PA1: Data Link Routing and Forwarding (10 Points)

1 Environment

To set up the environment for this assignment, you must follow the instructions from "`PA0`". The instructions can be found at "`Assignment/PA0/README.pdf`". Once the VM has booted (by using the command "`vagrant up`") and you have established a SSH connection (by using the command "`vagrant ssh`"), you should enter the assignment directory by running the command "`cd PA1`". Now you are ready to run the assignment!

2 Task

For this assignment, your task is to implement data link layer routing (using Spanning Tree Protocol (STP)), and MAC learning and forwarding. You will create a forwarding table at each switch, implement STP and MAC learning to populate those forwarding tables, and implement MAC forwarding to forward DATA packets. For this assignment, you can assume that switches and clients never fail. But the link status can change dynamically, i.e., during the run time, existing links can be removed, or new links can be added, or the link cost can change. Your implementation must be resilient to changes in link status.

3 Source Code

For this assignment, you will do your implementation inside a simulated network environment. The simulated network has switches, clients (end hosts), links, and packets just like a real network. Files "`switch.py`", "`client.py`", "`link.py`" and "`packet.py`" contain the implementations of a switch, a client, a link, and a packet respectively. You must **not** modify any of these files, however you can import the classes defined in these files and use their fields and methods for your implementation. You will do your implementation inside the file "`STPswitch.py`". "`STPswitch`" is a subclass of the "`Switch`" class and inherits all its fields and methods while overriding its "`handlePacket`", "`handleNewLink`", "`handleRemoveLink`", and "`handlePeriodicOps`" methods.

"`handlePacket`" method is called every time a switch receives a packet (DATA or CONTROL).

"`handleNewLink`" method is called every time a new link is added to the switch or the cost of an existing link changes.

"`handleRemoveLink`" method is called every time an existing link is removed from the switch.

"`handlePeriodicOps`" method is called periodically. The period is set using the "`heartbeatTime`" value in the json file as described in the next section.

You must **not** override any other "`Switch`" class methods, but you can add new fields and methods to "`STPswitch`" as you see fit.

Tip 1: For each switch, all the links directly connected to that switch are stored in the "`links`" data structure in "`switch.py`" file. Also, whenever a new link is added or removed or the link cost changes, this information is updated automatically in the "`links`" data structure and the respective "`handle...`" method is called.

Tip 2: Go through "`packet.py`" to understand packet format and types. In particular, there are two kinds of packets – DATA packets that are generated by clients and forwarded by the switches, and CONTROL packets that are generated and exchanged between switches to implement the routing algorithm. Refer to "`sendDataPackets`" method in "`client.py`" to understand how new packets are created. Go through "`link.py`" to understand various link parameters, such as "`cost`" and "`status`".

4 Running the Code

To create and start the network simulator, run the following command inside the VM,

```
$ python network.py [networkSimulationFile.json]
```

The json file argument specifies the configuration of the simulated network. This is explained in the next section.

To run the experiments with all the provided network simulation files using a single command,

```
$ ./run_all.sh
```

To clean temporary files from the previous run of the experiment, run the following command,

```
$ ./clean.sh
```

5 Network Simulation File

A sample network simulation file can be found at "`01.json`". The "`switch`" and "`client`" lists in the file specify the address of all the switches and clients in the network. Switches are addressed using numbers and clients are addressed using letters. Letter "X" is reserved for broadcast address and cannot be used as a client address. Next, given n clients in the network, each client periodically sends out n DATA packets, one of which is a broadcast packet (with destination address "X") and the remaining $n-1$ packets are destined to the $n-1$ clients (not including the source client). A link is represented as `[e1, e2, p1, p2, c]` where a node (switch or client) `e1` is connected to node `e2` using port `p1` on `e1` and port `p2` on `e2` and `c` is the cost of the link. Also, the links in the network can be added or removed dynamically as specified in the "`changes`" list. An existing link can be removed using the format `[t, [e1, e2], "down"]`, meaning the link between nodes `e1` and `e2` would be removed at time `t`. Similarly, a new link can be added using the format `[t, [e1, e2, p1, p2, c], "up"]`, meaning a link between nodes `e1` and `e2` would be added at time `t`. The above format for link add can also be used to change the cost of an existing link. The "`handlePeriodicOps`" method is called every "`heartbeatTime`". You can change the value of "`heartbeatTime`" in the json file to decide the rate of periodic operations. The network simulator runs for a fixed duration of time stated in the "`endTime`" field. The files provided have "`endTime`" set to 1000 simulation time units, which equates to $\sim$100 seconds in real time. **So, you should wait for each test case to run for $\sim$2 minutes before it prints the final output.** You may change the "`endTime`" value for your own testing.

6 Output

At the "`endTime`", simulator flushes out all the outstanding packets in the network, and generates a last batch of broadcast and non-broadcast packets between each pair of clients. It also tracks the route taken by each of those packets, and prints it as the final output on the stdout. Your solution must converge to the correct spanning tree and forwarding table entries before the "`endTime`" for the output to match the correct output. If the output path between a pair of clients is empty, i.e., [], then

it means the packet generated from the source client did not reach the destination client (probably because your solution converged to a wrong route). Below are some implementation suggestions:

1. Do not generate too many unnecessary CONTROL packets! Only generate them periodically (every "`heartbeatTime`" provided in the json file) and when your local state *changes*. Otherwise, it may overload the network, not allowing the DATA packets to go through to the destination before the experiment ends. This could result in either incorrect or empty paths in the final output! This is a very important consideration whenever you are implementing a network protocol in the real world — CONTROL packets must **not** overwhelm the network!
2. Do not send DATA packets (broadcast or non-broadcast) over the link/port packet arrived on or over an INACTIVE link/port.
3. Send CONTROL packets over all available links/ports at a switch.
4. When Case 2 in STP succeeds, i.e., you choose a new next hop link, make sure to make the new link ACTIVE.
5. After receiving each CONTROL packet, you should check whether the link on which the packet arrived needs to be ACTIVE or INACTIVE, and explicitly change the status of the link as one or the other each time.
6. Make sure to remove all entries from the forwarding table corresponding to a link that has become INACTIVE.

7 Grading

We will test your submission against 10 test cases (network simulation files), for a total of 10 test case runs. For each test case run, if your entire output matches the correct output, you will get 1 point, else a 0. **No partial credit.** We will provide you with 3 out of those 10 test cases for your testing. The remaining 7 test cases are private, and will **not** be released at any point. However, we will release your output for each test case run to let you know which test case runs you passed and which ones you failed. You are highly encouraged to create your own test cases to test the robustness of your implementation.

IMPORTANT: Your code must **not** print any custom / debug statements to the terminal (stdout) other than what the simulator already prints. Violations of this guideline will result in a 25% grade penalty per test case run.

8 Debugging

The sample network simulation files contain the correct routes between each pair of clients, which you can use to debug your implementation. The network simulator also generates .dump files for each switch and client. These files contain information about each packet received by a switch or a client during the run time of the simulator. The files also tag a received DATA packet as "DUP PKT" if the packet is a duplicate of some previously received DATA packet and "WRONG DST" if the destination address of the DATA packet does not match the address of the recipient client. You can use these files to debug your implementation.

9 Submission

You are required to submit a single file "`STPswitch.py`" on Brightspace.