CptS/EE 455
Fall 2015
Project 2
Assigned: Oct 21, 2015
Due: Monday, Nov. 16, 2015 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.

Logistics

You may write this program in C, C++, 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.

What to turn in

Turn in a zip (not bzip, not tar) file named project2.zip containing:
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

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. The supplied C files `readrouters.{h,c}` read these files and construct a data-structure representation. 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 beginning port number that they use. 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* `links`. ***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 `links` *is NOT used by your program)*

The files are in the zip file in which you found this assignment.

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

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 vector 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. The neighbors send distance vector update messages 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 distance vector.

Link cost messages

Your router may receive link-cost changes as well. These 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 of these messages your router should make the appropriate changes to its own distance vector and to its routing table, and if there are changes, send the changed distance vector immediately to its neighbors using `U`-messages. 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 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 to be simulated using ***connected*** *datagram sockets, using a separate socket to communicate with each neighbor*.

Note about *connected datagram sockets*: although it is not usually necessary to call `connect` for datagram sockets (after calling bind to establish the port number for the *local* end) in this project we will do so for the sockets used to communicate between routers. There are two advantages from your perspective and one from mine: from your perspective this means that 1) you can use `send` and `recv` instead of `sendto` and `recvfrom` 2) you don't have to look up the full IP address/port number when messages are received to know where they came from; from my perspective this forces you to use `select` to handle the multiple sockets which is one of the points of this project. Connected datagram sockets send and receive datagrams *only* to/from the host/port to which they are connected. Unlike with TCP calling connect for a datagram socket does not exchange setup messages with the other end – it simply sets the destination and port for the local socket. If you send datagrams to a connected socket from an address/port that it is not connected to the messages are discarded.

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

Thus, each router has a socket associated with *baseport*, and additional sockets with portnumbers *baseport*+1, *baseport*+2, …, *baseport*+*numneighbors*. If you run multiple routers on the same machine you must ensure that these ranges of port numbers do not overlap.

C implementation using `select`: at any given time, the next message that the router has to process may arrive on any of the sockets. Additionally, each router is required to send its entire distance vector to its neighbors, even in the absence of incoming messages, every 30 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 `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()`.

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.

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 `links` file that was mentioned previously as not being used by your program. `generateTest` reads the file `testdir/links` and produces the files `testdir/routers` along with `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

- 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
- Once again this is a group project; your submission will be graded on its technical content, and your contribution to the group will also be graded. Every member of the group is responsible for knowing the entire submission – why things are done as they are, how the code works, etc. Your group will meet with the TA and be quizzed about these things. This part of the grade is worth 10%.
- Finally, you will need to complete a survey at the end of the project in which you rate the contribution of members of your group. Extraordinary contribution will receive up to 4 points additional credit; lack of contribution may receive up to 4 points deduction from the group score.

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.
- 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. Is the behavior different if the -p flag is used (poisoned reverse)?
    - 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.
- 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 can you explain the differences.

- 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 groups

Some of the tasks associated with this project that might be good divisions of work for a group:
- Reading the configuration files
- Creating and connecting all 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.