# Load Balanced Virtual Data Center Network Using SDN Approach

**Rahul Cariappa Cheyanda, Varun Vijayakumar**

Department of Electrical Engineering, San Jose State University, San Jose, California 95192

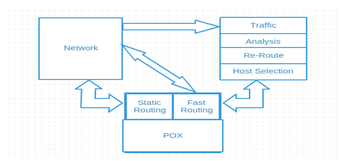
## Introduction

Data center networks have multiple farms of servers providing a variety of services. The increase in the number of services demands effective compute and network load balancing. Software Defined Networking (SDN) can enable an effective optimization of both computing and networking resources, i.e., network aware load balancing.

There are many load balancing methods available. Aster*k(SDN Scheme)[1] selects route and server for high traffic situations. In Plug n Serve (SDN Scheme)[2] model, the first packet of any request is used to determine a flow path. Wildcard rules(SDN Scheme)[3] reduces the load balancer computation by installing rules onto the switches. In OpenFlow based load balancing with multi-path support[4],the load balancing job is divided among switches which manage the devices connected to it. Probing based adaptive routing [5] method sends a probe packet to destination through different intermediate nodes

In this project, an Open-Flow based load balancing algorithm for SDN based data center networks is designed and implemented. The controller effectively chooses the best available server within the pool of servers and computes the least congested path to the selected server. Mininet along with POX controller and the NetworkX Python graph library is used to implement and test the algorithm on the created network. The results obtained are compared with the same network running with static load balancer.

## Modeling




Pictorial representation of the algorithm

### The Main Elements

- Running Network.
- Static and Dynamic Network information.
- Traffic generation.
- Host/Server Selection based on demand.
- Fast Routing based on priority.

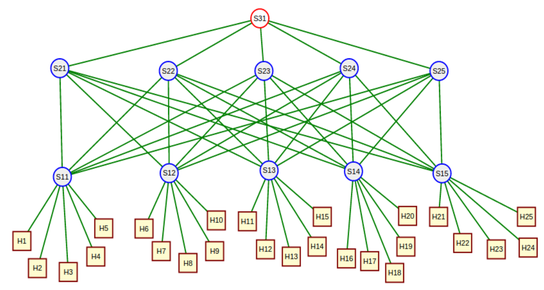
### Implementation Tools

- Mininet - A network emulator to emulate realistic virtual networks.
- OpenFlow - The protocol that defines the communication mechanism between an Open vSwitch and controller.
- POX - A Python based controller for Mininet.
- NetworkX - A Python library to create graphs.
- JSON- A lightweight data-interchange format.

## Design and Implementation

### Network

The following clos network topology is used throughout the implementation in this project.




The Mininet network.

### Path Selection

NetworkX has inbuilt algorithms to determine shortest paths between nodes in a graph. The two algorithms which drive the forwarding mechanism in this implementation are:

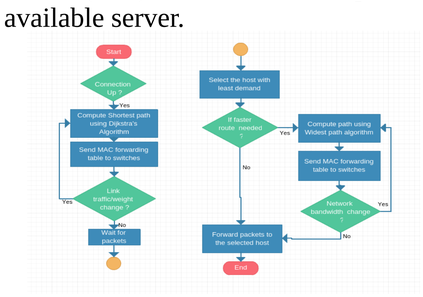
- Dijkstra's Shortest Path Algorithm
  - The algorithm involves computing least cost path between every switch (source) and host (destination).
- Widest path Algorithm
  - The algorithm involves determining path between any-switch and any-host with highest bandwidth capabilities.

### Server Selection

- A virtual IP and MAC address are defined in the POX Controller script which dons the name of Service IP such that it would translate or re-route the protocol specific packets to appropriate server farms.
- Controller loads the JSON file and checks the 'demand' value of all the servers, it finds the server with the least demand and uses its IP and MAC address to replace the Virtual IP and MAC in the incoming packets.

### The Algorithm

- Create the network in Mininet.
- Install static routes in the switches.
- Obtain the traffic details regularly.
- Analyze the traffic , compute dynamic routes and install the rules onto the switches.
- Observe the incoming packets and the protocol, compute faster route if required.
- Observe the incoming packets and the servers(host) and direct the packets to the available server.

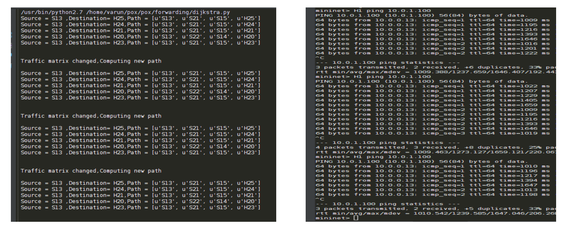



The implementation algorithm

## Results

### Static Load Balancing

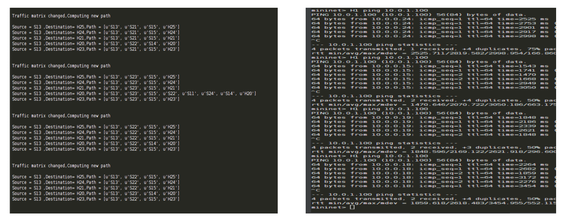
The routes remained constant. Since this routing is static, the packets are routed to the same server/host every time.



Static Load Balancing

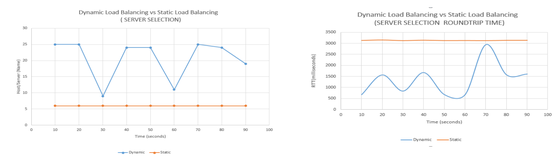
### Dynamic Load Balancing

The routes dynamically change at every interval. The packets are routed to different servers based on the demand value.



Dynamic Load Balancing

The following figure shows a graphical comparison of the server selection in the network using both static and dynamic load balancing. This graph clearly shows that the dynamic load balancer is selecting a different server at every interval but static load balancer selects the same server. The figure also shows a graphical comparison of the round trip time to the corresponding selected server. The graph clearly shows that using dynamic load balancer, a better round trip time can be achieved.




Dynamic Load Balancing vs. Static Load Balancing

## Conclusion

An SDN based dynamic load balancing algorithm was successfully designed and implemented for a data center network. The controller effectively chooses the best available server within the pool of servers and computes the least congested path to the selected server. The controller also prioritizes the packets based on the protocols defined. The implementation was done using Mininet,POX controller and the NetworkX Python graph library. The results were compared with the network running a static load balancer.

## Key References

[1]Nikhil Handigol , Srinivasan Seetharaman , Mario Flajslik , Nick McKeown Ramesh Johari, "Plug-n- Serve: Load-Balancing Web Traffic using OpenFlow", ACM SIGCOMM, Aug. 2009.

[2]Nikhil Handigol, Mario Flajslik , Srini Seetharaman, Ramesh Johari , Nick McKeown, "Aster*x: Load-Balancing as a Network Primitive".,9th GENI Engineering Conference..2010.p1-2.

[3]Richard Wang, Dana Butnariu, and Jennifer Rexford Princeton University; Princeton, NJ, "OpenFlow-Based Server Load Balancing Gone Wild"

[4]Yu Li and Deng Pan, "OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support", Florida International University of Miami, FL

[5]Santosh Mahapatra,XinYuan, "Load Balancing Mechanisms in Data Center Networks", Department of Florida State University,Tallahasse,Florida

## Acknowledgments

We would like to thank Dr.Greg Bernstein for his guidance, support and suggestions provided over the course of implementing this project.

## For further information

Please contact rahulcariappa.cheyanda@sjsu.edu. Mininet code, POX code and simulation results are available upon request.