Assignment 4

Computer Science 441

Due: 23:55, Friday December 9, 2016

Instructor: Majid Ghaderi

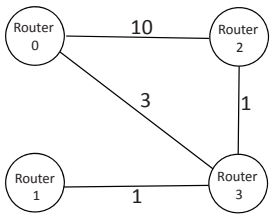
1 Objective

The purpose of this assignment is to implement a distributed asynchronous distance vector routing algorithm. By completing this assignment, you will learn about distributed computing in a realistic network environment.

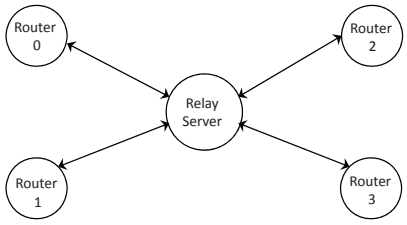
2 Overview

You will develop a program called *Router* to emulate the behavior of a router executing a modified version of the distance vector routing algorithm discussed in class. Several instances of your router program will be started on possibly different hosts that communicate with each other using TCP.

To have a controlled environment for this assignment, rather than directly communicating with each other, the routers will communicate through a *relay server* as depicted in Figure 1. The relay server forwards packets passing through it to the appropriate receiving routers, very much like a switch. Unfortunately, the relay server is lazy and sometimes drops packets that go through it!



(a) Network topology with 4 routers.



(b) Relay server forwards packets.

Figure 1: Routers and relay server.

3 Relay Server

The relay server is implemented as a TCP server that listens on a specific port to communicate with the routers. From the perspective of routers, the relay server forwards routing packets to appropriate destination routers.

- **Network Topology:** Once the relay server starts, it reads the network topology information from a text file. The topology information is presented as the adjacency matrix of the network formed by the routers. An example topology file is provided to you. It is possible to specify several topologies in a single file. All topologies in a file should have the same number of routers and be separated from each other by a blank line. If there are n routers

in the network, each is uniquely identified with an ID in $\{0, 1, \ldots, n-1\}$. During the execution, the server can be instructed to cycle through the specified topologies to emulate link cost change in the network.

- **Routing Packets:** The packets exchanged between routers and the relay server are of type `DvrPacket`, which is implemented as a Java class. The source code for `DvrPacket` is provided to you along with tester code showing how to use it. The class `DvrPacket` is *serializable* so that its instances can be conveniently read and write from and to the socket using object IO streams. Each `DvrPacket` contains the unique IDs of its sender and receiver. The relay server's ID is denoted by `SERVER`, which is defined in class `DvrPacket`.

4 Router Design

Every router has a unique ID that is assigned to it when the router object is created. In your implementation, in addition to the cost of the minimum cost path to each destination (*i.e.*, the mincost vector), you should also keep track of the neighboring router that is on the minimum cost path to the destination (*i.e.*, the next hop router).

In the following, we describe the main tasks performed by a router.

- **Initialization:** When started, each router must send a `HELLO` packet to the relay server and then wait to receive a `HELLO` response from the server. The `HELLO` response contains the link cost vector associated with the router. The link cost vector is sent via the instance variable `DvrPacket.mincost`. For the routers that are not directly connected to this router, the corresponding link cost values are set to `INFINITY`. The number of routers in the network is equal to the length of the link cost vector received in the `HELLO` message from the relay server. Upon receiving the initial distance vector from the server, the router should initialize its internal distance vector and any other data structures needed by your implementation.
- **Operation:** The router has to update its distance vector when it receives a routing packet from its directly connected neighbors. The values received in a routing packet (carried in `DvrPacket.mincost`) from some other router i, contain i's current mincost vector to all other routers in the network. Regular `DvrPacket`s sent by routers are of type `ROUTE`.
- **Termination:** When receiving a routing packet of type `QUIT` from the relay server, the router has to stop its operation, perform clean up and then return its forwarding table to the calling process. The forwarding table consists of both the mincost and nexthop vectors.

5 Routing Update

- **Updating Neighbors:** This part is slightly different from the algorithm described in class. Rather than updating neighboring routers whenever there is a change in the local mincost vector, your router should update its neighbors periodically (at fixed intervals) regardless of any change in its mincost vector. The update interval is specified as an input parameter. The updated mincost vector is sent in routing packets of type `ROUTE`.
- **Link Cost Change:** If a router receives a `ROUTE` packet with `DvrPacket.sourceid` set to `SERVER`, it means that the network topology has changed. The new link cost vector for the router is included in the received `DvrPacket.mincost` array.

6 Software Interfaces

For detailed information about the following methods and classes, refer to the Javadoc documentation provided in the source code.

Define a Java class named `Router`, which includes the following public methods:

- `Router(int routerId, String serverName,`
  `int serverPort, int updateInterval)`
  This is the constructor to initialize the router program.
- `RtnTable start()`
  Starts the router and runs the routing algorithm in a loop until terminated by a `QUIT` packet. The forwarding table of the router is returned once this method terminates. To return the forwarding table, simply create an object of type `RtnTable` and return it, as follows:

  ```
  return new RtnTable(mincost, nexthop);
  ```

  where `mincost` and `nexthop` are the mincost and nexthop vectors at the router.

Your implementation should include appropriate exception handling code to deal with various exceptions that could be thrown in the program. A skeleton class, named `Router`, is provided on D2L. Notice the import statement

```
import cpsc441.a4.shared.*
```

at the top of the file. The package `cpsc441.a4.shared` contains classes `DvrPacket` and `RtnTable` to be used in your program. A jar file named `a4.jar` containing the package `cpsc441.a4.shared` is provided to you on D2L. Make sure to add this jar file to your class

path when compiling and running your program. For example, if you are using Java command line tools, use the option `-cp` to include `a4.jar` in your class path.

The source code for classes `DvrPacket` and `RtnTable` is also provided to you as a source of documentation for using these classes:

- `DvrPacket`: This class defines the structure of the messages exchanged between routers (via the relay server). Each `DvrPacket` has a mincost array and specifies the packet's sender and receiver IDs, and the type of the packet.
- `RtnTable`: This is a wrapper class used by the `Router.start()` method to return the forwarding table of the router.

The relay server is provided to you in a Java jar file called `rserver.jar`. The server comes with a `readme` file that explains how to run it and set its various parameters.

Restrictions

- You are not allowed to change the signature of the methods provided to you. You can however define other variables, methods or classes in order to complete the assignment.
- Your program should use the library file `a4.jar`. Do not use `DvrPacket.java` or `RtnTable.java` in your code. The source code is provided to you only for documentation purposes.
- You have to write your own code for sending and receiving `DvrPacket`s over TCP. Ask the instructor if you are in doubt about any specific Java classes that you want to use in your program.