

WIDE AREA NETWORKS
SIMPLE LINK STATE ROUTING PROTOCOL

Using C Programming

Prabhat Kuchibhotla
Vaideesh Ravi Shankar

SPRING 2016

Table of Contents

OBJECTIVE	1
DESIGN	1
STRUCTURES USED	1
ROUTER FUNCTIONALITIES	1
RESULTS	6
Name Resolver – REGISTER & RESOLVE.....	6
Router – REGISTER & RESOLVE.....	7
Cost Matrix and Dijkstra’s Shortest path	7
Link Failure.....	8
File Transfer	9
Client – Start File Transfer	9
Client – End of File Transfer	9
Server – Received File Transfer Segment.....	10
DRAWBACKS OF ROUTING PROTOCOL.....	10
REFERENCES.....	10

OBJECTIVE

To design and implement a Simple Link State Routing Protocol (SLSRP).

DESIGN

The LSRP design consists of routers, client and a host. The number of routers in the design is dynamic (set to 5 by default). The client and the host are connected to an edge router and can perform file transfer.

STRUCTURES USED

- 1) **Struct data**
 - a. Data structure is used to store the router's IP address, IP address of other routers, ports used by all routers
- 2) **Struct header**
 - a. Used to fill the header of every segment before sending.
- 3) **Struct rtt**
 - a. Used to calculate the cost of the link based on timestamps.
- 4) **Struct alive**
 - a. Used to check if neighbor router link is alive.
- 5) **Struct lsa**
 - a. Used to advertise the cost matrix.
- 6) **Struct nacq**
 - a. For neighbor acquisition
- 7) **Struct nr**
 - a. To resolve and store the neighbor data from the Name Resolver.

ROUTER FUNCTIONALITIES

The router consists of the following functionalities:

Listen (Thread):

- 1) The router listens for any incoming packet in the receive buffer. If there is a packet in the buffer, the router checks the checksum of the packet in the buffer. It accepts the packet only if the checksum is correct
- 2) A switch case determines the type of packet from the packet header.
 1. ALIVE: Used to check if the neighbor is still connected
 - a. Types of ALIVE message: ALIVE_MSG, ALIVE_REPLY
 - b. If the alive message is of type ALIVE_MSG, reply back to the router from which the packet arrived with the same sequence number and type ALIVE_REPLY.

- c. If the alive message is of type ALIVE_REPLY, Change the variable alive_check [neighbor router] =1, which indicates to the alive thread that the router is alive.
- 2. NACQ: Used to acquire or deny neighbor-ship with a neighbor
 - a. Types of NACQ messages: BE_NEIGHBOR_REQUEST, BE_NEIGHBOR_ACCEPT, BE_NEIGHBOR_REFUSE, CEASE_NEIGHBOR
 - b. If the NACQ message is of type BE_NEIGHBOR_REQUEST, reply back to the router from which the packet arrived with type BE_NEIGHBOR_ACCEPT if not blacklisted and change value of element corresponding to the router from which the packet arrived in the adjacency matrix value to 1.
 - c. If the router is blacklisted, reply back with a type NEIGHBOR_REFUSE.
 - d. If not blacklisted and message is of type BE_NEIGHBOR_ACCEPT, change value of element corresponding to the router from which the packet arrived in the adjacency matrix value to 1.
 - e. If the NACQ message is of type BE_NEIGHBOR_REFUSE, change value of element corresponding to the router from which the packet arrived in the adjacency matrix value to 0.
- 3. RTT:
 - a. Used to calculate the cost of the link. If two routers (or a router and a host) are connected, the router with the larger Router ID calculates the cost and shares the cost with the smaller router ID. This way they can have a constant link cost. Router with larger Router ID sends RTT1. Router with smaller Router ID replies back with RTT2 with a timestamp. The router with the larger Router ID calculates the cost and replies back with RTT3 with the link cost. RTT4, RTT5, RTT6 are used with the same functionalities as RTT1, RTT2 and RTT3 but used between router and host.
 - b. Types of RTT messages: RTT1, RTT2, RTT3, RTT4, RTT5, RTT6. RTT1, RTT2 and RTT3 are used between routers. RTT4, RTT5, RTT6 are used between router and host (client or server).
 - c. If the RTT message is of type RTT1, take timestamp and reply with RTT2.
 - d. If the RTT message is of type RTT2, calculate cost and reply with RTT3 having link cost.
 - e. If the RTT message is of type RTT3, change value of element corresponding to the router from which the packet arrived in the adjacency matrix to value of the cost.

- f. Similarly RTT4, RTT5 and RTT6 are used to calculate link cost between router and host.
4. LSA: Used to inform the neighbors the cost of its links.
 - a. Check if the lsa_id (router id of the router that created the LSA) in the LSA message is the same as router_id (from itself). If so discard.
 - b. Check if the previous lsa sequence number (lsa id) > lsa_id. If so, check if previous lsa age (lsa id) > lsa age. If so, discard the packet.
 - c. Update the values of previous lsa sequence number and previous lsa age.
 - d. Update the values of the adjacency matrix from the lsa_cost in the LSA message.
 - e. Check if the value of ttl is 0, if so discard the packet. If not, decrement the value of ttl.
 - f. Forward the LSA packet with the same lsa_id and lsa_cost to all the neighbors.
5. FILETRANS: The received packets for file transfer are forwarded to the next hop based on the routing table computed by Dijkstra's algorithm.
 - a. Using the pred [] array from Dijkstra's, the router first checks if the predecessor of the destination ID is the router itself. If it is, the packet is forwarded to the end host.
 - b. If not, it will check if the router is the predecessor of the predecessor of the destination end host ID. Checking if the router is the predecessor of the predecessor and so on of the destination ID is computed until the router is the predecessor of another router on the shortest path to the destination. Once this is computed, it will send the packet to the next hop.
6. REGISTER: In this case, the router listens to an Acknowledgment from the Name Resolver that the router has registered successfully.
7. RESOLVE: The resolve responses from the NR are received and stored in the appropriate data structure neighbors.
8. RESOLV: This is to check if the Resolve service at the NR is up. Only once the NR is ready to receive resolve requests, it sends a "Resolve UP" message to all routers, after which the resolve requests are made.

Adperiodic (Thread):

1. The router executes the following set of commands every 5 seconds.
 - a. Update the sequence number, ttl, age and lsa_id and lsa_cost.
 - b. Send a LSA message to all neighbors only if the neighbors are not blacklisted, and are not hosts (end systems).

Nacq (Thread):

1. The router executes the following set of commands every 10 seconds.

- a. Send a BE_NEIGHBOR_REQUEST to all routers that have a link only if the routers are not already connected, not blacklisted and not hosts.

Alive (Thread):

Types of Alive messages: ALIVE_MSG, ALIVE_REPLY

1. The router executes the following set of commands every 10 seconds.
 - a. Update the sequence number
 - b. Send a message of type ALIVE_MSG to all neighbors and only if the routers are not already connected, not blacklisted and not hosts.
 - c. Wait for 5 seconds
 - d. Check if alive_check =1. When a router receives an alive reply, alive_check is updated to 1. If not, link to that router has failed. Change value of element corresponding to the router in the adjacency matrix as 0.

Dijkstra (thread):

1. The router executes the following set of commands every 5 seconds.
 - a. Prints the cost matrix (cost matrix is the adjacency matrix but 0s replaced with 999s).
 - b. Computes the shortest path to all routers and hosts in the topology.
 - c. Displays the shortest path.
 - d. Stores the predecessor of each router via the shortest path. This is later used in computing the next hop while forwarding packets in file transfer.

Code Usage

The code will execute perfectly if the following is followed:

1. Start the NR first on **Oxygen** server.
 - a. This is because we had to manually assign the IP address and port number of the NR in each of the router codes. It is assigned to **136.142.227.10:55000** (IP address : Port #)
2. Compile each router code with different output file names and execute them in the form
 - a. **./<output file name><space><router id>**
 - b. Example for executing router 0: **./y0 0**
3. In order to change the topology, the links [] array in the main function of each router has to be set to **1** accordingly.
 - a. Example: In router 0 code, to have links to router 1 and 2, put links[1] =1; and links[2]=1;
4. To execute the client host, use execution command: **./<client output filename> 5**
 - a. 5 is the ID assigned to the client
5. Similarly for the server, except use ID 6. **./<server output filename> 6**
 - a. 6 is the ID assigned to the server

6. To start file transfer on client, type the file path to the .txt file in the command line interface when prompted.
7. To check for link failure, terminate one of the routers and observe the output on the routers having links to the failed router. The “**Link to router id 2(eg.) failed**” message will be printed and routing table changed accordingly.
8. Once the failed router is brought back up, it will connect to the router again and “**Connected to Router 2(eg.)**” message will be displayed and routing table changed accordingly.

Functions used:

changeadj() : This function is used to change the value of the adjacency matrix. A mutex is used to implement thread synchronization and for protecting shared data when multiple writes occur.

crc16() : This function returns checksum of the data. Take

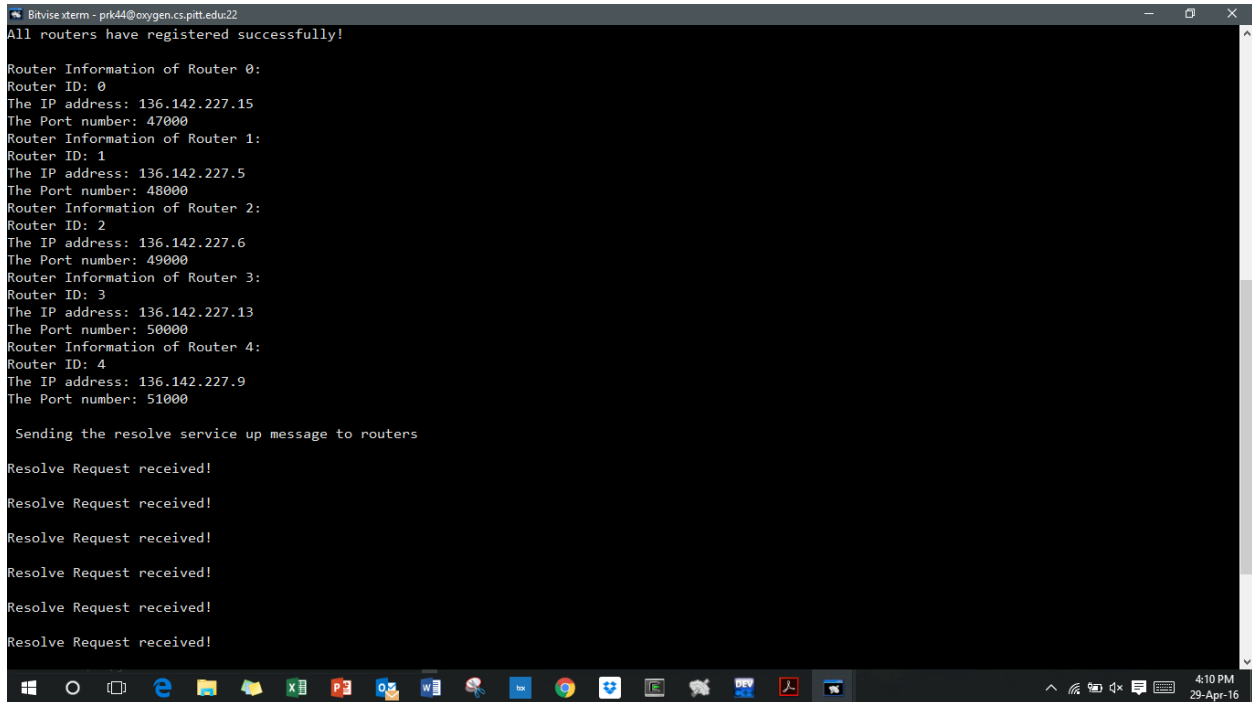
current_timestamp(): This function returns the time in milliseconds.

client_send(): This function sends a packet to the IP address and port specified. A mutex is used to implement thread synchronization.

create_pkt(): This function fills in the header for each packet to be sent with the source router ID, destination router ID, seq number and data.

RESULTS

Name Resolver – REGISTER & RESOLVE



```
Bitvise xterm - prk44@oxygen.cs.pitt.edu:22
All routers have registered successfully!

Router Information of Router 0:
Router ID: 0
The IP address: 136.142.227.15
The Port number: 47000
Router Information of Router 1:
Router ID: 1
The IP address: 136.142.227.5
The Port number: 48000
Router Information of Router 2:
Router ID: 2
The IP address: 136.142.227.6
The Port number: 49000
Router Information of Router 3:
Router ID: 3
The IP address: 136.142.227.13
The Port number: 50000
Router Information of Router 4:
Router ID: 4
The IP address: 136.142.227.9
The Port number: 51000

Sending the resolve service up message to routers

Resolve Request received!
Resolve Request received!
Resolve Request received!
Resolve Request received!
Resolve Request received!
Resolve Request received!
```

The above screenshot is at the NR when all routers register successfully when each router boots. After all routers are registered, the NR prints that all routers have registered and prints their respective data. Also it sends a “Resolve UP” message to all routers after which the NR receives the resolve requests.

Router – REGISTER & RESOLVE

```

Bitvise xterm - prk44@neptunium.cs.pitt.edu:22
.000000 0.000000 0.000000

Distance of node 0 = 999.000000
Path = 0 <- 3
Distance of node 1 = 999.000000
Path = 1 <- 3
Distance of node 2 = 999.000000
Path = 2 <- 3
Distance of node 4 = 999.000000
Path = 4 <- 3
Distance of node 5 = 999.000000
Path = 5 <- 3
Distance of node 6 = 999.000000

-bash-3.2$ ./y3 3
Starting router 3 at :50000
Socket created
Bound to 50000
Socket created
Bound to 6744

Registration Successful confirmation received from NR!

Got the Resolve UP message from the NR

Information of all neighbors obtained:

Router Information of Router 2:
Router ID: 2
The IP address: 136.142.227.6
The Port number: 49000
Router Information of Router 4:
Router ID: 4
The IP address: 136.142.227.9
The Port number: 51000

```

The above screenshot shows the router display when it receives the “**Registration Successful**” message from the NR and also receives the data of its neighbors **resolved** from the NR and prints it.

Cost Matrix and Dijkstra’s Shortest path

```

Bitvise xterm - prk44@neptunium.cs.pitt.edu:22
.000000 0.000000 0.000000

Distance of node 0 = 1.920000
Path = 0 <- 1 <- 2 <- 3
Distance of node 1 = 1.280000
Path = 1 <- 2 <- 3
Distance of node 2 = 0.640000
Path = 2 <- 3
Distance of node 4 = 0.840000
Path = 4 <- 3
Distance of node 5 = 999.000000
Path = 5 <- 3
Distance of node 6 = 999.000000
Path = 6 <- 3
Cost matrix:
  0      1      2      3      4      5      6
0  0.000000  0.512000  0.000000  0.000000  0.000000  0.000000  0.000000
1  0.512000  0.000000  0.512000  0.000000  0.000000  0.000000  0.000000
2  0.000000  0.512000  0.000000  0.712000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.712000  0.000000  0.840000  0.000000  0.000000
4  0.000000  0.000000  0.000000  0.840000  0.000000  0.000000  0.000000
5  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
6  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

Distance of node 0 = 1.736000
Path = 0 <- 1 <- 2 <- 3
Distance of node 1 = 1.224000
Path = 1 <- 2 <- 3
Distance of node 2 = 0.712000
Path = 2 <- 3
Distance of node 4 = 0.840000
Path = 4 <- 3
Distance of node 5 = 999.000000
Path = 5 <- 3
Distance of node 6 = 999.000000

```

The cost matrix of the entire network is displayed and the shortest path to each router from that router is shown with the distance as well.

Link Failure

```

Bitwise xterm - prk44@neptunium.cs.pitt.edu:22
Distance of node 1 = 1.224000
Path = 1 <- 2 <- 3
Distance of node 2 = 0.712000
Path = 2 <- 3
Distance of node 4 = 999.000000
Path = 4 <- 3
Distance of node 5 = 999.000000
Path = 5 <- 3
Distance of node 6 = 999.000000
Path = 6 <- 3
Link to router id 2 FAILED!!!
Link to router id 4 FAILED!!!

Cost matrix:
  0      1      2      3      4      5      6
0  0.000000  0.512000  0.000000  0.000000  0.000000  0.000000  0.000000
1  0.512000  0.000000  0.512000  0.000000  0.000000  0.000000  0.000000
2  0.000000  0.512000  0.000000  0.712000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
4  0.000000  0.000000  0.000000  0.872000  0.000000  0.000000  0.000000
5  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
6  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

Distance of node 0 = 999.000000
Path = 0 <- 3
Distance of node 1 = 999.000000
Path = 1 <- 3
Distance of node 2 = 999.000000
Path = 2 <- 3
Distance of node 4 = 999.000000
Path = 4 <- 3
Distance of node 5 = 999.000000
Path = 5 <- 3
Distance of node 6 = 999.000000

```

When links from 3 to two routers 2 and 4 failed, it displays the link failure message and updates the cost matrix.

File Transfer

Client – Start File Transfer

```

Bitvise xterm - prk44@arsenic.cs.pitt.edu:22

Please choose a file path
>>/afs/cs.pitt.edu/usr0/prk44/private/Sample.txt
filepath: /afs/cs.pitt.edu/usr0/prk44/private/Samp
le.txtFile opened

Sending packet with sequence number 0!
Data read is:This is a file transfer test text fil
e from client to server!!
Received an ACK

ACK with sequence number 1 received!

Sending packet with sequence number 1!
Data read is:This is a file transfer test text fil
e from client to server!!
Received an ACK

ACK with sequence number 2 received!

Sending packet with sequence number 2!
Data read is:This is a file transfer test text fil
e from client to server!!
Received an ACK

ACK with sequence number 3 received!

Sending packet with sequence number 3!
Data read is:This is a file transfer test text fil
e from client to server!!
Received an ACK

ACK with sequence number 4 received!

Sending packet with sequence number 4!
Data read is:This is a file transfer test text fil
e from client to server!!

```

The file path is first given as a Command Line Input. The file is opened, read and sent as packets with an MSS of 64 bytes. Stop and Wait flow control is implemented.

Client – End of File Transfer

```

Bitvise xterm - prk44@arsenic.cs.pitt.edu:22

e from client to server!!
Received an ACK

ACK with sequence number 4 received!

Sending packet with sequence number 4!
Data read is:This is a file transfer test text fil
e from client to server!!
Received an ACK

ACK with sequence number 5 received!

Sending packet with sequence number 5!
Data read is:This is a file transfer test text fil
e from client to server!!
Received an ACK

ACK with sequence number 6 received!

Sending packet with sequence number 6!
Data read is:This is a file transfer test text fil
e from client to server!!
Received an ACK

ACK with sequence number 7 received!

Sending packet with sequence number 7!
Data read is:This is a file transfer test text fil
e from client to server!!Received an ACK

ACK with sequence number 8 received!

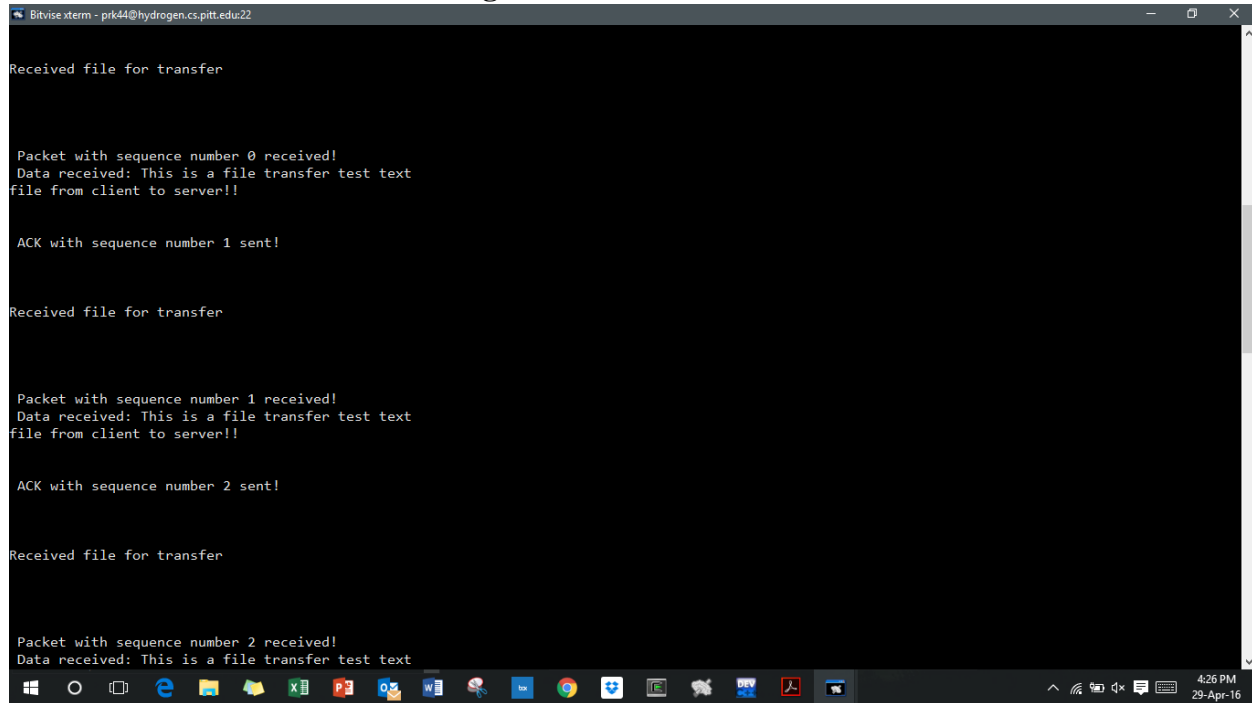
Reached the end of the file!

FILE TRANSFER COMPLETE!!!
^C
-bash-4.1$

```

After the read pointer reaches the end of the text file, it prints the “FILE TRANSFER COMPLETE” message. This does not affect the routers. Routers continue to run normally.

Server – Received File Transfer Segment



```

Received file for transfer

Packet with sequence number 0 received!
Data received: This is a file transfer test text
file from client to server!!

ACK with sequence number 1 sent!

Received file for transfer

Packet with sequence number 1 received!
Data received: This is a file transfer test text
file from client to server!!

ACK with sequence number 2 sent!

Received file for transfer

Packet with sequence number 2 received!
Data received: This is a file transfer test text

```

The server receives the segment, checks the sequence number and sends and ACK with the next sequence number indicating it is waiting for the next segment.

DRAWBACKS OF ROUTING PROTOCOL

1. Listen thread in each router does a lot of processing which could result in dropped packets.
2. ACK not implemented for all segments.
3. No queue management which might result in packet loss.

REFERENCES

- [1] Yan, Melissa. "Dijkstra's algorithm." (2014).
- [2] Dr. T. Znati. "IP address look up" and "CRC16" codes.
- [3] Timestamp function - <http://stackoverflow.com/questions/3756323/getting-the-current-time-in-milliseconds>
- [4] Use of UDP Sockets – abc.se
- [5] Stackoverflow.com used for debugging errors, idea of creation of packets.
- [6] Dijkstra's function code – www.scanftree.com
- [7] Kurose, James F. *Computer Networking: A Top-Down Approach Featuring the Internet*, 3/E. Pearson Education India, 2005.

