**Protocol document**

**Node start:** At first a node is updating the initial adjacency matrix from the file that contains it’s neighboring information and then it’ll start waiting for 5 seconds so that every router can run & making sure there’s no packet lost. To start a node please use the arguments like this.

./node <routerLabel> <portNum> <totalNumRouters> <discoverFile> [-dynamic]

the parameters are kept same as the project description. Please be sure to use the same port you defined as the file. We’re attaching some files for nodes. In that example six nodes are used. Forwarding table for that will be given later on this document.

File structure is the same as the project description.

**Flooding:** After using sleep for 5 seconds initially (after calculating initial adjacency matrix from the file that contains the cost between any node and the neighboring nodes) every node starts to send the neighboring information to all of its neighbors. Each node that receives an LSP from their neighbor, it decrements the value of hop counter so that the packet has a finite hop, and then it forwards this LSP to all of its links except that, from which it received the LSP. Every time a node receive the LSP, it updates the cost information of the network that is stored locally. When a node gets the LSPs generated by all the nodes of the network, it knows the all cost of existing routes in that network.

**Adjacency matrix:** At first adjacency matrix are created with a large value (5000 in here) to assume that as infinity. There’s a constant defined in node.h file (inf). After reading the file with every packet adjacency matrix is being updated with function adjMatrixChange(). Every function used in the project are declared in node.h and the defined in functions.c. Please see the documentation in there for return type and argument used for every function.

Steps to calculate final adjacency matrix:

1. If diagonal put 0, otherwise put inf in every element of the matrix.
2. Read the neighbor information from file and update the adjacency matrix depending on source router and neighbor router. For simplicity, in this project single capital letters are being used for routerLabel. Calculating the adjacency position using this formula in here. labelRouter[0] % 65.
3. The operating router updates individual neighbour information(as read from the file) into a LSP which has the following structure.

LSP(<routerLabel><sequenceNumber><nodeIP><nodePort><Cost>).

All LSP’s created from the file info is encapsulated into a single structure named “allLSP” which has the following structure.

allLSP(<numOfNeighbours><hopCount><sourceRouterLabel><LSP<array>>)

A copy of allLSP is sent to all the neighbours of the source Router.

4. A router on receiving an allLSP packet, extracts information from the individual LSPs and updates its own adjacency matrix. It decrements the hopCount by 1 and passes on the packet to the next router in the path.

5. A precondition for the forwarding is that, a router cannot return the packet to the router from which it receives in the first place.

6. An allLSP packet is disposed on two conditions

1. hopCount reaches 0.
2. hopCount > 0 and there are no neighbouring nodes to forward the packet.

**DJikstra Shortest Path:**

Djikstra Shortest Path algorithm is used to determine the shortest path and hence come up with the outgoing interface, for creating the forwarding table.Steps to calculate the djikstra algorithm.

1. Alongside the adjacency matrix, a cost matrix(2D) is also created to store the minimum cost to travel from the source router to all the other routers.

The cost matrix consists of the cost between two nodes as well as the previous node visited(which in turn helps us to retrace the path followed.)

We also maintain a visited matrix, which keeps in track, the nodes visited while traversing.

2. In the first row, the source to source cost matrix is marked as 0, while the rest of the elements in the row is filled with infinity.

3. At each iteration, we fill the rows with the value which is minimum of(previousCostValue, markedVal+offsetDistance from the markedNode). We select the next visited node as the node which has the minimum cost. We also update the previous node at each iteration.

4. We use a while loop to retrace back to the first path/first outgoing interface for the specific router.

5. The forwarding table is created using the above logic.

**Dynamic Routing:**

A random number is generated in (1,10) range, and the value obtained is added to the original cost in the adjacency matrix. A fresh LSP is created with an incremented sequence Number. This LSP is packed into the allLSP packet and sent to the neighbours.

A neighbour on receiving a packet with different sequence number from the previous, will update the new value on to the adjacency matrix, else it just drops the packet.

Routing tables are given below with the network posted in elearning.

The graph given in e-learning has been implemented in the project. The router information is stated below:

router A cs-ssh3.argo.uwf.edu, port 60085

router B cs-ssh3.argo.uwf.edu, port 60090

router C cs-ssh2.argo.uwf.edu, port 60080

router D cs-ssh2.argo.uwf.edu, port 60082

router E cs-ssh3.argo.uwf.edu, port 60080

we used only two servers because of some connection issue with server cs-ssh.argo.uwf.edu. We told Dr. Reichherzer about this.

Router A:

Destination Next Hop

B B

D D

C B

E B

Router B:

Destination Next Hop

A A

C C

D C

E C

Router C:

Destination Next Hop

B B

D D

E E

A B

Router D:

Destination Next Hop

A A

C C

E C

B C

Router E:

Destination Next Hop

C C

D C

B C

A C

**Note:**

If you wish to see the display of the costMatrix/adjacency Matrix/LSP info, add [-Debug] at the end of the command line argument.

A sample command line argument would be:

*./node A 60015 5 A.txt -Debug*

*./node A 60015 5 A.txt -dynamic -Debug*

The test files that contains neighboring information have to be stored in the folder named ‘routers’ before running. The files submitted for testing contains the node name with extension .txt

For node A the filename is : A.txt

For node B the filename is: B.txt

For node C the filename is: C.txt

For node D the filename is: D.txt

For node E the filename is: E.txt