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.

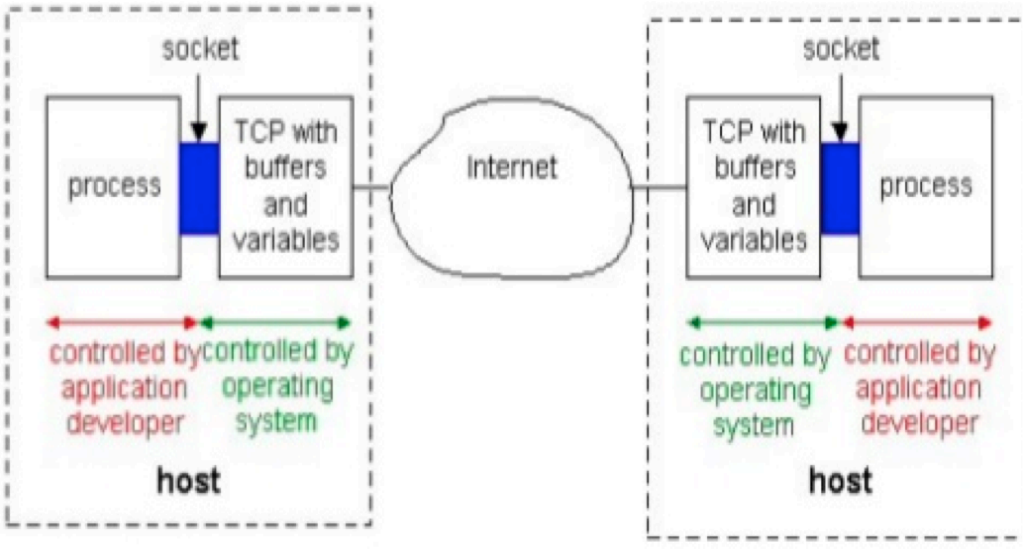
By defining the format of the messages transmitting between the program instances, as well as the parser and the handlers of these messages, you simulate the routing protocol with the user-space processes.

Prerequest

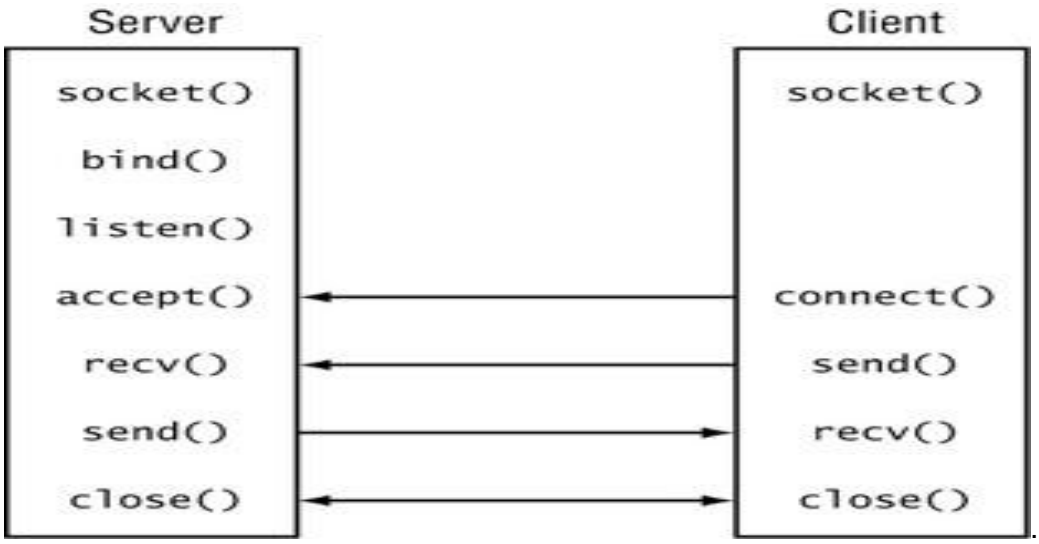
Before you start this project, please ensure that you understand the basic concept of routing, especially Link State Routing, which is taught in class.

Socket Programming 101

Socket is the interface between the application layer and tranmission layer



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



In `server` side, the program creates a socket instance by calling `socket()`. With this socket instance, you can `bind()` it to a specific IP address and port, call `listen()` to wait for the connecting requests, `accept()` to accept the connection. After you call `accept()`, you can transmit data with the client by calling `recv()` and `send()`. After you finish all tasks in server side, 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 the 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 the socket-based network communication. To understand it better, you are suggested to read the article in here (http://gnosis.cx/publish/programming/sockets.html). The article is described in C programming language, which exposes many details of network data transmission but helpful to understand the concepts.

Java Socket Programming

Different programming languages offer their own abstractions over socket interface to help the user to develop network-based programs. You are requestd to finish this project in Java. Java provides higher level abstraction for socket than C. 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)).

The blocking operations are always performance killer 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/messages 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 best next hop for all possible destinations independently. The key point for the correctness of Link State Routing Protocol is to synchronize the network description in all nodes.

In the following paragraphs, we will describe your tasks around the network description synchronization.

Router Design

One of the major tasks in this project is to implement the `Router` class which represents a router instance in the simulated network.

How Simulation Works

Before we introduce the 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 "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" to simulate the link state routing protocol. For example, you run two simulated router instances at port 50000 and 50001 respecitively, and the machine where you run the programs has the ip address of "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. On the other side, 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`. You have to use this `192.168.1.10` to describe the network topology, etc. This explains why you have to build a 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 are 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 the router's IP address to the link state description which is originated by the corresponding router. The shortest path algorithm runs over this database.

Command-line Console

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

- attach [Process IP] [Process Port] [IP Address] [Link Weight]: this command establishs a link to the remote router which is identified by [IP Address]. After you start a router program instance, the first thing you have to do is to run `attach` command to establish the new links to the other routers. This operation is implemented by adding the new Link

object in the port array (check the sketch code). 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 calculate 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` messages and `LSAUPDATE` to all connected routers for the Link State Database synchronization. This operation 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 [Port Number]: remove the link between this router and the remote one which is connected at port [Port Number] (port number is between 0 - 3, i.e. four links in the router). Through this command, you are triggering the synchronization of Link State Database by sending `LSAUPDATE` (Link State Advertisement 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].
- neighbors: output the IP Addresses of all neighbors of the router where you run this command.
- quit: exit the program. NOTE, this will trigger the synchronization of link state database.

Link State Database Synchronization and Update

Start

After you run `start` message, the router where this command is triggered should send `HELLO` messages to all routers which have been linked 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. The routers which receive the message will in turn broadcast to their own neighbors except the one which sends the `LSAUPDATE`.

`LSAUPDATE` contains one or more `LSA` (Link State Advertisement) structures which summarize the latest link state information 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.

Link State Database synchronization is triggered by `start`, `connect`, `disconnect` and `quit`.

Shortest Path Finding

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

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

Mapping This Document to Sketch Code

We provide the sketch code along with this document. Please note that you should feel free to add fields, methods, helper functions to the sketch code. For example, you can choose to add one more field in the Link class to describe the cost of the link or you can add your own code in LinkStateDatabase class to finish the same task.

In this section, we provide an brief introduction to the sketch code.

Router Design

We provide the sketch code of the Router in src/main/java/socs/network/node/Router.java:

- `ports` is the array of Link which stands for the 4 ports of the router;
- `rd` is an instance of RouterDescription, which is a wrapper of several describing fields of the router: `processIPAddress` and `processPortNumber` are where the server socket of the router program binds at; `simulatedIPAddress` is the simulated IP address and the identifier of this router; `status` is the instance of RouterStatus describing the stage of the database synchronization (see more explanations in the code);
- `lsd` is the instance of LinkStateDatabase (src/main/java/socs/network/node/LinkStateDatabase.java) (LSD). LSD contains a

HashMap which maps the linkStateId to the Link State Advertisement (LSA). LSA is the data structure storing the `LinkDecription` of in the router which advertised this LSA.

In the Router class, you have to implement the functionalities of creating socket instances (Server Socket and Client Socket) and communicating via them. Besides that, you have to complete the implementation of the functions with the name starting with `processing` which are the handlers of different commands triggered by the user. When you implement the handler of the `detect` command, you also need to fill the implementation of the method named `getShortestPath` in LinkStateDatabase Class.

Messages

The class in `src/main/java/socs/network/message/SOSPFPacket.java` defines the message format transmission among routers. The messages are distinguished by the field of `sospfType`, where 0 stands for HELLO and 1 stands for LinkStateUpdate.

`LINKSTATEUPDATE` message contains a set of LSA instances, which are all LSAs stored in the router originating this message (To reduce the load amount of this project, we simplify this database synchronization process, please read the textbook for the complete definition of a link state routing protocol). Each `LSA` (src/main/java/socs/network/message/LSA.java) has three fields: `linkStateID`, this is the simulated IP address of the router which originates this LSA. `lsaSeqNumber` describes the version of the LSA, when synchronizing database, you have to compare this value of the existing and the newly arrived LSA from the same server to prevent updating the database with stale values. `links` are a set of `LinkDescription` instances which describes the links attached to the router which originated this LSA.

Configuration

The sketch code provides a configuration module to read the config file for the program, we have a predefined configuration entry named `socs.network.router.ip` defining the simulated ip address of the router instance. You can add other entries like the ip address and the port which the socket binds to for your necessary.

Compile the Program

We use maven to compile and package the program into a jar file.

- compile (you have to install maven first, http://maven.apache.org/download.cgi)

```
mvn compile
```

Package all class files into a fat jar file

```
mvn compile assembly:single
```

Then you can run the program with the java command, e.g. "java -jar -cp router.jar config.conf"

Evaluation

The project is divided into three Programming Assignment, worthing 10%, 10% and 10% of the score respectively. The evaluation of the assignment is in the form of demo. The TA will ask the students to show different functionalities of the program and will also ask the students to explain the implementation of some code snippets.

Programming Assignment 1

In this assignment, you are supposed to finish the topology building functionality of the program. The TA will ask you to run `attach`, `start` and `neighbors` commands.

When you run `start`, you have to print out the log of change of the state. For example, if you `attach` Router 2 to Router 1 (192.168.1.2) and run `start` in Router 2 (192.168.1.3). The terminal window of Router 1 should output

```
received HELLO from 192.168.1.3;

set 192.168.1.3 state to INIT;

received HELLO from 192.168.1.3;

set 192.168.1.3 state to TWO_WAY;
```

The window of Router 2 should print

```
received HELLO from 192.168.1.2;

set 192.168.1.2 state to TWO_WAY;
```

When you run `neighbors`, the window of the router where you run the command should print

```
IP Address of the neighbor1

IP Address of the neighbor2

IP Address of the neighbor3

...
```