CptS/EE 455
Spring 2002
Project 3
Assigned: Mon Mar 25, 2002
Due: in class Fri Apr 12, 2002
10% per day penalty for late submissions

Summary

In this project you will create a network of simulated routers implementing a distance-vector routing algorithm.

The purposes of this project: to become familiar with the dynamic behavior of the distance-vector algorithm; to learn about programming with datagram sockets; to learn about socket demultiplexing using the select() system call; to learn about managing time using the select() system call. (Alternatively, if programming in Java or with threads in C, to practice using multi-threaded socket and timeout programming requiring synchronization.)

Logistics

You may write this program in C, C++ or Java. If you want to use a different language, please ask me first. You may work with a partner or alone on this project. If you work with a partner, please understand that both partners are responsible for completely understanding both the problem and the solution. The interview and demonstration that you will have with the TAs will be designed to uncover your personal understanding of what you have developed.

What to turn in

In class on the due date, turn in printed listings (in this order please) of your:

1. Cover sheet with your name, your partner's name, project #, submission date, and submission directory name.
2. README file - As in Project 1, the README file should contain complete instructions for creating your router from source and for running it on the supplied test cases.
3. Makefile
4. COMMENTS file – see below. Even if you work with a partner on the programming, the COMMENTS file is to be your *personal, individual* work
5. Test case output
6. Program code
7. All of these files are also to be stored in the /net/cs455/students/YOURNAME/project3/ directory prior to class on the due date. Partners should each store a copy of the code in her or his own directory. Make sure that each file clearly indicates who your partner is.

During the week following the due date you will each need to schedule an individual appointment with one of the TAs to demonstrate your router and to describe its operation.

Specifications

You are to create a program that implements a distance-vector routing algorithm like the one suggested in problem 4 of Homework 3. That is, each router keeps only the cost and next hop for each destination and relies on periodic updates from neighbor routers to allow calculation of improved paths. For this project we will define infinity to be 64.

Each router instance will read the configuration of its local links from files at the beginning of the run. One file contains a list of all the routers in the system along with the hostname on which they run and the beginning port number that they use. The predefined test cases use localhost for the hostname and a fixed set of port numbers. You are free to edit this file to allow different routers to run on different hosts or to use different port numbers.

The other file contains information about only this router's links.

We have supplied C files that read these files and construct a data-structure representation. `/net/cs455/assignments/proj3/readrouters.{h,c}`

Your program should be invoked like this:

```
router testdir routername
```

Where `testdir` is the name of a directory containing the configuration files and routername is a single-letter router name corresponding to an entry in the list of routers. You must run your routers against `/net/cs455/assignments/proj3/test{1,2}` and provide us with the output.

Your router should initialize its routing table with entries for all directly-connected neighbors. Non-neighbors can either be added dynamically or entered initially with infinite cost.

Messages

Router update messages

Each instance of your router will learn of better paths to other routers by receiving routing table updates from its neighbors. The neighbors send datagram messages that look like:

```
U d cost
```

That is, the letter U followed by a space, followed by a single-letter destination name, followed by a space, followed by the cost of reaching the destination from the neighbor expressed as a sequence of ascii decimal digits.

Link cost messages

Your router may receive link-cost changes as well. These look like:

```
L n cost
```

Where n is a neighbor and cost is the new cost of reaching that neighbor directly. (Link cost messages will only be received for existing links – you do not have to deal with new links coming into existence, but the cost of a link may change to infinity – and back).

Upon receiving either of these messages your router should make the appropriate changes to its routing table, and if there are changes, send the changed entries immediately to its neighbors using U-messages. Sending updates immediately like this is called "Triggered Update". Do not implement poison-reverse. Whenever a routing table entry is added or changed your program should print on standard output a message like this:

```
(r – dest: d cost: c nexthop: n)
```

Where r is the name of the router making the change, d, c, and n are the destination name, cost, and nexthop name, and the remainder of the pattern is literal characters. That is, you might use something like

```
printf("(%s – dest: %s cost: %d nexthop: %s)\n", r, d,
c, n)
```

to print these messages.

In addition to sending triggered updates, your router should send U-messages for all its routing table entries to each of its neighbors every 30 seconds.

Print messages

Finally, your router should respond to an incoming datagram consisting of:

```
P d
```

by printing its routing table entry for destination d. Use the above format but without the surrounding parentheses. If the d is omitted print all the entries in the routing table.

Implementation Notes

The links between neighboring routers are simulated using connected datagram sockets. A connected datagram socket sends to and receives from only its connected peer. For those using C, the function `createConnections in readrouters.c` will construct the connected datagram sockets for talking to neighbors and provide you with an array of (neighbor,socket) pairs. Your code will consult this array when sending to a neighbor to find out what socket to use. When receiving a message this array is consulted to determine from which neighbor the message came.

In addition to these connected sockets you will also need to create an unconnected socket bound to the local *baseport* to receive L and P messages (never U messages – those come only from neighbors).

C implementation without threads: The "next message" may arrive on any of these sockets (the connected ones or the unconnected one) and each router has to send its entire routing table to its neighbors, even in the absence of incoming messages, every 30 seconds. Therefore, you will need to use the `select()` system call to find out which socket descriptor(s) have available messages and issue `recv()` calls only for those descriptors. `select()` also provides the mechanism you need for determining when 30 seconds have passed. In addition to reading the relevant sections of "The Pocket Guide" book, I strongly suggest that you look at the unix man page for select().

Java or C with threads: since Java doesn't have select, you will need to use one thread per file descriptor and one for the timeout. Use synchronization between your threads to ensure that the routing table is updated, transmitted, and printed consistently.

Since you will need to start and stop multiple processes each time you change your code, I suggest that you develop shell scripts for this purpose. Please, please, please do not leave programs running unattended on the lab machines. And be alert to programs that are looping without doing useful work. The TAs and I reserve the right to unceremoniously kill any looping or unattended processes, but if we have to do it very often we're going to get real cranky. **Clean up your own garbage.**

Additional resources

RFC 1058 describes RIP, which is an internet routing protocol very much along these lines.
I've provided programs in the project directory to send the P and L control messages. These are python scripts so they will work on any of the lab machines (or anywhere else the python interpreter is installed as /usr/bin/python).

To send print messages:

```
printtables testdir r
printtables testdir
```

triggers printing of router r's table or all routers' tables, respectively by sending P messages.

To update the cost of a link

```
changelink testdir r1 r2 c
```

sends the appropriate L messages to r1 and r2.

The testdir argument is needed because the programs read the router configuration file to find the appropriate host and port destination for the messages that they send.

Another handy program is generateTest, which you can use to make up your own test networks.

```
generateTest testdir
```

reads the file testdir/links and produces testdir/routers and testdir/<routername>.cfg for each router. testdir/links is just a list of links, one per line, consisting of the names of the two endpoints and the cost of the link. Note that if any node of the graph has degree greater than 9 this program will work but the generated configuration will have problems. You can fix it by changing port = port + 10 near the bottom to port = port + 20.

Grading

- 65% of the grade will be for working code, defined as your routers computing the correct routing table for input test cases and respond correctly when link costs change.Your demonstration of your server to the TAs will be part of this grade.
- 5% for the README file
- 5% for the Makefile

- If your program will not compile or core-dumps when run the maximum credit is 50% for the project.

The remaining 25% of the grade will be for your COMMENTS file

COMMENTS file

Note: Even if you work with a partner, you are required to submit your ***own*** COMMENTS file, that is your *personal, individual* work.

The COMMENTS file for this project will be a mini-paper about routing and your implementation of it. Another way of looking at it is as a lab write-up. In your paper:

- Describe the routing problem
- Describe the solution implemented in your program.
- For each of the two test cases.
    - Describe the test case
    - What routing tables did your algorithm converge to? Are they correct?
    - In test1, what happens when link cost BE changes to 2, the algorithm is allowed to converge, and then BE changes back to 8.
- Describe any additional experiments you have performed with your router and the results of those experiments.
- Point out decisions that either I or you have made that could significantly affect the behavior of this distributed routing algorithm.
- Point out any discrepancies between the behavior you observe in your program and the behavior of D-V routing described in the textbook. One particularly likely discrepancy concerns "counting to infinity" when a link's cost increases.
- In your conclusion, assess how realistically your router models the function of a real internet router? What simplifying assumptions have we made? What aspects of the environment are different and might have significant influence on the performance of a real router? Again, I encourage you to read RFC 1058 as background.
- Finally, please tell me how long you spent on this project.

With your permission (see accompanying form), the COMMENTS file will be submitted to the WSU Critical Thinking Project for independent analysis as part of a research project on the use of the Critical Thinking Rubric at WSU. Your voluntary participation in the project is most appreciated (and in no way affects your grade, whether or not you participate).