No. 2015001

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF ELECTRONIC INFORMATION AND COMMUNICATIONS
INTERNET TECHNOLOGY AND ENGINEERING R&D CENTER

TECHNICAL REPORT

Towards SDN-based Routing for Smart Homes

Author : Afzal Muhammad, Kenyon Nsunza
and Samuel Rutunda
Supervisor : Ass. Prof. Xiaojun Hei

May 1, 2015

Abstract

Based on extensive research and study of Software Defined networks (SDN), this project determines the construction and feasibility of a SDN based smart home router. The router comprises of OpenWRT operating system embedded with Openflow in order to allow separate control of data plane and control plane. The project developers starts from the ground up in order to utilize Software defined network for commercial purposes. We integrate Openflow with OpenWRT and determine the performance on various off the shelf devices.

Key words: Software defined networking, Smart home router, OpenFlow, Perfomance, traffic analysis, OpenWRT

Contents

List of Figures

Chapter 1 Introduction

1.1 Overview

Since the world war enormous efforts were made to invent new ways of communication, whether it's between people or machines [1] [2]. Then different methods were created to make the communication reliable, efficient and secure. All the major developments were seemed as a kind of race against other nation's developments. If we look at the advancements of past in the field of communication, We witness that it has evolved from simple words transfer by means of regulating voltage to complex packets which act as a letter with the address of sender and receiver on it. After the advent of packet data and internet various methods were devised to cope with the growing usage of such kind of networks. For example to increase the bandwidth new error correction algorithms were invented to reduce the noise of wire transfer. While now a day's fiber optic cables circle the globe which provide high bandwidth to various continents, countries and regions, in order to connect people and infrastructures in different parts of the world. Still the burden of usage is increasing constantly on such types of networking infrastructure. Hence inventing new method to cope with the problem becomes a necessity.

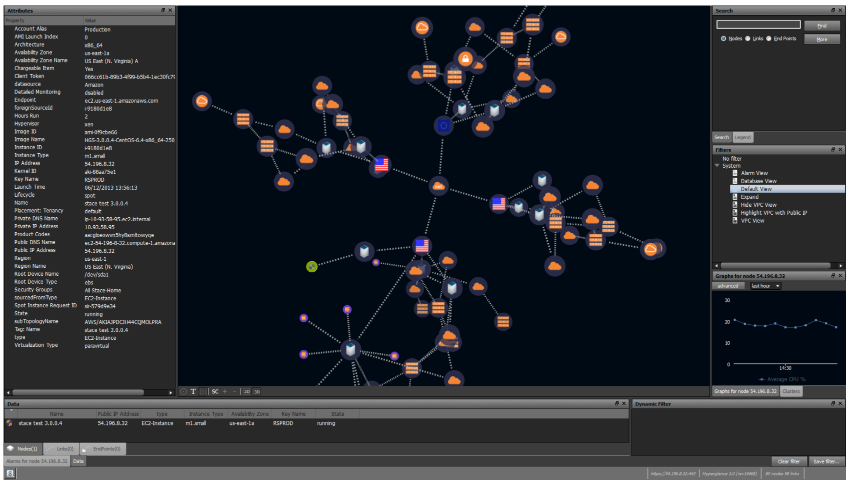
1.2 Software Defined Networks

Software Defined Network (SDN) is a way of networking computers by separating and taking control of the lower level functionality of networks such as data plane and control plane. The control function of such type of networking infrastructure is performed by a centralized controller (As shown in Fig.).

This tends to simplify the troubleshooting and establishment of fast and reliable networking by reporting the malfunctions of individual switches to the controller. SDN has been under development since the release of Java. Communication companies started developing SDN infrastructures and protocols as early as 1995. Openflow is an example of SDN protocol. SDN solves a lot of issues when it comes to everyday problems with networking such as traffic overload and disconnected links. SDN is praised for its adaptability to various scenarios of traffic flow, for example in case of heavy burden of traffic on one device or switch a SDN based switch can reroute the traffic through another SDN based switch which dramatically increases the reliability and life of a networking infrastructure. SDN reduces the cost of networking infrastructure dramatically with a little or no maintenance. There is a lot of interest currently growing in the development and implementation of SDN networks both for Commercial and Military usage.

1.3 Commercial Deployments of SDN

There are many SDN based networks currently operating worldwide. These networks are mostly to minimize the troubleshooting of the huge networking infrastructures. Most renowned internet companies use SDN based networking to optimize the user experience, such as the closest path is decided in order to make smoother user experience. As shown in the figure below is a small SDN company making use of commercial SDN software Hyperglance.



Property	Value
Account Alias	Production
AMI Launch Index	0
Architecture	x86_64
Availability Zone	us-east-1a
Availability Zone Name	US East (N. Virginia) A
Chargeable Item	Yes
Client Token	066cc61b-89b3-4f99-b5b4-1ec30fc79
datasource	Amazon
Detailed Monitoring	disabled
Endpoint	ec2.us-east-1.amazonaws.com
foreignSourceId	i-9180d1e8
Hours Run	2
Hypervisor	xen
Image ID	ami-0f9cbe66
Image Name	HGS-3.0.0.4-CentOS-6.4-x86_64-250
Instance ID	i-9180d1e8
Instance Type	m1.small
IP Address	54.196.8.32
Kernel ID	aki-88aa75e1
Key Name	RSPROD
Launch Time	06/12/2013 13:56:13
Lifecycle	spot
Name	stace test 3.0.0.4
Placement: Tenancy	default
Private DNS Name	ip-10-93-58-95.ec2.internal
Private IP Address	10.93.58.95
Product Codes	aacglxeowvn5hy8sznltowyqe
Public DNS Name	ec2-54-196-8-32.compute-1.amazona
Public IP Address	54.196.8.32
Region	us-east-1
Region Name	US East (N. Virginia)
Root Device Name	/dev/sda1
Root Device Type	ebs
Security Groups	All Stace-Home
sourcedFromType	EC2-Instance
Spot Instance Request ID	sir-579d9e34
State	running
subTopologyName	AWS/AKIAJPDCJH44CQMOLPRA
Tag: Name	stace test 3.0.0.4
type	EC2-Instance
Virtualization Type	paravirtual

Name	Public IP Address	type	Instance Type	Availability Zone	Key Name	State
stace test 3.0.0.4	54.196.8.32	EC2-Instance	m1.small	us-east-1a	RSPROD	running

Figure 1-1 Commercial SDN Software Hyperglance

However underneath the network of ISP's and big internet companies, most of the networking infrastructure is still old fashioned, upgrading and maintaining that infrastructure costs millions. The biggest hurdle to convert the following infrastructure to SDN based networks is that the user hardware unlike hardware's of various internet companies varies tremendously. The normal consumers hardware comprises of different manufacturers which make use of different chipsets and technology, such as wimax or free space optics. Which not only various in maximum bandwidth they can provide but also work on different mechanisms. Below is an graphical illustration of usage and popularity of different types of networks other than broadband or fiber optics.

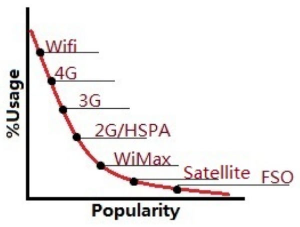


Figure 1-2 Different types of networks

Chapter 2 Programmable Routers

2.1 OpenFlow

OpenFlow is a protocol developed by Open Network Foundation (ONF) used in communication which gives access to the forwarding plane of a network switch or router to a controller over the network. Openflow on switches enables remote controllers to determine the path of network packets through the network of switches this allows for a more sophisticated traffic management than its achievable by normal routing protocols. The openflow protocol is layered on top of the (TCP) protocol, while use of Transport layer security protocol (TLS) is recommended.

2.2 OpenWRT

OpenWRT is an operating system and distribution of Linux kernel designed for embedded systems [3]. It is primarily used in networking devices such as routers to route the traffic. Now a day's most of the routers that are on the market use custom OS designed by the manufacturer. While most of them can support OpenWRT OS some cheap routers are unable to meet the hardware requirements of OpenWRT. OpenWRT brings vast majority of control and monitoring functions to the router. Just as an android cell phone is known as smart phone an OpenWRT based router is known as smart router.

OpenWRT allows its root file system to be modified without reflashing the whole firmware. While package manager OPKG allows installing over 3500 software which is also a huge benefit as one doesn't need to rebuild the firmware. The software can be just as easily uninstalled as it was installed. OpenWRT supports a wide range of networking hardware drivers.

2.3 Available router platforms

As of 2014-2015 various cheap wifi routers are on the market to enable wifi access to public. As the case we utilize these hardware's to determine the performance and feasibility of software defined networking on these devices. The list of hardware we use is as follows:

1. Asus RT-N16 WiFi router: This router comes with its custom OS provided by the manufacturer and has medium spec hardware. As of 2014 the router does not officially fully support OpenWRT, hence we selected to port OpenWRT with openflow enabled on it to test the performance of the hardware.
2. Raspberry Pi: Raspberry Pie is the most popular hand-held computer with the Low-Mid Specs [4]. It is not a networking device and lacks WiFi chip which can be included in it by usb interface. It is cheap and hence one of the ideal devices to test SDN infrastructure.

3. Intel Galileo Gen 2: it is a development board based on the on the Intel® Quark™SoC X1000 application processor with a 32-bit architecture [5].
4. NetGear 3700v4: The router is one of the most popular routers on the Market. It has fairly moderate specs and officially supports OpenWRT which makes it ideal to test performance against other low to mid spec hardware's.

Aside from the devices we use a point that should be kept in mind is that the version of OpenWRT we are trying to port is r36088 (Altitude Adjustment) or recent, which cannot run below 20 MB of Memory. Some devices on market under 100RMB lack the requirements. The routers we used contain ARM architect, while the chipset and WiFi drivers differ.

Chapter 3 ASUS-based OpenFlow Router

3.1 Setting up the Testbed

3.1.1 Compiling and integrating OpenFlow with OpenWRT

As discussed above we tried to compile OpenWRT on unsupported ASUS N-16 router as it is middle class hardware, also integrating the system image with Openflow. We use Linux system to compile OpenWRT. In order to start the compilation process, we need to make sure that the necessary space is available on the system. For this particular process around 20GB of space is required. Now we check the libraries by opening non root linux terminal and execute

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

we make a directory name Openwrt for the rest of the work to store in and navigate to it by.

```
cd ~/openwrt
svn co svn://svn.openwrt.org/openwrt/branches/backfire
cd backfire
./scripts/feeds update -a
./scripts/feeds install -a
```

This will checkout the packages from the svn directory of openwrt. Then we make the configuration file for our particular system. As we already know that our system uses Broadcom bcrmxx47 chipset for processing and b40 for wireless so we select it from the menu.

```
make menuconfig
Then we check the dependicies by using the command
make prereq
make
```

After the above steps are done we need to add openflow to build request menu. We use

```
git clone https://github.com/CPqD/openflow-openwrt.git
.cd ~/openwrt/backfire/packages/
ln -s ~/openwrt/openflow-openwrt/openflow-1.3/
cd ~/openwrt/backfire/
ln -s ~/openwrt/openflow-openwrt/openflow-1.3/files
cd ~/openwrt/
make menuconfig
```

we select openflow from the manu under network and start the operation by typing

```
make kernel_menuconfig
make
```

This took approximatly 15 hours on a moderate internet connection. After 15 hours of compiling we will have compiled os in the same directory as well as openflow package seprated in different folders. The openflow package is in .ipk format and bcrm xx47 is labeled on its name as it is compiled for our chipset. During the compiling process we encountered tens of errors. All most all of them were due to the fact that package was not found or it was no longer supported. This is becuase some url are blocked by GFW or are no longer in maintanace. We solved this issue by downloading the packages separately and putting them in OpenWRT directory. The compilation process starts exactly where it left.

3.1.2 Loading the system

Every router has its own mechanism flashing the firmware, In case of our N-16 we need to do the following to make it ready to accept the new firmware.

1. Take off the power of the router by unplugging the power cord.
2. Hold the reset switch and power it ON.
3. Keep holding the Power switch until the power LED on the router start to blink rapidly.
4. Plug in the Ethernet port to the LAN 1 port
5. Connect it with your computer and make sure that IP allocation is set to auto.
6. Use Asus flashing utility to flash the image compiled.

After the system reboots the router is set up to work with Openwrt but if there is a problem we can always revert back to the original factory firmware by using the above process again with the factory image. As the system boots we can see that the Wifi LED is turned OFF there are two reasons for this, first one being that the as the OpenWRT is not officially supported by the router so it needs to be tweaked before the LED starts to blink again and the second being that the wifi is not turned off by default on initial boot. We need to turn on the wifi in order to use its functionality. We use telnet to connect with router. On windows we can use the putty for this purpose. On the first connect we need to setup the password for administrator after which we can switch to SSH. The system is complete but need to be tweaked in order to make things easier and have a web GUI. The windows is the command line interface to the router which s same as the Linux terminal. Make sure that the router has internet access by connecting the net wire to the WAN port. We use the following commands to update the package directory list of the router. OPKG update

3.1.3 Tweaking and improving performance

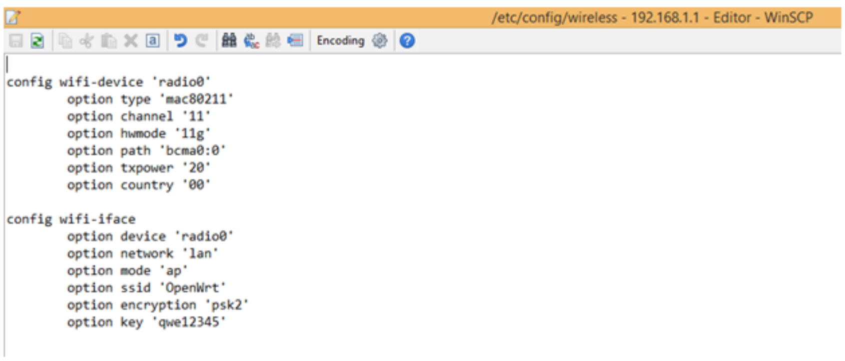
As seen in most of the routers in market we need to setup the web gui on our OpenWRT router. There are numerous web gui packages available for OpenWRT but we will go ahead with the Luci as It is one of the most developed web UI. To setup Luci there are two methods depending on the user situation either we can set it up by downloading the required libraries from the web within the router or we can use offline mode and transfer the libraries using SCP protocol. Now we use the following commands in ssh command line to setup Luci when router has access to the internet:

```
opkg install luci   *To install Luci*

/etc/init.d/uhttpd start *To start the web server*

/etc/init.d/uhttpd enable *Setup the server to start automatically at each
    boot*
```

Now the web server is running and we can open it by typing the gateway IP (Normally 192.168.1.1) we need to login using the username and password, here the username is root while the password is the one we set up in the telnet session. After logging in we can enable Wifi and connect to it by searching for the default name OpenWRT. We can remove unnecessary packages to improve the process by searching in the running service. As we can see openflow listed in the installed packages means the system has recognized openflow as a package but it's still not running. In the file system we can see the WIFI configuration file which allows us to tweak the performance of the WIFI by adjusting the frequencies or the TX power level.

3.1.4 Enabling Openflow and setting it up with WIFI

Before Enabling open flow we use winSCP on windows to access the file system of the router using SCP protocol. We need to access the file system so that we can modify the wifi configuration file in order to make it work with wifi adapter as shown in Figure 3-1.



```
config wifi-device 'radio0'
        option type 'mac80211'
        option channel '11'
        option hwmode '11g'
        option path 'bcma0:0'
        option txpower '20'
        option country '00'

config wifi-iface
        option device 'radio0'
        option network 'lan'
        option mode 'ap'
        option ssid 'OpenWrt'
        option encryption 'psk2'
        option key 'qwe12345'
```

Figure 3-1   Parameter settings for enabling WiFi on the Asus router

The file is located at "/etc/config/wireless" this file is for wifi configuration and we can optimize the wifi throughput by configuring this file. Now we need to configure Openflow file before starting it's services. In the configuration we can include or exclude wifi adapter. There are two ways an Openflow enabled switch can connect to the controller. (I) Inbound (II)

Outbound Inbound makes the switch listen on a specific port (TCP 6633) while controller tries to connect to it. While Outbound lets the switch to search for a controller on an IP and port.

3.1.5 Setting up the controller

Controller is the central brain of SDN based network, Hence setting up the controller properly is the key to a successful software defined network. There are various controllers now a days to choose for the sdn network. For this project we choose the Opendaylight controller. Opendaylight controller is a java based sdn controller and can run on any system that supports java. Opendaylight is one of the most popular controller which has a huge developer support community. We setup the controller on Windows operating system in our project. In order to do this we download the controller setup and extract the .zip file. We need to make sure that Java is installed on our system and environmental variables are set. Then we need to open windows command prompt and need to navigate to the directory of extracted files. There we run run.bat Program which is in the directory of the extracted files. After few minutes the server should run smoothly on the local system. To check we can open browser and open the local ip 127.0.0.1 or if the computer is located n the LAN we can check it through the LAN IP using another system.

We should see an Opendaylight login screen as shown in Figure 3-3. We can login by typing the default username and password admin.

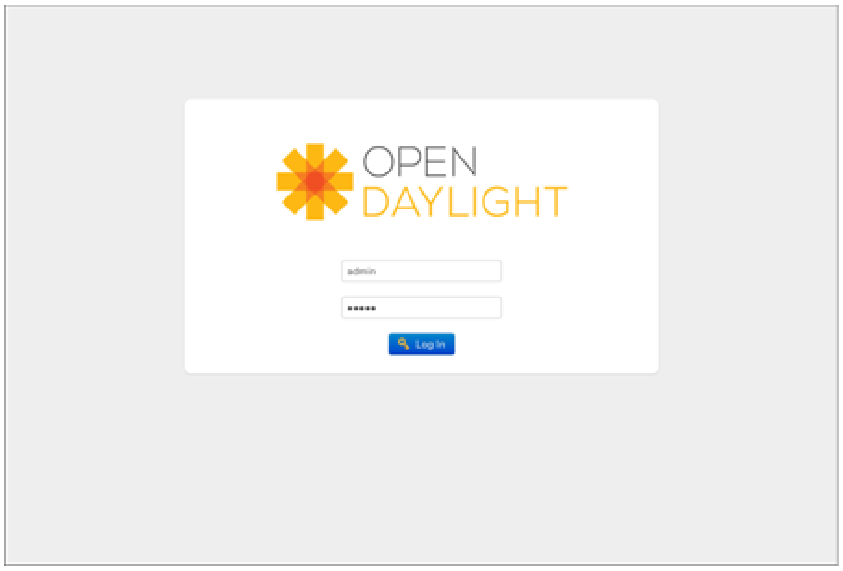
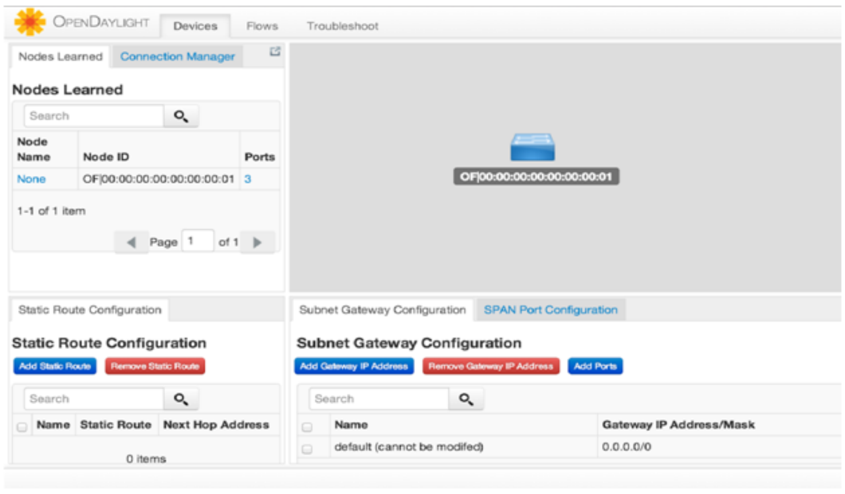


Figure 3-2 Login page of OpenDaylight

3.1.6 Networking

As we have setup the controller and switch, now we need to link the infrastructure. As we know that we have setup our switch to accept an Inbound connection so we need to provide the ip and the port of the switch we want to connect to. After a few seconds the switch will appear on the right side. As shown in the Figure 3-3, the blue box that we see on the right side is the switch. We can connect the controller to the switch either by Ethernet cable or wifi as we have already configured the wifi on openflow. Now in order to see the network in action we need to connect some clients. Keep in mind that it doesn't matter whether we connect clients on LAN or wifi.



Nodes Learned

Node Name	Node ID	Ports
None	OF|00:00:00:00:00:00:00:01	3

1-1 of 1 item

Static Route Configuration

Name	Static Route	Next Hop Address

0 items

Subnet Gateway Configuration

Name	Gateway IP Address/Mask
default (cannot be modifed)	0.0.0.0/0

Figure 3-3 Device page of OpenDaylight

3.1.7 Prototype Test

We will test our network and connect some clients with it to see whether our SDN based switch is also working as a simple DHCP switch. So now we connect 3 clients and we can see from the picture below that we can monitor the clients connected to the router.

The IP 192.168.122.161 is set manually while others are on automatic allocation in order to check whether the router has any issues with the clients connected automatically. In the same setup we can set up multiple switches to one controller and distribute the traffic and balance the load. This is necessary for commercial networks and wifi hotspots as in rush hours at different places during the day can me handled with relative ease. We used the following setup to test the traffic load as compared with the normal household router In order to check efficiency and to check whether there are any connection drop outs during the interval, All connections at this point seem stable.

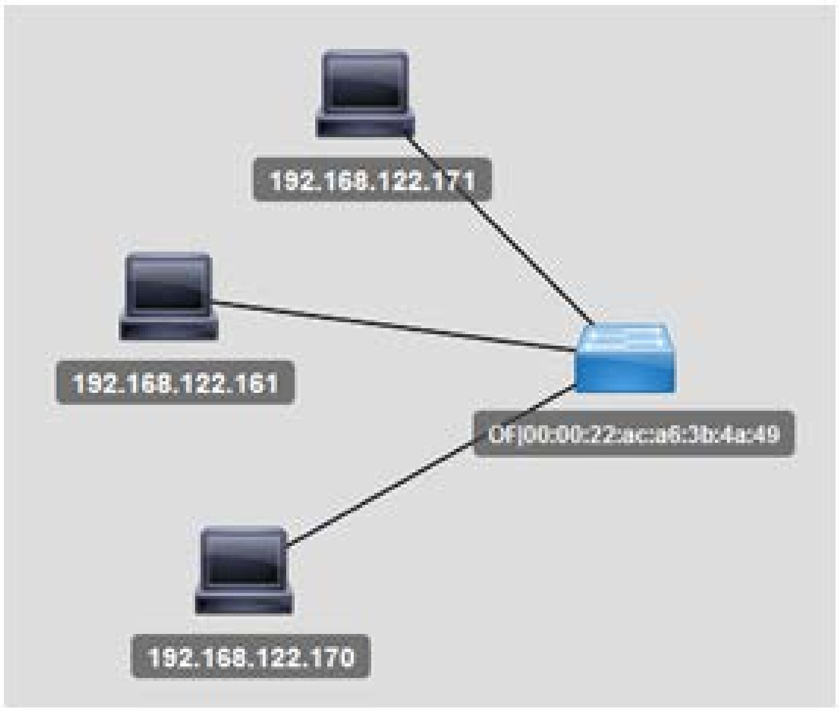
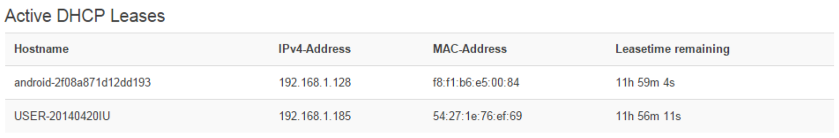


Figure 3-4 Clients connected to switch

3.2 Analysis and Comparison

3.2.1 Traffic Load

As now we managed to successfully port Openflow and SDN architecture on our wifi router we need to analyze whether usual traffic flows through this network and how does it stand against the non SDN based network. The first on the list of testing is obviously the local area ping test which is one of the important functions of switches these days, as printers and LAN games are popular in many offices and house holds. In order to test this we used an android cell phone connected to the router. We can get the IP of the device by going to the DHCP list in Web GUI as shown in figure below The below is the comparison of the LAN ping test. Figure 3-6 is the



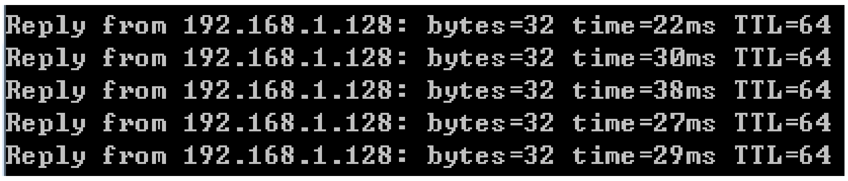
Active DHCP Leases

Hostname	IPv4-Address	MAC-Address	Leasetime remaining
android-2f08a871d12dd193	192.168.1.128	f8:f1:b6:e5:00:84	11h 59m 4s
USER-20140420IU	192.168.1.185	54:27:1e:76:ef:69	11h 56m 11s

Figure 3-5 OpenWRT DHCP Clients list

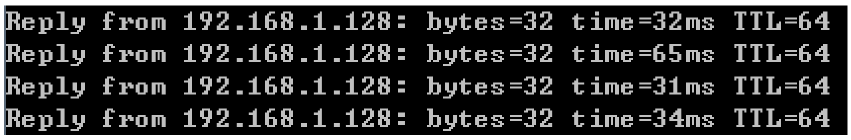
Asus router with Openflow disabled, while Figure **??** is the same router with Openflow enabled.

As from the above figure we can see that the difference in performance is visible in the form of slightly higher ping accompanied by high ping spikes. We can conclude from the above that Openflow exerts latency in the network when enabled. Which may reduce the maximum



```
Reply from 192.168.1.128: bytes=32 time=22ms TTL=64
Reply from 192.168.1.128: bytes=32 time=30ms TTL=64
Reply from 192.168.1.128: bytes=32 time=38ms TTL=64
Reply from 192.168.1.128: bytes=32 time=27ms TTL=64
Reply from 192.168.1.128: bytes=32 time=29ms TTL=64
```

Figure 3-6 Ping with Openflow Disabled



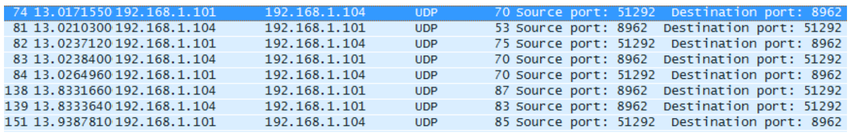
```
Reply from 192.168.1.128: bytes=32 time=32ms TTL=64
Reply from 192.168.1.128: bytes=32 time=65ms TTL=64
Reply from 192.168.1.128: bytes=32 time=31ms TTL=64
Reply from 192.168.1.128: bytes=32 time=34ms TTL=64
```

Figure 3-7 Ping with Openflow Enabled

amount of users connected to the router.

3.2.2 Voice and Video Streaming

Voip and video streaming are some of the most commonly services provided by the networking infrastructure which utilize high bandwidth over the network. It is necessary to keep this services as real time as possible hence both VOIP protocol and video streaming have developed rapidly during previous years. It is also vital for SDN based networks to prove them self in consumer space as they have already did in the network switches of ISP's. Hence we will compare and monitor the VOIP and video streaming over SDN based router. As from the figure below u can see that VOIP packets in comparison with Video buffering packets.



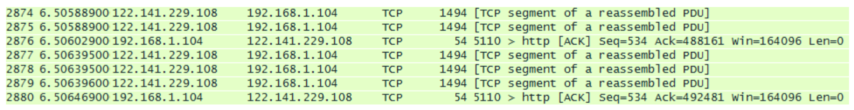
```
 74 13.0171550 192.168.1.101   192.168.1.104   UDP   70 Source port: 51292  Destination port: 8962
 81 13.0210300 192.168.1.104   192.168.1.101   UDP   53 Source port: 8962   Destination port: 51292
 82 13.0237120 192.168.1.101   192.168.1.104   UDP   75 Source port: 51292  Destination port: 8962
 83 13.0238400 192.168.1.104   192.168.1.101   UDP   70 Source port: 8962   Destination port: 51292
 84 13.0264960 192.168.1.101   192.168.1.104   UDP   70 Source port: 51292  Destination port: 8962
138 13.8331660 192.168.1.104   192.168.1.101   UDP   87 Source port: 8962   Destination port: 51292
139 13.8333640 192.168.1.104   192.168.1.101   UDP   83 Source port: 8962   Destination port: 51292
151 13.9387810 192.168.1.101   192.168.1.104   UDP   85 Source port: 51292  Destination port: 8962
```

Figure 3-8 Voip (Skype) Video Chat

From the figures above we can illustrate that VOIP protocol has taken it's distinctive place as it checks for the user on the same LAN to Improve performance while video buffering packet has segments over the internet. In terms of SDN we can see that VOIP can have some latency issues when transferred to LAN IP depending on the hardware. Both VOIP and video buffering can still some what be optimized by closing the gap between host and server using SDN.

3.2.3 Hardware Constraints

We know that more user constraint the limited resources of networking hardware, we are interested to know that how much constraint does SDN puts on a specific hardware. As we have seen from the tests above that enabling SDN increases the ping and introduces spikes, hence we will monitor the performance of our Asus router in both cases. We need to make sure



```
2874 6.50588900 122.141.229.108     192.168.1.104       TCP      1494 [TCP segment of a reassembled PDU]
2875 6.50588900 122.141.229.108     192.168.1.104       TCP      1494 [TCP segment of a reassembled PDU]
2876 6.50602900 192.168.1.104       122.141.229.108     TCP        54 5110 > http [ACK] Seq=534 Ack=488161 Win=164096 Len=0
2877 6.50639500 122.141.229.108     192.168.1.104       TCP      1494 [TCP segment of a reassembled PDU]
2878 6.50639500 122.141.229.108     192.168.1.104       TCP      1494 [TCP segment of a reassembled PDU]
2879 6.50639600 122.141.229.108     192.168.1.104       TCP      1494 [TCP segment of a reassembled PDU]
2880 6.50646900 192.168.1.104       122.141.229.108     TCP        54 5110 > http [ACK] Seq=534 Ack=492481 Win=164096 Len=0
```

Figure 3-9 Standard Video Buffering

that the number of users on the network are as minimum as possible in order to get accurate understanding of the amount of resources Openflow uses. Also we measured the resources when Openflow is not enabled. Figure 3-10 is the performance of our Asus router with Openflow disabled at first and then later enabling it.

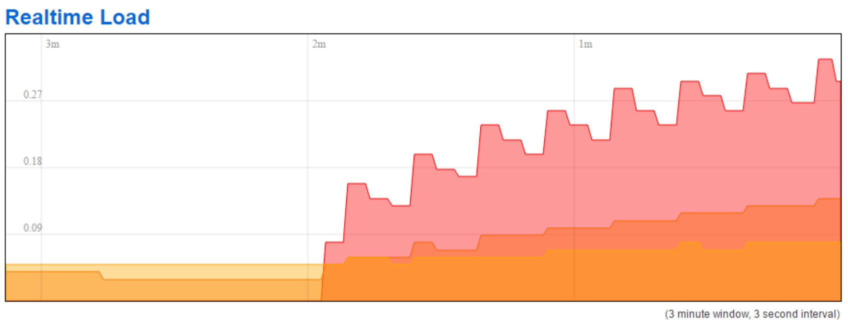


Figure 3-10 Traffic load of the OpenWRT-based Asus router

From the graph above we can see that the hardware usage increases when Openflow protocol is enabled. While the strain put on hardware may be decreased by inbound connection the overall effect depends on the memory and CPU resources of the hardware. The side effect of this constraint could be the maximum amount of users connected to this particular device because as the number of users increase the resources of hardware become important. Moreover as this work done by us is in early stages we can assume that there is room for improvements by cutting on some other software's that are not required after the addition of Openflow.

3.2.4 SDN vs Non-SDN Networks

After converting a non-SDN router to an SDN based router and evaluating the performance we can directly compare the benefits and flaws of both networking systems. As with the SDN we have seen that some of performance is lost while openflow a necessary protocol to enable sdn on various systems is turned on, this effect is negligible when we compare it with a lot of user on a non sdn based router. The router and user are necessarily locked out to constraint each other if the system is focused on one place. While SDN may take some performance out of the device it can efficiently flow the load between other openflow enabled routers. Which dramatically reduces the cost that is required to upgrade the networking infrastructure every year or so. While so many benefits come with a price and that price is that SDN is heavily centralized network which makes it curious and desirable target for network intruders. Companies like Google, Microsoft are actively developing and deploying there own SDN based networks. Which enables the fact that most of the maintenance cost is circling around internet service providers and consumers. While consumers and the demand for higher bandwidth grows, the upgrade and maintenance cost of equipment grows exponentially, which paves the way for development of SDN based network on consumer end.

Chapter 4 Raspberry-Pi-based OpenFlow Router

The Raspberry Pi is a credit card sized single-board computer, The original Raspberry pi is based on the Broadcom BCM2835 system on a chip(SoC) originally shipped with 256 megabytes of RAM, later upgraded(Models B and B+) to 512 MB. The system has secure digital (SD) (models A and B) or microSD (models A+ and B+) sockets for boot media and persistent storage, in Our case we used Rasberry Pi model A.

Raspberry Pi is cheap and can cost less than 200 RMB , it is power efficient:it uses 5V and 1A . Raspberry pi can handle different operating systems the latest being windows 10 , others include: linux, Raspbian, openSUSE, FreeBSD, Xtian, and the one we will be trying to port is OpenWRT which will also include Openflow.

4.1 Building OpenFlow on the OpenWRT image of Raspberry Pi

It is similar to the one we runned on the assus router we just changed some features:

We were trying to build the latest version called: “BarrierBreaker”. The procedures to build “BarrierBreaker” is described as follows.

First we download the necessary packages.

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

Then we need to install we follow the following steps.

```
git clone http://git.openwrt.org/14.04/openwrt.git
cd openwrt
mv feeds.conf.default feeds.conf
./scripts/feeds update -a
./scripts/feeds install -a
./scripts/feeds install -a luci
```

Then we configure the menu.

```
make menuconfig
Target systems -> Broadcom BCM2708/BCM2838
target profile ->Rasberry pi
Luci -> collection -> <*> luci
Luci ->Translations -> <*> luci-i18n-chinese
           <*> luci-i18n-english
Luci->Applications -> luci-app-ddns
Kernel modules -> Native Language Support -> <*> kmod-nls-utf8
Kernel modules -> Wireless Drivers -> <*> Kmod-ath9k-htc
           <*> Kmod-rtl8192cu
```

```
Kernel modules -> USB Support -> <*> Kmod-usb-eth-gadget
                <*> Kmod-usb-storage-extras
                <*> Kmod-usb-wdm
                <*> Kmod-usbip-client
                <*> Kmod-usbip-server
```

Next, install OpenFlow.

```
git clone https://github.com/CpqD/openflow-openwrt.git
cd ~/openwrt/package
ln -s ~/openwrt/openflow-openwrt/openflow1.3/
cd ~/openwrt/
ln -s /openwrt/openflow-openwrt/openflow1.3/files
```

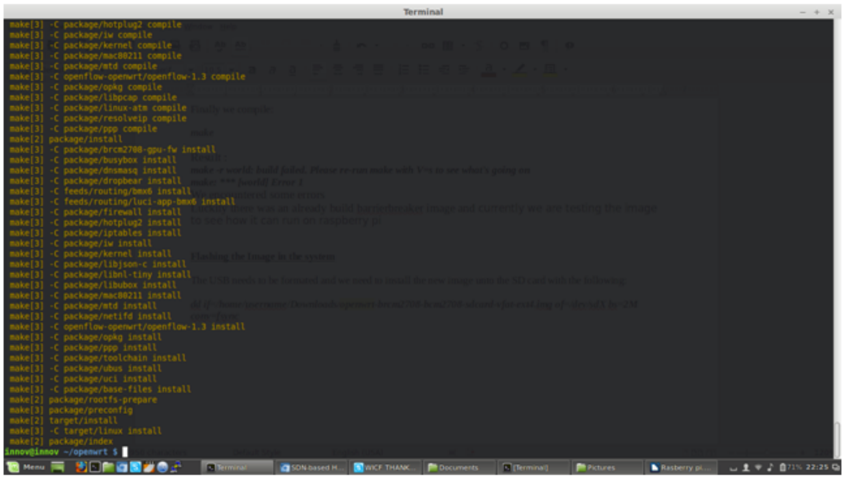
Configure the menu of the kernel.

```
make menuconfig
network---> <*>openflow... Open Flow 1.3 Switch Userspace Package
```

Finally, we compile the booting image for the board.

```
make
```

The compilation screenshot of the Raspberry Pi is shown in Figure 4-1.



```
make[3] -C package/hotplug2 compile
make[3] -C package/iw compile
make[3] -C package/kernel compile
make[3] -C package/mac80211 compile
make[3] -C package/mtd compile
make[3] -C openflow-openwrt/openflow-1.3 compile
make[3] -C package/opkg compile
make[3] -C package/libpcap compile
make[3] -C package/linux-atm compile
make[3] -C package/resolveip compile
make[3] -C package/ppp compile
make[2] package/install
make[3] -C package/brcm2708-gpu-fw install
make[3] -C package/busybox install
make[3] -C package/dnsmasq install
make[3] -C package/dropbear install
make[3] -C feeds/routing/bmx6 install
make[3] -C feeds/routing/luci-app-bmx6 install
make[3] -C package/firewall install
make[3] -C package/hotplug2 install
make[3] -C package/iptables install
make[3] -C package/iw install
make[3] -C package/kernel install
make[3] -C package/libjson-c install
make[3] -C package/libnl-tiny install
make[3] -C package/libubox install
make[3] -C package/mac80211 install
make[3] -C package/mtd install
make[3] -C package/netifd install
make[3] -C openflow-openwrt/openflow-1.3 install
make[3] -C package/opkg install
make[3] -C package/ppp install
make[3] -C package/toolchain install
make[3] -C package/ubus install
make[3] -C package/uci install
make[3] -C package/base-files install
make[2] package/rootfs-prepare
make[3] package/preconfig
make[2] target/install
make[3] -C target/linux install
make[2] package/index
innov@innov ~/openwrt $
```

Figure 4-1 The compilation screenshot of the Raspberry Pi.

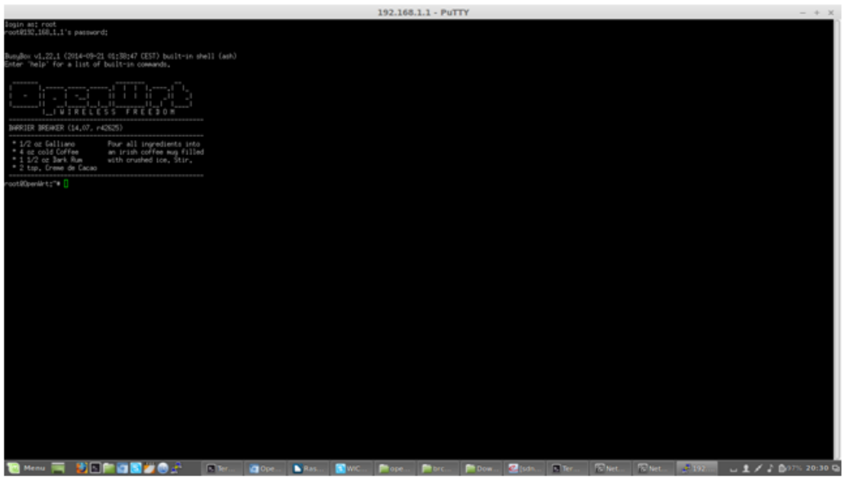
It takes about 2 hours for finish the compilation.

4.2 Flashing the Image in the system

The SD card needs to be formated and we need to install the new image unto the SD card with the following:

```
dd if=/home/username/Downloads/openwrt-brcm2708-bcm2708-sdcard-vfat-ext4.
    img of=/dev/sdX bs=2M conv=fsync
```

The image preview of the OpenWRT is as shown in Figure 4-2.



```
login as: root
root@192.168.1.1's password:


BusyBox v1.22.1 (2014-09-21 01:38:47 CEST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

         |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 BARRIER BREAKER (14.07, r42625)
 -----------------------------------------------------
  * 1/2 oz Galliano         Pour all ingredients into
  * 4 oz cold Coffee        an irish coffee mug filled
  * 1 1/2 oz Dark Rum       with crushed ice. Stir.
  * 2 tsp. Creme de Cacao
 -----------------------------------------------------
root@OpenWrt:~#
```

Figure 4-2 The image preview of the OpenWRT on the Raspberry Pi.

4.3 OpenWRT Configuration

To configure the raspberry we first connect the Ethernet cable from the raspberry pi to the computer, then using putty with the assigned ip: 192.168.1.1, we choose the telnet connection

After connecting we set a password for the root user:

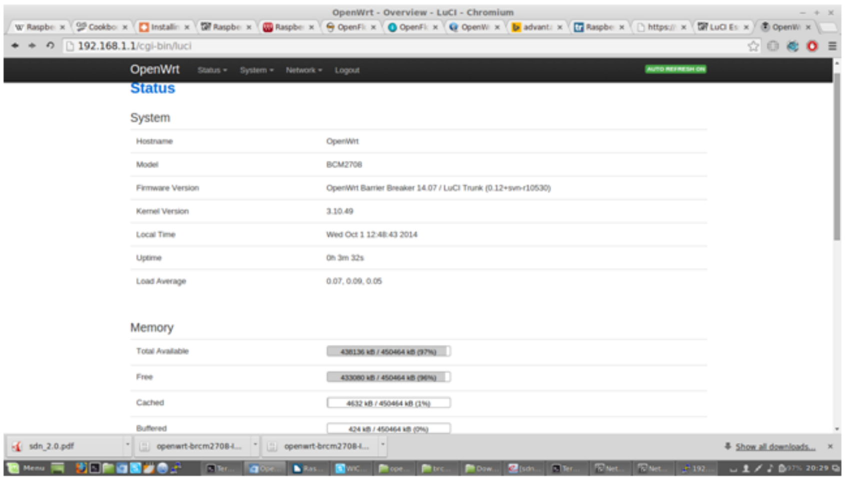
```
passwd
```

The password is now “qwe12345”.

From there we try to enable Luci:

```
/etc/init.d/uhttpd start
/etc/init.d/uhttpd enable
```

The configuration of the OpenWRT Luci on the Raspberry Pi is as shown in Figure 4-3.



Status

System

Hostname	OpenWrt
Model	BCM2708
Firmware Version	OpenWrt Barrier Breaker 14.07 / LuCI Trunk (0.12+svn-r10530)
Kernel Version	3.10.49
Local Time	Wed Oct 1 12:48:43 2014
Uptime	0h 3m 32s
Load Average	0.07, 0.09, 0.05

Memory

Total Available	438136 kB / 450464 kB (97%)
Free	433080 kB / 450464 kB (96%)
Cached	4632 kB / 450464 kB (1%)
Buffered	424 kB / 450464 kB (0%)

Figure 4-3 The configuration of the OpenWRT Luci on the Raspberry Pi.

4.4 Benefits of Raspberry Pi SOC

Raspberry Pi has a variety of uses, it can work as a personal computer, can be used in electronic systems , but here we make it work as a router. Raspberry pi is cheap, power efficient and it can help expand the connectivity in poor regions where high tech routers is not affordable, and electricity is mostly inexistent, it can be solar powered, with the sheer number of developers; raspberry pi has a promising future. As shown in Figure 4-4, we depict a demo of an OpenWRT router based on the Raspberry pi with solar cell.

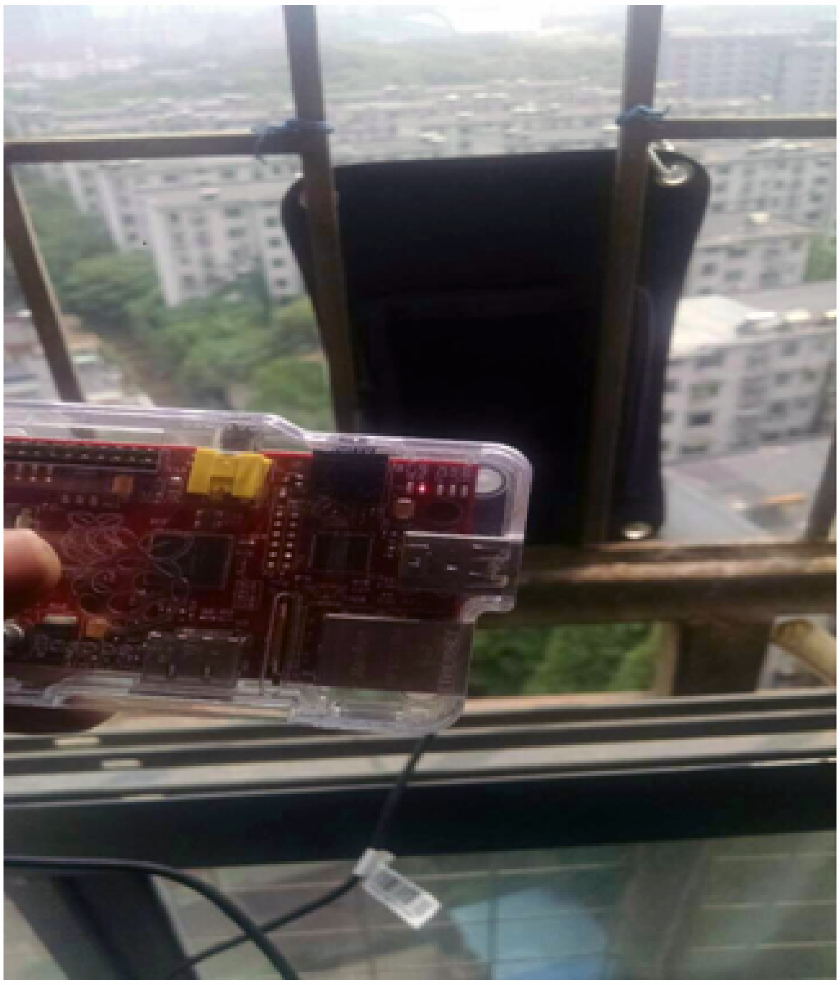


Figure 4-4 A demo of an OpenWRT router based on the Raspberry pi with solar cell.

As you can see from the above photo that raspberry pie's red LED is turned on, Which indicates that it's working normally. As with other CPU extensive operating systems the voltage fluctuation takes place and if the power is not sufficient it may cause damage to CPU and shutdown the system, but with OpenWRT there is minimum load on the resources of raspberry pi. Even with high traffic flow.

Another benefit of Raspberry pi is that it supports multiple USB adapters which can extend the branch of an SDN based network. It can also store a lot of packet data as it utilizes the micro SD card slot to extend it's memory at maximum of 32GB. However the performance of such type of interface is slow which is why it is just a test platform. It illustrates the benefit such a system could have to expand networking infrastructure. Raspberry pi can also act as a back up switch in case of malfunction, as it is cheap so it can almost certainly be used on different emergency situations.

Chapter 5 Intel-Galileo-based OpenFlow Router

5.1 Overview

Intel Galileo Gen 2 is a development board based on the on the Intel® Quark™SoC X1000 application processor with a 32-bit architecture as shown in Figure 5-1. This board can easily be mistakenly thought of as an only advanced, more powerful version of Arduino [6] Uno layered ontop of a Linux system in that, it is the first board based on Intel® architecture designed to be hardware and software pin-compatible with shields designed for the Arduino Uno* R3. With this architecture, even though it is a single-threaded system, it has the ability of functioning as multi-threaded device with linux running in the background and an Arduino sketch running in the core. With these tools, we able to also install a Linux image on Galileo, and even OpenWRT and OpenFlow embedded on this system since they're based on the Linux kenel.

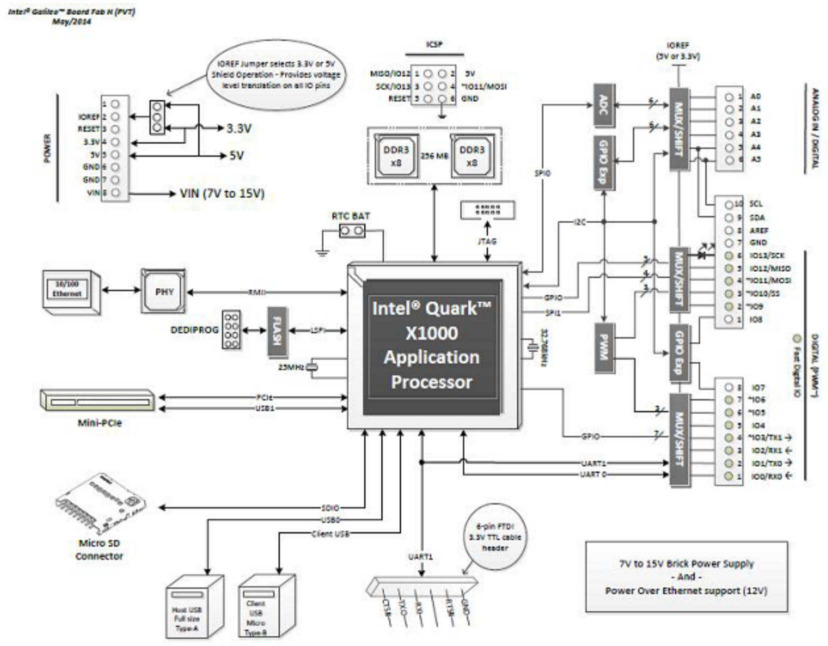


Figure 5-1 The block diagram of the Intel Galileo Gen 2.

5.2 Procedure

To develop OpenFlow for this system, it was first necessary to get familiar with the Intel Galileo board. This was done by breaking down the workload three separate procedures.

The first procedure was to install a Yocto linux image unto the system and begin communicating with the board through SSH or Telnet via Putty on linux. Yocto is the operating system that powers the Intel Galileo board, a developer image of Yocto is necessary in order to install and update system packages or enable networking on the system. In this setup, the Arduino IDE (Integrated Development Environment) serves as the shell access to Galileo while using the Ethernet port on the board to simply keep the board connected to the internet via Local Area Network. The Ethernet port is also used to communicate with the shell via Putty by enabling the telnet server using the following line in the arduino sketch:

```
system("telnetd -l /bin/sh");
```

Then, assign the board with a valid IP address using the following command.

```
system("ifconfig eth0 169.254.x.xxx netmask 255.255.0.0 up");
```

To test for internet connectivity, the Arduino "WebClient" sketch can be uploaded unto Galileo to ping a Google server or modified to ping any accessible servers.

As observed from the architecture, the system doesn't come with a preinstalled wireless (wlan0) configuration, the second precedure was to install a wireless adapter specically the (Netgear G54/N150 Wifi USB Micro Adapter). The most common method for installing wifi on Galileo is mounting a standard PCI card unto Galileos PCI slot, but for convinience and mobility; enabling the USB wifi adapter is the viable method. The Netgear G54/N150 Wifi USB Micro Adapter is built on the Realtek RTL8188CUS chipset, installing the drivers for this chipset and editing the network configuration file then enabling some specific network pereferences are required to configure the wireless (wlan0) on the system. With Galileo connected to the Internet, the following sketch developed int the Arduino IDE can be run to install all the necessary drivers for any Wifi USB adapter built on the Realtek RTL8188CUS chipset:

```
/*
  G_Shell22_Script

 This sketch updates Galileo system packages via Ethernet connection
 using an Arduino Wiznet Ethernet shield.

 Circuit:
 * Ethernet shield attached to pins 10, 11, 12, 13

 created 18 Dec 2009
 modified 14 Apr 2015
 by Kenyon W. Nsunza

 */

#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the
    shield
byte mac[] = {  0x98, 0x4F, 0xEE, 0x01, 0xA4, 0xA5 };
IPAddress server(220,181,57,217); // baidu

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
EthernetClient client;

void setup() {
 // Open serial communications and wait for port to open:
  system("ifup eth0");
  delay(5000);
  Serial.begin(9600);
   while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    for(;;)
      ;
  }
  // give the Ethernet shield a second to initialize:
  delay(1000);
  Serial.println("connecting...");

  // if you get a connection, report back via serial:
  if (client.connect(server, 80)) {
    Serial.println("connected");
    // Make a HTTP request:
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  }
  else {
    // if you didn't get a connection to the server:
    Serial.println("connection failed");
  }
  //system("opkg install --force-overwrite uclibc > /dev/ttyGS0");
  //Serial.println("uclibc overwritten");
  //delay(1000);
  //system("opkg install packagegroup-core-buildessential > /dev/ttyGS0");
  //Serial.println("package group installed");
  //delay(1000);
  //system("opkg update > /dev/ttyGS0");
  //Serial.println("Updated list from repository");
  //delay(1000);
  //system("opkg install usbutils > /dev/ttyGS0");
  //Serial.println("Installed packages to support usb");
  //delay(1000);
  //system("opkg install usbutils-dev > /dev/ttyGS0");
  //Serial.println("Dev USB Utillities installed");
  //system("lsusb > /dev/ttyGS0");
  //while(Serial.read()!='>'){
    //Serial.println("Check to see if system recognized the USB wifi dongle
        ");
   //delay(5000);
  //}
  //system("opkg install linux-firmware > /dev/ttyGS0");
  //Serial.println("Installed the driver for usb wireless");
  //delay(1000);
  //system("opkg install wpa-supplicant-dev > /dev/ttyGS0");
   //Serial.println("usb wireless drivers installed");
   //delay(1000);
   //system("opkg install kernel-module-rtl8192cu > /dev/ttyGS0");
   //Serial.println("installed the driver for RealTek wifi dongle");
   //delay(1000);
   //system("modprobe rtl8192cu > /dev/ttyGS0");
   //Serial.println("ran modprobe for the card");
   //delay(1000);
}

void loop()
{
  // if the server's disconnected, stop the client:
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();

    // do nothing forevermore:
    for(;;)
      ;
  }
}
```

Note: /dev/ttyGS0 is the serial port connected to Galileo, the output results are displayed on the Arduino serial monitor using this port.

After the sketch is run on Galileo, the internet connection to Galileo is not needed, the ethernet port can be assigned for SSH to Galileo through Putty to edit the following configuration files:

Edited the Wireless Interfaces section of /etc/network/interfaces

```
added “auto ”wlan0 to #Wirless interface
#nano /etc/network/interfaces
```

```
        ...
# Wireless interfaces
auto wlan0
iface wlan0 inet dhcp
    wireless_mode managed
    wireless_essid any
    wpa-driver wext
    wpa-conf /etc/wpa_supplicant.conf…
        .
```

Edit the settings of the network authentication on the campus network.

```
/etc/wpa_supplicant.conf
```

Added in a section with the SSID name and command that the Yun wasn't using a password. If you want to access a wifi router that has a password, then you need to change this.

```
#nano /etc/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
ssid="M.B.P."
scan_ssid=1
key_mgmt=WPA-PSK
proto=WPA
pairwise=TKIP
group=TKIP
psk="my-wlan-passwd"
id_str="M.B.P."

Remove connman
#opkg remove connman
--- if connman isn't removed it is very hard to get a wifi connection
---This might cause a problem elsewhere, but so far hasn't

Restarted networking
#/etc/init.d/networking restart
You can also restart just the wlan0 interface with:
#ifdown wlan0
#ifup wlan0
```

The wireless adapter is detected by Galileo and the system now runs flawlessly with the following script running in the background running after every reboot to start the necessary module and start the telnet server then assign an IP address to Galileo.

```
/*Galileo Startup*/
void setup() {
  // put your setup code here, to run once:
  system("telnetd -l /bin/sh"); //Start the telnet server on Galileo
  system("ifconfig eth0 169.254.8.248 netmask 255.255.0.0 up");
  system("modprobe rtl8192cu > /dev/ttyGS0");
  system("ifup wlan0 > /dev/ttyGS0");
  delay(5000);
  system("ifconfig > /dev/ttyGS0");
}
```

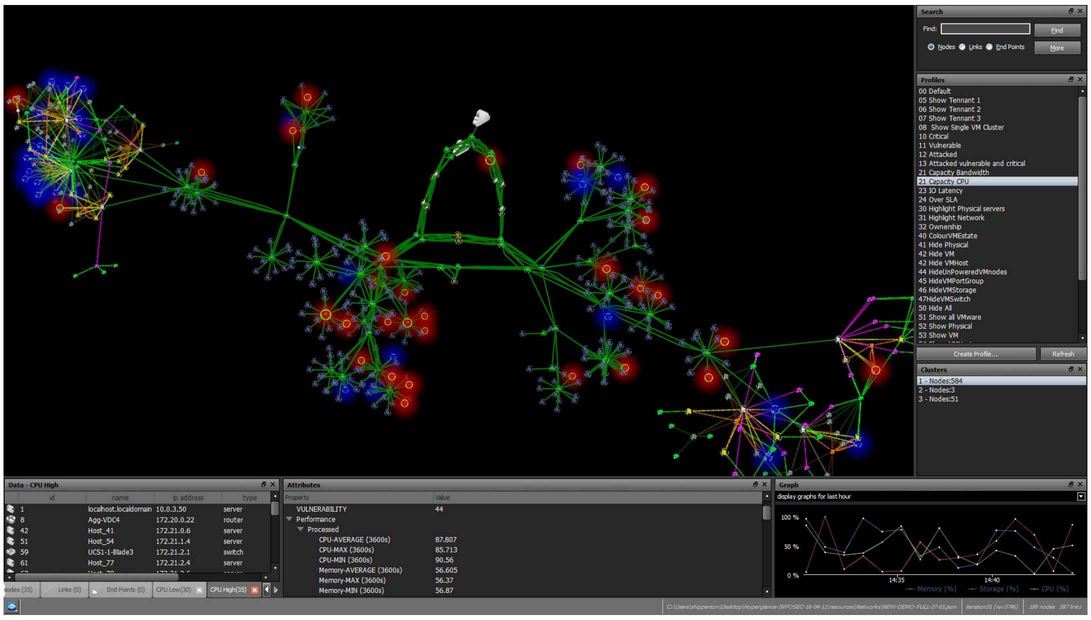
After understanding the Galileo architecture and learning how to communicate with the embeded system on Galileo through the Arduino and Linux enviroments, it was conclusive that OpenFlow can be developed on this system. The third step was to install an OpenWRT version compatible with the Galileo processor architecture. The OpenWrt “Trunk Version” is more compatible with Galileo because it supports x86 processor architectures. Though compatible with the CPU architecture, it’s not certain as of now how well this system will be running under OpenWRT/ OpenFlow. The Image is still currently being developed and after the system is updated with this firmware of OpenWRT trunk, procedures to enable the wireless(wlan0) on this system and rectify software malfunctions if any will be taken.

5.3 The Future of Galileo

Galileo has two powerful tools that can make it a very useful, reliable, and affordable bed for openFlow available today. The first is it’s Arduino features. Arduino can run sketches in the C programming language which can extend to the digital and analog input/output pins on Galileo and control many other smart home features such as temperature, lights, security and so much more. The second feature is the Yocto OpenWRT image running on this 32-bit Intel® Pentium® brand system chip (SoC); this image can control network features in the smart home network such as traffic control to prioritize the bandwith usage on the smart home network to improve the Quality of Service (QoS) for the network users. A system of keeping the whole family connected to the home network no matter their location can also be developed on this network to keep location the location status of each member of the home and always keep the family connected to each other over WiFi no matter their location. When these tools are working together, the home network serve the necessities of a smart home.

Chapter 6 Conclusion and Future Work

As the developers develop the future of SDN architecture, we can conclude that software defined networking would be the enabler of many new designs for home automation. It can also take place of cellular networks as we have shown it's capability to be deployed in clusters in small cheap SOC such as raspberry Pie or can be integrated in cellular networks to improve performance per area by distributing the traffic load and for a wide coverage than before. SDN can become more common networking architecture in the future specially in the consumer space making routing devices. As shown in Figure 6-1, the Commercial SDN software Hyperglance is one example along this direction.



Profiles

00 Default
05 Show Tennant 1
06 Show Tennant 2
07 Show Tennant 3
08 Show Single VM Cluster
10 Critical
11 Vulnerable
12 Attacked
13 Attacked vulnerable and critical
21 Capacity Bandwidth
21 Capacity CPU
23 IO Latency
24 Over SLA
30 Highlight Physical servers
31 Highlight Network
32 Ownership
40 ColourVMEstate
41 Hide Physical
42 Hide VM
42 Hide VMHost
44 HideUnPoweredVMnodes
45 HideVMPortGroup
46 HideVMStorage
47HideVMSwitch
50 Hide All
51 Show all VMware
52 Show Physical
53 Show VM

Clusters

1 - Nodes:584
2 - Nodes:3
3 - Nodes:51

Data - CPU High

id	name	ip address	type
1	localhost.localdomain	10.0.3.50	server
8	Agg-VDC4	172.20.0.22	router
42	Host_41	172.21.0.6	server
51	Host_54	172.21.1.4	server
59	UCS1-1-Blade3	172.21.2.1	switch
61	Host_77	172.21.2.4	server

Attributes

Property	Value
VULNERABILITY	44
Performance	
Processed	
CPU-AVERAGE (3600s)	87.807
CPU-MAX (3600s)	85.713
CPU-MIN (3600s)	90.56
Memory-AVERAGE (3600s)	56.605
Memory-MAX (3600s)	56.37
Memory-MIN (3600s)	56.87

C:\Users\shipperson\Desktop\Hyperglance-INFOSEC-16-04-11\resources\Networks\NEW-DEMO-FULL-27-03.json iteration31 (rev3746) 189 nodes 587 links

Figure 6-1 The Commercial SDN software Hyperglance in action

The development in all software intelligence fields would almost certainly benefit SDN. As AI algorithms will make the network more smart and reliable. We would like to further investigate the possibility of implementing it on FPGA devices as and find out the possibility of hardware reprogramming itself according to different load and traffic pattern. The further development is almost certainly immanent which can be in the networking of many cluster devices which can be industrial robots which require real time inspection. SDN can enable centralized control of such type of robots regardless of there individual function. Which is really beneficial for companies looking forward to deploy automation

Though at current stage SDN has some issues with the security as centralized system can be targeted by the network intruders to interrupt the traffic or to take control of the switches but the issues can definitely be solved as they are already solved at higher level and have proven themselves by serving on the public internet providing services. We as the enabler of SDN in common house router think that this field of networking has a bright future ahead as it is already in the stage of evolution, specially at a higher level than consumer space.

Acknowledgement

As bachelors students in the major of telecommunication engineering, we thought a little ahead of books and tried to do something that we thought is fun to learn and experiment with. However there were many difficulties that we faced during the project including lack of human resources and skills at our level. We were slow but making progress consistently as we were eager to see that how it all turns out be. There were also times that we had to reduce the target in mind and go towards the root of the problem.

Our teacher Prof. Xiao Jun Hei helped us a lot during our blockade of lack of information. He provided us with necessary equipments, support and knowledge to accomplish this project. We are very grateful to him for this wonderful opportunity to discover something new and create something new.

Also our team members who made a lot of contributions and the determination to work on this until the end of the project include Mr. Muhammad Afzal who worked closely with all the group members. He was responsible for enabling SDN on Asus rt-n16 which was a popular home router unsupported by OpenWRT. Mr. Kenyon Nsunza working on the popular Intel Galileo board and enabling WiFi and SDN on a development board for the developers to experiment with. Our team also include a skill-full member Mr. Samuel Rutunda who worked on enabling SDN on the popular SOC computer, the Raspberry Pi, which he accomplished successfully. Make it a networking device effectively.

All in all, it was great multi national experience for everyone and the memory of every achievement and applause will always be in our heart. We will never forget this experience.

Bibliography

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[2] N. Feamster, J. Rexford, and E. Zegura, “The road to sdn,” *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2559899.2560327

[3] “Openwrt,” 2015. [Online]. Available: http://openwrt.org/

[4] “Raspberry Pi,” 2015. [Online]. Available: https://www.raspberrypi.org/

[5] “Intel Galileo gen 2 development board,” 2015. [Online]. Available: http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-overview.html

[6] “Arduino,” 2015. [Online]. Available: http://www.arduino.cc/