CptS/EE 455
Fall 2016
Project 2
Assigned: Oct 19, 2016
Due: Monday, Nov. 14, 2016 at 11:59:59PM

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.

You may work individually or in teams of 2. If you work in a team, only one person is to turn in the work, but put both people's names in all files.

Logistics

You may write this program in C, C++, C#, Java, or Python. If you want to use a different language supporting demultiplexing using `select`, please ask me first. You may NOT use a multi-threaded solution.

The test cases and supporting python programs are to be obtained from the EECS github server:

```
git clone https://github.eecs.wsu.edu/hauser/CptS455-Fall16-Project2.git project2
```

This will prompt you for your eecs username and password and then download the files into the `project2` directory in which you can then develop your code.

What to turn in

Turn in a zip (not bzip, not tar, or anything else) file named project2.zip containing ONLY the following:

1. README.txt file - containing complete instructions for creating your router from source and for running it on the supplied test cases.
2. Makefile or equivalent
3. COMMENTS.pdf file. See below.
4. OUTPUT.txt - Test case output
5. Program code files (no stray IDE files, object code, etc.)

Specifications

You are to create a program that implements a distance-vector routing algorithm like the one section 4.5.2 of the text. You will implement it on multiple, communicating routers (emulated by processes).

Each router instance will read the configuration of its local links from files at the beginning of the run. One configuration file, `<testdir>/routers`, contains a list of all the routers in the system along with the hostname on which they run and the port numbers for a *command port* and and *update port*. Each of the other files, `<testdir>/<routername>.cfg`, contains the link information for router `<routername>`. 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. (***Note:*** *in each* `<testdir>/` *there is another file named* `net.cfg`. ***This file is not used directly by your program but can be used with the generateTest script described below to create the other files in the directory; let me repeat that: the file named*** `net.cfg` ***is NOT used by your program****)*

The test case files are included in the github project that you downloaded.

Your program should be invoked like this:

```
router [-p] 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, for example:

```
router test1 A
```

You must run your routers against the `test{1,2,4,cti}` directories (provided) and turn in the output in file `OUTPUT.txt`.

The `-p` switch controls use of poisoned-reverse. If `-p` is present on the command line use poisoned reverse, otherwise don't.

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

Infinity

For this project we will define infinity to be 64.

Message Protocol

Router update messages

Each router learns of better paths to other routers by receiving distance vector updates from its neighbors. Routers send distance vector update messages from their own update port to each neighbor router's update port that look like:

```
U d1 cost1 d2 cost2 … dn costn
```

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. The `di` and `costi` fields are repeated for as many entries as the sending router has in its own routing table.

Link cost messages

Your router may receive link-cost changes as well. These will be received on the router's command port and look like:

```
L n cost
```

Where `n` is the single-letter name of a neighbor and cost is the new cost of reaching that neighbor directly (as decimal digits). 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 an L message or a U message your router should make the appropriate changes to its own distance vectors and to its routing table, and if there are changes in the routing table, send the changed routing table immediately to its neighbors using a `U`-message. Sending updates immediately like this is called "Triggered Update".

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 a U-message containing all of its costs to each of its neighbors every 10 seconds. Prior to sending this message print on standard output a line like:

```
<r> - <time>
```

where <time> is expressed in seconds and has 3 digits after the decimal point. The reason for these messages is to ensure that your timeouts are actually occurring every 10 seconds so a lot of digits either before or after the decimal point are not needed.

Print messages

Finally, your router should respond to an incoming datagram on its command port consisting of:

```
P d
```

by printing its routing table entry for destination d. *Use the format specified above for printing the entry.* **If the d is omitted print all the entries in the routing table.**

Implementation Notes

The links between neighboring routers are to be simulated using *datagram sockets.*

Each router has a socket associated with its *command port*, and another socket associated with its *update port.* If you run multiple routers on the same machine you must ensure that every router is associated with a different pair of port numbers.

C implementation using `select`: at any given time, the next message that the router has to process may arrive on either of the two sockets. Additionally, each router is required to send its routing table to its neighbors, even in the absence of incoming messages, every 10 seconds. To implement this behavior you will need to use the `select()` system call to find out which socket descriptor(s) have available messages and issue `recvfrom()` calls only for those descriptors. `select()` also provides the mechanism you need for determining when 10 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()`.

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. You will need to kill all the processes robustly or they will pile up and cause your machine to eventually run out of memory (or even crash), and will certainly interfere with your ability to reuse port numbers from one run to the next. And be alert to programs that are looping without doing useful work.

Notice that U messages do not contain, in the message, any information about which router sent the message. You need to use the `recvfrom` system call (instead of `recv`) in order to obtain host ip address and port information about the sender. This along with the information in the `<testdir>/routers` file will let you figure out who sent the message.

Additional resources

RFC 1058 describes RIP, which is an internet routing protocol very much along these lines.
I've provided programs in the zip file to send the P and L control messages. These are python scripts so they will work on any machine where the python interpreter is installed as /usr/bin/python.

To send print messages use the `printtables` script:

```
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 use the `changelink` script:

```
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
```

Remember the `net.cfg` file that was mentioned previously as not being used by your program. `generateTest` reads the file `<testdir>/net.cfg` and produces the files `<testdir>/routers` along with `<testdir>/<routername>.cfg` for each router. `<testdir>/net.cfg` is just a list of links, one per line, consisting of the names of the two endpoints and the cost of the link.

Grading

- 75% of the grade will be for working code, defined as your routers computing the correct routing table for input test cases and responding correctly when link costs change, as well as sending periodic update messages at the correct times. Your demonstration of your server to the TA will be part of this grade.
- 5% for the README file
- If your program will not compile or core-dumps when run the maximum credit is 50% for the project.
- 10% of the grade will be for your COMMENTS file (see below)
- Once again you may work alone or with a partner; your submission will be graded on its technical content. Both partners are responsible for knowing the entire submission – why things are done as they are, how the code works, etc. You (or you and your partner) will meet with the TA and be quizzed about these things. This part of the grade is worth 10%.

COMMENTS file

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.
- Are the timeouts occurring properly every 10 seconds? (A few milliseconds variation is fine, but certainly not variation of more than a second from the once-every-10-seconds schedule.)
- For each of the four test cases (test1, test2, test4, and cti)
    - Briefly 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. Is the behavior different if the `-p` flag is used (poisoned reverse)?
    - In cti, what happens when the link cost AE link cost CD change to 64 and the system is given time to converge? Is the behavior different when the -p flag is used? Now change the two links back to cost 1. What happens?
    - DO NOT include all the output of these runs in the COMMENTS file. Summarize the results in a few sentences and figures. The output of the

runs should be turned in in the file name OUTPUT.txt; make sure to label the sections as corresponding to the different tests.

- 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. How do you explain the differences (if any).
- 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 each member of your team spent on this project – individually and in group sessions.

Division of work in teams

Some of the tasks associated with this project that might be good divisions of work for a group:

- Reading the configuration files
- Creating the sockets
- Designing data structures to hold information about which router is associated with which socket, etc.
- Designing data structures for the distance vector storage and routing table storage.
- Implementing the select loop that determines which socket to read
- Implementing the routing table/distance vector update algorithm (I strongly urge that the code for this NOT be embedded in the select loop, but rather in a separate function that the select loop code can call.
- Code to produce console output
- Writing the initial draft of the COMMENTS file. All team members should participate in writing and editing the final file that gets turned in.

Version history:
10/18/2016: Updated with requirements for 2016, including using separate command and update sockets and work in pairs or alone. Also added the `cti` test case and info about retrieving files from the eecs github server.