Report

1.Introduction

In this assignment, I implemented the update of shortest distances to other nodes by using Distance Vector algorithm, which aims to get the shortest distances only by the information received from the neighbors.The language I chose was Java and the transport layer protocol is UDP. There are 5 java files in this assignment and I will explain the main details of those files and the functions in them.

2. Description of files and functions

2.1 DvrPr.java

This is the main file of this application which contains the main method. When start the application, this file will read certain file containing information about neighbors according to the arguments input at terminal. And then initiate the distance table and generate the initial distance vector(Functions initialize() and initialize_pr() will be called in terms of whether “-p” appears at terminal). At the same time, it will create two thread to send DVs and heartbeat packets to its neighbors respectively (sendDVThread.java and sendHeartBeatThread.java). As the main thread, it will keep monitoring the coming packets from its neighbors. When receiving a packet, will invoke the function msgController(), which is used to differentiate the heartbeat packets and DV packets. When receiving a heartbeat packet, the timer used to detect whether a specific neighbor goes down will be refreshed. And if a neighbor’s heartbeat packet has not been received for a certain period, the HBTask will be executed to set this neighbor as a dead neighbor. When receiving a DV packet, the function UpdateDist_Table() will be invoked to update a distance table for a certain neighbor(The neighbors will be differentiated by their ports in DV packets). After updating distance tables, function recompMin_dist() will be called to re-compute the Distance Vector according to those distance tables.

2.2 HBTask.java

When a neighbor’s heartbeat packets have not been received for a certain period, the HBTask will executed to delete that neighbor from the node’s neighbor list and distance table. And set the distance to that neighbor to -1 in the Distance Vector. When other neighbors received the DV packets and find out the distance to that dead neighbor is -1, they will know that neighbor has gone and update the information in their distance tables and Distance Vector.

2.3 sendDVThread.java and sendHeartBeatThread.java

These two files will send DV packets and heartbeat packets to all its neighbors every 5 seconds and every 2 seconds respectively. They are defined as two threads. In sendDVThread.java file, the static function getDV() will read information from the Distance Vector and build a string to be transferred.

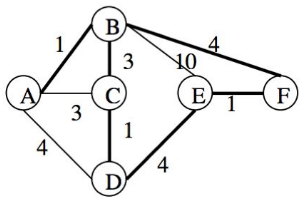
2.4 changeDistTask.java

This task aims to change the distances to some neighbors according to the config files after 60 seconds(this time could be long enough for the original network to convergent) in the “-p” mode. And update the changed distance to the certain distance tables.

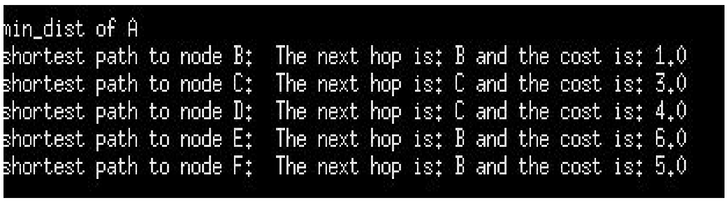
3. Possible improvement

When a neighbor gets back after going down and keeps running as before, the node should detect this situation and add this neighbor to be a new neighbor and update the distance tables. And this function will be used in the next version.

4. Sample output

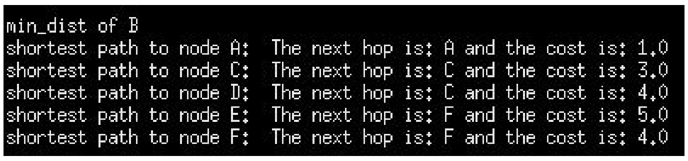


NodeA:



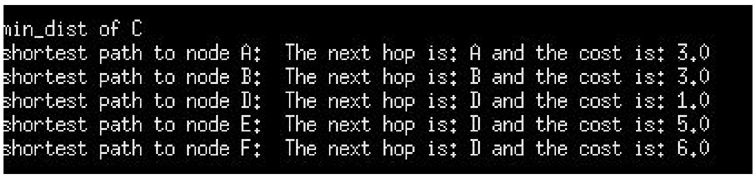
```
min_dist of A
shortest path to node B:  The next hop is: B and the cost is: 1.0
shortest path to node C:  The next hop is: C and the cost is: 3.0
shortest path to node D:  The next hop is: C and the cost is: 4.0
shortest path to node E:  The next hop is: B and the cost is: 6.0
shortest path to node F:  The next hop is: B and the cost is: 5.0
```

NodeB:



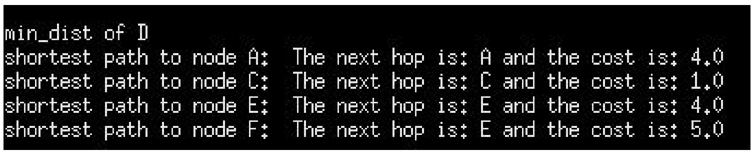
```
min_dist of B
shortest path to node A:  The next hop is: A and the cost is: 1.0
shortest path to node C:  The next hop is: C and the cost is: 3.0
shortest path to node D:  The next hop is: C and the cost is: 4.0
shortest path to node E:  The next hop is: F and the cost is: 5.0
shortest path to node F:  The next hop is: F and the cost is: 4.0
```

NodeC:



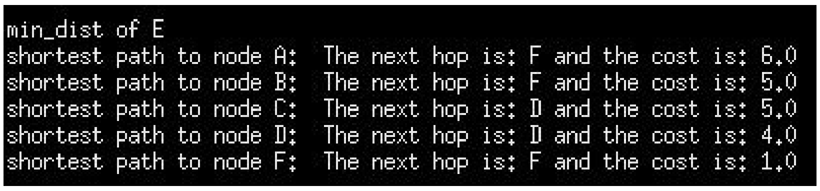
```
min_dist of C
shortest path to node A:  The next hop is: A and the cost is: 3.0
shortest path to node B:  The next hop is: B and the cost is: 3.0
shortest path to node D:  The next hop is: D and the cost is: 1.0
shortest path to node E:  The next hop is: D and the cost is: 5.0
shortest path to node F:  The next hop is: D and the cost is: 6.0
```

NodeD:



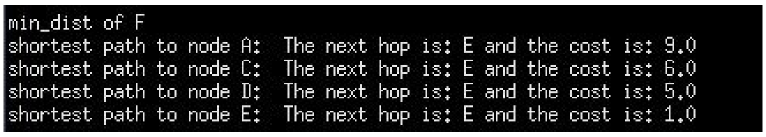
```
min_dist of D
shortest path to node A:  The next hop is: A and the cost is: 4.0
shortest path to node C:  The next hop is: C and the cost is: 1.0
shortest path to node E:  The next hop is: E and the cost is: 4.0
shortest path to node F:  The next hop is: E and the cost is: 5.0
```

NodeE:



```
min_dist of E
shortest path to node A:  The next hop is: F and the cost is: 6.0
shortest path to node B:  The next hop is: F and the cost is: 5.0
shortest path to node C:  The next hop is: D and the cost is: 5.0
shortest path to node D:  The next hop is: D and the cost is: 4.0
shortest path to node F:  The next hop is: F and the cost is: 1.0
```

NodeF:



```
min_dist of F
shortest path to node A:  The next hop is: E and the cost is: 9.0
shortest path to node C:  The next hop is: E and the cost is: 6.0
shortest path to node D:  The next hop is: E and the cost is: 5.0
shortest path to node E:  The next hop is: E and the cost is: 1.0
```