Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Group 15

Members Name	Student ID
Sankalp Panghal(all coding,implementation and design)	17317444
Gaurav Gupta(help in coding and design)	17321073
Rahul Yadav(help in typing report and design)	17338816
Daniel Kelly(help in design)	14312252

Implementation of Code:

-1- Code Running

➔ The code runs by entering the ***"./start-router***" command,

It automatically uses makefile and **runs 6 bash terminals** for each router

➔ In case of problems, Other way to start the code is to run "***./make"*** command

And then running ***"./router A" "./router B" "./router C" "./router D" "./router E" "./router F" respectively.***

In both cases

To test **node failure or reappearance you have to manually stop the respective node terminal by pressing CTRL+C or start failed router code again**

Also in both cases, it creates a **data-sender executable, which can be used to inject data into converged network**

Every terminal will display DATA packet received which can be used to check the path travelled by this DATA packet

We made sure and verified our code is **working fine to handle and number of router failure and reappearance**, only problem is that we are not reading or writing any file so **its visibility is on terminal screen. Every DV causing a change in routing table makes it print onto the terminal, so you can check the routing table by looking at respective node's terminal** (Sorry about that honestly!)

IMPLEMENTATION

-- Distance Vectors Implementation

Here in our code we made a 2-D array named as selfdv [7][7] of integer type. It is basically a 7x7 matrix used to store the dv's

Over here each router is supposed to save:

- His own DV from every other router.

- His neighbor DV's detail.

Suppose the router B's DV comes ,it stores in such a way that, which are the neighbors (directly connected to B) of the router B it save the distance corresponding to them and which are not the neighbors of B it make the distance as 99.

This all storing occurs in the selfdv[7][7] matrix. Every router has a **selfno.**

(Router A 's **selfno** is 0, B 's **selfno** is 1, C 's **selfno** is 2, D 's **selfno** is 3, E 's **selfno** is 4, F 's **selfno** is 5)

Where [3][4] signifies DV distance from **D to E** for example.

-- Routing Table implementation

Routing Table is implemented using the stuct which is defined below.

```
struct routing_table
{
        char destination ;
        int cost;
        char nexthop ;
};
```

Here the **destination** means that the where the packet to be sent.

cost is basically the distance from source to the destination.

nexthop is the nexthop packet with this destination will be forwarded from source.

-- Functions Implementation

//Code is very well documented, almost every line and paragraph is explained so you can easily read!!

In `main()` , first there is initialization of routers.

using `in()` , we first initialize DV to infinite to every other node in starting. Then by using `chartono(),`

we assign a self number to each router i.e. 0 for A, 1 for B, 2 for C, 3 for D, 4 for E, 5 for F, 6 for G

Using `Initialize_Function()`, we initialize DV for every neighbour.

`socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP)` is a predefined function and AF INET is IPV 4 , DGRAM is DATAGRAM PAROTOCOL and IPPROTO_UDP is UDP PROTOCOL, with this function SOCKET TYPE is DEFINED. Self details are stored in a structure.

using `bind()` ,the socket type defined is bound to the port number.

`set_routing_table()` creates and maintains routing table for neighbors according to above selfDV.

There are three main threads in main() which are passed on to the following functions as following:

`sendDV()`: sends DV to all of its live neighbours, and

`sendDV_5sec( )` sends own DV's to its live neighbor every 5 seconds, using thread.

keep receiving function – `void *keeprecv (void)`, A thread is passed onto this function which keeps receiving data from socket, ready to handle every kind of incoming packet.

if the incoming packet is **CONTROL** then it checks whether to apply Bellman Ford Algorithm or not. **DV's are exchanged in format of CONTROL:[source]:(dist0), (dist1), (dist2), (dist3), (dist4), (dist5)**

where dist0 signifies distance of router "source" from A (0 number represents A), similarly dist5 signifies distance of router "source" from F (5 number represents 5)

If it is a **DATA** packet then it forwards it to the next-hop router according to routing table and according to destination of the **DATA** packet.

If the packet is dead packet then it keeps that router DEAD by setting **dead[flag]**, and restarts its routing table and **selfDV** implementation.

If the incoming packet is ALIVE then it considers that router dead, no more.

➔ keepALIVE function : `void *keepALIVE(void)`

A thread is passed on this function in which if the flag changes from 0 to 1, the router stays ALIVE but if it changes from 1 to 0 the router is declared DEAD (cause no response from that particular router)

`void * dead_router_action ()`

- when packet is dead, it sends sends dead packet to every neighbour declaring which router number is dead, and sets dadflag to 1 for that router, and restarts.

-when packet is alive , router changes dead flag from 1 to 0 and sends ALIVE packet to every neighbor declaring router is alive and restarts itself.

-whenever any router receives any alive packet it changes its dead flag from 1 to 0.

Problems Faced and How we solved:

➔ Detecting Dead Router

We tried to use different time functions from already made C library cause only way to DETECT was to use timer. So every router is supposed to send his self DV to neighbours every 5 seconds and if no response from any neighbor for 15 seconds, that router declares it dead and sends **DEAD(routernumber)** packet to all neighbours so that they can set their **dead[flag]** to 1.

➔ Handling dead router and reappearing router

Our routing table implementation included only
Destination, cost and nexthop
Suppose a router in between detination fails, then before just by setting distance of neighbor who detected it dead to INFINITE (INFINITE is constant 99 in our code) , was creating problems. It was creating problems because the way bellman was implemented.
So we solved it by introducing conept of **DEAD(number) packet** which the neighbor sends to every neighbor after detecting a dead node before restarting itself.
Every router getting DEAD packet restarts after setting corresponding dead flag where number above represents whih router failed.
So this way fresh routng table is implemented every time a node fails or reappears

➔ To segmentation fault

Many times while implementing string, unsual, deeply saddening **segmentation fault** was occurring.
Slowly we figured out that some char arrays are not ending with NULL character (‘\0’) and some contain garbage value causing problems

➔ We had to do handle multi tasks together
So Sankalp learnt and applied multi-threading to various functions to keep everything parallel, abstract and smooth.

➔ 2 thread-functions (`keeprecv and keepALIVE` were using **recvfrom()** function)
This caused problems because both were emptying the receive buffer of the socket individually giving impression to function keepalive (which periodically checks which router is not responding) that some routers are not sending any packets, which in reality they were.
It took me many days to figure out that I can use global char array which only one thread function (keeprecv) will use to empty the buffer and both thread functions can use the same global char array.