19/04/23

19/04/23

COSC364 RIP Assignment

Tim Lindbom (tli89) 46808065

& Ben Ireland (bir16) xxxxxxxx

Contributions:

Ben 60%

• Bellman-Ford algorithm: BellmanFordAlgorithm.py is used throughout the project to compute the

shortest path to each router, add new routers to a routing table, and update routing tables if any changes need to be made.

• Response handler: The ResponseHandler.py file is used to Generate packets and packet headers, read, and unpack messages received from neighbours, and send responses.

• RIP demon: RIPDemon.py is the main file of the program. This file calls the helper functions and initiates the main infinite loop of the program.

• Timer: Timer.py contains the helper functions for all the timers used in the program, such as the timer

initialiser, timer checks, and the garbage collection timer.

Tim 40%

• Router: the router.py file is where the router class is defined. After the txt file is parsed the output is passed into a Router class constructor.

• Router Configuration: The routerConfig.py file parses the input file in which the router descriptions are contained and returns a list containing the id, input, output ports, and timers.

• Error Handling: The ErrorHandler.py file is where most of the error check functions for the Id, input and output ports, timers, and packet generation are held.

Testing:

When we first begin testing one big issue we ran into while testing was how we were going to run the

demon if we had all the routers in one text file and only one instance of the program running. The code we had at that time was expected to parse the single txt file and create a router for each line in the file. Once the routers had been initialised and the ports had been bound, we expected the code to call the select method on each router and send a packet to each router in its output list. However, when we ran this, we got stuck in what seemed to be an infinite loop since we had forgotten to account for the fact that select blocks the CPU until it receives something from a port or reaches a timeout. Because of this, no router ended up sending a packet and the program halted. To combat this issue, we separated the routers into separate txt files and ran multiple instances of the program, which resulted in the routers being able to communicate with each other.

Some of the tests we’ve done throughout the production of the program can be found as commented-out code in the routerTesting.py file. The first test in the file is the test of the response handler. This text would manually set up a router and then run the routing algorithm. Once the routing had been completed the test would then generate and print a response packet

Example Configuration:

Each router in the topology is defined in a separate txt file and formatted like

“router\_id 1, inputs 1106 1107 1102, outputs 1201-1-2 1701-8-7 1601-5-6”

Where the router ID is unique to each router, the input port numbers are only used as inputs for one router. The outputs are formatted so that the first number is the outputting router, the next number is the cost to the router, and the third number is the id of the receiving router.

To execute the program in the terminal, the program would be called with the line “python3 RIPDemon.py <filename>” as shown in figure 1. This would have to be called in a new terminal window/tab for each file in the configuration. Once the line has been called the program will first output the router's ID, input, and outputs defined in the txt file. Next, it will print the routing table of its router, which at this point in the program will only contain itself. Throughout the lifetime of the program, the routing tables will update after a predetermined amount of time given in a list in the source code.

A picture containing text

Description automatically generated

Figure

Calendar

Description automatically generated

Figure

Figure 2 shows that output for router 7 once all the routers have been turned on and the routers a have converged and found the shortest paths to each other router based the topology given in Figure 3.

A picture containing map, different

Description automatically generated

Figure