of h1 which was accessed on step 1. The parameters of the command is interpreted as (-s) server listening on (-p) port 5566 with an (-i) interval of 1 second and finally the output is (>) redirected to a file name called "result.txt" [14].

Step 4: Configure host 50 (for scenario 1) as the client by typing the command "iperf3 -c 10.0.0.1 -p 5566 -t 150" on the console of h50 which was accessed on step 1. The parameters of the command is interpreted as a client(-c) connected to a server whose ip is 10.0.0.1 and the port(-p) 5566 then tcp packets(-t) are send to the server for a duration of 150 seconds.

Step 5: Once the packet transfer is completed terminate the client server connection by pressing "ctrl+c" on the console of h1.


```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def emptyNet():
  net = Mininet( topo=None, build=False )
  net.addController('c0',
    controller=RemoteController,
    ip="127.0.0.1",port=6633)

  hosts=[]
  n=50 #change here for number of hosts

  for h in range(n):
    hosts.append('h%s' % (h+1))

  #creating switches
  switches = []
  s=6 #change here for number of switches
  for i in range(s):
    switches.append('s%d' % (i+1))

  #creating hosts
  for h in hosts:
    globals() [h]= net.addHost(h)

  #creating switches
  for s in switches:
    globals() [s] = net.addSwitch(s, cls=OVSSwitch)

  #creating links
  i=n=0;
  #creating links between hosts and switches
  for h in hosts:
    net.addLink(h,switches[i])
    n+=1
    if(n==10):#number of host per switch
      i+=1
      n=0

  #creating links between switches
  net.addLink(s1,s6)
  net.addLink(s2,s6)
  net.addLink(s3,s6)
  net.addLink(s4,s6)
  net.addLink(s5,s6)
  net.addLink(s1,s2)
  net.addLink(s2,s3)
  net.addLink(s3,s4)
  net.addLink(s4,s5)
  net.addLink(s5,s1)
  net.start()

  #Enable spanning tree
  for s in switches:
    globals() [s].cmd('ovs-vsctl set bridge %s stp-enable=true' % (s))

  CLI( net )
  net.stop()

if name == ' main ':
  setLogLevel( 'info' )
  emptyNet()
```

Fig. 5. Python Script

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

Step 6: Type "more results.txt" on the console of h1 to display the data present inside the result.txt file. The file will have details of Id, interval, transfer and bandwidth.

Step 7: Extract the time and the transfer rate from the result file by typing the command

"cat result.txt | grep sec | head -150 | tr - ' ' | awk '\{print \$3,\$5\}' >newresult.txt".

Step 8: Once the values are filtered and saved to the "newresult.txt" gnuplot tool is used to generate graph for the values that are present inside the file [15]. Open a new terminal and type "gnuplot".

Step 9: Type plot "newresult.txt" title "Tcp Flow - 50 Nodes" with lines; set xlabel "Time (sec)";set ylabel "throughput (GBps)";set xrange[0:150];set xtics 10;set yrange[0:5];set ytics .05;set terminal png;set output 'output.png';replot;

Here "newresult.txt" is the name of the filtered file. The type of graph that we want is with lines and title is set to "Tcp Flow - 50 Nodes". The label for X and Y axis is "Time (sec)" and "throughput (GBps)" respectively. X axis range is till 150 with an interval of 10 and Y axis range is till 5 with an interval of 0.5. The generated graph is saved in png format by the name "output.png" .

The above process is repeated for each scenario that is shown in table II by changing the last host in step number 2 to 200,400,600,800 and 1000 respectively. The script is also modified respectively as indicated in comments.

TABLE II
Experimental Scenarios

Scenario No	Nodes Per Switch	Total Hosts
1	10	50
2	40	200
3	80	400
4	120	600
5	160	800
6	200	1000

VI. Performance Analysis

This section analyzes the results obtained in the form of graph after logging the events of each experimental run on 6 different scenarios with difference in number of nodes. In this paper, authors have made attempt to obtain and test the characteristic of Beacon controller with respect to the scalability in diversified scenario. To set up the mesh topology, issue of loops was the most complex situation, while experimenting with large scale networking topology where, the authors were stuck for the long duration. It was observed during the experimental setup that mesh network was getting trapped in infinite loops which was later resolved using spanning tree protocol implementation by generating loop free topological scenario. Once the experiment was set up and the topology was up running, diversification of nodes with a large scale was introduced incrementally. The first scenario obtained the results is shown in fig 6. It was observed that there was no packet drop and the experiment was running for continuously 150 seconds by generating live traffic on the simulation network. It was found to be stable and constant, comfortably managing the load on the network. Initially for 10 RTT, the obtained throughput was increasing, but, later on, it became stable with the average of 1.81346 Gbps.

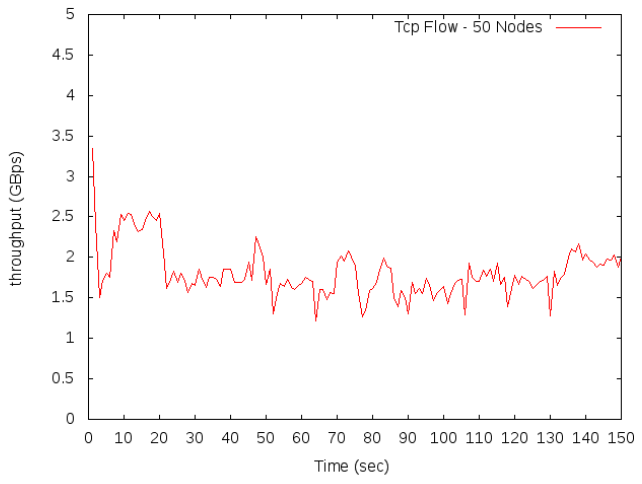


Fig. 6. Throughput of scenario 1

Further authors started testing the behavior of the network for 200 nodes with the same networking scenario to obtain the transition and observe the success and fail ratio of the packets delivered. While observing the transmission under scenario 2 for 150 seconds, it was observed that only two events of packet drops happened as shown in figure 7. The overall throughput became stable in comparison of previous scenario. The stability was observed with reduction in throughput having the average of 1.57227 Gbps. With this observation, in comparison of previous scenario, authors would like to throw light on the stability behavior of Beacon controller recommending it for the ad hoc network of approximate 200 nodes.

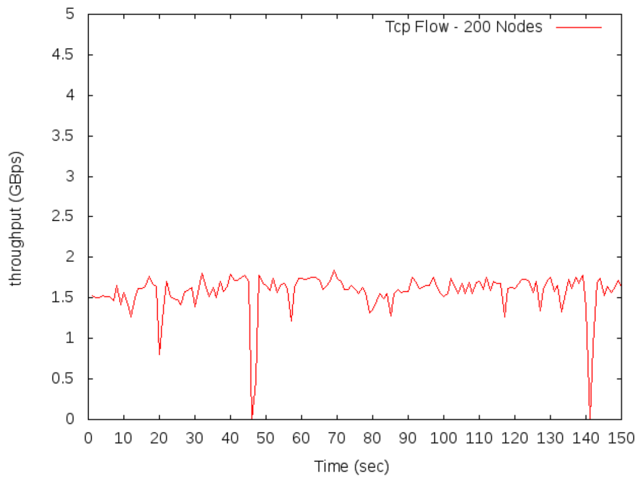


Fig. 7. Throughput of scenario 2

While testing 400 nodes scenario with tcp flows connecting the nodes with 5 separate switches, the experiment was conducted in similar networking circumstances. As shown in figure 8, it was observed that throughput is further degrading by increasing the number of nodes reaching the average of 1.39998 Gbps.

The load over the central 6th switch connecting all the 5 switches studded with numerous nodes seems to be getting overloaded with the congestion, according to the authors of this paper.

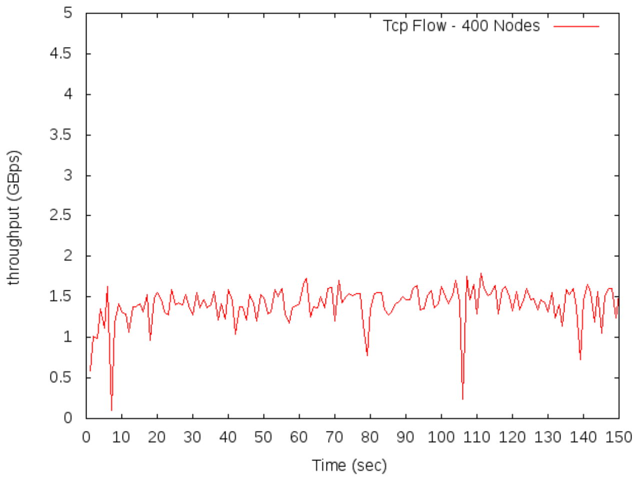


Fig. 8. Throughput of scenario 3

As we moved forward to adding nodes to our experimental test bed we further analyzed the output of the scenario with 600 nodes. Average of 1.48175 Gbps was observed while performing the experiment in similar networking conditions, it is improving in comparison of 400 nodes scenario. The bottleneck situation of 6th central switch is providing this average with multiple events of packet drop, network seems to be struggling to maintain the throughput high as shown in figure 9.

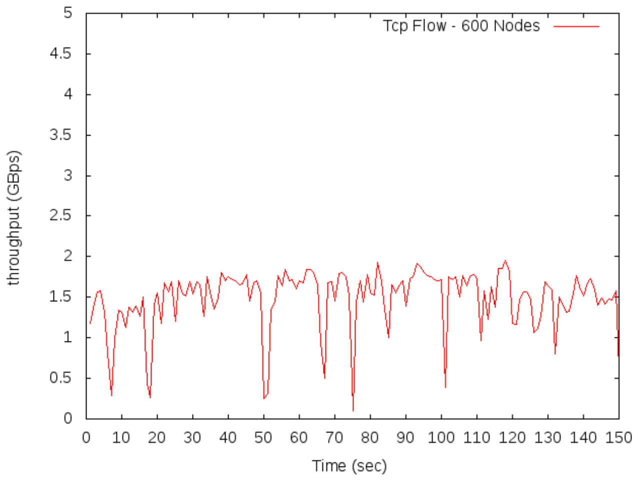


Fig. 9. Throughput of scenario 4

Further, experiment was conducted on 800 nodes which has brought the average throughput down to 0.97323 Gbps in comparison of previous scenario. This reduction of throughput was done by the networking devices while avoiding the events of packet drops for providing reliable tcp service for communication among the nodes. The stability factor seems to be highest in this scenario with a tradeoff lower throughput as shown in figure 10.

Further, while experimenting and adding nodes to 1000, the behavior of the controller became excessively unstable and multiple events of drops were observed along with average throughput of 1.09296 Gbps.

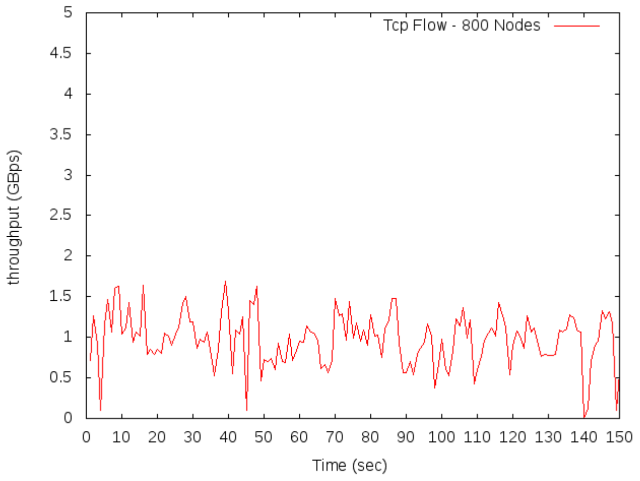


Fig. 10. Throughput of scenario 5

The figure 11 is clear indication of capacity limit of controller to handle the number of nodes in constant networking scenario. Further we did not experiment because of the limitations of system resources and the purpose seems to be achieved testing the scalability of Beacon controller to its upper limit.

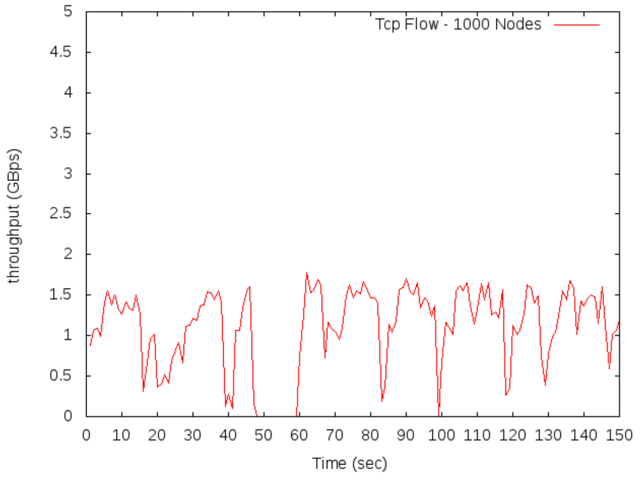


Fig. 11. Throughput of scenario 6

VII. Conclusion

With this paper, authors attempt to demonstrate the experimental environment on beacon controller addressing the scalability behavior of the Beacon controller. The paper starts with the discussion on the architecture and requirement of bringing the Beacon into existence which is a leap jump towards practicality of sdn. While experimenting, authors have noted down step by step procedure, which is listed in this paper under the section of IV experimental scenario. Also the issues while experimenting were clearly defined with its solution to save the time of many researchers experimenting with similar networking conditions. If the step by step procedure provided in the section of IV of experimental scenario is followed by any researcher, exact networking conditions will be created and can be further enhanced and experimented to dive through the Beacon controller and explore further.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

The paper not just show the experimental setup but also allows the readers to start from scratch where the new bees can also follow Beacon installation steps. The obtained results are analyzed in depth to draw the conclusion providing red light to the researchers for moving further on large scale networks with the controller of Beacon because, the graph of average of every scenario is reducing gradually as shown in figure 12.

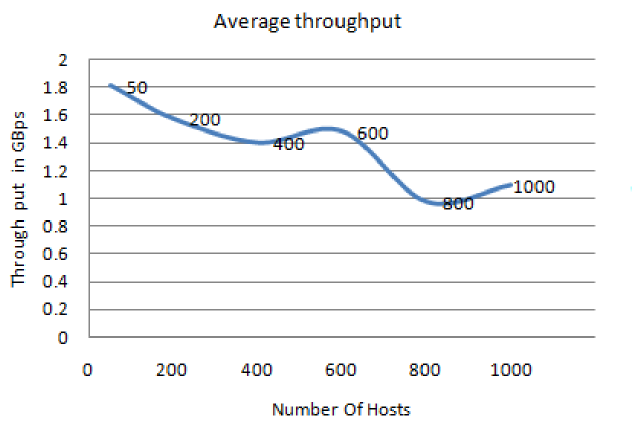


Fig. 12. Average throughput

In most of the scenarios, the controller is struggling to maintain the connectivity or the throughput is at bottleneck at the verge of burst packet drops. The controller is recommended to the networking conditions where the number of nodes is not more than 600. The statistics clearly indicates that the controller is not meant for large scale networking requirements and falls short in maintaining stability which also shows green light to the researchers who are willing to provide the solutions to gradual reduction in throughput while increasing the number of nodes. The new bees may like this article as it tells clearly about concept and practical step by step guidelines for implementing Beacon controller. Further, the researchers are working to come up with multiple controllers' implementation on single network in upcoming papers.

REFERENCES

1. Goswami B., Asadollahi S.S. (2018) Enhancement of LAN Infrastructure Performance for Data Center in Presence of Network Security. In: Lobiyal D., Mansotra V., Singh U. (eds) Next-Generation Networks. Advances in Intelligent Systems and Computing, vol 638. Springer, Singapore.
2. Saleh Asadollahi, Bhargavi H Goswami, (2017) "Revolution in Existing Network under the Influence of Software Defined Network", Proceedings of the 11th INDIACom, Pages: 1012-1017, IEEE, New Delhi, India.
3. S Das, B Goswami, S Asadollahi, Investigating Software-Defined Network and Networks-Function Virtualization for Emergent Network-oriented Services, IJIRCCE, Vol.5, Special Issue 2, April 2017, Pg. No. 201 – 205, DOI:10.15680.
4. Saleh Asadollahi ; Bhargavi Goswami ; Ahmad Sohaib Raoufy ; Hedmilson, (2017), "Scalability of software defined network on floodlight controller using OFNet", International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Pages: 1 – 5, IEEE, Mysore, India.
5. Nick McKeown; et al. (April 2008). "OpenFlow: Enabling innovation in campus networks". ACM Communications Review. Retrieved 2018-09-01.
6. Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In Proc. 1st workshop on Hot topics in software defined networks, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM..
7. S Asadollahi ; B Goswami (2017) "Experimenting with Scalability of Floodlight Controller in Software Defined Networks", International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Pages: 1 – 5, IEEE, Mysore, India.
8. S Asadollahi, B Goswami, Investigating Software-Defined Network and Networks-Function Virtualization for Emergent Network-oriented Services, IJIRCCE, Vol.5, Special Issue 2, April 2017, Pg. No. 211 – 217.
9. OpenFlow: Beacon controller. Available at https://openflow.stanford.edu/display/Beacon/Home.html (last accessed on September 2018).
10. Mininet: Emulator. Available at http://mininet.org/ (last accessed on September 2018)
11. Justin Pettit (August 20, 2018). "[ovs-announce] Open vSwitch 2.10.0 Available". openvswitch.org. Retrieved September 01, 2018.
12. Python: Scripting network topologies. Available at https://www.python.org/ (last accessed on September 2018)
13. Xterm: Emulator. Available at https://invisible-island.net/xterm/ (last accessed on September 2018)
14. IPERF: Networks tool. Available at https://iperf.fr/ (last accessed on September 2018)
15. Gnuplot: Graph tool. Available at http://www.gnuplot.info/ (last accessed on September 2018)
16. S Asadollahi ; B Goswami ; MSameer (2018) "Ryu controller's scalability experiment on software defined networks", Proceedings of IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), Pages: 1 – 5, IEEE, Bangalore, India.
17. B Goswami, S Asadollahi, Implementation of SDN using OpenDayLight Controller, IJIRCCE, Vol.5, Special Issue 2, April 2017, Pg. No. 218 – 227
18. S Asadollahi, B Goswami, Software Defined Network, Controller Comparison, IJIRCCE, Vol.5, Special Issue 2, April 2017, Pg. No. 211 – 217

AUTHORS PROFILE

Tony Manuel
Master of Computer Applications,
Christ (Deemed To Be University)
Bangalore, Karnataka, India

Dr. Bhargavi H Goswami,
Assistant Professor,
Christ (Deemed To Be University)
Bangalore, Karnataka, India