**TEL510E**

**Telecommunication Network Planning and Management**

**2015-2016 Spring**

# OpenFlow based Load Balancing for Fat-Tree Networks

**Detailed Design Report**

**Büşra TEMEL - 504141337**

**Şehriban ALPAYDIN – 504151346**

**Emre YALÇIN - 504121341**

**Fat-Tree Network Topology**

To start with, any network makes sense only if it is efficiently connected, so that, all end nodes can communicate to all other end nodes. In a switched fabric 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. By cleverly connecting switching elements and forming a topology, a network can interconnect an impressive amount of endpoints. Such network is a tree, and processors are connected to the bottom layer. The distinctive feature of a fat-tree is that for any switch, the number of links going down to its siblings is equal to the number of links going up to its parent in the upper level. Therefore, the links get "fatter" towards the top of the tree, and switch in the root of the tree has most links compared to any other switch below it as seen in Fig.1.

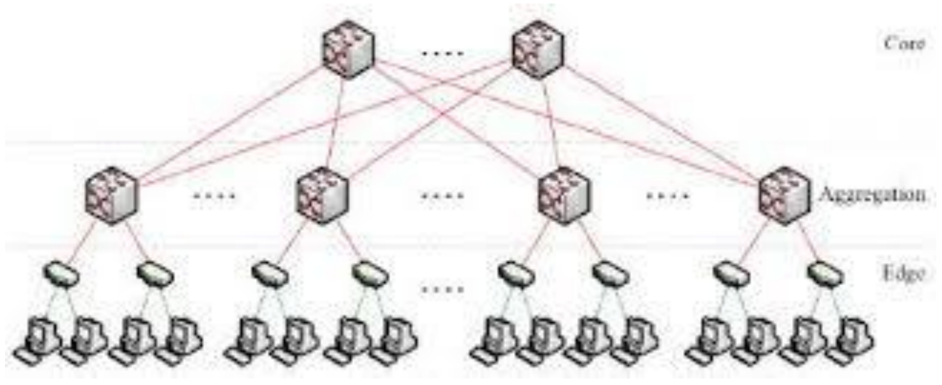

*Fig.1 : Fat-Tree topology*

**Mininet - Network topology emulator**

Mininet is a network emulator. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs. The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middlebox, with a given amount of queuing.

Mininet already has its own topology samples and one of them can be created as follow:

*$ sudo mn --topo=tree,2*
*,2*

This command creates a 2 level tree topology and these topology codes are stored as a *.py* file under */mininet directory*. Thus, it was replaced with the new one which was written in python for the fat-tree topology as shown below:

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.node import Controller, RemoteController
from mininet.cli import CLI

class SingleSwitchTopo(Topo):
	def build(self, n=2):
		host1 = self.addHost( ‘h1’ )
		host2 = self.addHost( ‘h2’ )
		host3 = self.addHost( ‘h3’ )
		host4 = self.addHost( ‘h4’ )


		switch1 = self.addSwitch( ‘s1’ )
		switch2 = self.addSwitch( ‘s2’ )
		switch3 = self.addSwitch( ‘s3’ )
		switch4 = self.addSwitch( ‘s4’ )
		switch5 = self.addSwitch( ‘s5’ )
		switch6 = self.addSwitch( ‘s6’ )


		# Add links
		self.addLink( host1, switch3, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( host2, switch4, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( host3, switch5, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( host4, switch6, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )

		self.addLink( switch1, switch3, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( switch1, switch4, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( switch1, switch5, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( switch1, switch6, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( switch2, switch3, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( switch2, switch4, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( switch2, switch5, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )
		self.addLink( switch2, switch6, bw=1000, delay='0ms', loss=0, max_queue_size=10000, use_htb=True )

def netStart():
	"Create network and run simple performance test"
	topo = SingleSwitchTopo(n=4)
	net = Mininet(topo=topo, controller=None, link=TCLink)
	net.addController( 'c0', controller=RemoteController, ip='192.168.1.2', port=6653 )
	net.start()
	CLI( net )
	#net.stop()

if __name__ == '__main__':
	setLogLevel('info')
	netStart()
```

This code represent for just as the topology given in Fig. 1 can be run by typing the following commands:

```
cd ./mininet/custom/
sudo ./topoGI.py
```

**Software Defined networking(SDN)**

The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices, is what called a Software Defined networking (SDN). It is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forwards traffic to the selected destination (the data plane), thus enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol is a foundational element for building SDN solutions. A typical SDN configuration can be thought of as shown in the Fig.2.

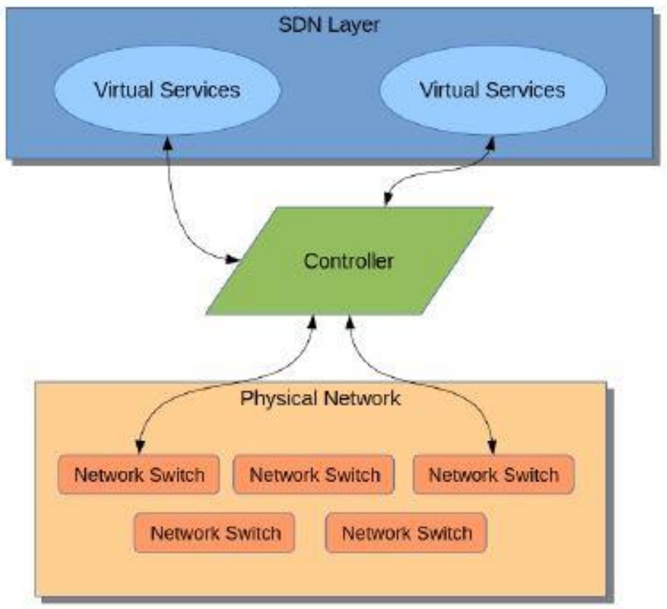

*Fig.2 : The controller connects the physical network to the SDN layer*

**OpenFlow - A type of SDN mechanism**

An SDN architecture has two distinct networking APIs: northbound and southbound. OpenFlow is a southbound API, which allows the path of network packets through the network of switches to be determined by software running on multiple routers.

***How does Open Flow work?***

In a classical router or switch, the fast packet forwarding (data path) and the high level routing decisions (control path) occur on the same device. An OpenFlow Switch separates these two functions. The data path portion still resides on the switch, while high-level routing decisions are moved to a separate controller, typically a standard server. The OpenFlow Switch and Controller communicate via the OpenFlow protocol, which defines messages, such as packet-received, send-packet-out, modifyforwarding- table, and get-stats. The data path of an OpenFlow Switch presents a clean flow table abstraction; each flow table entry contains a set of packet fields to match, and an action (such as send-out-port, modify-field, or drop). When an OpenFlow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends this packet to the controller. The controller then makes a decision on how to handle this packet. It can drop the packet, or it can add a flow entry directing the switch on how to forward similar packets in the future. Fig.3 shown below is an idealized open flow switch. Flow table is controlled by a remote controller via secure channel.

**OpenFlow message structure:**

OpenFlow has a predefined message structure. These messages are called OF messages that are passed between the controller and the OpenFlow switch. Every switch will have it's own Flow Table as opposite to MAC address table and routing table in the traditional switches. Every new incoming frame/packet is matched against the existing Flow table on the switch and take the necessary action specified in the 'Action' field of the Flow table. You can see the OF message structure in Fig.4 below.

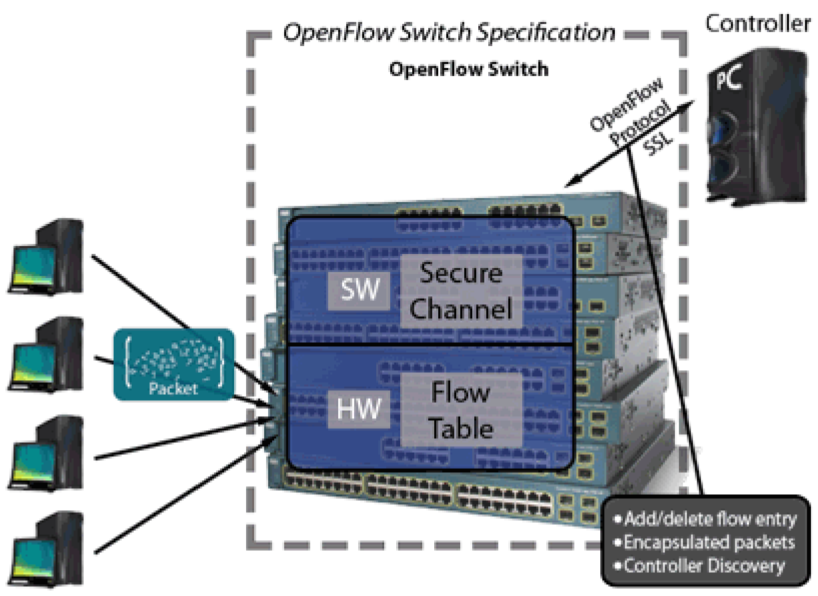

*Fig.3 : idealized open flow switch*

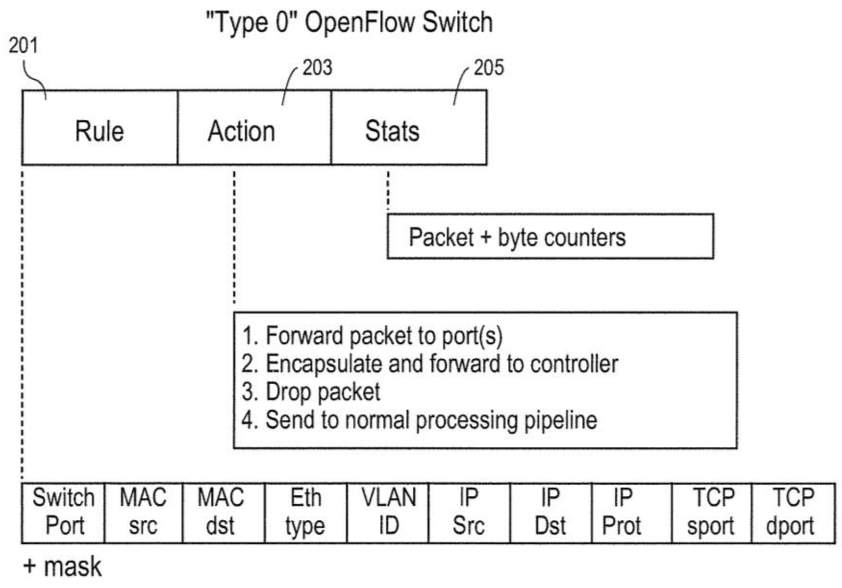

*Fig.4 : OF message format*

**Floodlight Controller**

The second part of the project is coding the load balancing algorithm on floodlight. Floodlight is a remote controller for mininnet emulator so remote command should be used while connecting as follows:

*$ sudo mn --controller=remote*

The port and IP address are not required due to running both of the controller and emulator locally.

After this steps, the routing configuration should be done. There are lots of running modules in floodlight based on java. The *ForwardingBase.java* class is used to achieve the desired load balance process.

***ForwardingBase.java***

This class implements the *IdeviceListener* interface. Implementors of this interface can receive updates from the device manager about the state of devices under its control.

It also use another subclass is called *Forwarding.* This subclass implements the forwarding module. This module determines how to handle all PACKET_IN messages according to the routing decision.

It extends a abstract class *OFModule* which is mother of all Torpedo modules. OFModule represents a reusable unit in defining a custom controller.

The most interested method in this class is ***pushRoute*** method. This method pushes routes from back to front and it returns ***true*** if the source switch is included in this route. It uses some parameters as follows:

| | |
|---|---|
| **conn** | - the connection to switch |
| **route** | - a route to push |
| **match** | - openFlow fields to match on |
| **wildcard_hints** | - wildcard information as integer |
| **pi** | - packetin |
| **pinSwitch** | - the switch of packetin |
| **cookie** | - the cookie to set in each flow_mod |
| **cntx** | - the MessageContext |

**reqeustFlowRemovedNotifn** - true when the switch would send a flow mod removal notification when the flow mod expires

**doFlush** - true when the flow mod would be immediately written to the switch

**flowModCommand** - flow mod. command to use, e.g. OFFlowMod.OFPFC_ADD, OFFlowMod.OFPFC_MODIFY etc.

The above method pushRoute used for changing the path. In Floodlight, each link has a same cost. For the fat-tree topology all route from host to host will have same cost. Based on this information, it could be implemented a statistic module which could be used for getting how much consumption in each link, but in this paper we assume that we obtain this statistic thus we directly give the information to the switch which port will be more available to forward packets. According to this available port knowledge, we will redeploy the ***pushRoute*** method to obtain desired routing process.