

OpenFlow Based Load Balancing for Fat-Tree Networks

Büşra TEMEL, Şehriban ALPAYDIN, Emre YALÇIN
Department of Electronics and Communications Engineering
Istanbul Technical University, Istanbul, Turkey
{temelb, yalcinmr, alpaydin16}@itu.edu.tr

Abstract—Data center environment requires high data traffic demand in each hosts. Fat –tree network commonly used for this need. In this network there are two important target; maximizing system throughput and minimizing network latency. Therefore lots of different load balancing methods are used to offer higher bandwidth utilization in fat tree network [1]. In traditional networks which use static switches, load balancing process is based on hash calculations of packets. An alternative solution for this issue is to use Software Defined Networking (SDN). In this project, we aim to make some improvements over load balancing in fat-tree topology networks by using SDN technology. SDN is a concept where a central controller makes the decision where the packet is send, contrariwise switches decide this in traditional We will design a dynamic routing algorithm to obtain these directions by controller thus we will be achieved an efficient load balancing for each seperate flow.

Keywords: SDN; Floodlight; Mininet; fat-tree; load balancing

I. INTRODUCTION

Nowadays data center environment which commonly uses fat-tree networks requires high data traffic demand in each hosts. In this network there are two important target; maximizing system throughput and minimizing network latency. Therefore lots of different load balancing methods are used. Load balancing methods offer higher bandwidth utilization in fat tree network, besides this type networks contain more than one link between hosts so this topology can offer higher available link which means also higher bandwidth usage.

In traditional networks which use static switches, load balancing process is based on hash calculations of packets. In this approach, each packet of a flow follows the single pre-defined path through the network. When something happen like a switch breakdown or physical layer damage, packets tend to drop or the other switches need to be manually configured for choosing a different path. This becomes a difficult task for large networks. Also, disadvantage of hashing is that all links gets the same percentage of hash values, this means all paths have the same capacity. Even if the network is hard coded to work in as multipath network, because of the equal capacity issue, efficient load balancing might not be achieved. An

alternative solution for this issue is Software Defined Networking (SDN).

In this project, we aim to make some improvements over load balancing in fat-tree topology networks by using SDN technology. SDN is a concept where a central controller makes the decision where the packet is send, contrariwise switches decide this in traditional networks. Controller dynamically detects the topology by listening to the switches and calculates available path with less load. Controller then directs the switches with forwarding entries needed for the paths [2]. We will design a dynamic routing algorithm to obtain these directions by controller thus we will be achieved an efficient load balancing for each seperate flow.

The implementation of this project will be done by using Mininet as a network emulator that creates a network of virtual hosts, switches, controllers, and links [3]. A Mininet host behaves just like a real machine, you can ssh into it and run arbitrary programs that is installed on the underlying Linux system. Packets can be handled by the Mininet switches, with a given link speed and delay. Mininet switches support OpenFlow as a communication protocol. Connection of the switches and the controller software, which is Floodlight in this project, will be established by OpenFlow. Due to generate traffic, Iperf network testing tools will be used so that TCP or UDP data streams will be created accordingly. Bandwith, delay jitter and datagram loss can be monitored on Iperf [4].

While Floodlight SDN controllers beginning based on Beacon, open source controller, it was built using Apache Ant which is a very popular software build tool that makes the development of Floodlight easier and flexible [5]. Floodlight has an active community and has a large number of features to create a system that meets the requirements of the specific organization. Floodlight is designed to work with the growing number of switches, routers, virtual switches, and access points that support the OpenFlow standard. Web based and Java based GUIs are available on this controller.

II. TOOLS INSTALLATION

A. Install java components

In this project, floodlight installed on Ubuntu 14.04.4 LTS OS. Floodlight is written in java thus install jdk and ant by running;

```
$ sudo apt-get install build-essential default-jdk ant python-dev eclipse
```

B. Download and build the floodlight

```
$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ git checkout stable
$ ant;
$ sudo mkdir /var/lib/floodlight
$ sudo chmod 777 /var/lib/floodlight
```

C. Run the floodlight

```
$ java -jar target/floodlight.jar
```

D. Simulate a network

It is possible to run Mininet against the locally running Floodlight (just type “sudo mn”) but you can also run it against a remote controller [3].

III. BASIC CONCEPT

After the installation process, virtual machine is started where the mininet emulator and floodlight controller are installed. There are two basic process while improving the project. One of them is coding the fat-tree topology on python for mininet and the other one is designing the load balancing system for floodlight. In this report, just basic concepts not entire coding steps are examined to understand usage of these two tools.

A. Obtaining a Topology on Mininet

Mininet already has its own topology samples and some of them can be created as follows:

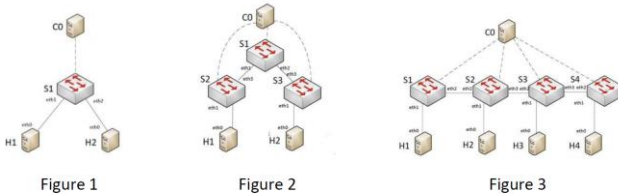


Figure 1 Mininet default topologies

Command for Figure 1: `$ sudo mn` (default topology)

Command for Figure 2: `$ sudo mn --topo=tree,2,2`

Command for Figure 3: `$ sudo mn --topo=linear,4`

This topology codes are stored as a .py file under /mininet directory. Thus, it will be replaced with the new one which will be written in python for the fat-tree topology.

Floodlight is used as an external controller and it runs in local virtual machine. It should be configured with an ip address and this address should be determined in mininet as follows:

```
$ sudo ifconfig eth0 192.168.1.10
```

```
$ sudo mn --controller=remote,ip=192.168.1.10,port=6653
```

6653 : Default port for the Open Flow Protocol

After providing this connection between network devices in the emulator and controller as floodlight, software defined networking which is the main target can be process. In this project load balancing will be aimed as a network process. The algorithm will be designed in next steps of the project is stored as a .java file under /floodlight directory.

B. Initial Outputs

The controller activation state is observed when Floodlight is run.

```
floodlight@floodlight:~/floodlight$ java -jar target/floodlight.jar
15:19:52.435 INFO [n.f.c.n.FloodlightModuleLoader:main] Loading modules from src/main/resources/floodlightdefault.properties
15:19:54.589 INFO [n.f.c.i.Controller:main] Controller role set to ACTIVE
```

Figure 2 Floodlight activation

It is learned after running floodlight which port will be used to connect devices to controller. It is observed port by 6653 is used in this version of floodlight and it is also flexible to change.

```
ClusterConfig [allNodes={32767=Node [hostname=localhost, port=6642, nodeId=32767, domainId=32767]}, authScheme=CHALLENGE_RESPONSE, keyStorePath=/etc/floodlight/auth_credentials.jcks, keyStorePassword is unset]
15:12:26.153 INFO [o.s.s.i.r.RPCService:main] Listening for internal Floodlight RPC on localhost/127.0.0.1:6642
15:12:26.190 INFO [n.f.c.i.OFSwitchManager:main] Listening for switch connections on 0.0.0.0/0.0.0.0:6653
```

Figure 3 Floodlight default port

A basic tree topology is produced in mininet and connected to floodlight controller via 6653 port. Due to both of mininet and floodlight are run in the same virtual machine, it is not required to specify ip address of the controller. It is already known as localhost by default.

```
floodlight@floodlight:~$ sudo mn --mac --controller=remote,port=6653 --topo tree,2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3
*** Starting CLI:
mininet>
```

Figure 4 Floodlight default topologies

Connected switches and added links are shown in the controller interface.

```
15:12:54.885 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:01 connected.
15:12:55.210 INFO [n.f.r.RouteManager:Scheduled-1] *** within the overridden topologyChanged() function in RouteManager
```

Figure 5 The connection of the switches

Connection between hosts is controlled by ping in mininet.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=5.90 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.296 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.058 ms
```

Figure 6 The connection between hosts

The controller session can be killed by Control + C. There is no controller running at that moment but ping process will be still working for a while. Because controller has been sent a flow modification message to switch which results in flow entries on the switch and these entries are cached until a timeout expires.

Based on these reviews and studies, we will keep working on our project with above-examined simulation tools.

IV. PROPOSED LOAD BALANCING SYSTEM

We use mininet as a network emulator and a floodlight virtual machine as a controller. We create below topology in mininet after that we connect all devices to the controller.

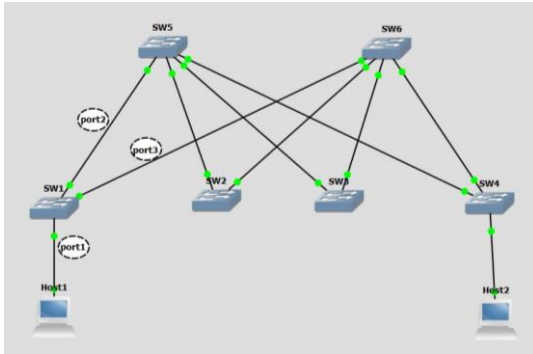


Figure 7 The designed fat-tree topology

A. Collecting statistics

For the purpose of our project we need to get statistical data for each link. In floodlight there is a statistics module. We activate this module from “floodlightdefault.properties” java file.

```
net.floodlightcontroller.statistics.StatisticsCollector.enable=TRUE
net.floodlightcontroller.statistics.StatisticsCollector.collectionIntervalPortStatsSeconds=10
```

Figure 8 Settings of statistics module

We change above field with “TRUE” and “10”. In this way, controller collects port statistics from switches in every 10 seconds.

After that we use REST API to log the output of this module into the terminal. In this command “00:00:00:00:00:00:01” specifies the DPID. Every switch has DPID in floodlight, controller generates this when mininet topology script runs. It assigns first number to first switch.

```
curl -X GET
http://192.168.1.10:8080/wm/statistics/bandwidth/00:00:00:00:00:00:00:00:01/1/json
```

This command gives each ports statistic as in shown below:

```
[{"dpid":"00:00:00:00:00:00:00:00:01","port":"1","updated":"Sun Apr 24 07:31:16 PDT 2016","bits-per-second-rx":"784","bits-per-second-tx":"844"}, {"dpid":"00:00:00:00:00:00:00:00:01","port":"local
```

```
","updated":"Sun Apr 24 07:31:16 PDT 2016","bits-per-second-rx":"0","bits-per-second-tx":"3391"}, {"dpid":"00:00:00:00:00:00:00:00:01","port":"2","updated":"Sun Apr 24 07:31:16 PDT 2016","bits-per-second-rx":"3876","bits-per-second-tx":"1203"}, {"dpid":"00:00:00:00:00:00:00:00:01","port":"3","updated":"Sun Apr 24 07:31:16 PDT 2016","bits-per-second-rx":"419","bits-per-second-tx":"60"}]
```

B. Static Push Route

After we obtain these data, we add static flow to switch with below command. In this command we can tell switch which direction it must use when forwarding traffic with setting in_port and output.

```
curl -X POST -d '{"switch": "00:00:00:00:00:00:00:00:01", "name":"flow-mod-1", "cookie":"0", "priority":"32768", "in_port":"1","active":"true", "actions":{"output=2"}}'
http://192.168.1.10:8080/wm/staticflowpusher/json
```

C. Evaluation

We start to generate traffic from h1(host1 – 10.0.0.1) to h2(host2 – 10.0.0.2) while using iperf in mininet.

```
mininet> h2 iperf -s &
mininet> h1 iperf -c h2 -t 100 -i 2
```

```
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
```

```
[ 3] local 10.0.0.1 port 33876 connected with 10.0.0.2 port 5001
[ ID] Interval    Transfer    Bandwidth
[ 3] 0.0- 2.0 sec 25.5 MBytes 107 Mbits/sec
[ 3] 2.0- 4.0 sec 34.8 MBytes 146 Mbits/sec
[ 3] 4.0- 6.0 sec 39.4 MBytes 165 Mbits/sec
[ 3] 6.0- 8.0 sec 38.8 MBytes 163 Mbits/sec
[ 3] 8.0-10.0 sec 41.6 MBytes 175 Mbits/sec
[ 3] 10.0-12.0 sec 47.1 MBytes 198 Mbits/sec
[ 3] 12.0-14.0 sec 47.4 MBytes 199 Mbits/sec
```

We implement linux bash script for achieving dynamic load balancer. This scripts collects port statistics for switch1 and deciding which port can be used for the outgoing traffic form port1.

```
#!/bin/sh
while true
do

tx1=`curl -X GET
http://192.168.1.10:8080/wm/statistics/bandwidth/00:00:00:00:00:00:00:00:01/1/json | jq '[2].bits-per-second-tx' | grep -o -E '[0-9]+'`

tx2=`curl -X GET
http://192.168.1.10:8080/wm/statistics/bandwidth/00:00:00:00:00:00:00:00:01/1/json | jq '[3].bits-per-second-tx' | grep -o -E '[0-9]+'`

echo "port2_tx=$((tx1)) port3_tx=$((tx2))"
```

```

if [ $((tx1)) -lt $((tx2)) ]

then

curl -X POST -d '{"switch": "00:00:00:00:00:00:01",
"name":"flow-mod-1", "cookie":"0", "priority":"32768",
"in_port":"1","active":"true", "actions":"output=2"}'
http://192.168.1.10:8080/wm/staticflowpusher/json

echo "traffic pushed to port2"

else

curl -X POST -d '{"switch": "00:00:00:00:00:00:01",
"name":"flow-mod-1", "cookie":"0", "priority":"32768",
"in_port":"1","active":"true", "actions":"output=3"}'
http://192.168.1.10:8080/wm/staticflowpusher/json

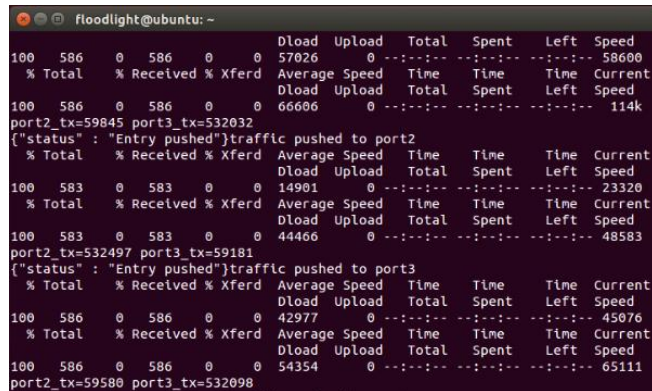
echo "traffic pushed to port3"

fi

sleep 10
done

```

In the output we see that scrips push route to port 2 if port 3 data rate higher than port 2 and inverse also as follows:



	Dload	Upload	Total	Spent	Left	Speed
100 586 0 586 0 0	57026	0	57026	0	58600	58600
% Total % Received % Xferd	Average	Speed	Time	Time	Time	Current
Dload Upload Total Spent Left Speed						
100 586 0 586 0 0	66606	0	66606	0	114k	114k
port2_tx=59845 port3_tx=532032						
{"status": "Entry pushed"}traffic pushed to port2						
% Total % Received % Xferd	Average	Speed	Time	Time	Time	Current
Dload Upload Total Spent Left Speed						
100 583 0 583 0 0	14901	0	14901	0	23320	23320
% Total % Received % Xferd	Average	Speed	Time	Time	Time	Current
Dload Upload Total Spent Left Speed						
100 583 0 583 0 0	44466	0	44466	0	48583	48583
port2_tx=532497 port3_tx=59181						
{"status": "Entry pushed"}traffic pushed to port3						
% Total % Received % Xferd	Average	Speed	Time	Time	Time	Current
Dload Upload Total Spent Left Speed						
100 586 0 586 0 0	42977	0	42977	0	45076	45076
% Total % Received % Xferd	Average	Speed	Time	Time	Time	Current
Dload Upload Total Spent Left Speed						
100 586 0 586 0 0	54354	0	54354	0	65111	65111
port2_tx=59580 port3_tx=532098						

Figure 9 The output of dynamic load balancing algorithm

And also we observe that in the evaluation process, there is no ping loss while route changes. To show this we start icmp test in a short interval in mininet.

```

mininet> xterm h1
root@ubuntu:~# ping 10.0.0.2 -i 0.1

```

In the output of this command there is no loss packet.

And also this process can be imlemented with using udp traffic as shown in below.

```

mininet> h2 iperf -s &
mininet> h1 iperf -c h2 -t 100 -i 2 -u -b 100M

```

V. CONCLUSION

In this paper, we present a dynamic load balancer(DLB) to efficiently push flows for fat-tree networks,which have

redundant links between each peer. DLB utilizes link usage and makes decisions based on real-time traffic statistics obtained via the OpenFlow protocol. We have implemented DLB as bash-script program. It has two main functions: monitoring traffic and pushing flows. With floodlight controller, we use the Mininet network emulator to evaluate the DLB. The results show that our DLB maintaining high link utilization and avoiding congestion.

References

- [1] Li, Y., & Pan, D. OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support.
- [2] B. Lantz, B. Heller & N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks.
- [3] Mininet. mininet.org
- [4] Iperf. <https://iperf.fr/>
- [5] Floodlight. <http://www.projectfloodlight.org/>