CSE 534 | Final Report
Implementation and Analysis of Virtualization in Software Defined Networking (SDN) for Data Center Networks

Prasoon Rai (110921754) Sayan Bandyopadhyay (110946522)

Goals

SDN is one of the hottest technological paradigm shifts under which the Networking industry is going through. Network functions at different layers are being virtualized. In this project we have created and measured performance of several topologies using OpenFlow which are used in Data Centers today. We have leveraged upon the work done by *Nader*[1], and evaluate performance . We have used an emulation software called *mininet*[2] which uses OpenFlow switches in turn. Then we used a performance measurement tool like iperf to measure bandwidth and RTT between nodes.
In the end it needs to be shown how different topologies behave in improving load balancing in Data Center Networks(DCNs).

Tools and technologies used

We used the following tools to reach our end goal ->

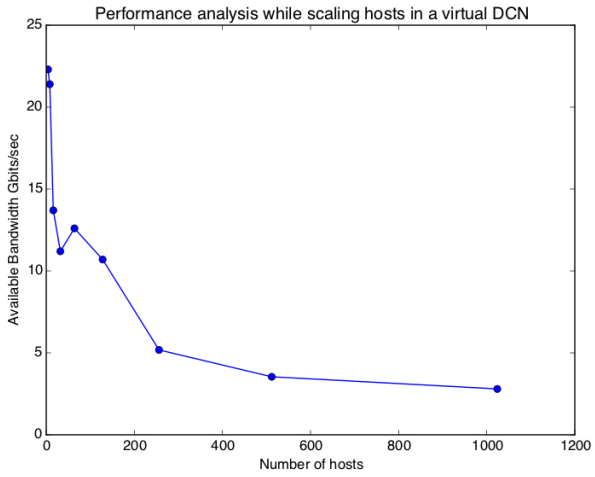
- OpenFlow
- mininet
- iperf

Project Objectives

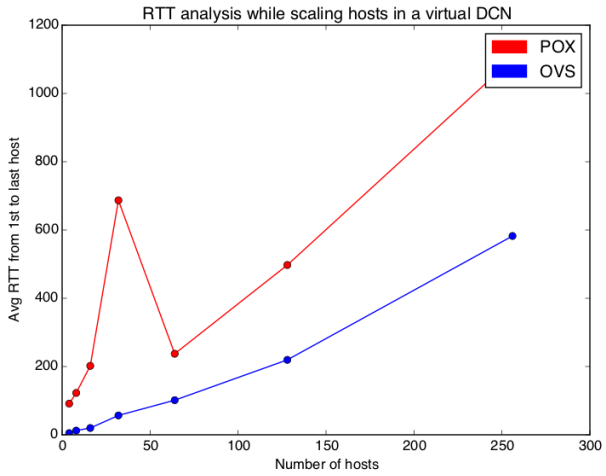
1. Setup mininet on a virtualization platform like VMware Fusion or Virtualbox.
2. Run mininet and configure a few example topologies using a controller, a switch and a few hosts in order to ensure that the setup works correctly.
3. Create a scalable Virtual Data Center interface that enables the programmer to configure desired topology with respect to the number of hosts and switches.
4. Conduct experiments to study the effect of scaling hosts in the DCN on network characteristics like available bandwidth, ping times (both OVS and POX controllers) etc.
5. Produce meaningful visualizations from the data collected, infer interesting observations about DCN scalability and appreciate the utility of SDN in implementing as well as benchmarking DCN virtualization.
6. Analysis of network characteristics for fat tree topology. Extended the study done in the paper to one of the most popular topologies utilized in DCN, the fat tree topology. Create module to create custom, scalable Fat tree topologies: For a given k, this module would be able to calculate and setup the appropriate number of hosts, edge, aggregate and core switches to simulate the DCN.
7. Performed similar performance analysis on bandwidth, ping times and latency as a comparative study against the conventional topology designs studied so far in the project. We expect to observe better performance in case of Fat trees and conclude that they are better configuration both theoretically and performance wise.

Results

1. **Mininet setup**:
   - We setup mininet 2.10 on VirtualBox and tested successfully some examples from the original code.
   - After successfully, testing the operations of default examples as well as experimenting with POX and OVS controller, remote vs local controllers and setting up flows using Openflow, we proceeded with setting up custom virtual DCN.
2. **Scalable DCN topologies**:
   - We were able to successfully write a wrapper to extend the tree topology construction feature of Mininet to allow configurable number of hosts and switches and their arrangement in the desired topology.
   - Our program takes in the number of hosts, depth and fan-out as input and allocates the appropriate number of switches and creates the topology to emulated our virtual DCN. We use this program throughout our study in subsequent sections.
   - Additionally, we created a wrapper to incrementally get the data for an increasing number of hosts and gauge performance metrics like available bandwidth and ping times.
     - This program collects data for the input list of hosts, has parsing utilities to extract computation numbers, extends the *iperf* utility to compute metrics like bandwidth and latency, and finally utilizes the Python's *matplotlib* library to generate plots.
3. **Bandwidth scaling**:

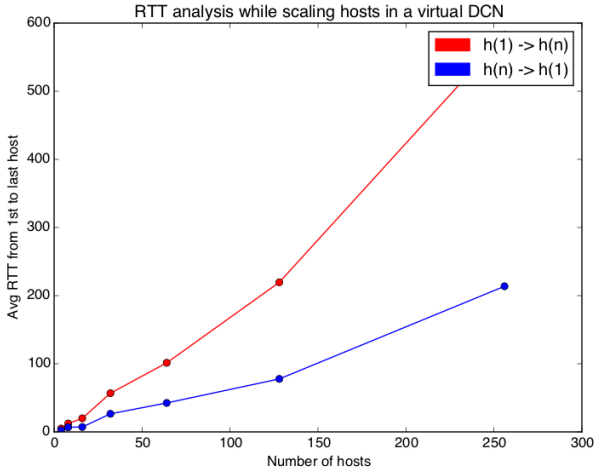


   - In the plot above, we observe that the available bandwidth in the DCN reduces as well scale the number of hosts.
   - It must be noted that the number of hosts have been scaled **height-wise**. Hence, as we keep on increasing the levels of switches, the available bandwidth starts decreasing.
   - Hence, keeping a flatter tree with scaling hosts while **fanning out breadth-wise** is a better approach instead of increasing them depth-wise.
   - We observed that the available bandwidth remains almost unaltered if we scale breadth-wise.
   - We analyzed performance of OVS vs POX controllers and found that OVS controller works far faster than POX controller in this scenario.



4. **Ping times analysis**:

   - We created a custom over-ride on top of mininet's ping() which provides RTTMin, RTTMax, Packets Received, Packets Send and RTTAvg for the scaling network, while taking the host list as input.
   - We observed that as we scale the network, the ping times from the **first** to the **last** host increases.
   - Another observation is that the RTT for the onwards ping is **considerably higher** than that of the opposite direction.
   - We summarize the results for this scaling in the plot below:

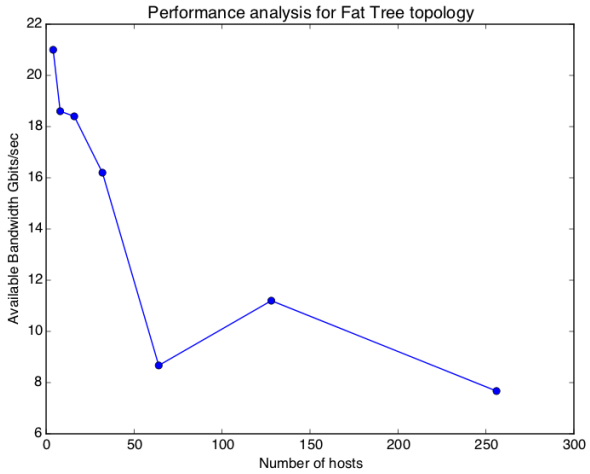


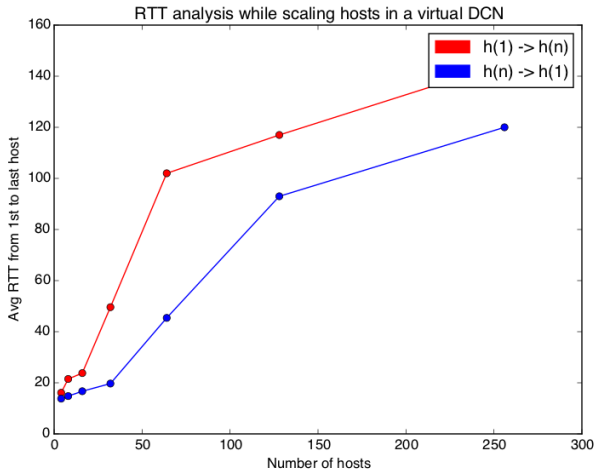
5. **Fat Tree Topology**:

   - We created a module to implement the Fat Tree topology. We used OVS switches with STP enabled on them using a wrapper to avoid loops in the topology. Here is a snapshot for $K = 4$

```
mininet> pingall
*** Ping: testing ping reachability
h_1 -> h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_2 -> h_1 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_3 -> h_1 h_2 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_4 -> h_1 h_2 h_3 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_5 -> h_1 h_2 h_3 h_4 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_6 -> h_1 h_2 h_3 h_4 h_5 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_7 -> h_1 h_2 h_3 h_4 h_5 h_6 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_8 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_9 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_9 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_10 h_11 h_12 h_13 h_14 h_15 h_16
h_10 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_11 h_12 h_13 h_14 h_15 h_16
h_11 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_12 h_13 h_14 h_15 h_16
h_12 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_13 h_14 h_15 h_16
h_13 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_14 h_15 h_16
h_14 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_15 h_16
h_15 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_16
h_16 -> h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_10 h_11 h_12 h_13 h_14 h_15
*** Results: 0% dropped (240/240 received)
mininet> nodes
available nodes are:
as_0_0 as_0_1 as_1_0 as_1_1 as_2_0 as_2_1 as_3_0 as_3_1 c0 cs_0 cs_1 cs_2 cs
_3 es_0_0 es_0_1 es_1_0 es_1_1 es_2_0 es_2_1 es_3_0 es_3_1 h_1 h_10 h_11 h_1
2 h_13 h_14 h_15 h_16 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9
```

- We detect when the STP converges and start our performance analysis after that. The Bandwidth and ping time analysis are as follows





- We compared the performance of both the Tree topology which we used earlier and the Fat tree topology and we made the following observations →

H1 – H2	24
H1-H4	18
H1-H8	16
H1-H16	14
H1-H24	13
H1-H40	12.5
H1-H64	11.7

Figure 1: GENERAL TREE TOPOLOGY: 64 HOSTS, 63 SWITCHES

H1-H2	19.8
H1-H4	15.3
H1-H8	15.4
H1-H16	13.4
H1-H24	13.3
H1-H40	13.1
H1-H54	13.4

Figure 2: FAT TREE TOPOLOGY: 54 HOSTS, 45 SWITCHES (K=6)

(a) From the above tables, we can see that in the general tree topology, the Bandwidth steadily decreases as we start to scale. This is because as the depth of the tree grows, the switches at the top begin to experience heavy load and hence start becoming bottle necks.

(b) We can observe that in the Fat tree topology, the bandwidth is high with communication between 2 hosts connected to the same edge switch (H1-H2). It is slightly lower in between 2 hosts connected to different edge switches but in the same pod (H1-H4, H1-H8). It is significantly lower in 2 hosts connected via the core switch layer (H1-H16, H1-H54, etc) but same for all of them

Project link is in https://github.com/saynb/FCN-mininet-dcn-topo.git

References

[1] Nader F. Mir, Jayashree N. Kotte, and Gokul A. Pokur, "Implementation of Virtualization in Software Defined Networking (SDN) for Data Center Networks", ICN 2016 : The Fifteenth International Conference on Networks

[2] Bob Lantz, Brandon Heller, Nick McKeown "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks"

[3] Mininet: http://mininet.org/

[4] Mininet open source code: https://github.com/mininet/mininet/tree/master/mininet

[5] POX Controller: openflow.stanford.edu/display/ONL/POX+Wiki

[6] CSE 534 class slides on *Data Center Networking* by Professor Aruna Balasubramanian: https://piazza.com/stonybrook/spring2017/cse534/resources

[7] CSE 534 class slides on *Software Defined Networks* by Professor Aruna Balasubramanian: https://piazza.com/stonybrook/spring2017/cse534/resources