CS422 - Spring 2020
Programming Assignment 3
Distance Vector Routing Algorithm

Due Date: April 10th, 2020 11:59 pm Total points: 100

I. OBJECTIVES

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 Fig. 1.

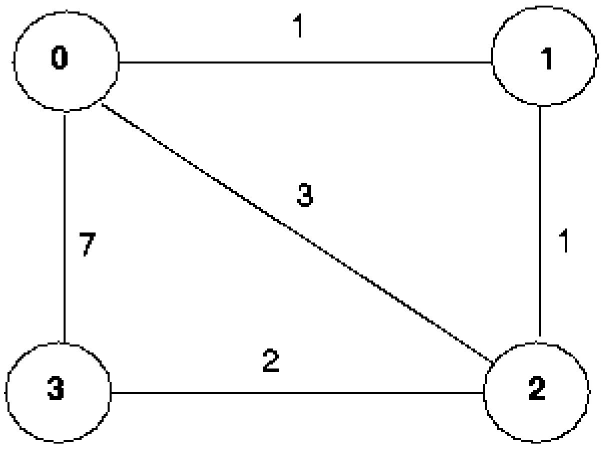

Fig. 1. Network topology and link costs

II. IMPLEMENTATION

A. Programming environment

You must work individually on this assignment. You will write C code that compiles and operates correctly on the XINU machines (xinu01.cs.purdue.edu, xinu02.cs.purdue.edu, etc.).

B. Routines you must write

For the basic part of the assignment, you are to write the following routines which will “execute” asynchronously within the emulated environment that has been provided for this assignment.

For node 0, you will write the following 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 Fig. 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(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* contains *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 because 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.

You will find it convenient to declare the distance table as a 4-by-4 array of integers, where entry [i,j] in the distance table in node 0 is node 0's currently computed cost to **node j** via **direct neighbor i**. If 0 is not directly connected to i, you can ignore this entry. We will use the convention that the integer value 999 is "infinity".

Fig. 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().**

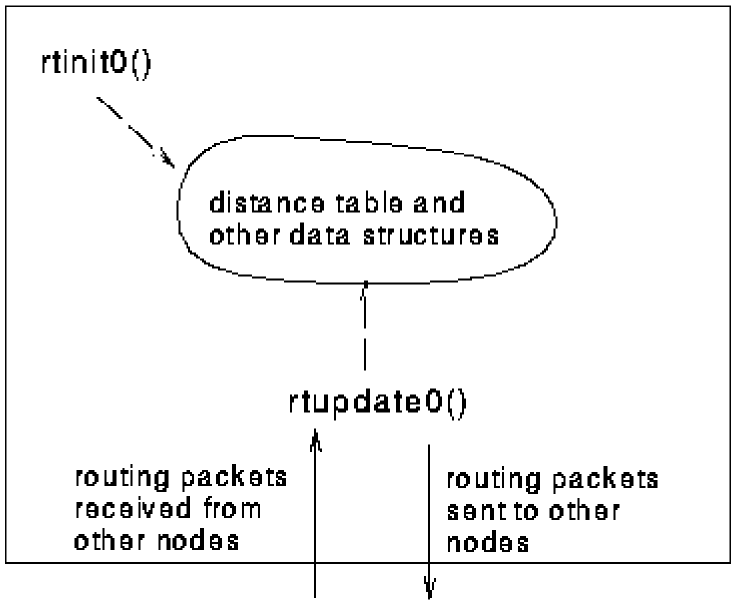
rtinit0()
distance table and other data structures
rtupdate0()
routing packets received from other nodes
routing packets sent to other nodes

Fig. 2. Relationship between procedures inside node 0

C. Software Interfaces

The procedures described above are the ones that you will write. You are provided with the following routines that can be called by your routines:

- **tolayer2(rtpkt pkt2send) where rtpkt is the following structure,** which is already declared for you.

  ```
  typedef struct rtpkt {
          int sourceid; /* id of router 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 */
  } rtpkt;
  ```

  The procedure tolayer2() is defined in the file prog3.c. Note that tolayer2() is passed a structure, not a pointer to a structure.
- **printdt0(distance_table *dtptr)** will pretty print the distance table for node 0. It is passed a pointer to a structure of type distance_table. printdt0() is declared in node0.c and the structure declaration for the node 0 distance table is in prog3.h. Similar pretty-print routines are defined for you in the files node1.c, node2.c and node3.c.

D. 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 a sender and receiver is variable (and unknown).

When you compile your procedures and all other procedures 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 emulator debugging purposes. A tracing value of 2 may be helpful to you in debugging your code.

E. Implementation

You are to write 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 Fig. 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, use:

```
gcc prog3.c node0.c node1.c node2.c node3.c -o PA3
```

For your sample output, your procedures should print out a message whenever your rtinit0(), rtinit1(), rtinit2(), rtinit3() or rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3() procedures are called, giving the time (available via the global variable clocktime). 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 pretty-print routines already provided to you), and a description (content) of any messages sent to neighboring nodes as a result of any distance table updates. **Make sure your sample output meets all these requirements.**

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

III. SUBMISSION AND GRADING

A. What to Submit

You will be submitting your assignment on blackboard. Your submission directory should contain

- All source files.
- PA3.pdf with your sample output
- Makefile

You should submit all your files in a zip folder with the name YourPurdueID_PA3.zip.

Note:

- Please document any reasonable assumptions you make or information in this file, e.g., if any parts of your assignment are incomplete.
- You may change prog3.h and prog3.c while implementing only for debugging purposes but should be aware that these two files will be replaced (including the parameters in them) after submission for grading.
- You may submit other source files, e.g. .c or .h files shared by all nodes.
- **Make sure your sample output meets all the requirements** – your lab will be graded based on it.

Questions about the assignment should be posted on ***Piazza****. In most of the cases, you might find your question already answered by some existing posts. There will be a post summarizing frequently asked questions. Please make sure you check it. If you need help understanding the concepts or the requirement of the project, you can contact Siyuan Cao at cao208@purdue.edu.*