Simulate Link State Routing Protocol with Java Socket Programming

Goal

In this project, you are supposed to develop a pure user-space program which simulates the major functionalities of a routing device running a simplified Link State Routing protocol.

To simulate the real-world network environment, you have to start multiple instances of the program, each of which connecting with (some of) others via socket. Each program instance represents a router or host in the simulated network space; Correspondingly, the links connecting the routers/hosts and the IP addresses identifying the routers/hosts are simulated by the in-memory data structures.

By defining the format of the messages transmitting between the program instances, the parser and the handlers of these messages, you simulate the routing protocol with the user-space processes.

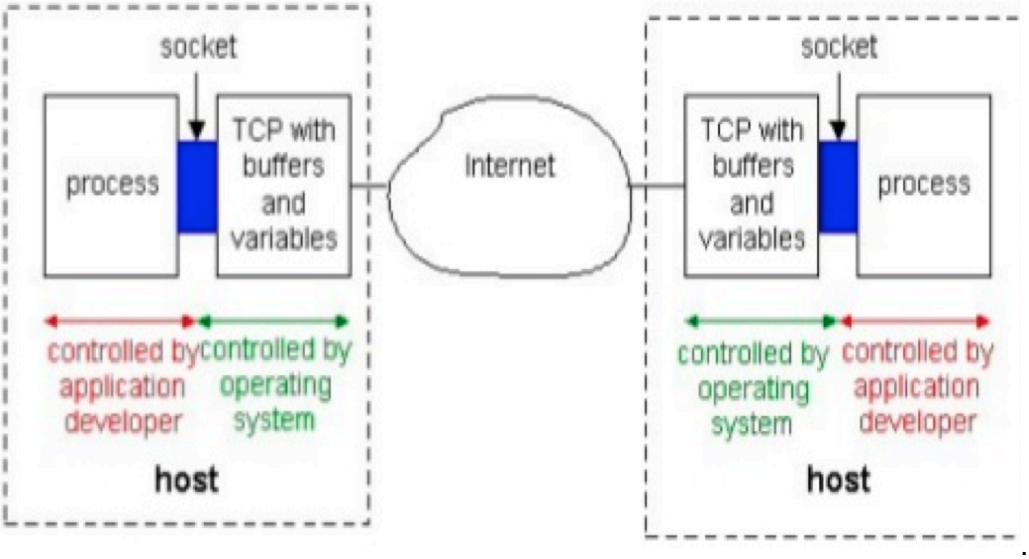
Prerequest

Before you do this PA, please ensure that you understand the basic concept of routing, especially Link State Routing, which is taught in class.

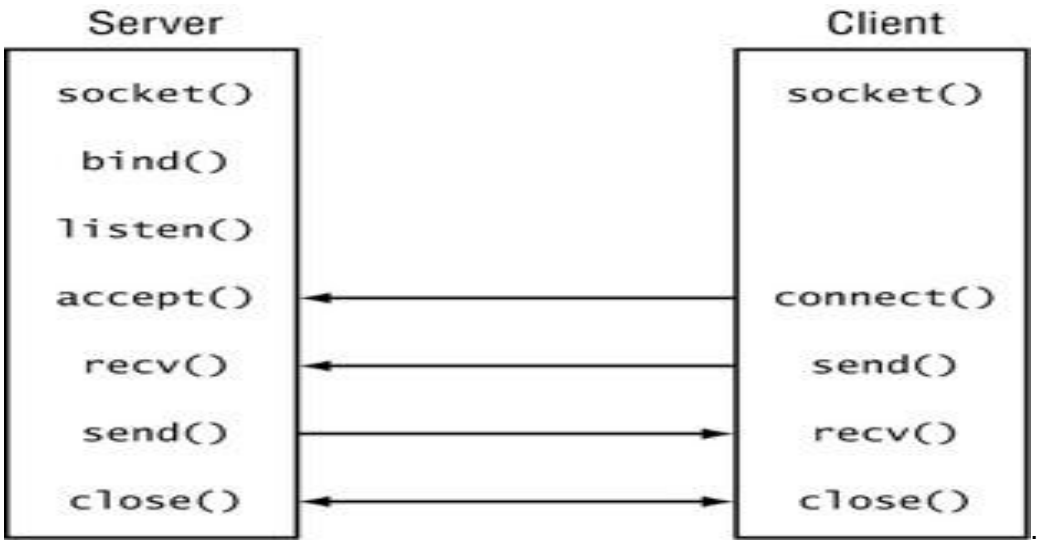
Recommended Reading: Link State Routing

Socket Programming 101

Socket is the interface between the application layer and tranmission layer



The existence of Socket interface greatly reduce the complexity of the developing network-based applications. When the processes communicate via socket, they usually fall into two categories of roles, `server` and `client`. The usage of socket in these two roles are different.



In `server` side, the program create a socket instance by calling `socket()`. With this socket instance, you can `bind()` it to a specific IP address and port, call `listen()` for waiting to active connections, `accept()` to accept the connection. After you call `accept()`, you can then call `recv()` and `send()` to transmit data between the client and itself. After you finish all tasks, you can call `close()` to shutdown the socket.

In `client` side, the story seems a bit simpler, after you call `socket()` to create a socket instance, you only need to call `connect()` with the specified IP address and port number to

request service from the server side. After the connection is established, the following process is very similar to server side, i.e. transmit data with `recv()` and `send()` and shutdown with `close()`.

This is the general process of socket-based communication. To understand it better, you are suggested to read the article in here. The article is described in C programming language, which exposes many details of network data transmission but helpful to understand the concept.

Java Socket Programming

Different programming languages offers their own abstraction over socket programming. You are requestd to finish this project in Java. Java provides higher level abstraction for socket. In server side, You only need to call `ServerSocketserverSocket = new ServerSocket(port);` to create socket, bind, listen in one shot. In client side, `Socket client = new Socket(serverName, port);` creates socket instance and connect to the remote server.

The data tranmission between the server and client is through stream. For example, the following code snippet writes data to remote server and wait for the feedback.

```
OutputStream outToServer = client.getOutputStream();
DataOutputStream out =
              new DataOutputStream(outToServer);

out.writeUTF("Hello from "
             + client.getLocalSocketAddress());
InputStream inFromServer = client.getInputStream();
DataInputStream in =
               new DataInputStream(inFromServer);
System.out.println("Server says " + in.readUTF());
```

Here we say `wait for the feedback`, because `getInputStream().read()` is a blocking method. According to the Java API, "This method blocks until input data is available, the end of the stream is detected, or an exception is thrown." (Blocking Read)).

This is not a good thing for high-throughput scenarios, e.g. routers. We have to develop some way to handle concurrent socket requests.

Recommended Reading: Java Socket Programming

Handle Concurrent Socket Requests

In this project, you are supposed to develop a multi-threaded server to handle concurrent socket requests from the client. Recommended Reading: Multi-threaded Java Server

Simulation of Link State Routing

The basic idea of link state routing is that every router maintains its own description of the connectivity of the complete network. As a result, each router can calculates the next best hop for all possible destinations in the network. The key point in Link State Routing Protocol is to synchronize the network description in all nodes (In this project, we only consider a single LAN deployment).

In the following paragraphs, we will describe your tasks in this project in details.

Router Design

The first task in this project is to design the `Router` class which represents a router instance in the simulated network.

How Simulation Mechanism Works

Before we introduce the necessary components and functionalities in router, we have to emphasize how the "simulation" mechanism (in this project) works.

In the Socket introduction part, we have described that each socket-based program has its own IP address and port, which are the identifiers used to communicate with other processes. In this project, you need to use these "Process IP" and "Process Port" to establish connection via socket. On the other side, in the "simulated network", we assign a "simulated ip address" to each router. This ip address is only used to identify the router program instance `in the simulated network space`, but `not` used to communicate via sockets.

You have to map between this simulated ip address and the "Process IP" and "Process Port". With this map, when you are requested to establish a link to a router with the specified simulated ip. For example, you run two simulated router instances at port 50000" and 50001 respecitively in the machine with the ip address "172.12.1.10". When you create socket instance, you have to use `172.12.1.10:50000` or `172.12.1.10:50001` to connect two program instances. However, in this project, you will have to build links in a simulated network topology, the program instance started at `172.12.1.10:50000` might be assigned with a simulated IP address of `192.168.1.10`. This explains why you have to map between the simulated ip address and the "Process IP" and "Process Port".

Data Structures in Routers

Each router is identified by a simulated "IP address", it is simply a String variable. In this project, the routers is assumed to have 4 ports, which means that the router class contains a 4-length, `Link` typed array. When the router is connected with the other, you shall fill the array with a Link object. If the ports are all occupied, the connection shall not be established.

The most important component in router side is the `Link State Database`. Each router

maintains its own `Link State Database` which is essentially a map from router IP address and the neighborhood description which is originated by the corresponding router. The shortest path algorithm is ran over this database.

Command-line Terminal

Besides the components introduced above, you have to develop a terminal for the router. When you start the router program, the terminal interface (command line based) should popup, and it allows the user to input following commands:

- attach [Process IP] [Process Port] [IP Address] [Link Weight]: establish a connection to the remote router which is identified by [IP Address]. After you start a program instance, the first thing you have to do is to run `attach` command to establish the new link to the other routers. This process is achieved by update the local data structures by adding the new Link object in the port array. In this command, besides the Process IP/Port and simulated IP Address, you also need to specify the "Link Weight" which is the cost for transmitting data through this link and is useful when you decide the shortest path to the certain destination.
- start: start this router and initialize the database synchronization process. After you establish the links by running `attach`, you will run `start` command to send `HELLO` message to all connected routers to establish the link and `LSAUPDATE` to synchronize the Link State Database. This process will be illustrated in the next section.
- connect [Process IP] [Process Port] [IP Address] [Link Weight]: similar to `attach` command, but it directly triggers the database synchronization without the necessary to run `start` (this command can only be run after `start`).
- disconnect [IP Address]: remove the link between this router and the remote router which is identified by [IP Address]. Through this command, you are triggering the synchronization of Link State Database by sending `LSAUPDATE` (Link State Abstract Update) message to all neighbors in the topology. This process will also be illustrated in the next section.
- detect [IP Address]: output the routing path from this router to the destination router which is identified by [IP Address].
- quit: exit the program.

Link State Database Synchronization and Update

Start

After you run `start` message, the router where this command is started should send `HELLO` messages to all routers which have been connected via `attach` command.

The process of handling `HELLO` is as following:

a. Router 1 (R1) broadcast Hello messages through all ports
b. the remote end (R2) receives a `HELLO` message, set the status in the RouterDescription of R1 as INIT, then sends Hello to R1
c. R1 receives the `HELLO` from R2, set the status of R1 as TWO_WAY, sends `HELLO` to R2
d. R2 receives `HELLO` from R1, set status of R1 as TWO_WAY

Synchronize Link State Database

The synchronization of Link State Database happens when the link state of a router changes. The router where the link state changes broadcasts `LSAUPDATE` which contains the latest information of link state to all neighbors, which in turn broadcast to their own neighbors except the one which sends the `LSAUPDATE`.

`LSAUPDATE` contains one or more `LSA` (Link State Abstract) structures which summarize the latest link state of the router. To update the local link state database with the latest information, you have to distinguish the `LSAUPDATE` information generated from the same router with a monotonically increasing sequence number. The router receiving the `LSAUPDATE` only update its Link State Database when the `LSAUPDATE`'s sequence number is larger than maximum sequence number from the same router it just received.

Note that the synchronization of link state database is triggered in three cases: 1) after two connected routers are both set as TWO_WAY; 2) connect; 3) disconnect.

Shortest Path Finding

Based on the information saved in Link State Database, you can build the weighted graph representing the topology of the network. With the weighted graph, you can find the shortest path from the router to all the other peers with Dijkstra algorithm.

When you run `detect [IP Address]` command, the Dijkstra algorithm is run over the database and output the result.

Evaluation

1.