PA2: Network Routing and Forwarding (10 Points)

1 Environment

To set up the environment for running this assignment, you must follow the instructions from `PA0`. Download the code for this assignment from Brightspace. The code will be inside the directory "`PA2/`". Open the terminal, and "`cd`" to the directory where you downloaded the assignment. Then enter the assignment directory by running the command "`cd PA2`". Now you are ready to run the assignment!

2 Task

For this assignment, your task is to implement network layer routing (i.e., Distance Vector (DV) / Link State (LS)) and forwarding. You will create a routing table at each router, implement DV/LS to populate those routing tables, and use those routing tables to forward DATA packets. For this assignment, you can assume that routers 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 routers, clients (end hosts), links, and packets just like a real network. Files "`router.py`", "`client.py`", "`link.py`" and "`packet.py`" contain the implementations of a router, 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 files "`DVrouter.py`" and "`LSrouter.py`". "`DVrouter`" and "`LSrouter`" are subclasses of the "`Router`" class and inherit all its fields and methods while overriding its "`handlePacket`", "`handleNewLink`", "`handleRemoveLink`", and "`handlePeriodicOps`" methods.

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

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

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

"`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 "`Router`" class methods, but you can add new fields and methods to "`DVrouter`" and "`LSrouter`" as you see fit.

Tip 1: For each router, all the links directly connected to that router are stored in the "`links`" data structure in "`router.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 routers, and CONTROL packets that are generated and exchanged between routers 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`".

Tip 3: In this assignment, you will need to exchange routing tables and neighbor lists between routers. This information will typically be stored using data structures such as a dictionary or a list. However, a packet's content can only be of type String (refer to "`content`" field in "`packet.py`"). To convert a data structure to String, use "`dumps(data structure)`", and to convert the String back to the data structure, use "`loads(String)`".

4 Running the Code

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

```
$ python network.py [networkSimulationFile.json] [DV|LS]
```

Use "`DV`" option to run the Distance Vector implementation and "`LS`" option to run the Link State implementation. 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 "`router`" and "`client`" lists in the file specify the address of all the routers and clients in the network. Routers are addressed using numbers and clients are addressed using letters. Next, given n clients in the network, each client periodically sends out $n-1$ DATA packets, one to each of the other $n-1$ clients in the network (excluding itself). A link is represented as `[e1, e2, p1, p2, c]` where a node (router 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 value of "`infinity`" in the json file specifies the cost of infinity for Distance Vector. 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 ~100 seconds in real time. **So, you should wait for each test case to run for ~2 minutes before it prints the final output.** You may change the "`endTime`" value for your own testing.

A Note about Addressing. At the network layer, the addresses tend to be *hierarchical* to save space in the routing tables. But for simplicity, in this assignment all the addresses are *flat*, just as in `PA1`. So, each routing table will need to store information about every other router and host in the network.

6 Output

At the "`endTime`", simulator flushes out all the outstanding packets in the network, and generates a last batch of 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 routing 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).

IMPORTANT: 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!

7 Grading

We will test both your DV and LS submissions against the same 10 test cases (network simulation files), for a total of 20 test case runs. For each test case run, if your entire output matches the correct output, you will get 0.5 points, 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 router and client. These files contain information about each packet received by a router or a client during the run time of the simulator. The files also tag a received DATA packet as "DUP PKT" if the DATA packet is a duplicate of some previously received DATA packet and "WRONG DST" if the destination address of the 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 two files "`DVrouter.py`" and "`LSrouter.py`" on Brightspace.