Lab 2: Distributed Network Routing Protocols (10 Points)

1 Download Lab

Download the Lab 2 files from Brightspace. The source code will be inside the directory "`Lab2/`". You must use the environment from Lab 0 to run and test your code. Next, open a terminal and "`cd`" into the "`Lab2/`" directory. Now you are ready to run the lab!

2 Task

For this lab, your task is to implement network routing (Distance Vector (DV) and Link State (LS)) and network forwarding. You will create a *routing table* at each router, and implement DV/LS to populate those routing tables. Next, you will implement forwarding logic at each router to forward DATA packets using the information in the router's routing table. For this lab, you can assume that we **never add** to the initial set of routers and clients, and that routers and clients **never fail**. But link status can **change dynamically**, i.e., during the runtime, existing links can be removed, or new links can be added, or the link cost can change. Your solution must be resilient to such changes.

3 Source Code

For this lab, 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 (including the initial set of links) 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`". Whenever a new link is added or removed or the link cost changes, **this information is updated automatically in the "`links`" data structure by the starter code**, and the "`handleNewLink`" or "`handleRemoveLink`" method is called inside "`DVrouter.py`" and "`LSrouter.py`".

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: To implement DV and LS, you will need to exchange routing tables and neighbor lists between routers using the CONTROL packets. 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

You must run and test your code on `ecegrid` using the environment from Lab 0. If your code does not run in that environment, you will not get any credit! Start the simulator using the command,

```
$ python3 network.py [networkConfigurationFile.json] [DV|LS]
```

The json file argument specifies the configuration of the simulated network. This is explained in the next section. You are provided with three sample json files ("`01.json`", "`02.json`", "`03.json`"). Use "`DV`" option to run Distance Vector implementation and "`LS`" option to run Link State implementation.

To run the experiments with all the provided network configuration json files, run the command,

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

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

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

5 Network Configuration Json File

A sample network configuration json 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 $n-1$ unicast 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 connecting `e1` and `e2`. You can assume that there will be **at most** one link between any two nodes at any given time. Further, 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 port `p1` of node `e1` and port `p2` of node `e2` would be added at time `t`, and the cost of that link would be `c`. The above format for link addition 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 provided json files have "`endTime`" set to 1000 simulation time units, which equates to $\sim$100 seconds in real time. **So, you should wait for each experiment to run for $\sim$2 minutes before it prints the final output.** You may change the "`endTime`" value for your own testing.

NOTE: The simulator will start adding the initial set of links specified in the json file at time t=0, but it may take a few simulation time units for all the links to be added. The "`handlePeriodicOps`" method will also be first triggered at time t=0, and then periodically every "`heartbeatTime`". But you cannot assume that all the initial links will already be added before the first trigger of "`handlePeriodicOps`" because of the reason mentioned above.

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 lab all the addresses are *flat*, just as in Lab 1. So each routing table will need to store information about every other router and host in the network.

6 Output

At the "`endTime`", the simulator cleans up all the active/queued packets in the network, and generates a last batch of unicast packets between each pair of clients. It also tracks the route taken by each packet in the last batch, and prints it as the final output on the terminal. For your output to match the correct output, your solution must converge to the correct spanning tree and forwarding table entries before the "`endTime`". 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 any destination client (probably because the packet was incorrectly dropped by your solution or the packet is in a loop).

7 Grading

We will test both your DV and LS submissions against the same 10 test cases (network configuration json 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 have provided 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 final code must **not** print any custom / debug statements to the terminal other than what the simulator already prints, as that may break the auto-grader parser.
Violations of this guideline will result in a 10% grade penalty.

8 Debugging

The provided network configuration json files also contain the **correct routes** between each pair of clients, which you can use to debug your implementation. Besides, 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 runtime of the simulator. A received DATA packet in the dump file will be tagged as "DUP PKT" if the packet is a duplicate of some previously received DATA packet, and tagged as "WRONG DST" if the destination address of the DATA packet does not match the address of the recipient client. You can use these dump files to debug your implementation.

9 Submission

You are required to submit **two** files "`DVrouter.py`" and "`LSrouter.py`" on Brightspace.
Do not submit a compressed (e.g., .zip) file.