Department of Computer Science and Information Systems, BITS Pilani
First Semester 2019-2020
Advanced Computer Networks (CS G525)
ASSIGNMENT #2

Date of posting: 27-10-2019 (1:30 AM) **Submission Deadline: 05-10-2019 (Mid-Night)**
Max. Marks: 18 [Weightage: 9%] **Mode: Individual**

Guidelines:

1) All students are expected to work with full honesty and must avoid to involve himself/herself in any kind of unethical practices. Consider this assignment as a learning opportunity.
2) Submitted code should not have unwanted code, which is not relevant to the problem asked to solve. Add comments wherever necessary or required.
3) Any doubts/clarification can be posted on the discussion forum at NALANDA.

QUESTION-1 Write NS3 Simulation script for the following scenarios.

a) Create a Topology

Create a "full binary tree" topology in NS3, as shown in Fig.1. The code supplied by you should be dynamic, i.e., if '7' is passed as an argument to the code, a full tree topology should be created.

A **full binary tree** (sometimes proper **binary tree** or 2-**tree**) is a **tree** in which every node other than the leaves has two children. The code supplied by you should work for the number of nodes(n) as 7,15,31......

The link parameters like delay, bandwidth, etc. can be specified as per your choice.

For the Netanim part of the topology, write Netanim code in the simulation file to specify the position for exactly **seven** nodes only (using **SetConstantPosition()**), i.e. the **coordinates** of exactly 7 nodes should be specified for the animation. You can make Netanim part dynamic as well but it is not desired.

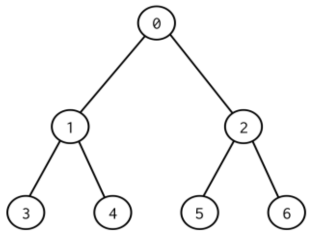

Fig.1

b) Adding New TCP congestion control algorithm

In this part, you have to create a new congestion control algorithm that supports the following methods:

```
virtual std::string GetName () const;
virtual uint32_t GetSsThresh (Ptr<const TcpSocketState> tcb, uint32_t bytesInFlight);
virtual void IncreaseWindow (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked);
virtual Ptr<TcpCongestionOps> Fork ();
```

For sample implementations of the above functions you can refer the **tcp-congestion-ops.cc** file <ns-allinone-3.28.1/ns-3.28.1/src/internet/model/tcp-congestion-ops.cc>

The congestion control algorithm comprises two phases (i) Slow Start and (ii) Congestion Avoidance while increasing the congestion window, i.e., IncreaseWindow(). Selection of phase needs to be implemented correctly in IncreaseWindow().

And for decreasing the congestion window GetSSThresh() is used.

You have to customize these functions: SlowStart(), CongestionAvoidance(), GetSSThresh() as described below:

From the options described below, you have to implement only one case (1, 2 or 3) for each of the function which is mentioned in the file named as "*question1_prob_allocation*.pdf"

For example, student with ID as 2014HS120498P have to implement case (3) for SlowStart (), case (1) for Congestion Avoidance () and case (1) for GetSSThresh ()

Variants of SlowStart ():

(1) Congestion window increases by the number of segments in-flight for the previous packet drop event.

(2) Congestion window increases by the amount given as -
(Initial Slow Start Threshold in Segments)/10000
Note: Initial threshold is defined in bytes in NS3.

(3) Slow Start of the New Reno TCP.

Variants of Congestion Avoidance ():

(1) Congestion window increases by the amount equals to the-
(Initial Slow Start Threshold in Segments)/10000.
Note: Initial threshold is defined in bytes in NS3.

(2) Congestion window increases by the number of segments in-flight of the previous packet drop event.

(3) Congestion Avoidance of New Reno TCP.

Variants of GetSSThresh ():

(1) Congestion window is calculated as product of Segment_Size and the ratio of Congestion Window to the Initial Slow_Start_Threshold.
(2) Congestion window should become the ratio of Square of Congestion Window and the Segment Size.
(3) Congestion window is calculated as the product of Segment_Size and the ratio of Congestion Window to (Bytes in-Flight + 1).

c) Throughput and Congestion Window Calculation

For this part, you can use the topology of 7 nodes created in the first part, i.e. (a). Create a TCP flow and UDP traffic flow between any two nodes in the topology. Flow parameters can be selected as per your choice, including the TCP variant selection.
If you have implemented the part (b), then you MUST use that implementation for TCP flow.
Your simulation script should calculate the throughput for both the flows and congestion window variation for TCP flow as specified below:

(i) Calculate the **throughput per second** for both the flows using the FlowMonitor class in NS3. Write the throughput values with time stamp in a separate file named as throughput.txt
Hint: You can put the throughput calculation code in a function and schedule the same for every second of the simulation using the below line
```
Simulator::Schedule(Seconds(1),&Function_name, parameter1, parameter2,
parameter3, parameter4, parameter5);
```
Where parameter1, parameter2 ... are the parameters that are passed to the function named ***'Function_name'***

(ii) Calculate the congestion window for the TCP flow and store the values in a file named as cwnd.txt.

Deliverables for each part of the question: (See submission instructions on NALANDA)

a) Topology simulation script implemented in C++ language. [2M]

b) Submit two files named as tcp-modxxx.cc and tcp-modxxx.h. Where xxx is last three digits of your BITS ID. [5M]

c) Submit the C++ simulation script with throughput and congestion window calculations along with the throughput.txt and cwnd.txt files. [2M]

QUESTION-2 Write python scripts for the following SDN scenarios.

a) **Create a Topology**

Write a python script to create a "dynamic" dumbbell network topology, as shown in Fig.2, which comprises two switches (S1 and S2) and four hosts (H0, H1, H2, and H3).

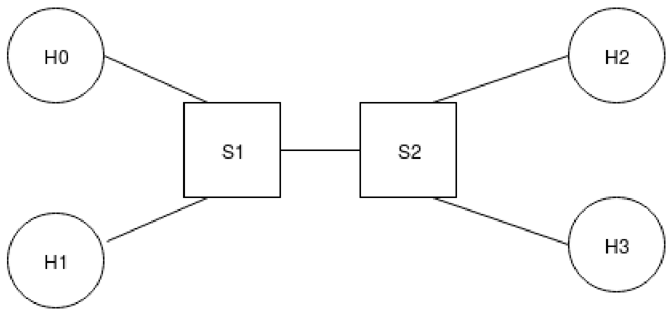

Fig.2

Your python script should take two arguments, 'm' and 'n' where 'm' is the number of hosts connected to the switch S1, and 'n' is the number of hosts connected to switch S2. You can assume number of switches are always fixed to two.

b) **Selective Switch Behavior Implementation**

For this part, use the topology file created in part (a) with m = 2 and n = 2 and add the code to connect the topology to a remote controller. Implement a controller which pushes the flow rules in S1 & S2 such that H1 can ping H0 and H3, whereas it should not be able to send HTTP request to H3.
Note: The hostnames H0, H1, H2, and H3 are just for the representation. You can use different names.

c) **Malicious SDN Controller Implementation**

Write a python script to create a topology in which three hosts (H1, H2, and H3) are connected to a switch (Switch is not shown in the Fig. 3), and a controller controls the switch.

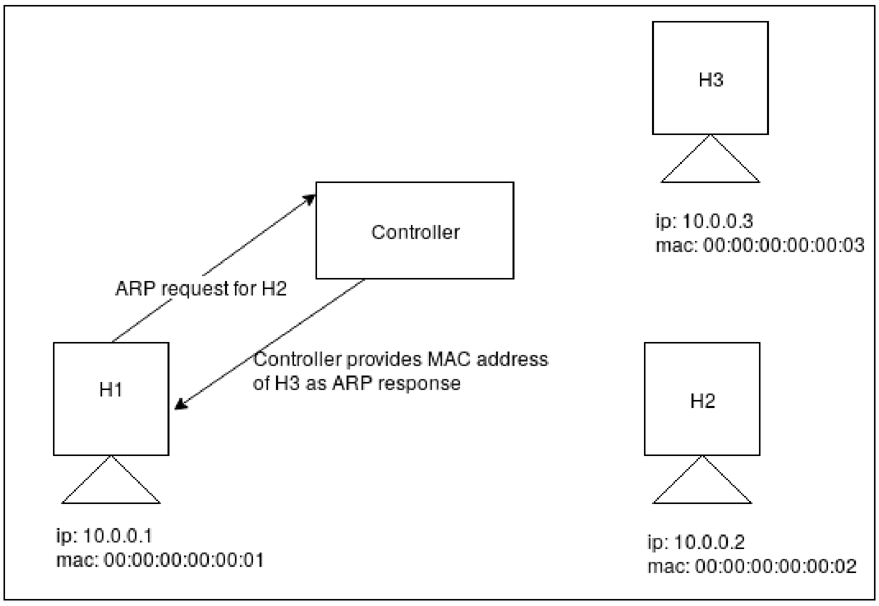

Fig.3

Implement the scenario presented by Fig.4 and described below:

- Host H1 (shown below) wants to ping H2.
- The malicious controller acts as an intruder, changes the required packet header fields (addresses) in the packet, and sends a ping request to H3.
- Host H3 responds to the ping with the assumption that the request is intended for it. The controller modifies the response again and sends the response back to H1.
- Host H1 considers that it has successfully communicated with H2, but actually it has communicated with H3.

Note: To have fixed MAC Addresses, this code can be used in your topology code.

```
net = Mininet(topo=None,
              link=TCLink,
              build=False,
              autoSetMacs = True,
              ipBase='10.0.0.0/8'
              )
```

As part of the implementation, you are supposed to follow the below-mentioned steps in the controller code:

1) Handle ARP request from host H1 as shown in the Fig.3 above and give an ARP response with MAC address of host H3.
2) Handle ICMP echo request and response messages for the ping. The required changes to the header fields of the packet are also shown in the Fig.4.

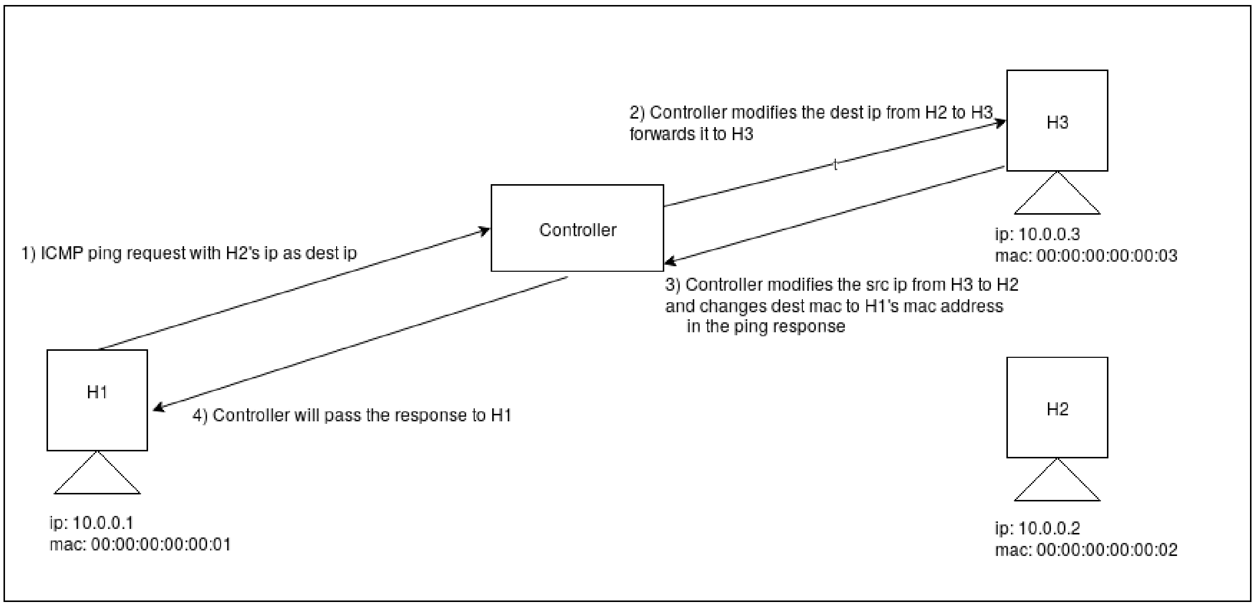
2) Controller modifies the dest ip from H2 to H3, forwards it to H3
H3
Controller
1) ICMP ping request with H2's ip as dest ip
ip: 10.0.0.3
mac: 00:00:00:00:00:03
3) Controller modifies the src ip from H3 to H2 and changes dest mac to H1's mac address in the ping response
H1
4) Controller will pass the response to H1
H2
ip: 10.0.0.1
mac: 00:00:00:00:00:01
ip: 10.0.0.2
mac: 00:00:00:00:00:02

Fig.4

Note:

- You can hard code the MAC and IP addresses of the hosts in the controller code wherever required.
- Test your implementation by pinging from host H1 to H2 and use the tcpdump tool to see the request and reply being generated by H1 and H3.

Deliverables for each part of the question: (See submission instructions on NALANDA)

a) Topology file to be coded in python language. [2M]

b) Controller file to be coded in python language. [3M]

c) Controller file to be coded in python language. [4M]
