

---

# Enabling OpenFlow v1.0 on Mikrotik RouterBoard 750GL: A Tutorial

---

Deployment and Testing of a SDN Switch

Version 1.0

OCTOBER 31, 2014

Uzzam Javed <13mseeujaved@seecs.edu.pk>  
Azeem Iqbal <13mseeaiqbal@seecs.edu.pk>

Applied Network and Data Science Research (AN-DASH) Group

School of Electrical Engineering and Computer Science (SEECS)  
National University of Sciences and Technology (NUST), Islamabad, Pakistan

# Document History

Version	Date	Author(s)	Changelog
1.0	October 31, 2014	Uzzam Javed, Azeem Iqbal	Initial version .

# Acknowledgment

We would like to thank our supervisor, Dr. Muhammad Usman Ilyas, for guiding us and we also appreciate the efforts made by Zaafar Ahmed, SysNet Lab Pakistan, in helping us installing OpenWRT.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Deployment of OpenFlow enabled Network</b>	<b>3</b>
2.1	OpenFlow Switch . . . . .	3
2.1.1	OpenFlow Switch Platform . . . . .	3
2.1.2	Installing OpenWRT with OpenFlow Package . . . . .	4
2.1.2.1	STEP 1: Build Files . . . . .	4
2.1.2.2	STEP 2: Load Ramdisk Image . . . . .	7
2.1.2.3	STEP 3: Flash Target Image . . . . .	9
2.1.2.3.1	Option 1: Flash files by downloading from server . . . . .	9
2.1.2.3.2	Option 2: Flash files using SCP . . . . .	10
2.2	OpenFlow Contoller . . . . .	12
2.2.1	POX Controller . . . . .	12
2.2.1.1	Installation Procedure . . . . .	12
2.2.2	Floodlight Controller . . . . .	12
2.2.2.1	Installation Procedure . . . . .	12
<b>3</b>	<b>Testing of OpenFlow enabled Network</b>	<b>14</b>
3.1	Starting OpenFlow Protocol . . . . .	14
3.2	Experimenting with OpenFlow Protocol . . . . .	17
	<b>Bibliography</b>	<b>17</b>

# List of Figures

1.1	SDN Architecture . . . . .	2
2.1	Steps for installing OpenWRT with OpenFlow Support. . . . .	4
2.2	OpenWRT's package selection menu . . . . .	5
2.3	Terminal output after starting dnsmasq . . . . .	8
2.4	Wireshark showing file tranfer . . . . .	8
2.5	CLI showing file tranfer . . . . .	8
2.6	Network configuration file . . . . .	10
2.7	View of POX controller after being started . . . . .	12
2.8	View of Floodlight controller after being started . . . . .	13
3.1	Warning message . . . . .	14
3.2	OpenFlow configuration file . . . . .	15
3.3	POX Controller Window . . . . .	16
3.4	Router Window . . . . .	16
3.5	Wireshark showing exchange of messages . . . . .	17

# Chapter 1

## Introduction

The rapid evolution of new technologies, such as cloud computing, have complicated the way in which traditional networking was done [1]. Software-Defined Networking (SDN), a new networking paradigm, provides a solution for it by smartly managing and configuring the network elements [2]. It makes the network programmable by separating the control plane of the network from the data plane. This leaves behind data plane having only switches with packet forwarding capabilities. SDN provides the network administrators more control over the entire network through the centralized control plane running in software. It has also allowed researchers to experiment on the real world networks without causing any interference to their traffic [3].

The centralized control plane consists of a southbound Application Programming Interface (API), for communication with the networks hardware, and a northbound API, for communication with applications of the network [4]. OpenFlow is the main southbound API, promoted by the Open Networking Foundation (ONF). It aims to provide interoperability between networking equipments of different vendors, as previously only proprietary attempts has been made in this regard. This effort improves the performance of SDNs [1]. Figure 1.1 depicts the structure of a SDN network having OpenFlow as its southbound API.

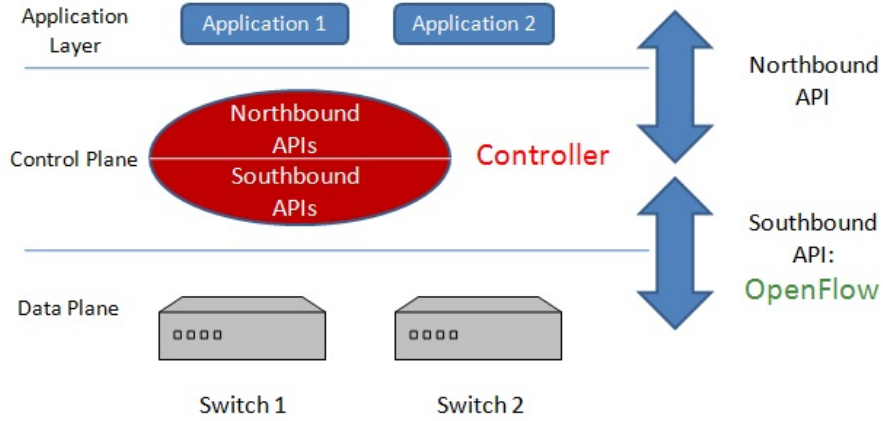


Figure 1.1: SDN Architecture

Controller and OpenFlow switch communicate through the OpenFlow protocol. The OpenFlow switch has a flow table entry for the packets arriving; if not, then the respective packet is directed to the controller for further processing [5].

Further information about the OpenFlow protocol can be found the white paper titled 'OpenFlow: enabling innovation in campus networks' [6], and one could visit <https://www.opennetworking.org/sdn-resources/onf-specifications> [7] for the OpenFlow switch specifications.

## Chapter 2

# Deployment of OpenFlow enabled Network

### 2.1 OpenFlow Switch

OpenFlow switch can be of two types, software based or hardware based. The choice of switch varies according to the requirement of your network. Software based OpenFlow switches includes Reference Linux Switch, NetFPGA Switch, Open vSwitch, and OpenWRT. HP Procurve 5400zl, NEC PF5240 and Pronto are the hardware based OpenFlow switches. The OpenFlow switch and the controller have to be checked for the support of the OpenFlow version you desire to work on [8].

#### 2.1.1 OpenFlow Switch Platform

We used a software based implementation of OpenFlow switch by porting OpenFlow on a router running OpenWRT and turning it into a OpenFlow enabled router.

We used **Mikrotik RouterBoard 750GL** [9] as the router for our choice, which is easily available and is an economical router. It has 64 MB RAM and a 64 MB Flash Memory, which contains the full Operating System image. It has 5 wired Gigabit ports and also supports VLAN configuration.

Mikrotik RouterBoard 750GLs RouterOS (factory firmware) supports OpenFlow but still we installed OpenWRT over it. OpenWRT is basically embedded Linux distribution, providing more flexibility to the router. With time new versions of OpenWRT are released. The history regarding different versions of OpenWRT can be found on <http://wiki.openwrt.org/about/history> [10]. Other routers that support OpenWRT can be found on <http://wiki.openwrt.org/toh/start> [11].



## 2.1.2 Installing OpenWRT with OpenFlow Package

The whole procedure is a three step process, depicted the figure below. In this section each step is individually explained in detail. They are all implemented in the terminal window of Ubuntu.

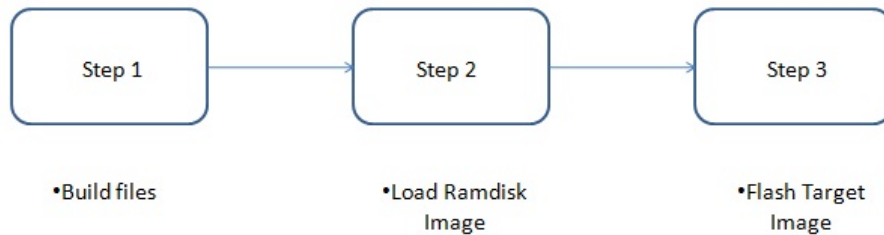


Figure 2.1: Steps for installing OpenWRT with OpenFlow Support.

### 2.1.2.1 STEP 1: Build Files

The process first requires to build files that need to be ported in the router. The computer used to build files has 64 bit Ubuntu 12.04 LTS. Ubuntu 32 bit can also be used. Following steps explains this process [12][13]:

Note: Do not implement step 1 process as a root user.

#### Installation of prerequisite packages

- `$ apt-get install build-essential binutils flex bison autoconf gettext texinfo sharutils \ subversion libncurses5-dev ncurses-term zlib1g-dev gawk`
- `$ sudo apt-get update`
- `$ sudo apt-get install git-core build-essential`
- `$ sudo apt-get install subversion`

#### Downloading OpenWRT files

- `$ git clone git://git.openwrt.org/openwrt.git`
- A folder of the name 'openwrt' will be made in your home directory
- `$ cd openwrt`
- `$ ./scripts/feeds update -a`
- `$ ./scripts/feeds install -a`

## Compilation of OpenWRT

- `$ make defconfig`
- `$ make prereq`
- `$ make menuconfig`

Following menu will open up in front of you after the previous command.

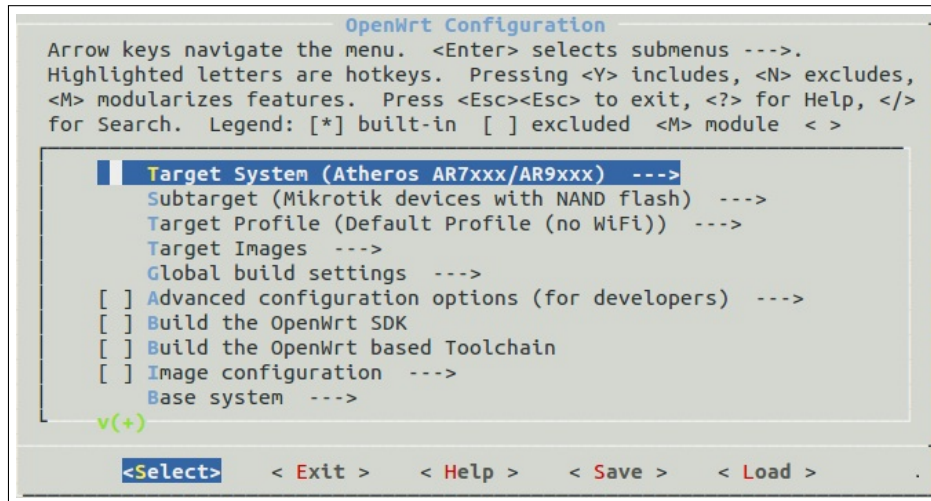


Figure 2.2: OpenWRT's package selection menu

Following packages need to be selected:

1. *Target System: Atheros AR71xx*
2. *Subtarget: Devices with NAND flash (mostly Mikrotik)*
3. *Target Images: ramdisk and tar.gz*

For Web Interface

4. *LuCI → Collections: luci and luci-ssl*

Now, select *Exit* and save your configurations.

- `$ make`

This command will take quite some time, and if it fails in between, for some reason, execute it again.

## Adding OpenFlow extension

*Pantou* converts a router into OpenFlow-enabled switch. Based on the Stanford reference implementation (userspace), the OpenFlow package is implemented over OpenWRT [14]. *Pantou* provides OpenFlow version 1.0 support, which is a stable release and is supported by many controllers.

- `$ cd ~/openwrt/`
- `$ git clone git://gitosis.stanford.edu/openflow-openwrt`

A folder will be made in your working directory with the name of ‘openflow-openwrt’. In it there will be a folder of ‘openflow-1.0’. Copy this folder to the destination as directed by the command in the next line.

- `$ cp -a openwrt/openflow-openwrt/openflow-1.0 openwrt/package/network/services`

Now, execute

- `$ make menuconfig`

Following packages need to selected:

1. *Network: openflow*
2. *Kernel modules → Network Support: kmod-tun*

Although, *Target System*, *Subtarget* and *Target Images* are selected already; but just in case verify they are still selected.

Now, select *Exit* and save the your configurations.

#### For Queuing Support

Execute,

- `$ make kernel.menuconfig`

A new menu will open up. Select,

1. *Network Support → Networking options → under the heading of Queueing/Scheduling : Hierarchical Token Bucket (HTB)*

Again select *Exit* and save the your configurations, and execute

- `$ make`

The image files created will be situated in your working directory, in the folder of ‘ar71xx’ located in ‘bin’ folder. The files that you will be requiring are the following:

1. *openwrt-ar71xx-mikrotik-vmlinux-initramfs.elf*
2. *openwrt-ar71xx-mikrotik-vmlinux-lzma.elf*
3. *openwrt-ar71xx-mikrotik-DefaultNoWifi-rootfs.tar.gz*

- Rename the file `openwrt-ar71xx-mikrotik-DefaultNoWifi-rootfs.tar.gz` to `openwrt-ar71xx-mikrotik-rootfs.tar.gz` [15].
- `$ mv bin/ar71xx/openwrt-ar71xx-mikrotik-DefaultNoWifi-rootfs.tar.gz bin/ar71xx/openwrt-ar71xx-mikrotik-rootfs.tar.gz`

Note: The files created through this process contains basic packages, if one wants more package he/she could install them from the menu of ‘make menu-config’ command. We saved the files created through this process on [<http://andash.seecs.nust.edu.pk/sdnfiles/>](http://andash.seecs.nust.edu.pk/sdnfiles/) [16]. If one wants generic files, without OpenFlow support, for any OpenWRT supported router, it can be found on [<https://downloads.openwrt.org/>](https://downloads.openwrt.org/) [17].

### 2.1.2.2 STEP 2: Load Ramdisk Image

Step 2 involves booting the Ramdisk image of OpenWRT. After loading this image OpenWRT will be running on this router until router is powered on. Following steps will guide you in installing the Ramdisk image [18].

- Open a root shell.
- `$ sudo -s`
- Place the three files created in the previous step in a separate folder.
- Connect your workstation to the router at its port 1, through a ethernet cable. Do not power up the router yet.
- `# service network-manager stop`
- `# ifconfig eth0 192.168.1.99`
- Start Wireshark to see the file transfer and select interface eth0. If wire-shark is not installed, install it with the command ‘apt-get install wire-shark’.
- `# wireshark &`
- Starting DHCP and TFTP server.
- `# dnsmasq -i eth0 --dhcp-range=192.168.1.100,192.168.1.200 --dhcp--boot=openwrt-ar71xx-vmlinux-initramfs.elf --enable-tftp -- tftp-root=<directory path>-d -u root -p0 -K --log-dhcp --bootp-dynamic`
- In our case the directory path is `/home/uzzam/Downloads/`, so the above instruction is executed as follows:

```
dnsmasq -i eth0 --dhcp-range=192.168.1.100,192.168.1.200 --dhcp-
boot=openwrt-ar71xx-mikrotik-vmlinux-initramfs.elf --enable-tftp --
tftp-root=/home/uzzam/Downloads/ -d -u root -p0 -K --log-dhcp --bootp-dynamic
```

initramfs.elf  
file will be  
transferred  
to the  
router  
from the  
directory

Figure 2.3 below show the terminal output when dnsmasq was started in our case.

```
dnsmasq: started, version 2.59 DNS disabled
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n DHCP TFTP conntrack IDN
dnsmasq-dhcp: DHCP, IP range 192.168.1.100 -- 192.168.1.200, lease time 1h
dnsmasq-tftp: TFTP root is /home/uzzam/Downloads/
```

Figure 2.3: Terminal output after starting dnsmasq

- With the reset button kept pressed, power on the router. Keep the reset button pressed until the file is transferred, which can be verified by the following messages of wireshark and CLI respectively.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	BOOTP	342	Boot Request from d4:ca:6d:03:89:1e (Routerbo_03:89:1e)
2	0.025817	192.168.1.99	192.168.1.147	BOOTP	342	Boot Reply
3	0.026059	Routerbo_03:89:1e	Broadcast	ARP	64	Who has 192.168.1.99? Tell 192.168.1.147
4	0.026075	Wistron_79:88:dd	Routerbo_03:89:1e	ARP	42	192.168.1.99 is at 08:1d:72:79:88:dd
5	0.026192	192.168.1.147	192.168.1.99	TFTP	109	Read Request, File: openwrt-ar71xx-mikrotik-vmlinux-initramfs.elf
6	0.026305	192.168.1.99	192.168.1.147	TFTP	57	Option Acknowledgement, blksize=1452\000
7	0.026392	192.168.1.147	192.168.1.99	TFTP	64	Acknowledgement, Block: 0
8	0.026437	192.168.1.99	192.168.1.147	TFTP	1498	Data Packet, Block: 1
9	0.027035	192.168.1.147	192.168.1.99	TFTP	64	Acknowledgement, Block: 1
10	0.027084	192.168.1.99	192.168.1.147	TFTP	1498	Data Packet, Block: 2
11	0.027681	192.168.1.147	192.168.1.99	TFTP	64	Acknowledgement, Block: 2
12	0.027726	192.168.1.99	192.168.1.147	TFTP	1498	Data Packet, Block: 3

Figure 2.4: Wireshark showing file tranfer

```
dnsmasq-tftp: sent /home/uzzam/Downloads/openwrt-ar71xx-mikrotik-vmlinux
fs.elf to 192.168.1.147
```

Figure 2.5: CLI showing file tranfer

- `# service network-manager start`
- Put the ethernet cable to port 2 of the router. Make sure your wired network connection has IPv4 Setting which is ‘Automatic (DHCP)’.
- Access OpenWRT through telnet.
- `# telnet 192.168.1.1`

### 2.1.2.3 STEP 3: Flash Target Image

**Warning:** Step 3 will erase your original firmware permanently, make sure to backup your firmware, configurations and license.

Step 3 can be done in two ways, both options are explained in the coming sections.

#### 2.1.2.3.1 Option 1: Flash files by downloading from server

Two files that were built previously, named *openwrt-ar71xx-mikrotik-rootfs.tar.gz* and *openwrt-ar71xx-mikrotik-vmlinux-lzma.elf*, are to be placed on a FTP or a HTTP server from which they will be downloaded. Following steps are in continuation of step 2.

- Put the internet cable in router's port 1.

For Internet Connection with Dynamic IP

- `# wget2nand <http or ftp server address>`
- In our case http server was `<http://andash.seecs.nust.edu.pk/sdnfiles/>`, so the above instruction was executed as follows:

```
wget2nand http://andash.seecs.nust.edu.pk/sdnfiles/
```

- If this command fails, run `'rm -rf /tmp/wget2nand'` and try again to download the file.
- `# reboot`

For Internet Connection with Static IP

- `# cd /etc/config`
- `# vi network`
- A network configuration file will open. Edit the wan interface section by pressing 'i' according to your network settings. An example to it is shown in the figure below.

```

config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'fdfe:6170:b729::/48'

config interface 'lan'
    option ifname 'eth0.1'
    option force_link '1'
    option type 'bridge'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

config interface 'wan'
    option ifname 'eth0.2'
    option proto 'static'
    option ipaddr '10.3.93.48'
    option netmask '255.255.255.0'
    option gateway '10.3.93.1'
    option dns '8.8.8.8'

```

Figure 2.6: Network configuration file

- After completing the changes, press ‘Esc key’ then press ‘:wq’. This will save the changes in your document. If you do not want to save the changes, press ‘Esc key’ then press ‘:q!’.
- `# /etc/init.d/network reload`
- `# wget2nand <http or ftp server address>`
- In our case the above command was as follows:

```
wget2nand http://andash.seecs.nust.edu.pk/sdnfiles/
```

- If this command fails, run ‘rm -rf /tmp/wget2nand’ and try again to download the file.
- `# reboot`

#### 2.1.2.3.2 Option 2: Flash files using SCP

If you are implementing Step 3 through this method, then following steps are to be implemented right after the procedure of step 2 has ended within the same terminal window. Following steps will guide you through this procedure [19].

- Erase the content of 'kernel' and 'rootfs'.
- `# mtd erase kernel`
- `# mtd erase rootfs`
- Create yaffs2 partition.
- `# mkdir /mnt/kernel /mnt/rootfs`
- `# mount -t yaffs2 /dev/mtdblock1 /mnt/kernel`
- `# mount -t yaffs2 /dev/mtdblock2 /mnt/rootfs`

### Copying image files to the router

- Set the password for the router for secure file transfer.
- `# passwd`

Without closing the current terminal, open a new terminal window and implement the following command:

- `$ scp <directory path>openwrt-ar71xx-mikrotik-rootfs.tar.gz <directory path>openwrt-ar71xx-mikrotik-vmlinux-lzma.elf root@192.168.1.1:/tmp`
- In our case both files were located in `/openwrt_os`, so the instruction was executed as

```
scp /openwrt_os/openwrt-ar71xx-mikrotik-rootfs.tar.gz /openwrt_os/
openwrt-ar71xx-mikrotik-vmlinux-lzma.elf root@192.168.1.1:/tmp
```

### Flashing the files into the router

Continue typing the following commands into the previous terminal window.

- For kernel file
- `# mv /tmp/openwrt-ar71xx-mikrotik-vmlinux-lzma.elf /mnt/kernel/kernel`
- `# chmod +x /mnt/kernel/kernel`
- `# umount /mnt/kernel`
- For root filesystem
- `# cd /mnt/rootfs`
- `# tar -xvzf /tmp/openwrt-ar71xx-mikrotik-rootfs.tar.gz`
- `# cd /`
- `# umount /mnt/rootfs`
- `# reboot`



## 2.2 OpenFlow Controller

OpenFlow controller manages the flows of the data path in a SDN, using OpenFlow protocol. Many controllers are currently prevailing in the market written in different programming languages, including C, C++, Python and Java [20].

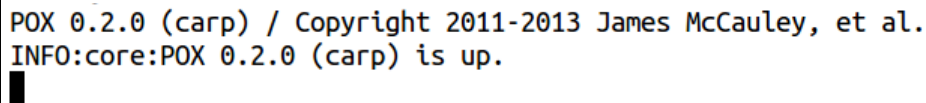
### 2.2.1 POX Controller

POX controller is an open source, Python based SDN controller [21]. It is the sister project of NOX, C based SDN controller.

#### 2.2.1.1 Installation Procedure

- `$ git clone http://github.com/noxrepo/pox`
- `$ cd pox`
- Start POX controller with the following command:
- `$ ./pox.py`

Following figure shows the output of the terminal after POX controller has started.

A terminal window showing the output of the POX controller startup. The text displayed is: "POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al. INFO:core:POX 0.2.0 (carp) is up." followed by a cursor icon.

```
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
```

Figure 2.7: View of POX controller after being started

POX controller can also be started by adding other arguments to it. These options in arguments can be found on <https://openflow.stanford.edu/display/ONL/POX+Wiki> [22].

### 2.2.2 Floodlight Controller

Floodlight is a Java based, Apache-licensed OpenFlow Controller. A number of industry partners are involved in its development, including engineers from Big Switch Networks [23].

#### 2.2.2.1 Installation Procedure

- `$ sudo apt-get install build-essential default-jdk ant python-dev eclipse`
- `$ git clone git://github.com/floodlight/floodlight.git`
- `$ cd floodlight`
- `$ ant`
- Start Floodlight controller with the following command:
- `$ java -jar target/floodlight.jar`

- GUI on the web browser can be viewed at `<http://localhost:8080/ui/index.html>` [24].

Following figure shows the output of the terminal after Floodlight controller has started.

```
16:24:46.602 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting autoportfast feature to OFF
16:24:46.717 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet configured; using fallback local configuration
16:24:46.719 INFO [o.s.s.i.SyncManager:main] [32767] Updating sync configuration ClusterConfig [allNodes={32767=Node [hostname=localhost, port=6642, nodeId=32767, domainId=32767]}, authScheme=CHALLENGE_RESPONSE, keyStorePath=/etc/floodlight/auth_credentials.jceks, keyStorePassword is unset]
16:24:46.880 INFO [o.s.s.i.r.RPCService:main] Listening for internal floodlight RPC on localhost/127.0.0.1:6642
16:24:47.273 INFO [n.f.c.i.Controller:main] Listening for switch connections on 0.0.0.0/0.0.0.0:6633
16:24:53.957 INFO [n.f.j.JythonServer:debugserver-main] Starting DebugServer on :6655
```

Figure 2.8: View of Floodlight controller after being started

## Chapter 3

# Testing of OpenFlow enabled Network

### 3.1 Starting OpenFlow Protocol

Open up a new terminal window in Ubuntu. Follow the following steps.

- `$ telnet 192.168.1.1`

#### Setting Password for SSH communication

- `# passwd`
- Access the router through ssh after exiting from it. Telnet won't work as password has been set.
- `$ sudo ssh 192.168.1.1`

Following message will appear on your terminal.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@  WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!  @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
5f:5e:75:0e:f7:97:8e:fb:99:21:9b:50:ba:01:42:a4.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending RSA key in /root/.ssh/known_hosts:1
  remove with: ssh-keygen -f "/root/.ssh/known_hosts" -R 192.168.1.1
RSA host key for 192.168.1.1 has changed and you have requested strict checking.
Host key verification failed.
```

Figure 3.1: Warning message

Run,

- `$ sudo rm -f /root/.ssh/known_host`
- Now ssh again into the router.

## Additional Router Settings

1. Copy functions.sh from lib to etc [25].
2. `# cp /lib/functions.sh /etc`
3. Add --no-slicing parameter to /lib/openflow/ofswitch.sh [26]. After the addition of this parameter, the modified lines should look as follows.

```
if [[ "$mode" == "inband" ]]
then
    echo "Configuring OpenFlow switch for inband control"
    [ -n "$dpid" ] && {
ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing
--local-port=tap:tap0 "--pidfile=$pidfile" -d "$dpid" &
    } || {
ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing
--local-port=tap:tap0 "--pidfile=$pidfile" &
    }
else
    echo "Configuring OpenFlow switch for out-of-band control"
    [ -n "$dpid" ] && {
ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing
--no-local-port "--pidfile=$pidfile" -d "$dpid" &
    } || {
ofdatapath punix:/var/run/dp0.sock -i "$dpports" --no-slicing
--no-local-port "--pidfile=$pidfile" &
    }
}
```

## Adding IP address of Controller

- `# cd /etc/config`
- `# vi openflow`
- A configuration file will open. Edit the controller IP address to the IP address of the computer on which controller is running. An example to it is shown in the figure below.

```
config 'ofswitch'
    option 'dp' 'dp0'
    option 'ofports' 'eth0.1 eth0.2'
    option 'ofctl' 'tcp:192.168.1.248:6633'
    option 'mode' 'outofband'
```

Figure 3.2: OpenFlow configuration file

Note: Add only those interfaces in the OpenFlow configuration file that are active. On a fresh installation of OpenWRT, only two interfaces are active.

## Connecting Controller

Assuming Controller is running, enter

- `# /etc/init.d/openflow start`

CLI view of controller and router windows are shown below respectively.

```
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
WARNING:openflow.of_01:<class 'pox.openflow.PacketIn'> raised on dummy OpenFlow
nexus
WARNING:openflow.of_01:<class 'pox.openflow.PacketIn'> raised on dummy OpenFlow
nexus
WARNING:openflow.of_01:<class 'pox.openflow.PacketIn'> raised on dummy OpenFlow
nexus
INFO:openflow.of_01:[00-23-20-65-ab-7c 1] connected
INFO:openflow.of_01:[00-23-20-65-ab-7c 1] closed
WARNING:openflow.of_01:<class 'pox.openflow.PacketIn'> raised on dummy OpenFlow
nexus
INFO:openflow.of_01:[00-23-20-7d-ae-7b 2] connected
█
```

Figure 3.3: POX Controller Window

```
root@OpenWrt:~# /etc/init.d/openflow start
eth0.1,eth0.2
Configuring OpenFlow switch for out-of-band control
Oct 31 13:25:58|00001|datapath|ERR|eth0.2 device has assigned IPv6 address fe80::d6ca:6dff:fe03:891e
No need for further configuration for out-of-band control
Oct 31 13:26:01|00001|vlog|INFO|opened log file /var/log/ofprotocol.log
Oct 31 13:26:01|00002|secchan|INFO|OpenFlow reference implementation version 1.0.0
Oct 31 13:26:01|00003|secchan|INFO|OpenFlow protocol version 0x01
Oct 31 13:26:01|00004|secchan|WARN|new management connection will receive asynchronous messages
root@OpenWrt:~# Oct 31 13:26:01|00005|rconn|INFO|unix:/var/run/dp0.sock: connecting...
Oct 31 13:26:01|00006|rconn|INFO|tcp:192.168.1.248:6633: connecting...
Oct 31 13:26:01|00007|rconn|INFO|unix:/var/run/dp0.sock: connected
Oct 31 13:26:01|00008|port_watcher|INFO|Datapath id is 0023207dae7b
Oct 31 13:26:01|00009|rconn|INFO|tcp:192.168.1.248:6633: connected
█
```

Figure 3.4: Router Window

## 3.2 Experimenting with OpenFlow Protocol

Start Wireshark on the workstation on which controller is running, by adding the interface on which router is connected to the workstation. It will show the exchange of messages between controller and router. Figure below shows the exchange of messages between the controller and router.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=1 Win=189 Len=0 TSval=4583406 TSecr=25051
2	0.000799000	192.168.1.1	192.168.1.248	TCP	150	49525 > 6633 [PSH, ACK] Seq=1 Ack=1 Win=1825 Len=84 TSval=25051 TSecr=4583406
3	0.001272000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=85 Win=189 Len=0 TSval=4583406 TSecr=25051
4	0.001978000	192.168.1.1	192.168.1.248	TCP	150	49525 > 6633 [PSH, ACK] Seq=85 Ack=1 Win=1825 Len=84 TSval=25051 TSecr=4583406
5	0.002448000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=169 Win=189 Len=0 TSval=4583407 TSecr=25051
6	0.003153000	192.168.1.1	192.168.1.248	TCP	150	49525 > 6633 [PSH, ACK] Seq=169 Ack=1 Win=1825 Len=84 TSval=25051 TSecr=4583407
7	0.003616000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=253 Win=189 Len=0 TSval=4583407 TSecr=25051
8	0.004323000	192.168.1.1	192.168.1.248	TCP	150	49525 > 6633 [PSH, ACK] Seq=253 Ack=1 Win=1825 Len=84 TSval=25051 TSecr=4583407
9	0.004789000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=337 Win=189 Len=0 TSval=4583407 TSecr=25051
10	0.005508000	192.168.1.1	192.168.1.248	TCP	150	49525 > 6633 [PSH, ACK] Seq=337 Ack=1 Win=1825 Len=84 TSval=25052 TSecr=4583407
11	0.005950000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=421 Win=189 Len=0 TSval=4583407 TSecr=25052
12	0.006655000	192.168.1.1	192.168.1.248	TCP	150	49525 > 6633 [PSH, ACK] Seq=421 Ack=1 Win=1825 Len=84 TSval=25052 TSecr=4583407
13	0.007099000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=505 Win=189 Len=0 TSval=4583408 TSecr=25052
14	0.007805000	192.168.1.1	192.168.1.248	TCP	150	49525 > 6633 [PSH, ACK] Seq=505 Ack=1 Win=1825 Len=84 TSval=25052 TSecr=4583408
15	0.008244000	192.168.1.248	192.168.1.1	TCP	66	6633 > 49525 [ACK] Seq=1 Ack=589 Win=189 Len=0 TSval=4583408 TSecr=25052

Figure 3.5: Wireshark showing exchange of messages

*dpctl* utility can be further used for the purpose of experimenting and debugging as it communicates directly with the switch.

# Bibliography

- [1] Steven J Vaughan-Nichols. “OpenFlow: The next generation of the network?” In: *Computer* 44.8 (2011), pp. 13–15.
- [2] Hyojoon Kim and Nick Feamster. “Improving network management with software defined networking”. In: *Communications Magazine, IEEE* 51.2 (2013), pp. 114–119.
- [3] Andrea Bianco et al. “Openflow switching: Data plane performance”. In: *Communications (ICC), 2010 IEEE International Conference on*. IEEE. 2010, pp. 1–5.
- [4] Raj Jain and Subharthi Paul. “Network virtualization and software defined networking for cloud computing: a survey”. In: *Communications Magazine, IEEE* 51.11 (2013), pp. 24–31.
- [5] *About OpenFlow*. <<http://archive.openflow.org/wp/learnmore/>>.
- [6] Nick McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.
- [7] *ONF Specifications*. <<https://www.opennetworking.org/sdn-resources/onf-specifications>>.
- [8] *OpenFlow, Components*. <<http://archive.openflow.org/wp/openflow-components/>>.
- [9] *RB750GL, Product specifications*. <<http://routerboard.com/RB750GL>>.
- [10] *OpenWRT Version History*. <<http://wiki.openwrt.org/about/history>>.
- [11] *OpenWRT, Table of Hardware*. <<http://wiki.openwrt.org/toh/start>>.
- [12] *OpenWRT Buildroot Installation*. <<http://wiki.openwrt.org/doc/howto/buildroot.exigence>>.
- [13] *Pantou : OpenFlow 1.0 for OpenWRT*. <[http://archive.openflow.org/wk/index.php/Pantou:\\_OpenFlow\\_1.0\\_for\\_OpenWRT](http://archive.openflow.org/wk/index.php/Pantou:_OpenFlow_1.0_for_OpenWRT)>.
- [14] *SDNCentral, Stanford University Pantou (OpenWRT)*. <<https://www.sdncentral.com/projects/pantou-openwrt/>>.
- [15] *OpenWRT, Mikrotik RouterBoard RB750GL*. <<http://wiki.openwrt.org/toh/mikrotik/rb750gl>>.

- [16] *Router image files on AN-DASH Website.* <<http://andash.seecs.nust.edu.pk/sdnfiles/>>.
- [17] *OpenWRT Downloads.* <<https://downloads.openwrt.org/>>.
- [18] *Mikrotik forum, OpenWRT for RouterBoard.* <<http://forum.mikrotik.com/viewtopic.php?t=55285>>.
- [19] *Wolfs Tech Blog.* <<https://blog.poettner.de/2011/05/27/openwrt-trunk-on-mikrotik-routerboard-411750/>>.
- [20] *Controller Performance Comparisons.* <[http://archive.openflow.org/wk/index.php/Controller\\_Performance\\_Comparisons](http://archive.openflow.org/wk/index.php/Controller_Performance_Comparisons)>.
- [21] J Mccauley. *Pox: A python-based openflow controller.*
- [22] *OpenFlow @ Stanford, POX Wiki.* <<https://openflow.stanford.edu/display/ONL/POX+Wiki>>.
- [23] *Project Floodlight.* <<http://www.projectfloodlight.org/floodlight/>>.
- [24] *Floodlight OpenFlow Controller GUI Applet.* <<http://localhost:8080/ui/index.html>>.
- [25] Arman Keyoumars. *“Openflow 1.3 for OpenWRT in MikroTik Router-Board 750GL”.*
- [26] *OpenWRT Forum.* <<https://forum.openwrt.org/viewtopic.php?id=43742>>.