3D3 Project 2

Srijan Gupta: 17317499

Caoimhe Mooney: 15315786

Jasmeet Singh: 17317440

Pragya Gupta: 17317460

Implementation

Our implementation consists of three structs and class.

Struct dv_info

The struct dv_info contains all the variables and functions relating to the distance vector protocol information including the router port and name, cost between routers and whether the router is dead.

Struct neighbour_info

The struct neighbour_info contains all the information relating to the neighbouring nodes such as their start time, name and port.

Struct header

The final struct header contains all the information in relation to the headers such as the source and destination.

Class DV

The class DV contains everything pertaining to the distance vector protocol from resetting to updating tables to calculating minimum distance. It also has private variables which take in size, each router's distance vectors, initial distance vectors for when routers reset and neighbours port id's.

Main programme

The program initialises the distance vector for itself and then sets up the UDP socket. It checks to see if the ID is 'H' as suggested in the project brief.

It then checks the neighbouring nodes and if the neighbour is 'A' and if it is it creates a packet to send to 'D' and sends it to all neighbours. It also creates a header. The program then calls fork(), this essentially creates a thread to run, while the thread is running the router will periodically sleep and wake itself up. Fork() creates another branch of the programme that allows the programme to essentially run two threads in parallel. It also listens for incoming data packets on another socket. When it receives a packet if it is the correct packet it forwards it on provided it is not the destination router. If a neighbour is the destination it will only send it to that router and not to all neighbours. In summary, we have two separate threads running through the fork() function, one periodically waking up and the other sending advertisements to neighbours

It checks the type of information coming in. If the information is an advertisement it updates alive time and payload. If the information is wakeup it will wake itself up and multicast to its neighbours each time it wakes up. If it resets it creates a new packet and sends it to its neighbours.

DV functions

DV: takes in the topology file. Initialises all the details and parses the file line by line. It sets the source router, destination router and destination port number. From the topology file it also initialises the neighbours. If the incoming id is not H it prints the initial rioting table.

Reset

If a router goes dead the cost goes to -1 and the router is set to dead. The function then prints reset routing table.

Update table

Checks if the source is dead and if it is sets it to alive. If it's not dead it loads the advertised distance vector and recalculates the distance vector. It sets the cost using the min function. It then moves onto the next router to recalculate its table and prints where the change was detected.

IndexOf, nameOf and portNoOf

All return their corresponding private variable.

initMyaddr

Initialises the address used.

Alive time

Gets the current time of the clock.

Dead timer

Checks to see if the router has been active in the past 10 seconds. If it hasn't been active it declares it as dead.

Min

Uses the Bellman-Ford algorithm to calculate the least distance between nodes and returns the smallest cost.

Print

Prints all relevant information to DV.

Create Packet

Creates an empty packet and a header. Fills the packet with the information from the header.

getHeader and getPayload

Both of these return information about the relevant variables.

Multicast

Send packets to all the nodes neighbours.

wakeSelfUp

Sends a packet to itself to wake itself up periodically.

Difficulties Faced

Initial implementation of the UDP server had problems with binding due to the socket being busy because it was never closing and taking the wrong client address however that was easily fixed. We found very similar solutions to what we wanted to achieve online however none of them achieved the correct routing table solutions. Achieving the correcting routing table solutions took a lot of time but eventually we managed it.

Ideas we Avoided Using

Tiny AODV (Given as a suggestion in project specification).
Characteristics of AODV-:
1) Only affected nodes are informed.
2) AODV reduces the networkwide broadcasts to the extent possible.
3) Whenever routes are not used -> get expired -> Discarded which-:
    Reduces stale routes
    Reduces need for route maintenance
4) AODV discovers routes as and when necessary
5) Does not maintain routes from every node to every other
6) Routes are maintained just as long as necessary
We didn't implement AODV as we didn't see the need for it, especially with such a small no. of routers in our project. We didn't need such a high level sophisticated network, though it would work better in a real-life network.

Routing Tables

Router A		
A	-1	-1
B	10001	3
C	10001	5
D	10001	7
E	10005	1
F	10001	4

Router B		
A	10000	3
B	-1	-1
C	10004	2
D	10004	4
E	10005	2
F	10004	1

Router C		
A	10004	5
B	10004	2
C	-1	-1
D	10003	2
E	10004	4
F	10004	1

Router D		
A	10002	7
B	10002	4
C	10002	2
D	-1	-1
E	10002	6
F	10002	3

Router E		
A	10000	1
B	10001	2
C	10001	4
D	10001	6
E	-1	-1
F	10001	3

Router F		
A	10001	4
B	10001	1
C	10002	1
D	10002	3
E	10001	3
F	-1	-1

Topology from topology file not from assignment specs