

Electrical, Computer and System Engineering
ECSE4670: Computer Communication Networking (CCN)

Lab2: Implementing a Routing Algorithm

Due date for on-campus and live students: Nov. 6, 2001

Due date for tape-delayed students: Nov.14, 2001

1 Overview

In this lab, you will be writing a "distributed" set of procedures that implement a **distributed asynchronous distance vector routing** for the network shown in Figure Lab.2-1.

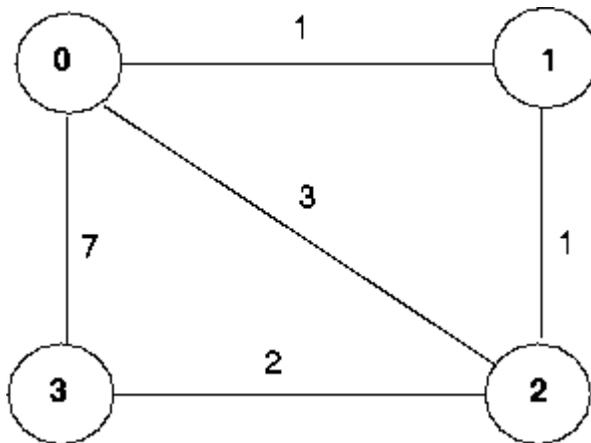


Figure Lab.2-1: Network topology and link costs for DV routing lab

The Basic Assignment

The routines you will write: For the basic part of the assignment, you are to write the following routines which will "execute" asynchronously within the emulated environment.

For node 0, you will write the routines:

- rtinit0() This routine will be called once at the beginning of the emulation. rtinit0() has no arguments. It should initialize the distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In Figure 1, all links are bi-directional and the costs in both directions are identical.

After initializing the distance table, and any other data structures needed by your node 0 routines, it should then send its directly-connected neighbors (in this case, 1, 2 and 3) the cost of its minimum cost paths to all other network nodes. This minimum cost information is sent to neighboring nodes in a *routing packet* by calling the routine `tolayer2()`, as described below. The format of the routing packet is also described below.

- `rtupdate0(struct rtpkt *rcvdpkt)`. This routine will be called when node 0 receives a routing packet that was sent to it by one of its directly connected neighbors. The parameter `*rcvdpkt` is a pointer to the packet that was received.

`rtupdate0()` is the "heart" of the distance vector algorithm. The values it receives in a routing packet from some other node *i* contain *i*'s current shortest path costs to all other network nodes. `rtupdate0()` uses these received values to update its own distance table (as specified by the distance vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change in minimum cost by sending them a routing packet. Recall that in the distance vector algorithm, only directly connected nodes will exchange routing packets. Thus nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will not communicate with each other.

As we saw in class, the distance table inside each node is the principal data structure used by the distance vector algorithm. You will find it convenient to declare the distance table as a 4-by-4 array of `int`'s, where entry `[i,j]` in the distance table in node 0 is node 0's currently computed cost to node *i* via direct neighbor *j*. If 0 is not directly connected to *j*, you can ignore this entry. We will use the convention that the integer value 999 is "infinity."

Figure Lab.2-2 provides a conceptual view of the relationship of the procedures inside node 0.

Similar routines are defined for nodes 1, 2 and 3. Thus, you will write 8 procedures in all: `rtinit0()`, `rtinit1()`, `rtinit2()`, `rtinit3()`, `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()`

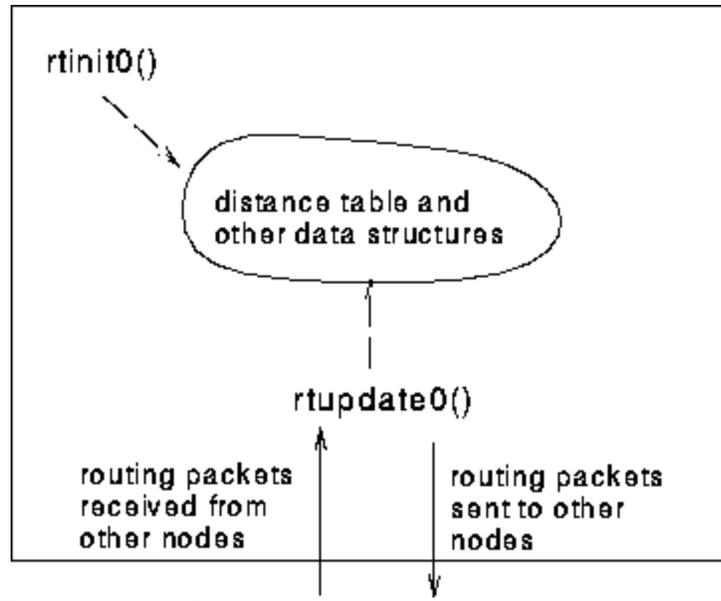


Figure Lab.2-2: Relationship between procedures inside node 0

Software Interfaces

The procedures described above are the ones that you will write. Following routines are available to be called by your routines:

`tolayer2(struct rtpkt pkt2send)`

where `rtpkt` is the following structure, which is already declared for you. The procedure `tolayer2()` is defined in the file [prog3.c](#)

```

extern struct rtpkt {
    int sourceid; /* id of node sending this pkt, 0, 1, 2, or 3 */
    int destid;   /* id of router to which pkt being sent
                  (must be an immediate neighbor) */
    int mincost[4]; /* min cost to node 0 ... 3 */
};
  
```

Note that `tolayer2()` is passed a structure, not a pointer to a structure.

`printdt0()` will pretty print the distance table for node 0. It is passed a pointer to a structure of type `distance_table`. `printdt0()` and the structure declaration for the node 0 distance table are declared in the file `node0.c`. Similar pretty-print routines are defined for you in the files `node1.c`, `node2.c` `node3.c`.

The simulated network environment

Your procedures `rtinit0()`, `rtinit1()`, `rtinit2()`, `rtinit3()` and `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()` send routing packets (whose format is described above) into the medium. The medium will deliver packets in-order, and without loss to the specified destination. Only directly-connected nodes can communicate. The delay between the sender and receiver is variable (and unknown).

When you compile your procedures and the simulator program together and run the resulting program, you will be asked to specify only one value regarding the simulated network environment:

- **Tracing.** Setting a tracing value of 1 or 2 will print out useful information about what is going on inside the emulation (e.g., what's happening to packets and timers). A tracing value of 0 will turn this off. A tracing value greater than 2 will display all sorts of odd messages that are for the emulator-debugging purposes.

A tracing value of 2 may be helpful to you in debugging your code. You should keep in mind that *real* implementers do not have underlying networks that provide such nice information about what is going to happen to their packets!

The procedures `rtinit0()`, `rtinit1()`, `rtinit2()`, `rtinit3()` and `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()` which together will implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in Figure 1.

You should put your procedures for nodes 0 through 3 in files called `node0.c`, ..., `node3.c`. You are **NOT** allowed to declare any global variables that are visible outside of a given C file (e.g., any global variables you define in `node0.c` may only be accessed inside `node0.c`). This is to force you to abide by the coding conventions that you would have to adopt if you were really running the procedures in four distinct nodes. To compile your routines: `cc prog3.c node0.c node1.c node2.c node3.c`. Prototype versions of these files are here:

<http://gaia.cs.umass.edu/kurose/network/node0.c>,

<http://www-net.cs.umass.edu/kurose/network/node1.c>,

<http://www-net.cs.umass.edu/kurose/network/node2.c>,

<http://www-net.cs.umass.edu/kurose/network/node3.c>.

You can pick up a copy of the file `prog3.c` at :

<http://gaia.cs.umass.edu/kurose/network/prog3.c>.

This assignment can be completed on any machine supporting C. It makes no use of UNIX features.

2 The Advanced Assignment

You are to write two procedures, `rtlinkhandler0(int linkid, int newcost)` and `rtlinkhandler1(int linkid, int newcost)`, which will be called if (and when) the cost of the link between 0 and 1 changes. These routines should be defined in the files

`node0.c` and `node1.c`, respectively. The routines will be passed the name (id) of the neighboring node on the other side of the link whose cost has changed, and the new cost of the link. Note that when a link cost changes, these routines will have to update the distance table and may (or may not) have to send updated routing packets to neighboring nodes.

In order to complete the advanced part of the assignment, you will need to change the value of the constant `LINKCHANGES` (line 3 in `prog3.c`) to 1. FYI, the cost of the link will change from 1 to 20 at time 10000 and then change back to 1 at time 20000. Your routines will be invoked at these times.

We would again **STRONGLY** recommend that you first implement the basic assignment and then extend your code to implement the advanced assignment.

3 Submissions

- 1) All submission must be done on line via WebCT drop-box
- 2) The lab will count for 10% of your final grade. 5% is for the “Basic Assignment” and 5% is for the “Advanced Assignment”. Partial credit may be awarded only if the code compiles correctly and evidence of progress in the lab is seen in the code.
- 3) You should submit:

- **Source code:** The C file containing the routines you have written, clear commented. It must compile to qualify any credit.
- **A text file** containing your output from the algorithms:
 - your procedures should print out a message whenever your `rtinit0()`, `rtinit1()`, `rtinit2()`, `rtinit3()` or `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()` **procedures are called**, giving the **current global time** (available via global variable `clocktime`).

Eg: Node0: `rtinit0()` is called at time 0.000000

Node1: `rtupdate1()` is called at time 93.569748

- For `rtupdate0()`, `rtupdate1()`, `rtupdate2()`, `rtupdate3()` you should **print the identity of the sender of the routing packet** that is being passed to your routine, whether or not the distance table is updated, the contents of the distance table (you can use the print routines provided in `Node.c`), and a description of any messages sent to neighboring nodes as a result of any distance table updates.

Eg: Node0: Receiving routing message from Node1

Node0: updating my routing table as follows:

D0	1	2	3
1	1	4	10
2	*	*	*
3	*	*	*

Note: "*" stands from the distance between Node 0 and Node x via Node Y.

- o For the "Advanced Assignment" you should print out "link cost change" messages and each node's updated routing table.

The sample output should be an output listing with a TRACE value of 2. Mark the final distance table produced in each node. Your program will run until there is no more routing packets in-transit in the network, at which point our emulator will terminate.

Note: the example output given above is only provided as a reference. You can design your output to be similar to the above format.

Please direct all technical questions regarding the lab to the bulletin board. The TAs and instructors will be able to help you in conceptual matters regarding the lab, but will not be available to debug your code

4 Q&A

Check out following webpage for it:

http://gaia.cs.umass.edu/kurose/network/programming_assignment_QA.htm