Yang Meng
01679623
11/29/2016

Program Design

I have implemented the Distance Vector Routing Algorithm for the network. All the programs I've written for this assignment are coded in C language.

The program design consists of 5 source files. They are: node0.c, node1.c, node2.c, node3.c, and prog3.c. The file prog3.c is the main routine which emulates the layer 2 and below network environment as follows:

- Emulates the transmission and delivery (with no loss and no corruption) between two physically connected nodes
- Calls the initializations routines rtinit0(), rtinit1(), rtinit2(), and rtinit3() only once during the initialization of the emulation

The other 4 source files, namely node0.c, node1.c, node2.c, and node3.c, implement the node configuration for the given network topology. The conceptual view of the relationship of the procedures inside node 0 that prog3.c uses is shown in Figure 2, and it's similar for node 1, node 2, and node 3.

The following routines are defined in node0.c, and similar routines are defined in node1.c, node2.c, and node3.c:

- rtinit0() is called only once at the beginning by routine init() in prog3.c for initialization of all the data structures used by the algorithm and it sends a packet to the other nodes.
- rtupdate0() is called by routine main() in prog3.c for whenever the current node receives a packet from some other node. This routine is used to update the distance table of that particular node which contains the distances to the other nodes directly connected to itself and also to send packets to its neighbors if the shortest[] variable which contains the shortest paths to the other nodes has changed in the process.
- compute_shortest_path0() is called within node0.c by rtupdate0() and linkhandler0() routines which updates shortest[] variable if a shortest paths are found. Basically it finds the shortest distance from the source node to all the other directly connected nodes and stores these values in the shortest[] variable.
- update() is called within node0.c to build a packet or update any information of packet and send it to all the directly connected nodes using the tolayer2() routine defined in the prog3.c, so in node0 packet is send to node 1, 2 and 3.

- printdt0 () is called within node0.c to pretty print the distance vector table.
- printf_sendinfo0 () is called within node0.c to print send information, which includes send time, source ID, destination ID and shortest path.

The files node0.c and node1.c have additional routines called linkhandler0() and linkhandler1(), respectively. I have implemented these routines as an extension to the program. The idea of these routines are to demonstrate the dynamic capability of the design by changing the link cost value at a certain time, and reverting it back to what it was before.

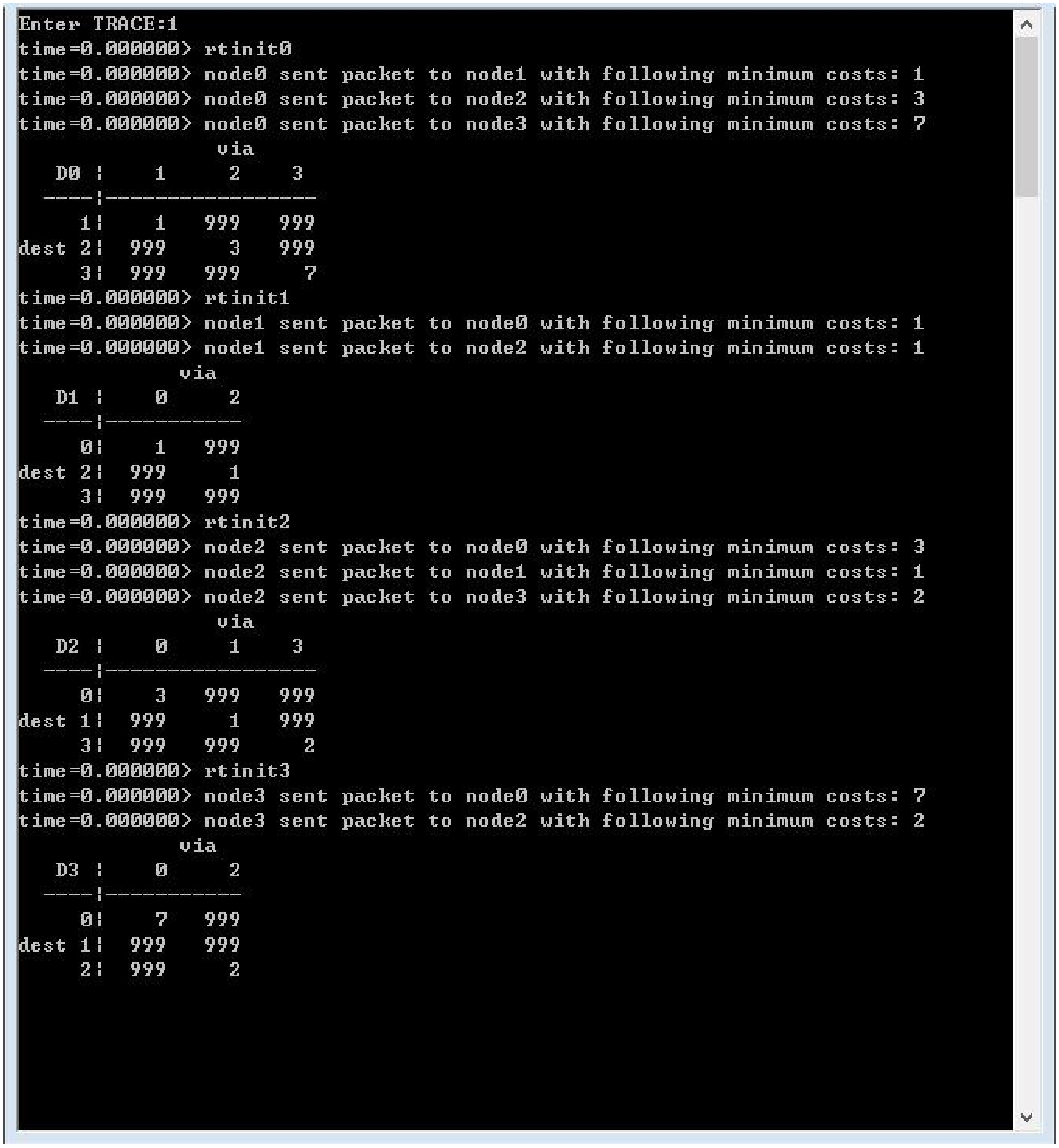
Test Cases

Initially every node has information about the cost of directly connected nodes to itself. This is setup in routines rtinit0(), rtinit1(), rtinit2(), and rtinit3(). Which are called at the beginning of the emulator by the main() routine in prog3.c. In the rtinit0() routine for each node, it initializes the costs of the node's directly connected nodes with the specified costs and the rest of the node costs are set to infinity.

Each node contains and maintains a data structure called distance table. It contains a 2-dimentioal array which has a row for each destination in the network and a column for each of its directly connected neighbors.

Figure 1 shows that initial result at time 0.0000 via rtinit0(), rtinit1(), rtinit2() and trinit3(). After initialization, every node send packet to direct node. Node 0 sends to node 1, node 2 and node 3; node 1 sends to node 0 and node 2; node 2 sends to node 0, node 1 and node 3; node 3 sends to node 1 and node 2.

When a packet is received by a given node, then that node's rtupdateX() routine is invoked and distance table is updated. For example, when node 0 received a packet from node 1 at time = 1.0, the packet contains the least cost from node 1 to node 0 and node 2 as {1, 0, 1, 999}. Then the rtupdate0() calculates values for distance table to all the nodes via node 1.



```
Enter TRACE:1
time=0.000000> rtinit0
time=0.000000> node0 sent packet to node1 with following minimum costs: 1
time=0.000000> node0 sent packet to node2 with following minimum costs: 3
time=0.000000> node0 sent packet to node3 with following minimum costs: 7
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1   999   999
dest 2|  999     3   999
     3|  999   999     7
time=0.000000> rtinit1
time=0.000000> node1 sent packet to node0 with following minimum costs: 1
time=0.000000> node1 sent packet to node2 with following minimum costs: 1
             via
   D1 |    0     2
  ----|-----------
     0|    1   999
dest 2|  999     1
     3|  999   999
time=0.000000> rtinit2
time=0.000000> node2 sent packet to node0 with following minimum costs: 3
time=0.000000> node2 sent packet to node1 with following minimum costs: 1
time=0.000000> node2 sent packet to node3 with following minimum costs: 2
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3   999   999
dest 1|  999     1   999
     3|  999   999     2
time=0.000000> rtinit3
time=0.000000> node3 sent packet to node0 with following minimum costs: 7
time=0.000000> node3 sent packet to node2 with following minimum costs: 2
             via
   D3 |    0     2
  ----|-----------
     0|    7   999
dest 1|  999   999
     2|  999     2
```

Figure 1

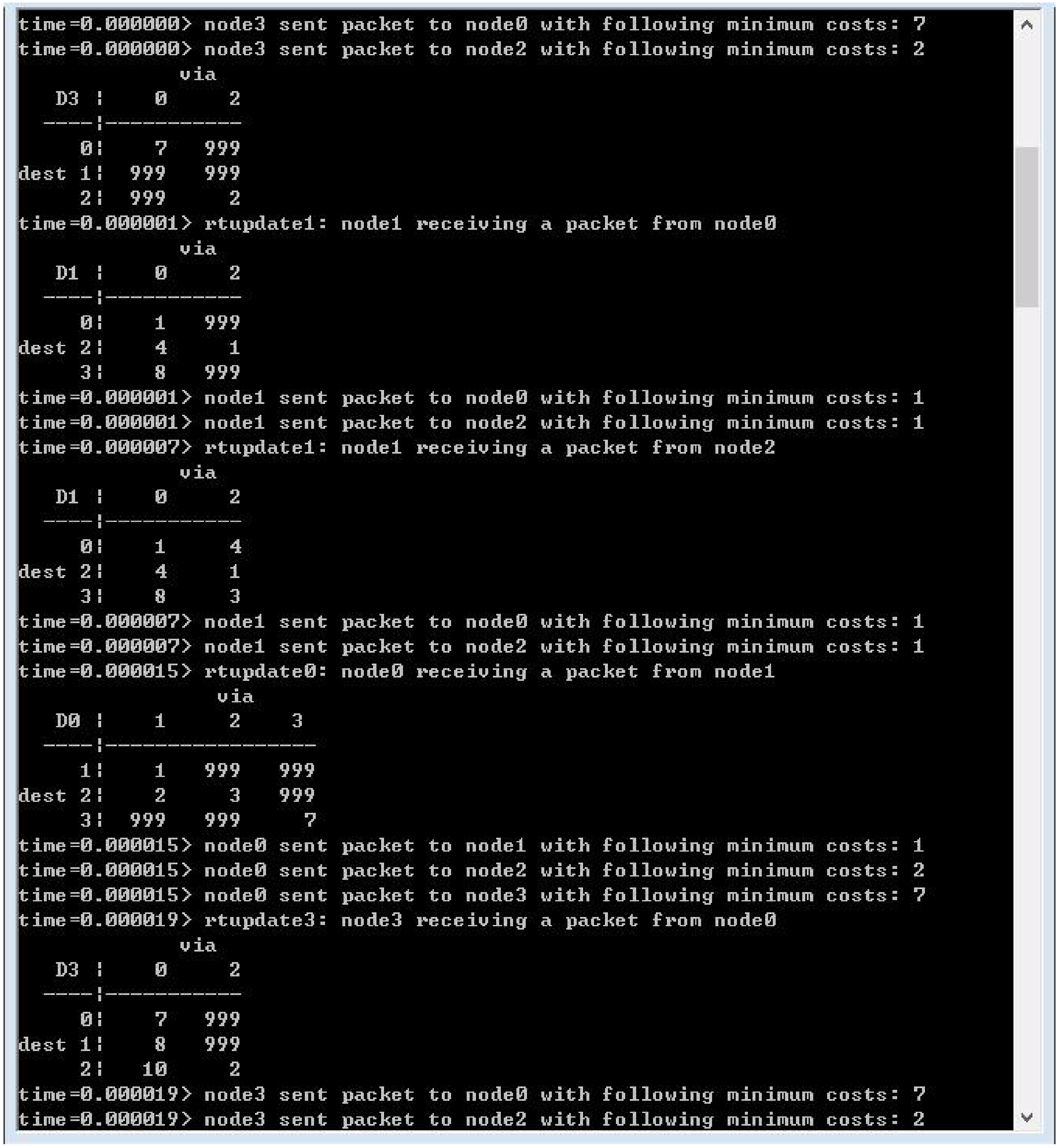
link[] for the link cost between node 0 and rest of the nodes, and shortest[] for the shortest path between the specific node and rest of the nodes, i.e. the least cost to all the nodes directly attached to itself. The difference between link and shortest is that link never update, and shortest update if we get new shortest path to other node. This information can be obtained from distance table, but it is better to have a separate array for more efficient way of implementing the algorithm.

The link cost are stored in the array link[]. During the initialization, the graph topology and its link costs are coded as follows: rtinit0() would set this array for node 0 to the values { 0, 1, 3, 7 }; rtinit1() would set this array for node 1 to the values { 1, 0, 1, infinity }; rtinit2() would set this array for node 2 to the values { 3, 1, 0, 2 }; and rtinit3() would set this array for node 3 to the values { 7, infinity, 2, 0 }.

When initialization is completed, then every node sends a packet, containing shortest[] values, to its directly connected neighbor. Hence, node 0 will send a packet to node 1, node 2 and node 3. This packet contains the shortest path information from the node itself to all the other directly connected nodes. Similar initialization is done for the other nodes in rtinit1(), rtinit2(), and rtinit3(). If this newly calculated distance changes any of the shortest[] values, then the shortest[] is updated and packets are sent to all its directly connected neighbors. For example, in this case the shortest[] for node 0 changes from values {0, 1, 3, 7} to {0, 1, 2, 7}. Then node0 sends a packet to all it neighbors.

The distance table of the other nodes are updated every time a packet is received from any of the other nodes. The iteration of updating each of the node's distance table continues until no more information is exchanged between neighbors. The program will run until there are no more routing packets in-transit in the network, at which point emulator will terminate. Hence, at the end of the iteration, distance table will contain the least cost paths to all the other directly connected nodes. There is a pretty print routine that has been implemented to print distance table in a nice format for easy reading of its contents for the node0, node1, node2, and node3.

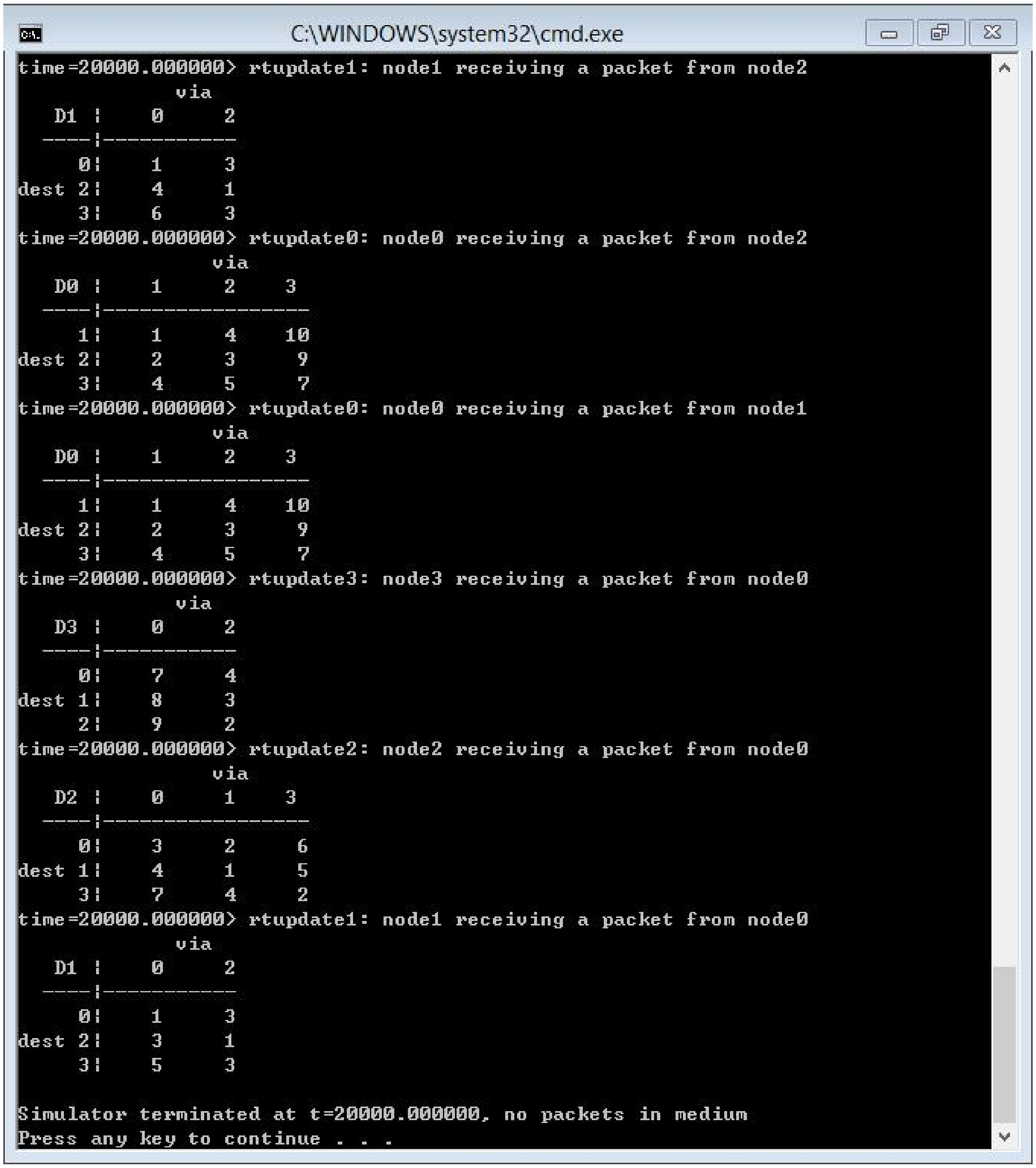
Figure 2 show that node send packet to other node at different time, and receive packet from other node. For example, node 1 receives packet from node 0 at time 0.00001. After distance tabled is updated, node 1 send packet to node 0 and node 2 at time 0.00001.



```
time=0.000000> node3 sent packet to node0 with following minimum costs: 7
time=0.000000> node3 sent packet to node2 with following minimum costs: 2
             via
   D3 |    0     2
  ----|-----------
     0|    7   999
dest 1|  999   999
     2|  999     2
time=0.000001> rtupdate1: node1 receiving a packet from node0
             via
   D1 |    0     2
  ----|-----------
     0|    1   999
dest 2|    4     1
     3|    8   999
time=0.000001> node1 sent packet to node0 with following minimum costs: 1
time=0.000001> node1 sent packet to node2 with following minimum costs: 1
time=0.000007> rtupdate1: node1 receiving a packet from node2
             via
   D1 |    0     2
  ----|-----------
     0|    1     4
dest 2|    4     1
     3|    8     3
time=0.000007> node1 sent packet to node0 with following minimum costs: 1
time=0.000007> node1 sent packet to node2 with following minimum costs: 1
time=0.000015> rtupdate0: node0 receiving a packet from node1
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1   999  999
dest 2|    2     3  999
     3|  999   999    7
time=0.000015> node0 sent packet to node1 with following minimum costs: 1
time=0.000015> node0 sent packet to node2 with following minimum costs: 2
time=0.000015> node0 sent packet to node3 with following minimum costs: 7
time=0.000019> rtupdate3: node3 receiving a packet from node0
             via
   D3 |    0     2
  ----|-----------
     0|    7   999
dest 1|    8   999
     2|   10     2
time=0.000019> node3 sent packet to node0 with following minimum costs: 7
time=0.000019> node3 sent packet to node2 with following minimum costs: 2
```

Figure 2

The Figure 3 shows the final results of the C programming version of the distance vector routing algorithm which runs to completion.



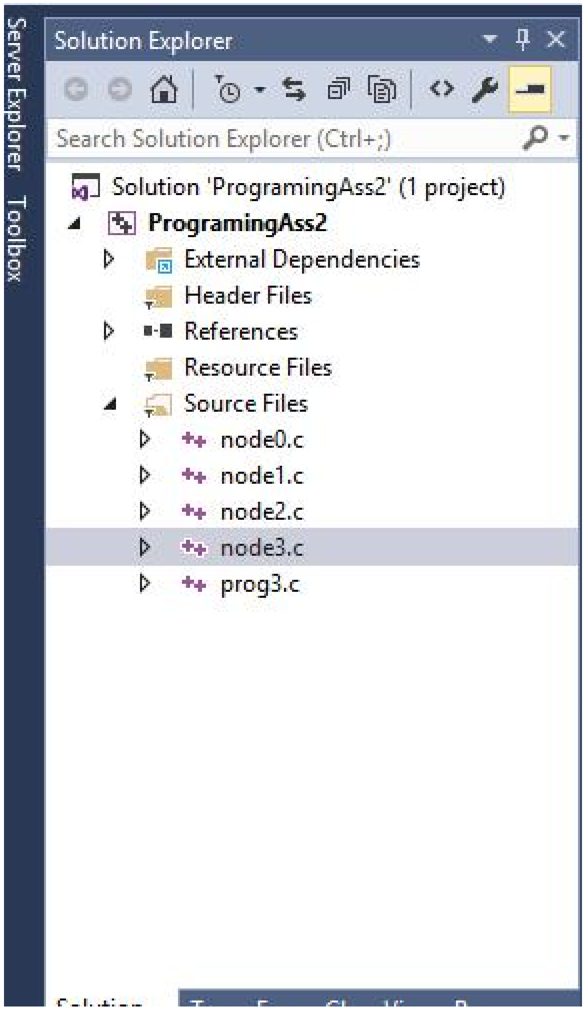
```
time=20000.000000> rtupdate1: node1 receiving a packet from node2
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    4     1
     3|    6     3
time=20000.000000> rtupdate0: node0 receiving a packet from node2
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    4     5     7
time=20000.000000> rtupdate0: node0 receiving a packet from node1
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    4     5     7
time=20000.000000> rtupdate3: node3 receiving a packet from node0
             via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|    8     3
     2|    9     2
time=20000.000000> rtupdate2: node2 receiving a packet from node0
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2
time=20000.000000> rtupdate1: node1 receiving a packet from node0
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    5     3

Simulator terminated at t=20000.000000, no packets in medium
Press any key to continue . . .
```

Figure 3

File list

Prog3.c is the main function of the project.

Node0.c, Node1.c, Node2.c and Node3.c are functions for 4 nodes.


Server Explorer
Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ProgramingAss2' (1 project)
ProgramingAss2
External Dependencies
Header Files
References
Resource Files
Source Files
node0.c
node1.c
node2.c
node3.c
prog3.c