PROJECT 2
ROUTING PROTOCOLS
REPORT

DATA STRUCTURES

Update Message

Source: akannan4_proj2.c

Line numbers: 36-52

```
/* distance vector format */
 struct  distance_vector {
        uint32_t server_ip;
        uint16_t server_port;
        uint16_t padding;
        uint16_t server_id;
        uint16_t cost;
} ;


/* routing packet format */
 struct routing_update_pkt{
        uint16_t num_of_updates;
        uint16_t sender_port;
        uint32_t sender_ip;
        struct distance_vector* updates;
} ;
```

Routing Table

Source: akannan4_proj2**.c**

Line numbers: 22-32

```
/* structure to save info about all servers in the network ( routing table ) */
 struct server{
        uint32_t server_ip;
        uint16_t server_id;
        uint16_t server_port;
        uint16_t cost;

        int is_neighbor;
        int num_of_skips;
        int is_alive;
        int next_hop;
};
```

IMPLEMENTATION

STARTUP

At program start, the topology file is read and information about all servers in the network are processed and stored in the routing table. If a server is a neighbor, the corresponding *is_neighbor* flag is set to 1.

In the routing table, self links are set to 0. If there exists no link to a neighbor, the cost is set to infinity. (**USHRT_MAX**, the maximum value that can be stored in 2 bytes of memory). If a link exists, the cost mentioned in the topology file is set in the routing table.

The program also identifies itself in the topology file to determine its ***server identifier*** in the network and the ***port number*** on which the UDP connection should be setup.

A new UPD socket is created and bound to the port number mentioned in the topology file.

The program uses the ***select function*** to wait for packets from neighbours. The last parameter of the select function is the ***timeout(update interval)*** variable. Whenever a timeout occurs, the distance vector is broadcasted to all neighbors(***send_update_pkt*** function).

When a distance vector is received from a neighbour, we process the update packet and store the new link costs in a adjacency matrix. The ***bellman ford algorithm*** is used to find the minimum distance to other routers/servers.

The variable ***num_of_skips*** denotes the *consecutive number of times* we didn't receive an update packet from the neighbor. The ***num_of_skips*** variable is reset to 0 if we receive a packet from the neighbour and it is incremented by 1 if an update packet is missed. If it is equal to 3, we assume the neighbour is dead and the link to neighbour no longer exists (***is_neighbor*** flag is set 0).

UPDATE COMMAND

This command changes the link cost to neighbors. ***update_link_cost*** function implements the update command.

STEP COMMAND

This command send the distance vector to all neighbours right away. ***send_update_pkt*** function is invoked to implement this command.

PACKETS COMMAND

This command displays how many routing update packets this local server has received since the last “packets” command was invoked.

DISPLAY COMMAND

This command displays the routing table. ***display_routes*** function implements the display command.

DISABLE COMMAND

This command disables the link a particular neighbour. ***disable*** function implements the disable command.

CRASH COMMAND

This command crashes the server/router by ***closing*** the UDP socket.