Load Balancing Using Open Daylight Controller

Kiran Noolvi[1], Swetha Nallamalli[2], Apoorva Godasu[3], Naga Sai Krishna Mohan Pitchikala[4]

[1,2,3,4]Graduate Student, University of Texas at Dallas, Texas, USA

Abstract

In this study, we focused on setting up load balancers and improving load balancing in a network that uses a Fat Tree design using Open Daylight system. We built the network using tools like Mininet and Instant Virtual Network (IVN). Open Daylight, which is an open-source controller for software-defined networks (SDN), was used to manage and control the traffic flow in the network. To test the network, we created traffic using a tool called IPERF and measured key factors like bandwidth, delay (latency), and overall network performance. The aim of this research is to study how the traffic flows through the network, analyze the performance of different ports, and improve how new traffic is scheduled using Open Daylight's load balancing features.

Keywords: software defined networks, load balancing, Open Daylight controller

Introduction

As the networks continue to increase in complexity and size, efficient traffic management has become a critical factor in maintaining network performance and reliability. One promising solution to the network management is the use of Software-Defined Networking (SDN). In Software Defined Networks the control plane is separated from the data plane to provide centralized network management. In this study, we utilized a Fat Tree network topology. Fat Tree topology is a commonly used architecture in data center networks and is known for its high bandwidth and fault tolerance. The Fat Tree topology is built using Mininet and Instant Virtual Network (IVN) tools to emulate a realistic network environment.

The core of our study is the Open Daylight controller. It is a popular open-source SDN platform that uses the OpenFlow protocol for controlling traffic flows. By integrating Mininet with Open Daylight, we can develop an efficient load balancer that can manage network traffic and improve overall performance. Traffic is generated using IPERF, which allows us to measure key network parameters such as bandwidth, latency, and link performance.

In this study we aim to analyze Open Daylight's load balancing algorithm, monitor network flows based on port statistics, and further optimize flow scheduling to improve traffic distribution.

Open Sources Used

Fat-Tree Network Topology

For having an efficient network connectivity we have used fat-tree network topology. Fat-Tree network is a tree structure and hosts are connected to the bottom layer. For any switch, the number of data links going down to its siblings is equal to the number of links going up to its parent in the upper level. The main goal is to connect a large number of endpoints (processors or servers) by using switches that only have a limited number of ports. This would also help us in

- Full Bisection BW: 1:1 Oversubscription ratio
- Low Cost: Commodity switches.

- Great Scalability: K-port switch's supports $K^{3/4}$ servers
- Each port supports same speed as end host.
- All devices can transmit at line speed if packets are distributed uniform along available paths.

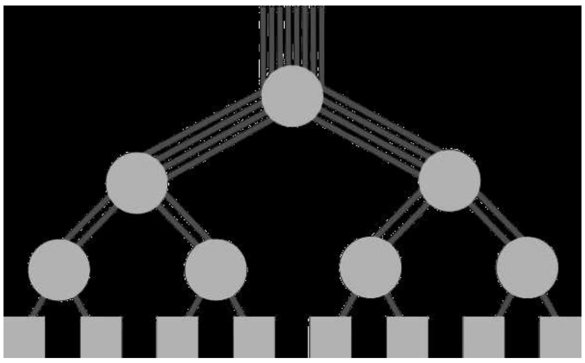


Fig-1: Fat Free Network Topology

MININET

Features of Mininet:

- **Custom topologies:** You can build any kind of network, from a simple one with just one switch to something as big as the Internet, or even a data center
- **Customize packet forwarding:** Mininet’s switches can be programmed to control how data moves through the network using the OpenFlow protocol.
- **Can share and replicate results:** Once you've set up a network and packaged the code, anyone with a computer can run the same setup easily
- **Can run real programs:** Since Mininet works on Linux, you can run actual programs like web servers, network monitoring tools (like Wireshark), or anything else that runs on Linux.
- Mininet creates a virtual network environment with hosts, switches, routers, and links, all running on a single Linux machine.
- It's easy to understand, use, and can connect to other network systems like Open Daylight smoothly.
- Unlike simulators, Mininet runs real, unmodified code, whether it's application code, the operating system, or the network control code, and it can easily connect to actual physical networks.

Limitations of Mininet:

- The virtual networks you create are limited by the CPU and bandwidth available on the single server you're running Mininet on. So, it might struggle with very large networks.
- Mininet doesn't work with OpenFlow switches that aren't running on a Linux platform

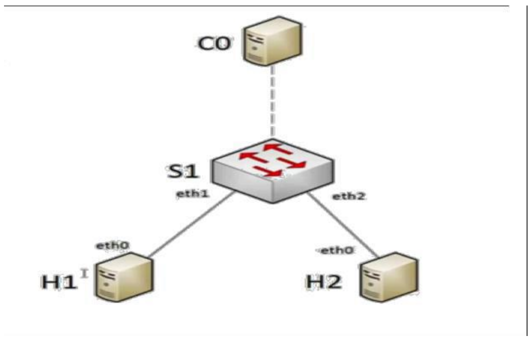


Fig-2: Mininet

Open Flow

- OpenFlow is a southbound API that enables software running on multiple routers to control the path of network packets through a network of switches.
- The fast packet forwarding function is still on the switch, but the high-level routing decisions is moved to a new controller
- OpenFlow has a message structure called OF messages that are passed between the controller and the OpenFlow switch. Every switch will have their own Flow Table like the MAC address table and routing table in the traditional switches. Every new incoming packet is matched against the existing Flow table on the switch and a necessary action is taken.

Open Daylight

- An open-source project with modular, pluggable and flexible controller platform and implemented strictly in software which runs in its own Java Virtual Machine.
    - It provides
    - SDN Controller
    - Overlay Virtual Network
    - Plugins
    - Interfaces
- Yang is the modeling language used in SDN.
- It is used for modelling tree-based data structured messages that help in interacting with RPCs. (Model Driven – Service Abstraction Layer)

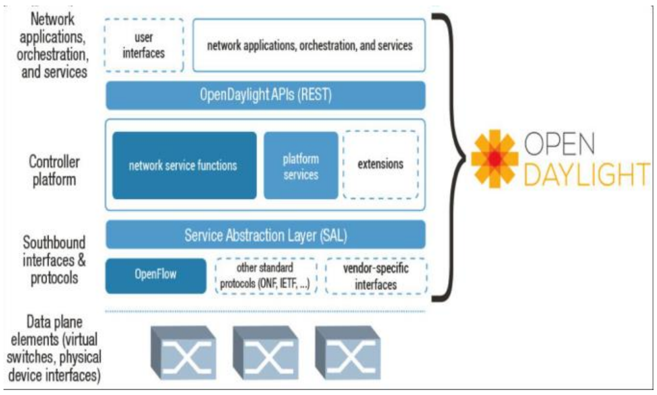
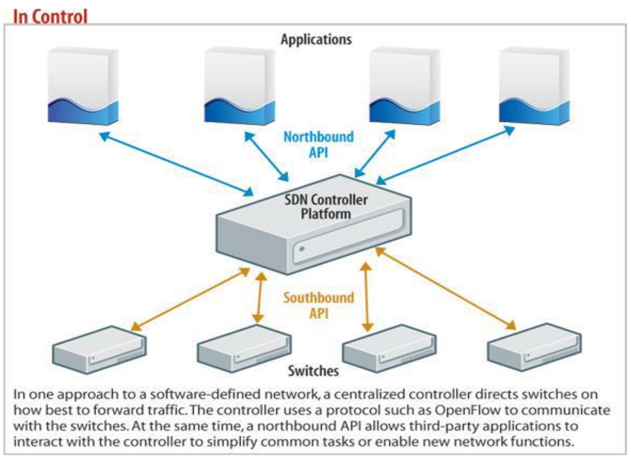

Network applications, orchestration, and services
user interfaces
network applications, orchestration, and services
OpenDaylight APIs (REST)
Controller platform
network service functions
platform services
extensions
OPEN DAYLIGHT
Southbound interfaces & protocols
Service Abstraction Layer (SAL)
OpenFlow
other standard protocols (ONF, IETF, ...)
vendor-specific interfaces
Data plane elements (virtual switches, physical device interfaces)

Fig-3: Open Daylight



In one approach to a software-defined network, a centralized controller directs switches on how best to forward traffic. The controller uses a protocol such as OpenFlow to communicate with the switches. At the same time, a northbound API allows third-party applications to interact with the controller to simplify common tasks or enable new network functions.

Fig-4: Design Overview

IPERF

IPERF is a commonly used tool that helps test and improve how well a network is performing. It's important because it works on different operating systems (like Windows, Linux, etc.) and provides consistent, standardized results that can be used to measure any type of network.

IPERF works by using two components: a client and a server. The client sends data to the server, and this exchange allows IPERF to measure how much data can move between the two devices in a given time (this is called throughput). It can measure data transfer in one direction (client to server) or both directions (client to server and server to client).

When IPERF runs, it provides a detailed report that shows how much data was transferred, how fast it moved, and when each part of the test happened (using timestamps). This information is very useful for understanding the speed and efficiency of a network and can help identify areas where improvements are needed.

Steps Involved

1. Create Mininet Topology
2. Connect it to Controller
3. Execute 'pingall' to check reachability of hosts
4. Use Rest API to configure load balancer service
5. Generate traffic using IPERF

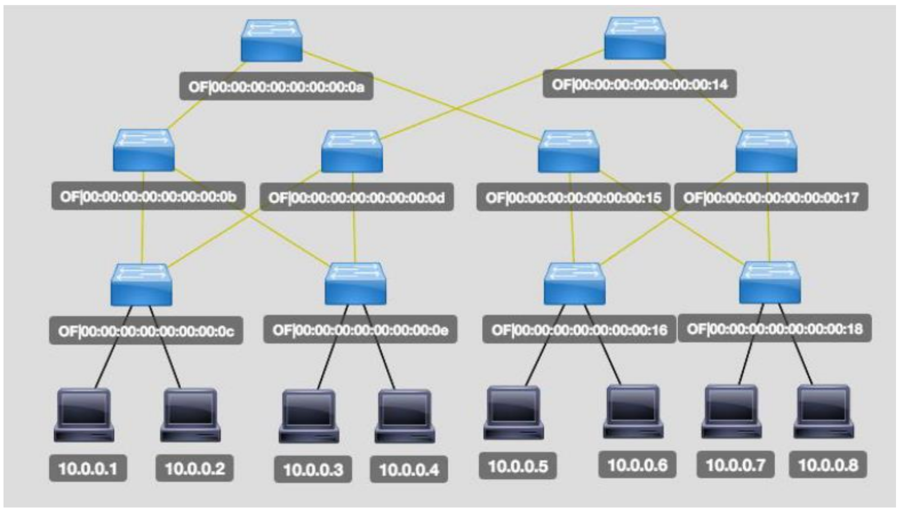
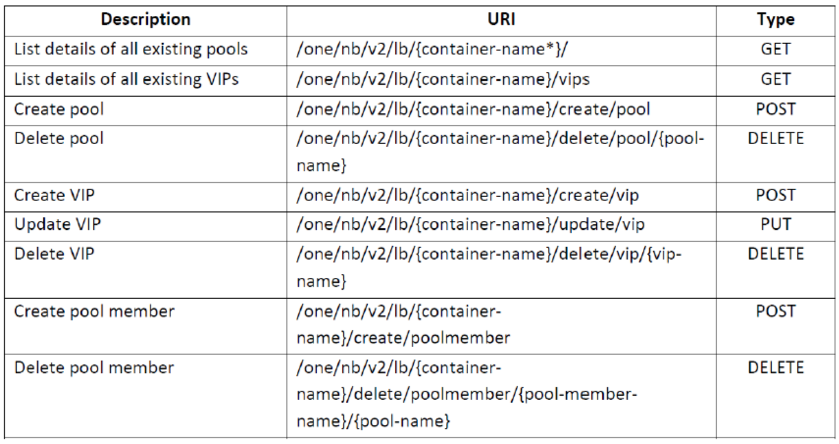

OF|00:00:00:00:00:00:00:0a
OF|00:00:00:00:00:00:00:14
OF|00:00:00:00:00:00:00:0b
OF|00:00:00:00:00:00:00:0d
OF|00:00:00:00:00:00:00:15
OF|00:00:00:00:00:00:00:17
OF|00:00:00:00:00:00:00:0c
OF|00:00:00:00:00:00:00:0e
OF|00:00:00:00:00:00:00:16
OF|00:00:00:00:00:00:00:18
10.0.0.1
10.0.0.2
10.0.0.3
10.0.0.4
10.0.0.5
10.0.0.6
10.0.0.7
10.0.0.8

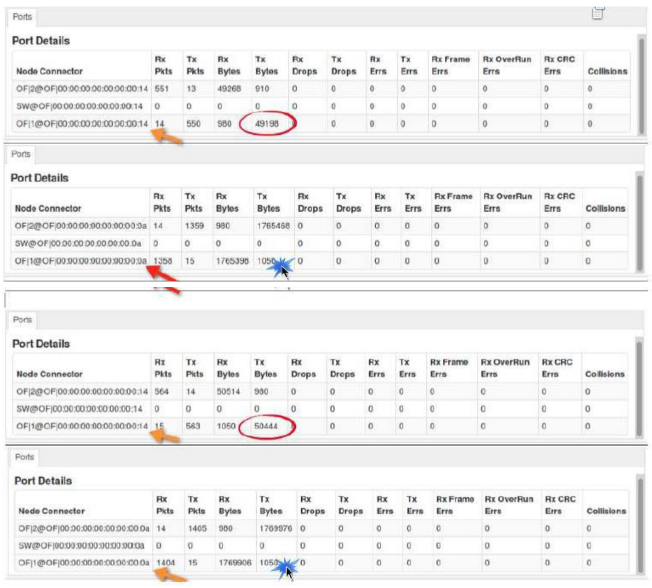
Fig-5: Connecting Topology to controller



Description	URI	Type
List details of all existing pools	/one/nb/v2/lb/{container-name*}/	GET
List details of all existing VIPs	/one/nb/v2/lb/{container-name}/vips	GET
Create pool	/one/nb/v2/lb/{container-name}/create/pool	POST
Delete pool	/one/nb/v2/lb/{container-name}/delete/pool/{pool-name}	DELETE
Create VIP	/one/nb/v2/lb/{container-name}/create/vip	POST
Update VIP	/one/nb/v2/lb/{container-name}/update/vip	PUT
Delete VIP	/one/nb/v2/lb/{container-name}/delete/vip/{vip-name}	DELETE
Create pool member	/one/nb/v2/lb/{container-name}/create/poolmember	POST
Delete pool member	/one/nb/v2/lb/{container-name}/delete/poolmember/{pool-member-name}/{pool-name}	DELETE

Fig-6: REST API Configuration

Results



Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:14	551	13	49268	910	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:14	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:14	14	550	980	49198	0	0	0	0	0	0	0	0

Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:0a	14	1359	980	1765468	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:0a	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:0a	1358	15	1765398	1050	0	0	0	0	0	0	0	0

Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:14	564	14	50514	980	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:14	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:14	15	563	1050	50444	0	0	0	0	0	0	0	0

Ports

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|2@OF|00:00:00:00:00:00:00:0a	14	1405	980	1769976	0	0	0	0	0	0	0	0
SW@OF|00:00:00:00:00:00:00:0a	0	0	0	0	0	0	0	0	0	0	0	0
OF|1@OF|00:00:00:00:00:00:00:0a	1404	15	1769906	1050	0	0	0	0	0	0	0	0

Problems faced

1. Issues faced while installing Beryllium open daylight

Beryllium open daylight failed to start with Installation of Java8 using apt install command and setting the Java path shows as follows:

```
sudo add-apt-repository ppa:webupd8team/java sudo apt-get update
sudo apt-get install oracle-java8-installer
```

2. Following is the error displayed while starting Beryllium:

```
Error: Could not find or load main class project.distribution-karaf-0.4.4-Beryllium- SR4.etc.odl.java.security
```

To resolve this, download jdk1.8.0_131 from Java webpage and set the java path

Conclusion

We began this project with the goal of developing an efficient load balancer for the Open Daylight controller in a network using the Fat Tree topology. Throughout the project, we successfully implemented and tested the load balancer, using OpenFlow to manage traffic switching and control how data flows through the network.

By analyzing the performance of the load balancer, we were able to monitor the traffic, assess how well the network handled different loads, and optimize the flow scheduling process. The flows we worked with in this project were Layer 2 (L2), meaning they involved switches and direct connections between devices.

While the load balancer worked well for L2 traffic, there is room for improvement. A potential future step is to extend the load balancing capabilities beyond just Layer 2 switches to include routers (Layer 3). This would allow for more advanced traffic management, covering not just direct device-to-device connections but also routing across different networks. Overall, the project provided a solid foundation for improving network performance, with clear opportunities for further development in load balancing across different network layers.

References

1. https://wiki.p2pfoundation.net/Network_Topology
2. https://clusterdesign.org/fat-trees/
3. https://blogchinmaya.blogspot.com/2017/04/what-is-fat-tree-and-how-to-construct.html
4. https://hackmd.io/@pmanzoni/BklqpKddS

5. https://mininet.org/walkthrough/
6. https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/
7. https://stackoverflow.com/questions/4527836/flow-based-routing-and-openflow
8. https://networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-sdn-opendaylight
9. https://medium.com/@blackvvine/sdn-part-2-building-an-sdn-playground-on-the-cloud-using-open-vswitch-and-opendaylight-a0e2de029ce1
10. https://docs.openstack.org/kolla-ansible/stein/reference/networking/opendaylight.html
11. https://openmaniak.com/iperf.php
12. https://iperf.fr/