CS5229 - Advanced Computer Networks - HW 2

Part1 Due : Sep 6, 2019 2359 Hrs
Part2 Due : Oct 7, 2019 2359 Hrs
Late Penalty : 20% per day (Applicable to both parts)
TA Email : Pravein G. Kannan <pravein@comp.nus.edu.sg>**,** Ghozali Hadi <ghadi@comp.nus.edu.sg>

Part 1 - Total : 35 points (7% of Grade)

Overview

This HW exposes you to Software Defined Networking (SDN) concepts and the related toolsets. The SDN architecture decouples network control and forwarding functions such that the network control is programmable and also abstracting the network infrastructure. You can further refer to the following references on SDN[1] and OpenFlow[2].

Problem Statement

You have recently joined as a Network engineer at an Organization. The organization has three internal departments, namely, Sales, Marketing and R&D as shown in Figure 1. They are interconnected by a network where the edge switches S1, S2 and S3 respectively are SDN Switches. The Organization has a specific policy on how the communication between the three departments should be maintained. As a network engineer, you have to ensure that the networking policies meet the overall company policy. Being adventurous, you wanted to implement the network using SDN (Software Defined Networking) . The advantage of using SDN to implement the network is that : 1) Networks are easily programmable, 2) Centrally Controlled, 3) Network entities can be abstracted to optimize various resources, and 4) Usage of Open Standards (E.g. OpenFlow). Now, in order to implement the policies, you plan to write an application on top of a standard SDN Controller which talks to each of the edge switches(S1/S2/S3) as shown in Figure 1. Consider just 3 hosts operating in the respective departments (Sales, Marketing and R&D) namely, H1, H2 and H3.

[1] https://www.opennetworking.org/sdn-resources/sdn-definition
[2] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

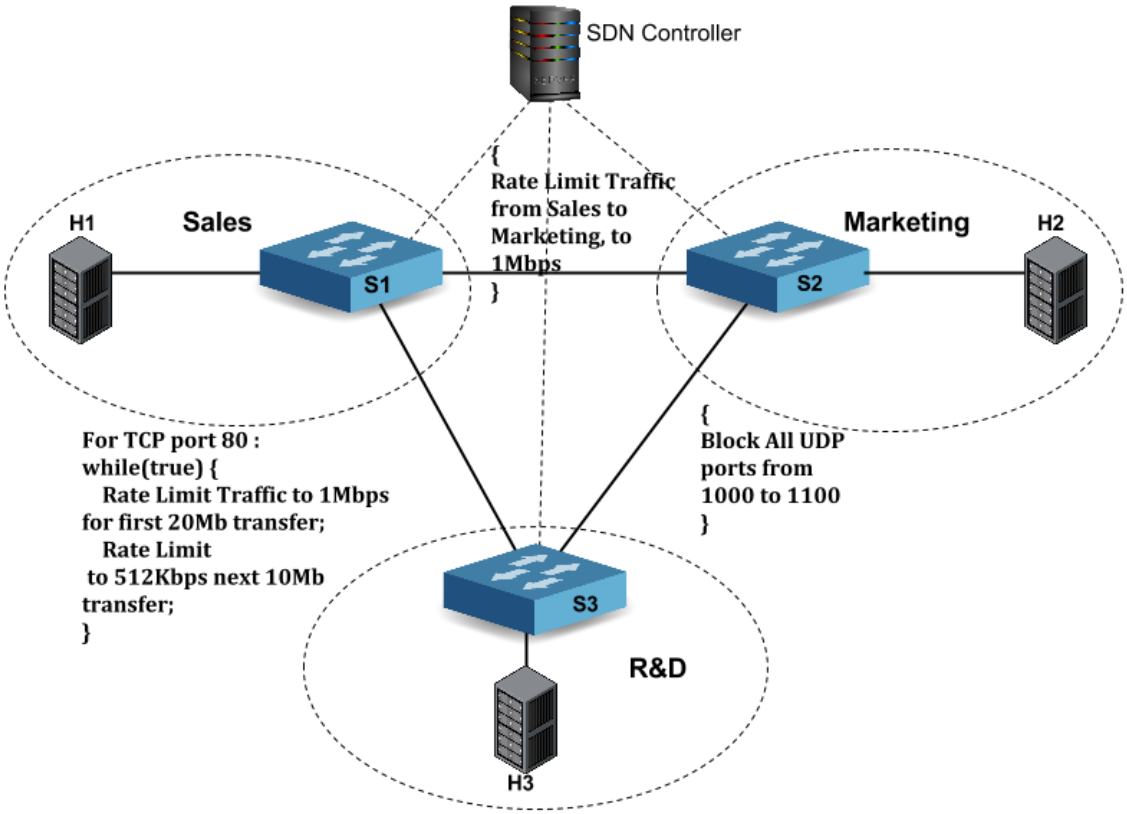


Figure 1: Network Topology

The Organizational Network Policy to be implemented by you is :

1) **Communication between H2 to H3 and vice-versa: (10 points)**
   Block all traffic using dest UDP ports from 1000-1100.

2) **Communication from H1 to H2: (10 points)**
   Rate limit traffic to 1 Mbps.

3) **Communication from H1 to H3: (15 points)**
   Regulate HTTP traffic using the below logic:
   While (True) {
   Rate Limit Traffic to 1Mbps for 20Mb transfer;
   Rate Limit to 512Kbps next 10Mb transfer;
   }

Where to Start?

1) Install VirtualBox[3] in your PC. VirtualBox supports Windows/Linux/MacOS.
2) Download the VM from http://www.comp.nus.edu.sg/~pravein/cs5229/cs5229.ova (Size: 1.6G)
3) Goto VirtualBox UI, and do "Import Appliance", and specify the location of the downloaded "cs5229.ova" file in order to import the VM.
4) Start the VM once it is imported. The VM(LUbuntu 16.04) comes with all the necessary softwares installed.
5) You can login to the vm using :

   username :cs5229
   password: cs5229
6) The topology, policy scripts and the SDN controller needed can be found in"~/CS5229/"

Getting System to work:

1. We will be using Mininet[4] to emulate our organizational network. Mininet uses OpenvSwitch to emulate the switches, and containers for the hosts connecting to the network. Start your mininet topology, by running the following:

   sudo ~/CS5229/Topology.py

   This will start the company topology. Please skim through the "Topology.py" to understand how the topology is defined in mininet

2. Start the SDN Controller, floodlight[5] (Floodlight is a java based SDN controller) run :
   a. *cd ~/CS5229/floodlight-1.2*
   b. *Java -Dlogback.configurationFile=logback.xml -jar target/floodlight.jar*

3. Verify the switches are connected to the floodlight controller. You must see the below logs in your floodlight console :

   WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:02 connected.
   WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 connected.
   WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:03 connected.

   Where, 00:00:00:00:00:00:00:01 is the fixed Datapath ID of Switch S1. 00:00:00:00:00:00:00:02 is the ID of Switch S2, and 00:00:00:00:00:00:00:03 is the ID of Switch S3

4. Now, we will be running our policies as an application over the floodlight controller. Before anything, we first setup static forwarding in our policy, so that communication happens via the shortest path. To run this policy file:
   a. ~/CS5229/Policy.py

[3] https://www.virtualbox.org/wiki/Downloads
[4] http://mininet.org/overview/
[5] http://www.projectfloodlight.org/floodlight/

5. Our static forwarding is now setup, Verify by pinging. To do this, goto mininet console, and type:

   h1 ping h2
   h2 ping h3
   h1 ping h3

   Verify if ping is successful.
   Alternatively, you can start a terminal on host h1 by running the command "xterm h1" in the mininet console. Note that the IP addresses of h1, h2 and h3 are 10.0.0.1, 10.0.0.2 and 10.0.0.3 respectively.

6. Now , its your turn to implement the company policies in "Policy.py". Generally, policies are implemented in the switches in terms of <Match, Action> rules, where a rule matches on certain header entries of a packet, and takes the corresponding action based on the rule(eg. Output to a certain port/drop, etc) This file contains basic things/layout needed to write a policy . You will need to fill(code) in the functions for the appropriate policies:

   S1toS2()
   S2toS3()
   S1toS3()

   The file is well-commented for you to figure out what is happening in "Policy.py"

   Please refer to https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Static+Entry+Pusher+API for the components and their pre-requisite matches. Typically, a policy string should contain : 1) Switch DPID, 2) Unique Policy Name, 2) Optional Cookie id, 3) Priority, 4) Matching Header components like eth_type/ipv4_src/ip_proto, etc, 5) actions.

   Some Hints:

   a) Make sure you add in your policies with a higher priority than the default static forwarding rules.
   b) For Policy 2, and 3, Make sure you read the topology.py completely to get an idea.
   c) You need to use queues defined using topology.py to implement policy 2 and 3
   d) You are supposed to use the keys supported for OpenFlow 1.3.

7. Once, you write in your functions in "Policy.py", you can run it, and verify if the flows are added to apply the policy by executing the below command, which lists the flows in the switch :

   Curl http://localhost:8080/wm/core/switch/all/flow/json | python -m json.tool

   Alternatively, you can open the firefox browser, and type url as "localhost:8080/ui/index.html" to view all controller information regarding the topology.

How to Test your policies?

1. You will have to generate traffic from the hosts (h1,h2,h3) in order to test the policies.

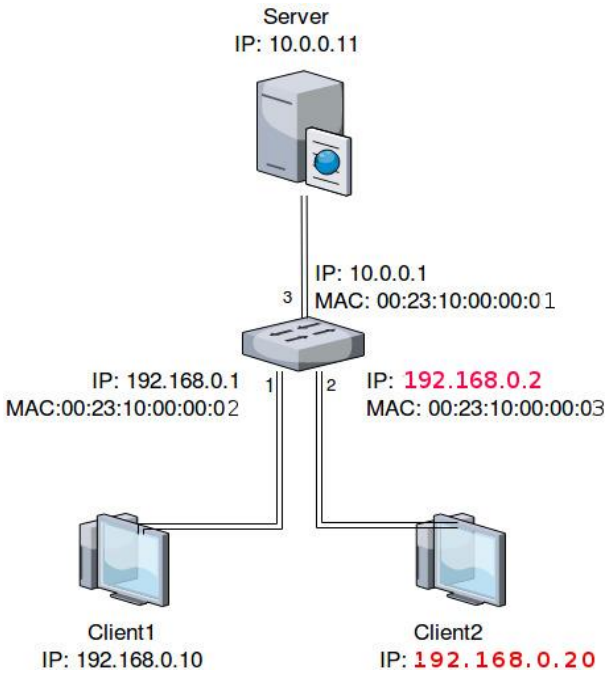
2. Initially, you can run a terminal on h1,h2 or h3. Run the following in mininet console :
    a. *xterm h1*
    b. *xterm h2*
    c. *xterm h3*
3. Generating TCP/ UDP Packets :
    a. You can use iperf3 tool to generate TCP/UDP packets between a pair of nodes, and measure the link bandwidth.
    b. TCP session b/w h1 and h2 for example :
        i. First, start server on h2 by running "*iperf3 -s -p <port no.>*"
        ii. Next, start the client on h1 to connect to h2 by running *"iperf3 -c 10.0.0.2 -p <port no.>*"
    c. UDP session b/w h1 and h2 for example :
        i. First, start server on h2 by running *"iperf3 -s -p <port no.>"*
        ii. Next, start the client on h1 to connect to h2 by running *"iperf3 -c 10.0.0.2 -p <port no.> -u "*

Submission Instruction

Please zip your completed 1) "Policy.py" and 2) "README" file, which explains how you configured the policies and iperf3 logs which show your solution actually working. Please note that we will be running the code to make sure, the iperf3 reflects the logs. Make sure, Policy.py and README also have your NAME/Matric No. filled in. Please name your zip file as <StudentID>_HW2.zip and submit to IVLE workbin "Student Submission-HW2-Part1".

Part 2 - Total : 75 points (15% of Grade)

This is the interesting part, which gives an insight on the programmability strength of SDN. You will need to implement a NAT[6] (Network Address Translator). The Network topology is shown in Figure 2.



[6] https://tools.ietf.org/html/rfc3022

Figure 2: Topology for implementing NAT

Requirements :

1) The NAT that you implement must be able to support ICMP messages (PING) from client to server.
2) It must be implemented according to the specification of https://tools.ietf.org/html/rfc5508 Refer Section 3. (Both 3.1 and 3.2)

Getting Started:

1) Let's start the topology by running the following
   sudo ~/CS5229/Nat_Topology.py

2) Before starting the SDN Controller, floodlight, we will need to disable default forwarding module, because we will be implementing a custom forwarding module which does NAT. Hence, do the following :
   Goto "~/CS5229/floodlight-1.2/src/main/resources/floodlightdefault.properties"
   And remove the line "net.floodlightcontroller.forwarding.Forwarding,\"

3) Run xterm client1 client2 server in mininet console, and in the corresponding xterm console, execute "***python setupRoute.py <client1/client2/server>***" depending on which host xterm you are on. Once it executes, check the route in the xterm terminal for each host.

4) Try running the floodlight controller, and see if the client and server ping each other. They should not be able to ping. Your main task would be to make this ping work by performing correct NAT of IP addresses (i.e client IP must be converted to public IP(10.x.x.x) while sending to the server, and in back forth). Client to Client ping must not work though.

5) There is a skeleton code in "src/main/java/net/floodlightcontroller/natcs5229/NAT.java"

Debugging/ Hacking Tips:

1. To build floodlight, you can just type "ant" in the floodlight project directory. Additionally, you could use eclipse/IntelliJ for development.
2. Start with pinging a server from client, and observe the packets received at the switch interface "S1-eth1" using wireshark. This will give you insights on what to do next.

3. Also, Tcpdump at the server interface would be helpful if the packet re-assembly had happened correctly.

4. For the syntax, you can check loadbalancer code in floodlight for reference.

Sub-Tasks:

1) Implementing Proxy-Arp Reply for the switch interface IP. (15 Points)
2) Perform outbound Translation (Client to Server). (15 Points)
3) Perform inbound Translation (Server to Client) based on ICMP Query ID. (15 Points)
4) Handle Query ID Timeout. (15 Points)
5) With most of the gateways in internet performing NAT, how can two client machines communicate with each other directly?
   Give some example clients/software which do that. (15 points)

Submission Instructions:

Please zip your completed 1) "NAT.java" and 2) "README" file, which explains the logic, and challenges you had faced.. Make sure, NAT.java and README also have your NAME/Matric No. filled in. Please name your zip file as <StudentID>_HW2.zip and submit to IVLE workbin "Student Submission-HW2-Part2".