ITCS 6166 Project 3 Report: Distance Vector Protocol Simulation

Spring 2018

Elizabeth (Lizzy) Thomas and Shrirupa Chowdhury

Language Used: Java (Version 8 Update 144)

Platform: Windows 10

IDE: Eclipse (Oxygen.2 Release (4.7.2))

Program Components:

- **Router.java**
    - This class is responsible for setting up and handling main router interactions and their UDP connection. It will read from a file input (given in arguments) and set up the initial distance vector for a router based on the data given. It will set up the router to listen and send over UDP, using the port given in arguments.
    - Each router will receive and send updates, maintaining its vector table accordingly.
    - Each router will print updates as it sends them, displaying changes in route information from the distance vector table for that router.
    - Routers are also responsible for changes in link costs that are implemented by data file changes while the program is running.
- **DistanceVector.java**
    - This class is responsible for setting up and handling the DistanceVector objects used in this program. This includes several methods which allow setting/getting for the distance vector table.
- **DVAlgorithm.java**
    - This class contains the logic for the execution of the Bellman Ford algorithm used in the distance vector protocol. It will be used to compare route information as routers receive updates, and update the distance vector table accordingly.
- **Scheduler.java**
    - This class is used for controlling and timing the periodic updates that routers send, which will occur every 15 seconds.

Instructions for Program Operation: (in command line prompt)**

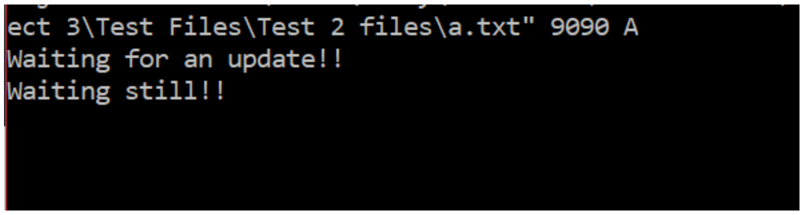
1. Compile all .java files using "javac"

2. To run the connections, create separate command line terminals for each router in the topography. Do this by typing into the command prompt, "java Router <"file path"> <port number> <router name>" where the file path corresponds to the data file for the particular router, the port number corresponds to the UDP port the router will listen on, and the router name is a single alphabetical character that identifies the router (e.g. A).

For example, to start router A on port 9090, one would type “java Router “D:\a.txt” 9090 A”

3. Note that a local file entitled “Ports.txt” will need to be kept in the same directory location as the .java files, it must contain the port numbers to be used for all the routers in the topography (this must be the same port entered in arguments in step 2). A test file of this nature has been included in the project.

4. You will know that the router is running when you see “Waiting for an update!” displayed on the command screen (see screenshot below).

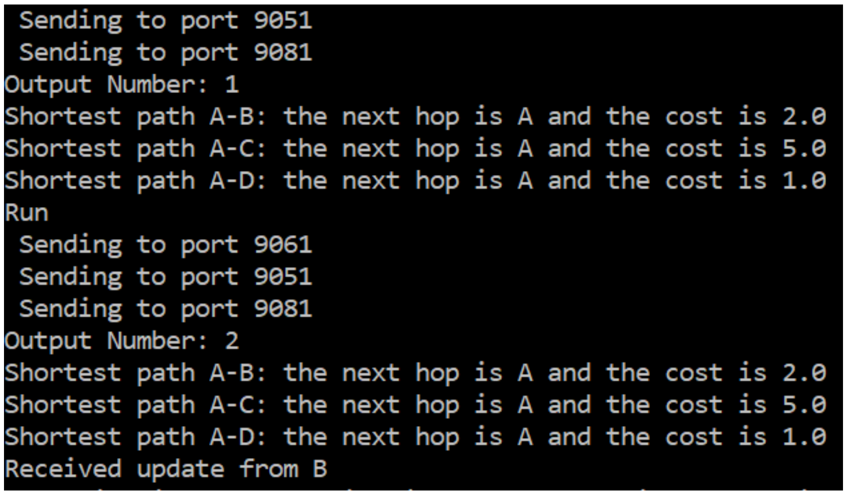


```
ect 3\Test Files\Test 2 files\a.txt" 9090 A
Waiting for an update!!
Waiting still!!
```

5. Open all the routers in their own terminals to watch the network interact.

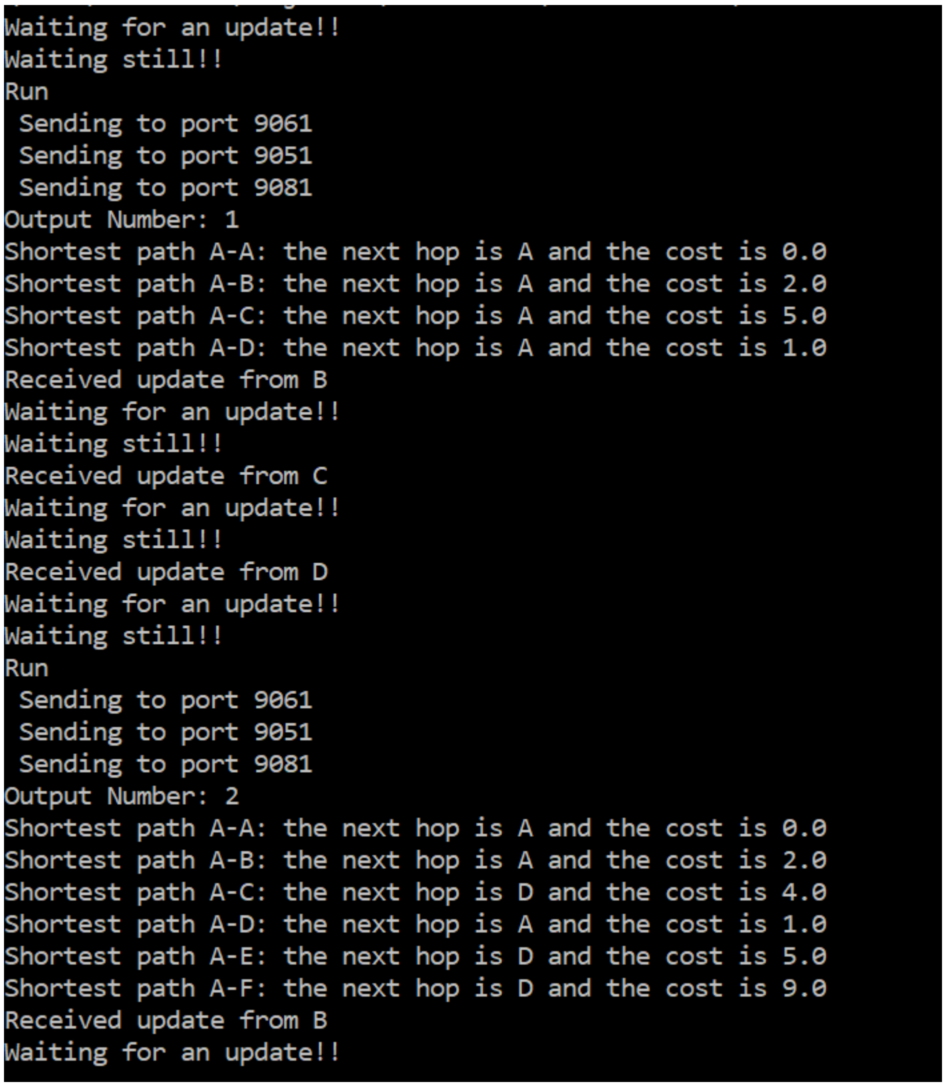
   **Note that two folders have been included for testing purposes, if desired (each with a different network topology)

6. The routers will print output each time they send an update to their neighboring routers. This output will be a readable version of the distance vector that router maintains. They will also print control information that tells the user when and where they are receiving and sending information (see screenshot below).



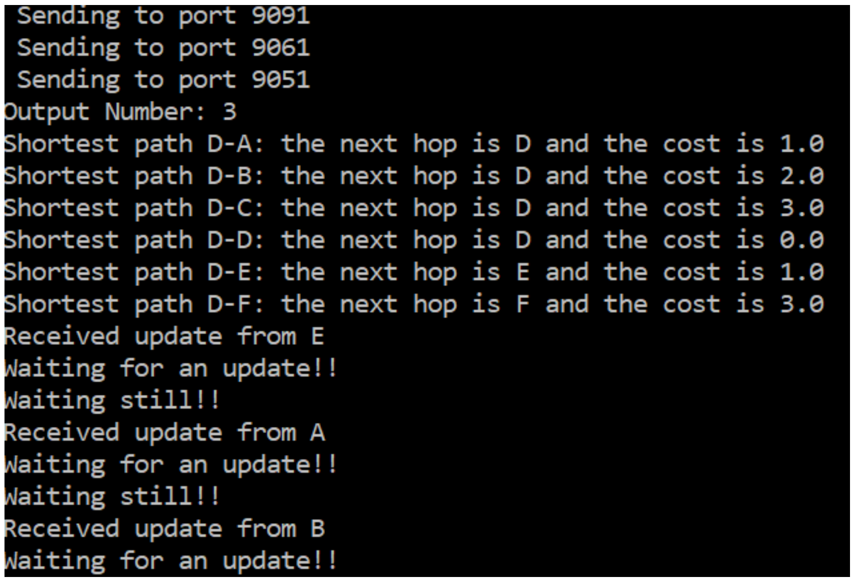
```
 Sending to port 9051
 Sending to port 9081
Output Number: 1
Shortest path A-B: the next hop is A and the cost is 2.0
Shortest path A-C: the next hop is A and the cost is 5.0
Shortest path A-D: the next hop is A and the cost is 1.0
Run
 Sending to port 9061
 Sending to port 9051
 Sending to port 9081
Output Number: 2
Shortest path A-B: the next hop is A and the cost is 2.0
Shortest path A-C: the next hop is A and the cost is 5.0
Shortest path A-D: the next hop is A and the cost is 1.0
Received update from B
```

7. The routers will update their own distance vector tables as they “discover” new routers in the topology. This will be reflected in the printed output (see screenshot below).

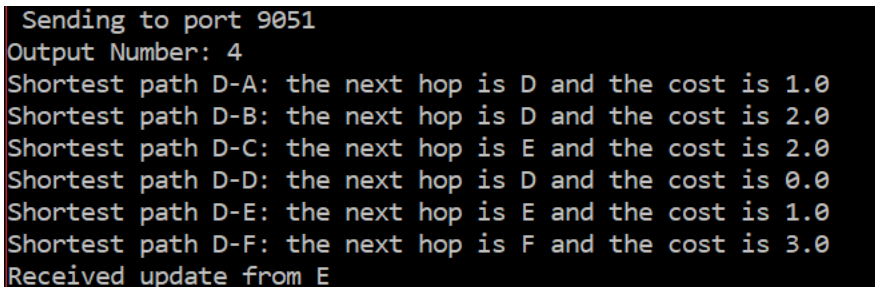


```
Waiting for an update!!
Waiting still!!
Run
 Sending to port 9061
 Sending to port 9051
 Sending to port 9081
Output Number: 1
Shortest path A-A: the next hop is A and the cost is 0.0
Shortest path A-B: the next hop is A and the cost is 2.0
Shortest path A-C: the next hop is A and the cost is 5.0
Shortest path A-D: the next hop is A and the cost is 1.0
Received update from B
Waiting for an update!!
Waiting still!!
Received update from C
Waiting for an update!!
Waiting still!!
Received update from D
Waiting for an update!!
Waiting still!!
Run
 Sending to port 9061
 Sending to port 9051
 Sending to port 9081
Output Number: 2
Shortest path A-A: the next hop is A and the cost is 0.0
Shortest path A-B: the next hop is A and the cost is 2.0
Shortest path A-C: the next hop is D and the cost is 4.0
Shortest path A-D: the next hop is A and the cost is 1.0
Shortest path A-E: the next hop is D and the cost is 5.0
Shortest path A-F: the next hop is D and the cost is 9.0
Received update from B
Waiting for an update!!
```

8. The routers will also update their distance vector tables as they discover shorter paths in the network. This will be reflected in the printed output (see screenshots below, which reflect a change from update 3 to update 4).



```
 Sending to port 9091
 Sending to port 9061
 Sending to port 9051
Output Number: 3
Shortest path D-A: the next hop is D and the cost is 1.0
Shortest path D-B: the next hop is D and the cost is 2.0
Shortest path D-C: the next hop is D and the cost is 3.0
Shortest path D-D: the next hop is D and the cost is 0.0
Shortest path D-E: the next hop is E and the cost is 1.0
Shortest path D-F: the next hop is F and the cost is 3.0
Received update from E
Waiting for an update!!
Waiting still!!
Received update from A
Waiting for an update!!
Waiting still!!
Received update from B
Waiting for an update!!
```



```
 Sending to port 9051
Output Number: 4
Shortest path D-A: the next hop is D and the cost is 1.0
Shortest path D-B: the next hop is D and the cost is 2.0
Shortest path D-C: the next hop is E and the cost is 2.0
Shortest path D-D: the next hop is D and the cost is 0.0
Shortest path D-E: the next hop is E and the cost is 1.0
Shortest path D-F: the next hop is F and the cost is 3.0
Received update from E
```

9. The programs will continue until stopped. Simply enter "<Ctrl> c" to end the program.

Sources Consulted for this Project:

https://docs.oracle.com/javase/7/docs/api/