CSE 589
MODERN NETWORKING CONCEPTS
PROJECT 2
ROUTING PROTOCOLS

Submitted by
Arvind Srinivass Ramanathan
Person Number: 50205659
UBIT Name: arvindsr
Email: arvindsr@buffalo.edu

SOURCE CODE: arvindsr_proj2.c

IMPLEMENTATION DETAILS

Specifying the topology file and routing update interval as command line arguments start the program. There is a structure server to store the list of servers, and their information. This structure stores the following information:

```
struct server
{
        uint16_t server_id;
        char server_port[200];
        char server_ip[16];
        int cost[10];
        int dest[10];
        //int destcount;
        int nexthop;
        int exists; // To keep track whether a server is alive or not
        uint16_t currentserver_cost;
        int count_to_kill; // To discover that a neighbour server is dead
        int neighbour_or_not; // To keep track whether a server is a
neighbour or not

};
```

exists=1 if a server exists, otherwise exists=0
neighbour_or_not=1 if a particular server is the neighbour of the current server, otherwise it is set to 0.
count_to_kill to keep track of how many update intervals it has been since an update from a particular server has been received.
nexthop to store the next node from the current node in the path to the destination
currentserver_cost to store the current cost to a particular node from this server.
server_id, server_port, server_ip for the current server details.

Initially, we read the topology file and store the details of all the servers along with all the above values.

We identify the current server by traversing the topology file by comparing IP addresses and store its IP, Port and ID.
The cost values initially are all set to infinity. Later on, the costs between two nodes are read from the topology file and updated. Self-costs are 0. These are set in matrix[][] and currentserver_cost.
A UDP socket **sock_fd** is created on the current Port and IP.

The **select** function is used to listen for data. When data is received from neighbours, it's socket receives it, and this data is received in a predefined UDP Message Structure, and it is packed this way on the Sender side.

If select returns 0(check_read==0), it means there is a timeout, and routing updates need to be sent to all neighbours. If the neighbour_or_not variable is 1, it is sent to that neighbour, otherwise it is not. Here we also keep track if a server is dead by keeping a count to kill, and we kill a server if this count reaches 3. This count becomes 0 when we receive a message from that server.

The sending message is sent in a pre-specified **UDP Message Format**, which goes as follows.

LINES : 172 - 220

uint16_t nouf : 2 Byte Number of Update Fields
uint16_t tempport1 : 2 Byte Sender Port
uint32_t bu :4 Byte Sender IP
uint32_t bu1: 4 Byte Server(n) IP
uint16_t tempport: 2 Byte Server(n) Port
uint16_t ttid: 2 Byte Server(n) ID
uint16_t ttcost: 2 Byte Server(n) Cost from current server

All these are converted from Host Byte Order to Network Byte Order using htons() or htonl() and sent to the neighbour IP's and Port's using sendto() function.

The data sent is received by the neighbours, converted back to Host Byte type using ntohs() or ntohl() and stored in the data(matrix and server structure) corresponding to the particular sender server.

Every time we receive data, we call the bellman ford algorithm. This algorithm helps us find minimum distance from the current node to all other nodes given the data just received.

Keyboard input is continuously checked for, for different commands.

STEP
This sends updates to all the server's neighbours by calling construct_packet(Sends distance vector) and following the same sending procedure as before.

PACKETS
Number of packets received is incremented every time a message is received. This is printed when packets input is given on keyboard and then it becomes 0 again, as we require the value from that point again.

DISPLAY
It displays the current routing table in the format Server ID, Next Hop, Cost for each of the servers from the current server.

UPDATE x y z
It updates the costs between two nodes and sends a message to that particular server about it. In this command, many conditions are checked for and then updatelinks() is called. The link between x and y is updated with cost z.

DISABLE x
This command disables a link between current server and x and sets neighbour_or_not to 0, cost to infinity, nexthop to -1. Various checks are made before executing this command.

CRASH
The socket is closed and the program exits.

The data structure for the routing table is a displayed in DISPLAY and is in LINES : 1080 to 1093. The table consists of Server ID, Next Hop and Cost. The three members are defined in struct server : nexthop, currentserver_cost, server_id.