CSE 489/589 Fall 2014

Programming Assignment 3

Routing Protocols

Due Time [Design Document]: 11/17/2014 @ 23:59:59 EST

Due Time [Project]: 11/26/2014 @ 23:59:59 EST

1. Problem Statement

In this assignment you will implement a simplified version of the ***Distance Vector Protocol***. The protocol will be run on top of servers (behaving as routers) using UDP. Each server runs on a machine at a pre-defined port number. The servers should be able to output their forwarding tables along with the cost and should be robust to link changes. You need to implement the basic algorithm: **count to infinity, NOT poison reverse**.

In addition, a server should send out routing packets only in the following two conditions: a) periodic update.
b) the user uses a command asking for one.

This is a little different from the original algorithm which immediately sends out update routing information when routing table changes.

NOTE: You should use UDP Sockets only for your implementation. Further, you should **use only the select() API** for handling multiple socket connections. Do not use multi-threading or fork-exec. **If you use multi-threading or fork-exec, your project will not be graded.**
In general, you are not allowed to use any non-standard third-party libraries for the socket programming part, which generally are a wrapper around the low-level function/system calls. **If you make use of any such libraries, your project will not be graded**.
You will need to use select timeout to implement multiple timers. **Any other implementation of multiple timers will not be accepted.**

2. Getting Started

2.1 Distance Vector Routing Algorithm

Text book: Page 371 – Page 377.

2.2 Read section 6 and submit the *mandatory design document.*

2.3 Get the PA3 template

Read the installation section of the document at http://goo.gl/28HL0L

It is recommended that you follow the Installation section and setup the template before reading further, as this handout may include references to the template components. ***It is mandatory to use the template.***

3. Protocol Specification

The various components of the protocol are explained step by step. Please strictly adhere to the specifications.

3.1 Topology Establishment

In this programming assignment, you will use **five** CSE student servers – {**stones**, **euston, highgate, embankment, underground**}**.cse.buffalo.edu**. Each server will need to be supplied with a topology file at startup that it uses to build its initial routing table. The topology file is local and contains the link cost to the neighbors. For all other servers in the network, the initial cost would be infinity. Each server can only read the topology file for itself. The entries of a topology file are listed below:

- ***<num-servers>***
- ***<num-neighbors>***
- ***<server-ID> <server-IP> <server-port>***
- ***<server-ID1> <server-ID2> <cost>***

num-servers: total number of servers.
server-ID, server-ID1, server-ID2**:** A unique identifier for a server.
cost**:** Cost of a given link between a pair of servers. Assume that cost is either a non-zero positive integer value.
E.g., consider the topology in Figure 1. We give a topology file for server 1 (timberlake).

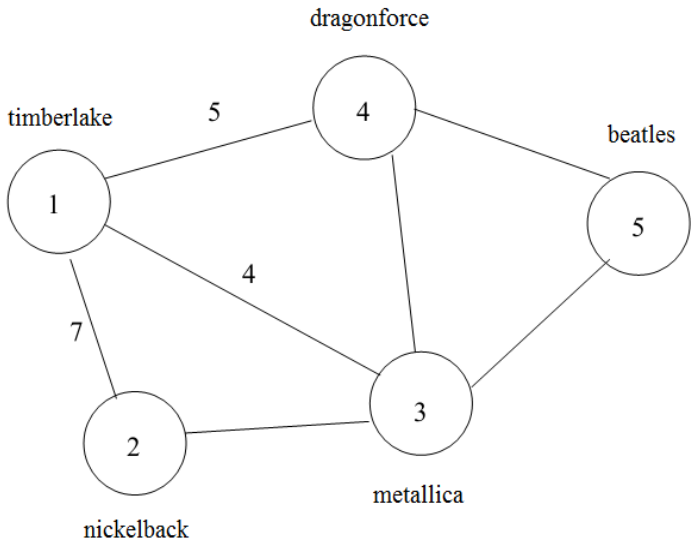


Figure 1: Example topology

Line number	Line entry	Comments
1	5	number of servers
2	3	number of edges or neighbors
3	1 128.205.36.8 4091	server-id 1 and corresponding IP, port pair
4	2 128.205.35.24 4094	server-id 2 and corresponding IP, port pair
5	3 128.205.36.24 4096	server-id 3 and corresponding IP, port pair
6	4 128.205.36.4 7091	server-id 4 and corresponding IP, port pair
7	5 128.205.36.25 7864	server-id 5 and corresponding IP, port pair
8	1 2 7	server-id and neighbor id and cost
9	1 3 4	server-id and neighbor id and cost
10	1 4 5	server-id and neighbor and cost

Your topology files will contain only the Line entry part (2nd column, see topology_example.txt). In each line, every two elements (e.g., server-id and corresponding IP, corresponding IP and port number) will be separated with a **space**. For cost values, **each topology file will contain the cost values of the host server's neighbors** (Host server here is the one which will read this topology file). You can use your own topology files to test your code. However, we will use our topology files to test your program. So please adhere to the format for your topology files.

IMPORTANT: In this environment, costs are bi-directional i.e. the cost of a link from A-B is the same for B-A. Whenever a new server is added to the network, it will read its topology file to determine who its neighbors are. Routing updates are exchanged periodically between neighboring servers. When a newly added server sends routing messages to its neighbors, they will add an entry in their routing tables corresponding to it. Servers can also be removed from a network. When a server has been removed from a network, it will no longer send distance vector updates to its neighbors. When a server no longer receives distance vector updates from its neighbor for three consecutive update intervals, it assumes that the neighbor no longer exists in the network and makes the appropriate changes to its routing table (**link cost to this**

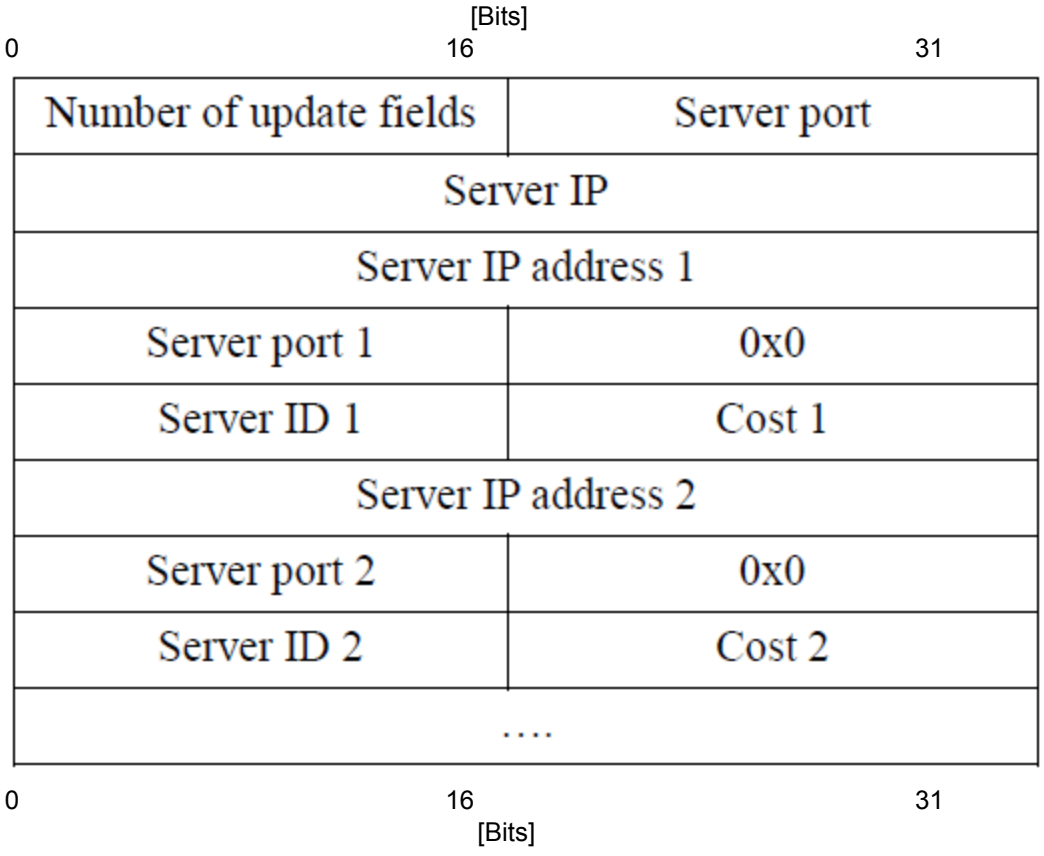
neighbor will now be set to infinity but not remove it from the table). This information is propagated to other servers in the network with the exchange of routing updates. Please note that although a server might be specified as a neighbor with a valid link cost in the topology file, the absence of three consecutive routing updates from this server will imply that it is no longer present in the network.

3.2 Routing Update

Routing updates are exchanged periodically between neighboring servers based on a time interval specified at the startup. In addition to exchanging distance vector updates, servers must also be able to respond to user-specified events. Events can be grouped into three classes: topology changes, queries, and exchange commands. Topology changes refer to an updating of link status (update). Queries include the ability to ask a server for its current routing table (display), and to ask a server for the number of distance vectors it has received (packets). In the case of the packets command, the value is reset to **zero** by a server after it satisfies the query. Exchange commands can cause a server to send distance vectors to its neighbors immediately.

3.3 Routing Update Packet Format

Routing updates are sent using the General Message format. All routing updates are UDP unreliable messages. The message format for the data part is:



- **Number of update fields:** (2 bytes): Indicate the number of entries that follow.
- **Server port:** (2 bytes) port of the server sending this packet.
- **Server IP:** (4 bytes) IP of the server sending this packet.
- **Server IP address n:** (4 bytes) IP of the n-th server in its routing table.
- **Server port n:** (2 bytes) port of the n-th server in its routing table.
- **Server ID n:** (2 bytes) server id of the n-th server on the network.
- **Cost n:** (2 bytes) cost of the **path** from the server sending the update to the n-th server whose ID is given in the packet. To represent **inf** (infinity) value, use the largest unsigned integer that can be represented in 2 bytes.

The update packets need to be exactly as described above, failing which will result in loss of points.

Note: First, the servers listed in the packet can be any order e.g., 5, 3, 2, 1, 4. Second, the packet needs to include an entry to reach itself with cost 0 e.g., server 1 needs to have an entry of cost 0 to reach server 1.

4. Server Commands

To start a server

- **./assignment3 -t <*path-to-topology-file*> -i <*routing-update-interval*>**
  path-to-topology-file: The topology file contains the initial topology configuration for the server, e.g., timberlake_init.txt. Please adhere to the format described in 3.1 for your topology files.
  routing-update-interval: It specifies the time interval between routing updates in seconds.
  port and server-id: They are written in the topology file. The server should find its port and server-id in the topology file without changing the entry format or adding any new entries.

The following commands can be specified at any point during the run of the server:
[The commands should be **case-insensitive; both uppercase and lowercase** versions should be accepted by your program]

- **update <*server-ID1*> <*server-ID2*> <*Link Cost*>**
  server-ID1, server-ID2: The link for which the cost is being updated.
  Link Cost: It specifies the new link cost between the source and the destination server. It can be any non-zero positive value or ***inf*** indicating infinity.
  Note that this command will be issued to **both** *server-ID1* and *server-ID2* and involve them to update the cost and no other server.

- **step**
  Send routing update to neighbors right away (triggered/force update). Note that except this, routing updates only happen periodically.

- **packets**
  Display the number of distance vector packets this server has received since the last invocation of this command.

- **display**
  Display the current routing table. Table should be displayed in a **sorted** order from small ID to big. The display should be formatted exactly as described in section 5.

- **disable <*server-ID*>**
  Disable the link to a given server. Doing this "closes" the connection to a given server with *server-ID*. Here you need to check if the given server is its neighbor.

- **crash**
  This simulates a server crash. Close all connections on all links. The neighboring servers must handle this close correctly and set the link cost to infinity.

- **dump**
  Build a routing update packet and write it as is to a binary file (DO NOT send it).

- **academic_integrity**
  We will NOT grade your submission, if this command fails to work.
  Prints the academic integrity statement (see sec. 5 for the exact string).

5. Server Responses/Output Format

We will use automated tests to grade this assignment. The grader, among other things, will also look at the output generated by your program. Towards this end, **ALL the *required* output generated by your program needs to written to BOTH stdout and to a specific logfile**. This section also provides exact format strings that to be used for output (for each individual command), **which need to be strictly followed.**

5.1 Print and LOG

We have already provided a convenience function for this purpose in the template (see src/logger.c and include/logger.h), which writes both to stdout and to the logfile. **You should use ONLY this function, for all output described in this handout**. **On the other hand, if you want to output something more to stdout, than what is described, you should NOT use this function and rather use native C function calls.**

To use the function you will need to have the following statement at the top of your .c file(s) where you want to use this function:

#include "../include/logger.h"

The function is designed to behave almost exactly as printf. You can use the function as:

cse4589_print_and_log(char format, ...)*

Read the comments above the function definition contained in the src/logger.c file, for more information on the arguments and return value.

5.2 Output Message Format

In this section, we will provide very specific format strings that you need to use for print-logging the output of each of the commands. **You should stick to these strictly to avoid loss of points.**

The following is the list of responses expected from a server on different events:

- On successful execution of an update, step, packets, display, disable, dump or academic_integrity command, the server must display the following message:

  ("%s:SUCCESS\n", <*command-string*>)

  where *command-string* is the command executed. Additional command-specific output (e.g., for display, packets, etc. commands) specified below, should follow the above output from the next line.

- Upon encountering an error during execution of one of these commands, the server must display the following response:

  ("%s:%s\n", <*command-string*>, <*error message*>)

  where *error message* is a brief description of the error encountered.

- On successfully receiving a route update message from neighbors, the server must display the following response

  ("RECEIVED A MESSAGE FROM SERVER %d\n", <*server-ID*>)

  where the *server-ID* is the id of the server which sent a route update message to the local server.

  This should be followed by the information contained in the update packet. The display should be formatted as a sequence of lines in **sorted** order from small ID to big, with each line has the following form:

  ("%-15d%-15d\n", <server-ID>, <cost-of-path>)

  To represent **inf** value, use the largest unsigned integer that can be represented in 2 bytes.

- On successfully executing **display** command, display the current routing table. The display should be formatted as a sequence of lines in **sorted** order from small ID to big, with each line has the following form:

  ("%-15d%-15d%-15d\n", <destination-server-ID>, <next-hop-server-ID>, <cost-of-path>)

  To represent **inf** value, use the largest unsigned integer that can be represented in 2 bytes. For a server with **<cost-of-path>** as *inf*, the **<next-hop-server-ID>** will be **-1**.

- On successfully executing the **dump** command, you should build a routing update packet and write it as is to a binary file (DO NOT send it). For this, you should use ONLY the function provided in the template.

  To use the function you will need to have the following statement at the top of your .c file(s) where you want to use this function:

  #include "../include/logger.h"

  The function is designed to behave almost exactly as fwrite. You can use the function as:

  cse4589_dump_packet(const void packet, size_t bytes)*

  Read the comments above the function definition contained in the src/logger.c file, for more information on the arguments and return value.

- On successfully executing **academic_integrity** command, you should display the following message:

  ("I have read and understood the course academic integrity policy located at http://www.cse.buffalo.edu/faculty/dimitrio/courses/cse4589_f14/index.html#integrity")

NOTE: There are no '\n' characters in all the above format strings, unless where stated explicitly. The line breaks that you see above are a result of word-wrapping. In your code, you should print the strings as is, without introducing extra '\n' characters at the visible line-breaks.

6. Design Document

A design document (max. 2 pages) for the project needs to be submitted before you start working on the assignment. **Without the submission of this design document within the deadline (mentioned at the very beginning of this handout), we will NOT grade your assignment.** At the very least it should contain the following items:

- C/C++ code of the data-structure you plan to use for defining the update packet structure.
- C/C++ code of the data structure you plan to use for storing the routing table.
- Pseudocode for the main select loop. It should be clear from your pseudocode how and where you plan to handle the different I/O events. You can use meaningful function names to convey your intent.
- Pseudocode/Strategy to implement multiple timers for each of the neighbors in the network graph.

The design document document needs to be submitted in a text (ASCII) file named as <ubitname>_pa3_design.txt. You should submit this file using the CSE submit script for cse489/cse589.

7. Submission

- Use ONLY the package script included in the template to package your files for submission.
- If your submitted tarball differs from what the template script produces, we will NOT be able to grade your assignment.
- You SHOULD NOT submit any topology files.