### Cover:

Student Name: Kieran Mc Gowan

Email: kieran.mcgowan9@mail.dcu.ie

Student ID Number: 19214800

Programme: MCM - M.Sc. in Computing

Module Code: CA687

Lecturer: Long Cheng

Assignment Title: Assignment 2 - Software Defined Network Application

Submission Date: Monday, 20th April 2020.

Word Count: 860

# 

# 

### Disclaimer:

An essay submitted to Dublin City University, School of Computing for module CA640 Professional and Research Skills in Computing.

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy.

I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. By signing this form or by submitting material for assessment online I confirm that I have read and understood [DCU Academic Integrity and Plagiarism Policy](http://www.dcu.ie/registry/examinations/index.shtml).

Name: Kieran Mc Gowan

Date: Monday, 20th April 2020.

Software Defined Network Application

## Introduction

Traditional networks up to recently are built using hardware based network devices that are single purpose, expensive and rigid. These devices run vendor specific software, are difficult to maintain and upgrade and often the release cadence of fixes and upgrades are controlled by the vendor.

Examples of such devices are

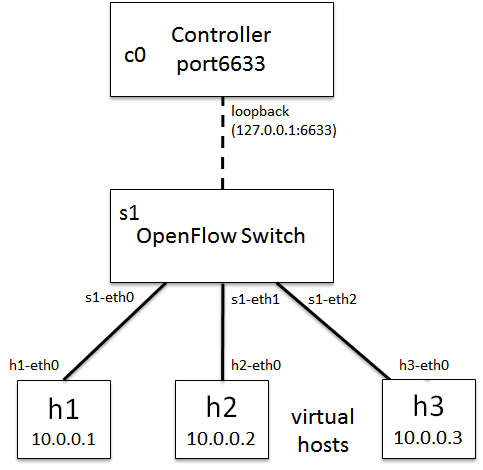
* Switches,
* Hubs
* Load balancers
* Firewalls

This leads to very expensive, inflexible and underutilised computing resources.

SDN aims to solve this problem by separating the software from the hardware, moving the software to programmable VM (Controller) and replacing the expensive hardware with commodity hardware. This means that physical devices can now be emulated in software otherwise known as SDN applications.

The motivation for choosing a Hub is that while simplistic in nature, it gives instant feedback that the fundamentals of SDN are understood quickly.

## SDN Hub



<http://sdnhub.org/resources/useful-mininet-setups/>

This project demonstrates the use of an SDN to implement a simple network hub. Mininet an SDN emulator is used as a platform to build, test and deploy software based networks that traditionally would be built using hardware components. We will program a switch to act as a network hub forwarding all packets to each host.

## Prerequisite and Caveat

This project requires the installation of Mininet, an open source network emulator. Mininet is a python based network emulation tool that comes with several pluggable controller implementations. The one demonstrated in our Futurelearn platform was Ryu but unfortunately after several attempts that application failed to execute due to python dependency failures. This is a huge disappointment for me and this project, but instead I have pivoted to another controller implementation known as POX.

## SDN Hub Implementation

### Setup:

It's outside the scope of the Assignment, but Mininet should be running on your machine. SSH into the mininet VM and ensure mininet is setup and running.



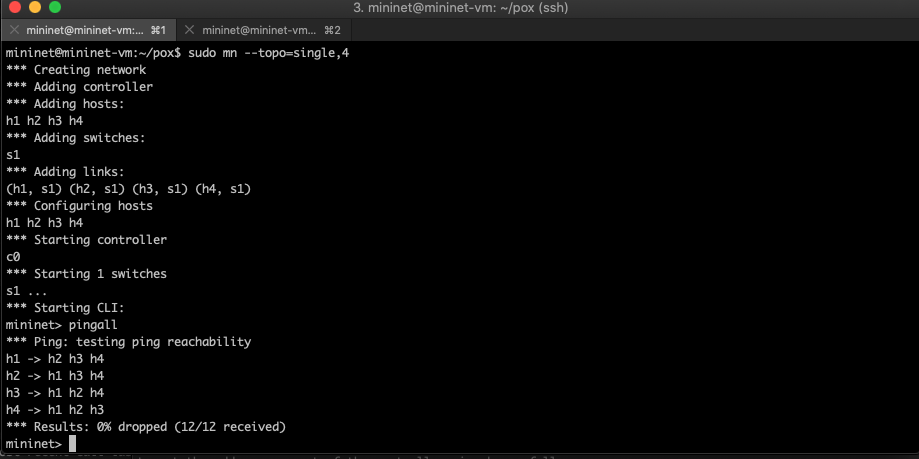
Verify OpenFlow is also installed



### Default Controller (Provided by Mininet)

Demonstrate a single topology 4 host network, 1 switch network.

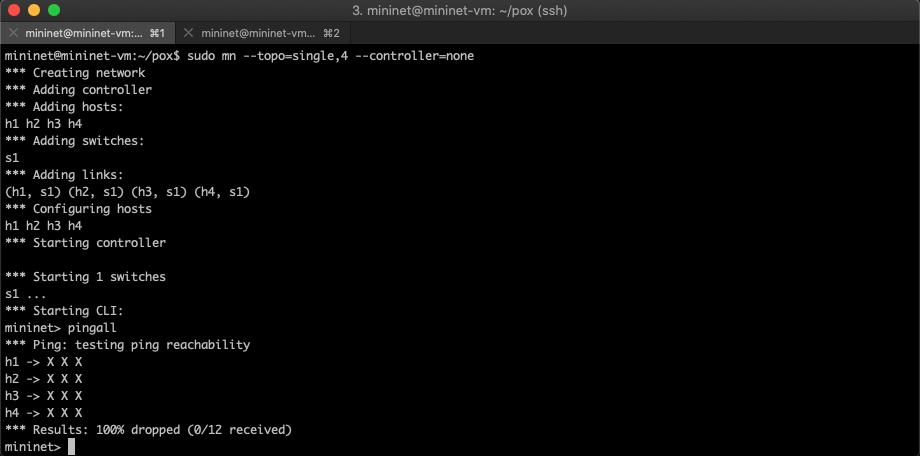
$ sudo mn --topo=single,4



### Remove the Default Controller (Provided by Mininet)

This time build a single topology 4 host network, 1 switch network but do not use the default controller. None of the hosts can see each other so all packets are dropped.

$ sudo mn --topo=single,4 --controller=none

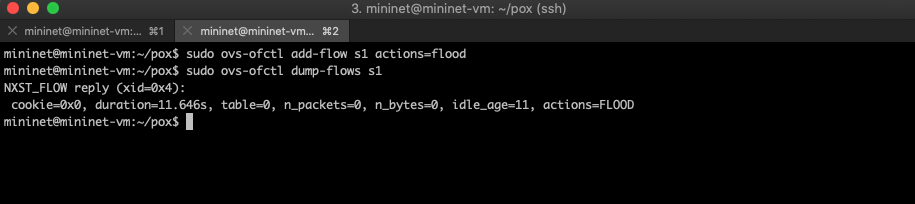


### Manually add a flood rule to the OpenFlow Flow table

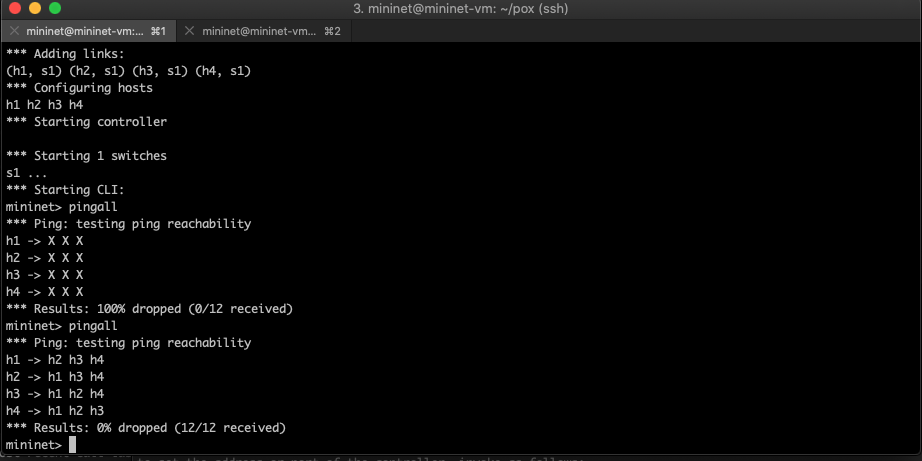
In another shell add a rule to the open flow table, his turns the switch into a hub

$ sudo ovs-ofctl add-flow s1 actions=flood

$ sudo ovs-ofctl dump-flows s1



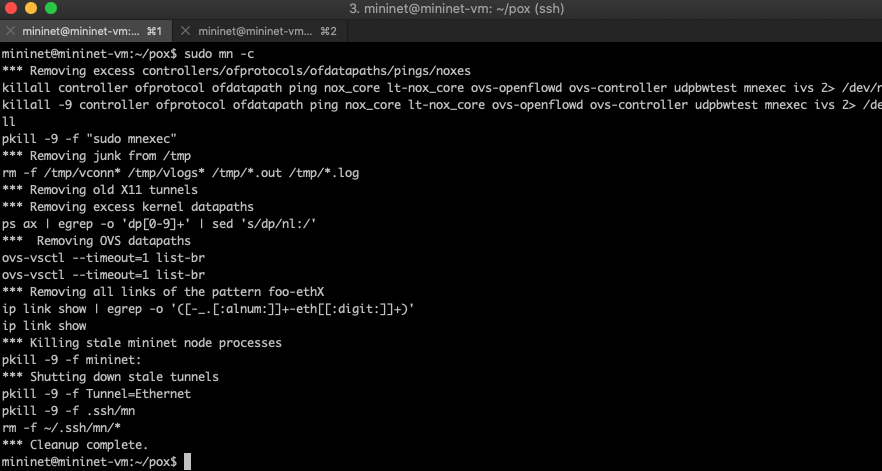
The pingall command now succeeds



### 

We have successfully programmed the switch into a hub that connects all hosts.

Finally let us reset the environment. Type exit to leave the mininet shell. Lets clean up

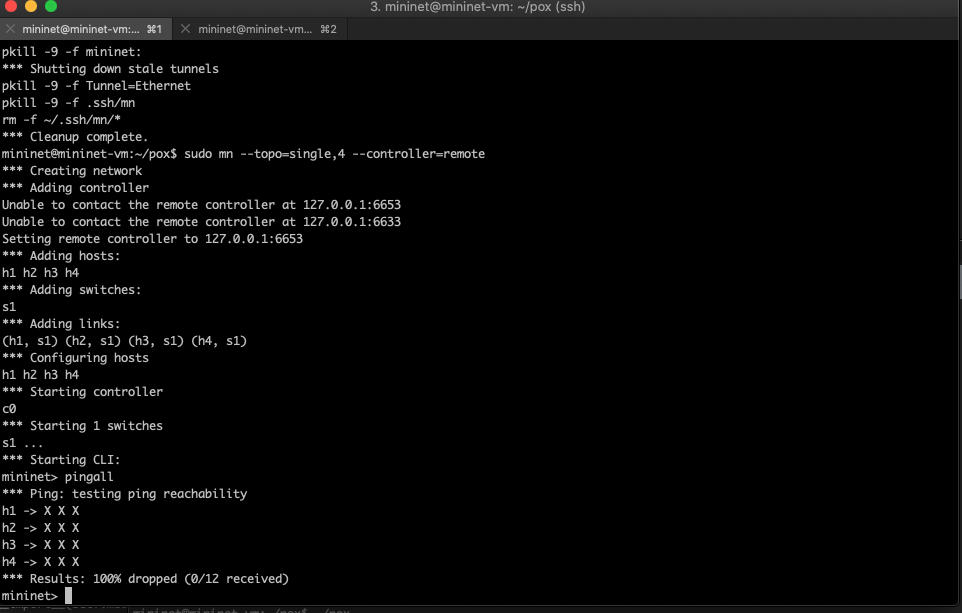


### 

### Programatically add a flood rule to the OpenFlow Flow table

Now let's re-introduce a controller only this time make it a third party controller that is more user friendly and programmable. For this demo we will use POX.

Start mininet only this time with a remote controller, since we have not started that controller yet we will receive a warning and our pingall will fail

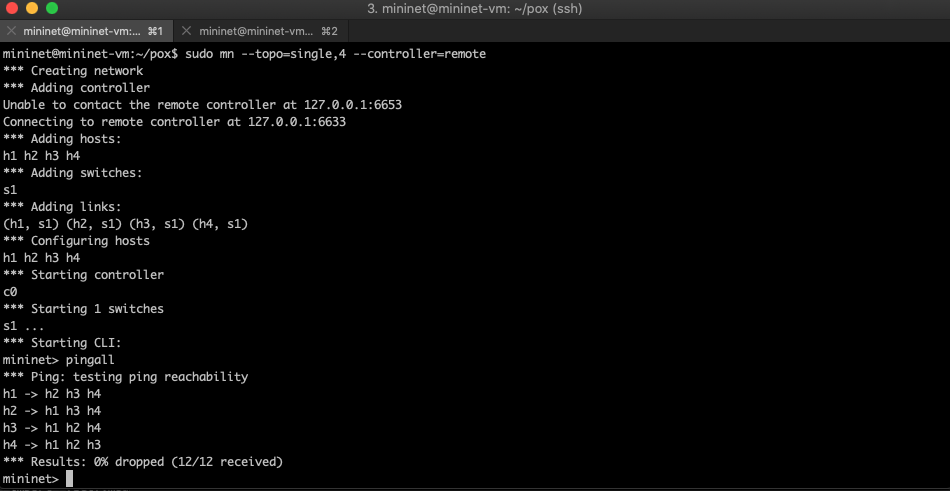


Exit mininet to return to the prompt, and In another shell let's start our remote controller passing it the pox implementation of a hub.

$ sudo ./pox.py pox.forwarding.hub



Finally when we start mininet again with the same remote controller specified, since our controller is now running, our switch is programmed as a hub and our pingall succeeds.



## Challenges and Lessons learned

The most interesting part of this assignment for me is it reminded me of a previous role I held at a well known investment bank in London. My role was an E-trading Software Engineer and that involved developing platform components for an e-trading platform known as Marketview. This was a pub / sub platform scalable to 1000’s of components. The platform connected trading Market Gateway data with Traders in real time. That platform was completely implemented in software a proprietary SDN but at its core it was composed of gateways, switches, routers, firewalls and load balancers.

Obviously from my experience with Marketview I know how customizable and flexible SDN networks can be as opposed to the traditional devices. I witnessed many times how a component upgrade such as a router version release managed to gain us valuable milliseconds in trade capture time over our competitors and at that level that type of improvement is worth millions of dollars in won revenue annually.

The largest challenge with this assignment came down to time and the tools available in python. I am an experienced programmer but not in python. Unfortunately I lost a lot of time trying to configure the python installation to get over the issues with the ryu controller. I know several of the class had the same issues I had, only we did not collaborate in time to solve them. There were many times over the time I had allocated to this project where I was tempted to abandon it such was the frustration with the python lib dependency failures. It was critical for my learning that I could reproduce the steps from the coursework in ryu and that was not possible.

## Demo.

### Sourcecode:

#### <https://github.com/sligokid/dcu-mininet-hub>

#### 

### Demo Screencast:

#### <https://youtu.be/q6EQEFTbPtA>

### 

### 