Advanced Computer Networks (CS G525)

Lab Sheet - 5

Objectives:

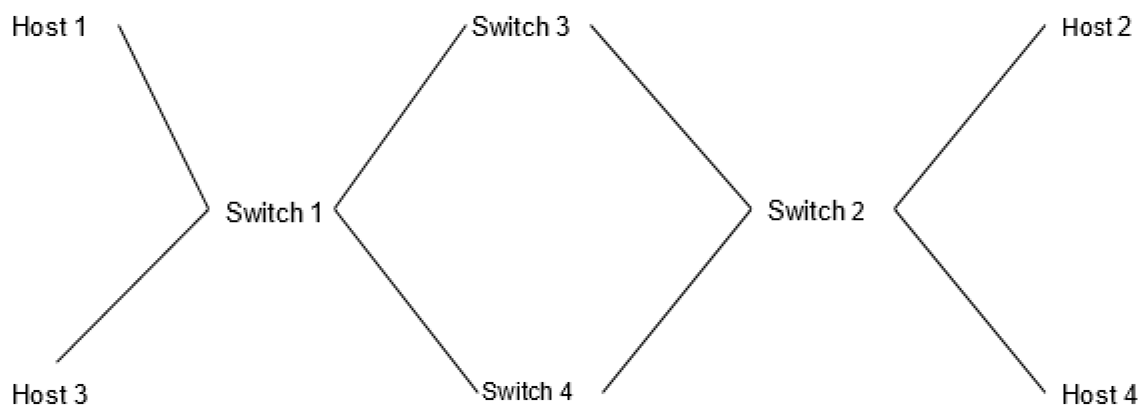
1. **To learn calculating link bandwidth using controller application**
2. **To learn modifying flow rules based on throughput**

Introduction:

In this lab session we will learn how to measure network state (e.g., link bandwidth, link latency, etc.) through a controller application and use them for traffic flow provisioning in the network.

Topology File Description:

Consider the network topology as shown in the Figure below. The topology is provided with the lab sheet in file **lab4_topo.py**. Also, the controller code for this module is provided in file **lab4_controller.py**.



Run the controller using the following commands
$ ryu-manager lab4_controller.py

Run the topology file using the following commands
$ sudo python lab4_topo.py

The topology file is similar to the lab3_topo.py file which we used in the Lab Sheet 3. If you open the topology you will find a few minor changes as listed below-

- Added 2 more hosts h3 and h4.
- Added new links and modified the existing ones such as, the link s_1-- s_3 has a bandwidth of 5 Mbps and the link s_1-- s_2 has a bandwidth of 10 Mbps.

Controller File Description:

In the **lab4_controller.py** file, the **`switch_features_handler`** function is called when a switch establishes a connection with the controller, so we are pushing the flow rules to the switches when that happens. You can use `sudo ovs-ofctl dump-flows <switch name>` command in the terminal to check the entries if needed.

Initially the flow rules are designed in such a way that all the traffic either be TCP or Non TCP flows, goes through the switch s_3. No traffic goes through switch s_4 initially. After that TCP flows forwarded through s_4 due to high utilization of the based on the utilization of the links connected to s_3. Hence the controller is acting as a "Load Balancer" for the network.

Along with the **`switch_features_handler`** function, we have defined a monitor thread which calls a **`Prober`** function which requests for **flowstats** from the switches using:

```
req = parser.OFPFlowStatsRequest(datapath)
```

The **flowstats** are requested again after waiting for 5 seconds. And we are calling that function every 10 seconds from the launch function.

Bandwidth Calculation:

Now for calculation of the bandwidth, we need to process the **flowstats** we received from the switches. By having a difference of 5 seconds between the queries we can calculate the numbers of bytes transferred and then dividing it by 5 seconds provides us the throughput of the link. The **Flow State Reply** object contains all the required information. The **Flow State Reply** is handled by the **`flow_stats_reply_handler`** function. This function extracts the required information from the object and performs some operations based on the results.

The **Flow State Reply** contains an entry corresponding to multiple flows passing through a switch and we can iterate over those flows using a loop. Sample output of all the entries in the object is:


```
[{'packet_count': 0, 'hard_timeout': 0, 'byte_count': 0,
'duration_sec': 2, 'actions': [{'max_len': 0, 'type': 'OFPAT_OUTPUT',
'port': 2}], 'duration_nsec': 214000000, 'priority': 32768,
'idle_timeout': 0, 'cookie': 0, 'table_id': 0, 'match': {'dl_type':
'IP', 'in_port': 1, 'nw_dst': '10.0.0.3/32'}}, {'packet_count': 0,
'hard_timeout': 0, 'byte_count': 0, 'duration_sec': 2, 'actions':
[{'max_len': 0, 'type': 'OFPAT_OUTPUT', 'port': 1}], 'duration_nsec':
177000000, 'priority': 32768, 'idle_timeout': 0, 'cookie': 0,
'table_id': 0, 'match': {'in_port': 3}}, {'packet_count': 0,
'hard_timeout': 0, 'byte_count': 0, 'duration_sec': 2, 'actions':
[{'max_len': 0, 'type': 'OFPAT_OUTPUT', 'port': 2}, {'max_len': 0,
'type': 'OFPAT_OUTPUT', 'port': 3}], 'duration_nsec': 177000000,
'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0,
'match': {'in_port': 1}}, {'packet_count': 0, 'hard_timeout': 0,
'byte_count': 0, 'duration_sec': 2, 'actions': [{'max_len': 0,
'type': 'OFPAT_OUTPUT', 'port': 1}], 'duration_nsec': 177000000,
'priority': 32768, 'idle_timeout': 0, 'cookie': 0, 'table_id': 0,
'match': {'in_port': 2}}]
```

You can iterate through each flow entry using a for loop. Some objects like actions are sub-lists inside the **stat list**. For accessing them you again need to iterate over **stat.actions** objects.

For Bandwidth calculation, we will rely on the ***byte_count*** variable of each flow object. The ***byte_count*** is a counter of the number of bytes flowing through a flow entry, i.e., it is **cumulative** also each switch has multiple flow entries.

Task 1: Calculate Instantaneous Bandwidth of each link in the topology

1) To calculate the instantaneous bandwidth of a link we need to subtract the earlier byte count from the later byte count and divide it by the elapsed time. We use 2 arrays for storing the initial and final byte counts and time durations. For e.g., the bandwidth of s_1-s_3 link = (ByteArray[1] (later byte count) - ByteArray[0] (later byte count)) / time.o

- Now open the xterm window for h2 and type iperf -s -t 1000, i.e., creating a TCP server at h1 for 1000 seconds.
- Now open xterm of h1 and type iperf -c 10.0.0.2 -t 1000, i.e., creating a TCP client for h2
- Monitor the throughput calculated using the controller.
- Now open the xterm window for h4 and type iperf -s -u -t 1000 i.e., creating a UDP server at h4 for 1000 seconds.
- Now open xterm of h3 and type iperf -c 10.0.0.4 -u -t 1000, i.e., creating a UDP client for h2.
- Again monitor the throughput with two flows at the controller.

The throughput will increase initially then stabilize later.

2) With the ping still running, run `iperf h1 h4` from the topology terminal i.e. “mininet>”. You should notice a significant jump in the bandwidth utilization of some links. Verify whether the links are associated with the traffic flows generated using iperf.

Task 2: Modify Flow Rules based on Instantaneous Bandwidth of link in the topology

- As discussed above, the flow rules are designed to only pass through s_3. Now, based on the throughput calculated, we may want to shift the TCP connection to a higher bandwidth link. The higher bandwidth link is present between s_1 and s_4.

- After sending the **Flow State** Request using **Prober** function, the **Flow State** reply message calculated the throughput and if it exceeds a certain threshold then the corresponding TCP flow which was running through s_3 are changed to s_4 using the **flow mods.**

Go through both topology and controller files to understand the scenario.

Exercise:

TASK: Calculate a link delay and use that to modify the flow rules.

Algorithm Description:
The controller creates a probe packet at time T1 and sends it to s_1. This packet is forwarded by s_1 to s_2. Switch s_2 sends this packet to the controller. Special nw_proto field value, i.e., 253 is being used to distinguish it from the controller. Let us assume the controller receives the packet at time T2.

This gives Link Latency = T2-T1

Explanation of the required code:

```
Timer(1, Prober, recurring = True)
```

Timer function schedules a call to Prober which is a function in the same file. It will be called after every 1 second. This is because of the recurring attribute.

```
    if switches ==2:
    for i in range(1,3):
        match=of.ofp_match()
        match.in_port=1
        msg=of.ofp_flow_mod()
        msg.match = match
        msg.actions.append(of.ofp_action_output(port = 2))
        SwitchMap[i].connection.send(msg)

        match=of.ofp_match()
        match.in_port=2
        msg=of.ofp_flow_mod()
        msg.match = match
        msg.actions.append(of.ofp_action_output(port = 1))
        SwitchMap[i].connection.send(msg)
```

This snippet helps in adding flow rules to the switches in an automated manner using a for loop. Understanding this now will be helpful in Module 2 Bandwidth measurement.

Switches communicate to the controller through a designated port to the controller. We specify the output port as the controller port using the below lines that create a flow rule to send packets having 253 nw_proto fields to the controller, i.e., the probe:

```
    match=of.ofp_match()
    match.dl_type = pkt.ethernet.IP_TYPE
```

```
      match.nw_proto = 253
      msg=of.ofp_flow_mod()
      msg.match = match
      msg.actions.append(of.ofp_action_output(port =
of.OFPP_CONTROLLER))
      SwitchMap[2].connection.send(msg)
```

TASK 2: Link s1---s2 latency measurement.

As you might have observed, in this method the time difference of T2 and T1 includes delay between controller and switch (s_1 or s_2). If delay observed between controller and switch is T3 then the correct measurement of link S1----S2 latency becomes T2-T1-2*T3

Now modify the controller program to incorporate the above-mentioned modification in link latency measurement.

Hint: Can we use another probe to achieve this delay T3?

Note: Repeat the experiment by running iperf traffic from h1 to h2.
Use the following commands to enable iperf traffic generator.

```
Open Xterm for h1 and h2
mininet> xterm h1 h2

On xterm of h1 create server:
$iperf -s

On xterm of h2 start client:
$iperf -c -t 100
```

-t specifies the number of seconds.

xx—00—xx