Assignment 3

Computer Networks (CS 456/656)

Spring 2011

Shortest Path Routing Algorithm

Due Date: Monday, July 25th 2011, at midnight (11:59 PM)

Work on this assignment is to be completed individually

1 Assignment Objective

For this assignment, you will implement a shortest path routing algorithm.

2 Overview

Like most link-state algorithms, OSPF uses a graph-theoretic model of network topology to compute shortest paths. Each router periodically broadcasts information about the status of its connections. OSPF floods each status message to all participating routers. A router uses arriving link state information to assemble a graph. Whenever a router receives information that changes its copy of the topology graph, it runs a conventional graph algorithm to compute shortest paths in the graph, and uses the results to build a new next-hop routing table.

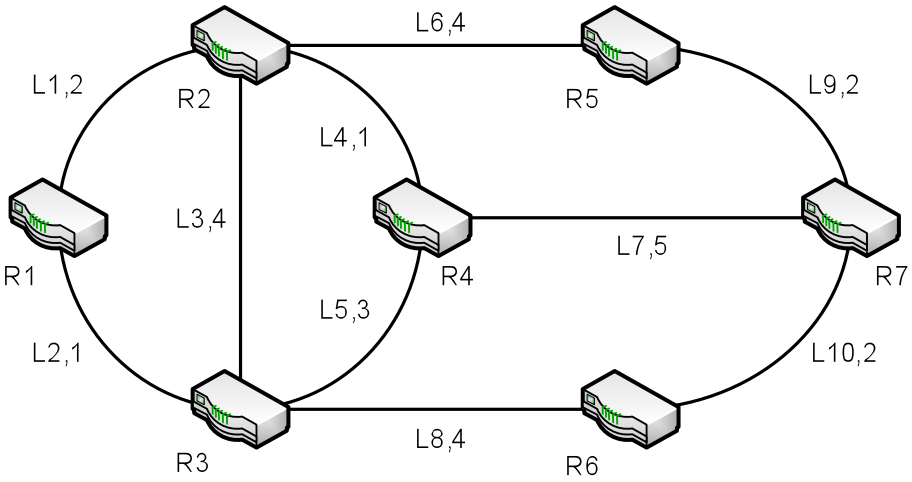

FIGURE 1: INTERNET TOPOLOGY

OSPF uses a directed graph to model an Internet. Each node in an OSPF topology graph either corresponds to a router or a network. If a physical connection exists between two objects in an Internet, the OSPF graph contains a pair of directed edges (one in each direction) between the two nodes that represent the objects. To enable routers to compute shortest paths, each edge in an OSPF graph is assigned a weight that corresponds to the cost of using the path.

The identities of the links that are attached to a router together with their cost metrics are entered into a table (known as the **circuit database**).

Circuit Database for R1	
Link	**Cost**
L1	2
L2	1

FIGURE 2: INITIALIZED CIRCUIT DATABASE

Before a router can send any PDU to its neighbors it should first send a HELLO PDU to tell its neighbor that the router is ready to participate in the OSPF procedure. So at the beginning of the procedure each router sends a HELLO PDU to each of its neighbors.

Each router sends to each of its neighbors a **link state (LS) PDU** containing the identifiers of the links to which the router is attached together with the cost values. In this way, a router receives some LS PDUs from each of its neighbors informing it of the links that are attached to it and their cost values. This information is stored in the **link state database** by the update process in the router.

R1 From R2 – R2: L1,2/L3,4/L4,1/L6,4 R3 – R3: L2,1/L3,4/L5,3/L8,4
R2 From R1 – R1: L1,2/L1,1 R3 – R3: L2,1/L3,4/L5,3/L8,4 R4 – R4: L4,1/L5,3/L7,5 R5 – R5: L6,4/L9,2
| | |
R7 From R4 – R4: L4,1/L5,3/L7,5 R5 – R5: L6,4/L9,2 R6 – R6: L8,4/L10,2

FIGURE 3: LINK STATE DATABASE AFTER FIRST SET OF LS PDUS

When it has done this, the update process in each router sends a copy of the LS PDU to each of its neighbors (except the one that has sent the PDU and those from which it has not received a HELLO PDU yet; also the same message should be sent to a router only once to avoid loops). As a result, each router receives a further set of LS PDUs which have effectively originated from its neighbors' neighbors. This procedure then continues. As can be deduced, over a period of time each router will receive a complete set of LS PDUs containing the identities of the links - and their path cost values - which are attached to all other routers in the Internet.

Whenever a new set of LS PDUs is entered into the link state database, the router performs the Shortest Path First (SPF) algorithm on the link state database, and determines, from all the entries in

the various databases, which neighbor R should be used to reach each of the other routers based on their corresponding path cost values.

To illustrate this procedure, consider its application to the example subnet shown in Figure 1.

The initialized circuit database is shown in Figure 2 and the first set of LS PDUs that are received by each router (R) is shown in Figure 3.

For example, R1 will receive 8 LS PDUs, 4 from R2 and 4 from R3. Similarly, R2 will receive 11 LS PDUs, 2 from R1, 4 from R3, 3 from R4 and 2 from R5.

As described earlier, on receipt of each of these PDUs and if those later affect its LS database, each R will then generate another set of LS PDUs, and pass them on to each of its neighbors. This procedure then repeats. The first, second and third sets of LS PDUs that will be received by R1 are shown in Figure 4, Figure 5, and Figure 6.

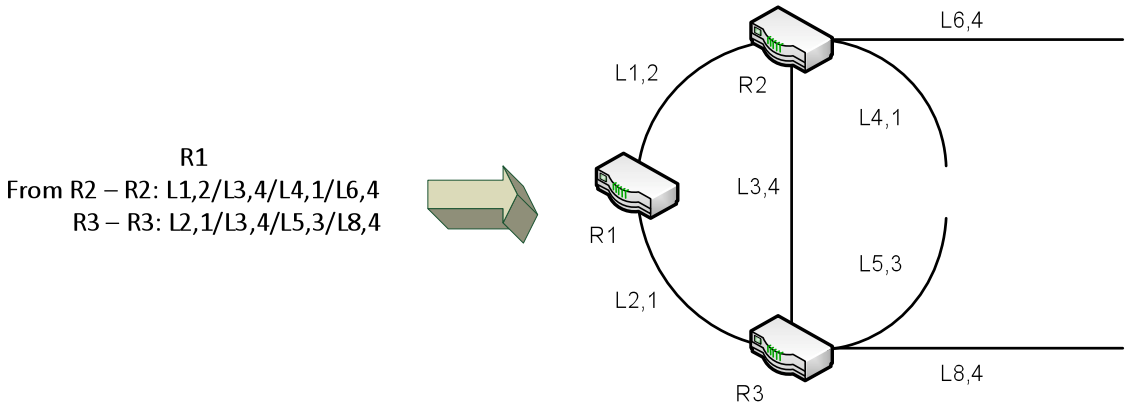

FIGURE 4: LINK STATE DATABASE DEVELOPMENT FOR R1 (FIRST SET OF LS PDUS)

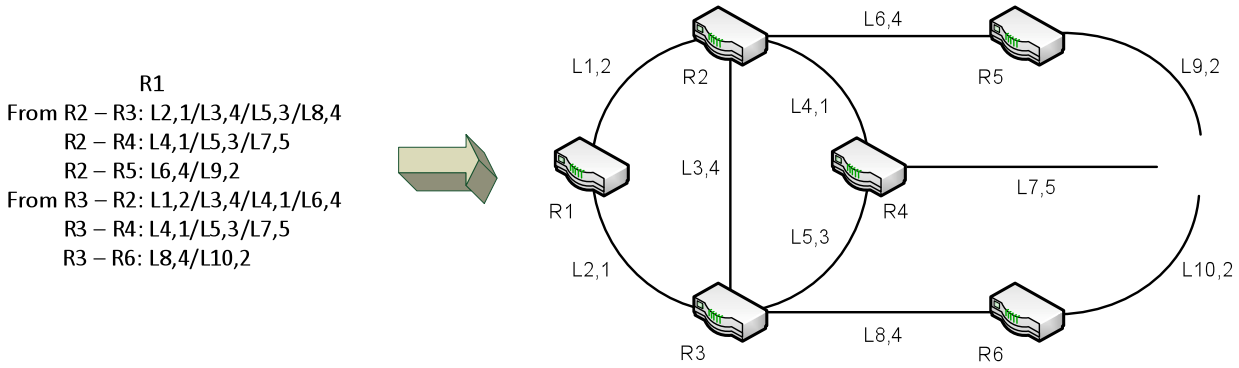

FIGURE 5: LINK STATE DATABASE DEVELOPMENT FOR R1 (SECOND SET OF LS PDUS)

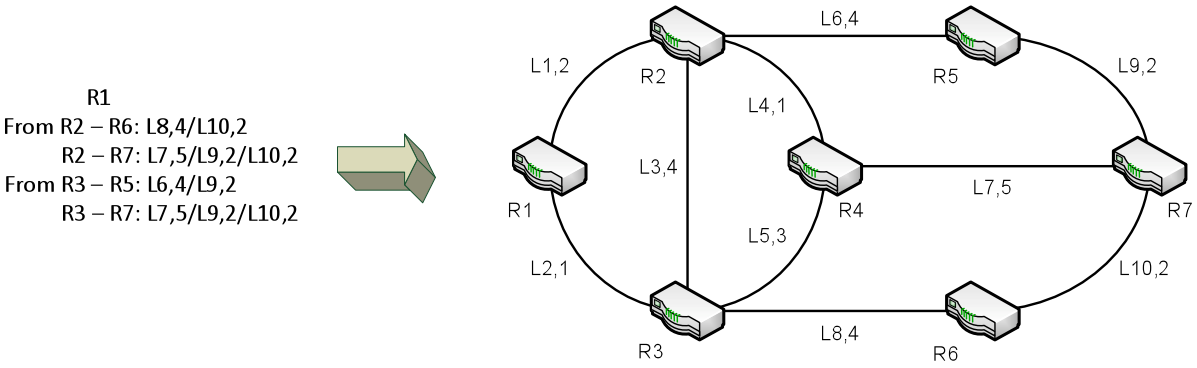

FIGURE 6: LINK STATE DATABASE DEVELOPMENT FOR R1 (THIRD SET OF LS PDUS)

Finally, Figure 7 shows the output of the decision process for R1. This is the routing information base (RIB) and is computed by the decision process performing the SPF algorithm on the contents of the link state database to determine the shortest (minimum) path cost to each destination R.

Destination R	Path, Cost
R1	Local,0
R2	R2,2
R3	R3 1
R4	R2,3
R5	R2,6
R6	R3,5
R7	R3,7

FIGURE 7: FINAL RIB FOR R1

3 Hints

The following structures are assumed (and should be respected):

```
#define NBR_ROUTER 5          /* for simplicity we consider only 5 routers */

struct pkt_HELLO
{
  unsigned int router_id;     /* id of the router who sends the HELLO PDU */
  unsigned int link_id;       /* id of the link through which it is sent */
}

strruct pkt_LSPDU
{
  unsigned int sender;        /* sender of the LS PDU */
  unsigned int router_id;     /* router id */
  unsigned int link_id;       /* link id */
  unsigned int cost;          /* cost of the link */
  unsigned int via;           /* id of the link through which the LS PDU is sent */
}
```

```
struct pkt_INIT
{
  unsigned int router_id;     /* id of the router that send the INIT PDU */
}

struct link_cost
{
  unsigned int link;          /* link id */
  unsigned int cost;          /* associated cost */
}

struct circuit_DB
{
  unsigned int nbr_link;      /* number of links attached to a router */
  struct link_cost linkcost[NBR_ROUTER];
  /* we assume that at most NBR_ROUTER links are attached to each router */
}
```

When a router needs to send packets to another router, it sends them to the Network State Emulator instead of sending them directly to the router. The Network State Emulator then forwards the routers packets to the right receiver.

4 What to do?

You should implement the router program, named `router`. Its command line input should include the following:

`router <router_id> <nse_host> <nse_port> <router_port>`

where ,

- `<router_id>` is an integer that represents the router id. It should be unique for each router.
- `<nse_host>` is the host where the Network State Emulator is running.
- `<nse_port>` is the port number of the Network State Emulator.
- `<router_port>` is the router port

You will be given the executable of the **Network State Emulator (nse)**, where circuit databases are *hardcoded* and will be sent to the routers during the initialization phase. Its command line is:

`nse <routers_host> <nse_port>`

where,

- `<routers_host>` is the host where the routers are running. For simplicity we suppose that **all routers are running on the same host**.
- `<nse_port>` is the Network State Emulator port number.

The Network State Emulator will not affect any packet; it simply forwards the packet to the right receiver. The topology used by the simulator is also provided in an accompanying PDF file.

First each router must send an `INIT` packet to the Network State Emulator containing the router's id. After the Network State Emulator receives an `INIT` packet from each router, the Network State Emulator will send to each router the circuit database associated with that router. The circuit database

will be sent in a `circuit_DB` structure. For the assignment, you should always run the five router programs in order from the 1st to the 5th.

Then each router must send a `HELLO` packet to all its neighbors. Each router will respond to each `HELLO` packet by a set of `LS PDU`s containing its circuit database. When receiving an `LS PDU` a router will update its topology database and then it will send to all its neighbors (except the one that send the `LS PDU` and those from which the router did not receive a `HELLO` packet yet) the `LS PDU`. The `LS PDU` is contained in the `pkt_LSPDU` structure. Before a router can send the `pkt_LSPDU`, it must change the field sender to its router id value. When receiving an `LS PDU` a router should also use the Dijkstra algorithm using its link state database to determine the shortest (minimum) path cost to each destination R.

For both testing and grading purpose, your router program should be able to generate a log file, named as `router (id).log`, where `id` is the id of the router (i.e. router 1 will generate a log file with name router1.log). The routers must record in the log file all the messages that they receive and all messages that they send. They must also record their topology database every time this later changes. The log file should contain the corresponding RIB for each topology. Before each line of the trace, the router must write its id.

An example of a message trace is the following:

R1 receives an `LS PDU`: sender 3, router_id 7, link_id 7, cost 5, via 2
/* it can be interpreted as : R1 receives from R3 via link id 2 that R7 has a link with id 7 and cost 5 */

An example of a *topology database* is shown in the following:

```
# Topology database
R1 -> R1 nbr link 2
R1 -> R1 link 1 cost 2
R1 -> R1 link 2 cost 1
R1 -> R2 nbr link 4
R1 -> R2 link 1 cost 2
R1 -> R2 link 3 cost 4
R1 -> R2 link 4 cost 1
R1 -> R2 link 6 cost 4
```

Notice that the topology database will be subject to change every time information about a new router becomes available

An example of a *RIB* is given below:

```
# RIB
R1 -> R1 -> Local, 0
R1 -> R2 -> R2, 2
R1 -> R3 -> R3, 1
R1 -> R4 -> R2, 3
R1 -> R5 -> R2, 6
R1 -> R6 -> R3, 5
R1 -> R7 -> R3, 7
```

4.1 Example Execution (with 5 routers)

- On the host **hostX**

```
nse hostY 9999
```

- On the host **hostY**

```
router 1 hostX 9999 9991
router 2 hostX 9999 9992
router 3 hostX 9999 9993
router 4 hostX 9999 9994
router 5 hostX 9999 9995
```

- Expected output

```
router1.log router2.log router3.log
router4.log router5.log
```

5 Procedures

5.1 Due Date

The assignment is due on Monday, July 25th 2011, at midnight (11:59 PM)

5.2 Hand in Instructions

Submit your all your files in a single compressed file (.zip, .tar etc.) using UW-ACE Assignment 3 Drop Box.

You must hand in the following files / documents:

- *Source code* files.
- *Makefile*: your code **must** compile and link cleanly by typing "*make*" or "*gmake*".
- *README* file: this file ***must*** contain instructions on how to run your program, which undergrad machines your program was built and tested on, and what version of *make* and *compilers* you are using.

If you choose not to use C, the makefile you submit must generate shell scripts that run your programs as specified. Name the script `router`.

Your implementation will be tested on the machines available in the **undergrad environment**.

5.3 Documentation

Since there is no external documentation required for this assignment, you are expected to have a reasonable amount of internal code documentation (to help the markers read your code).

You **will** lose marks if your code is unreadable, sloppy, inefficient, or not modular.

5.4 Evaluation

Work on this assignment is to be completed individually.