David Harris
300069566

COMP306 – Project 2b

Algorithm

My modified version of Dijkstra's algorithm is not really correct. The way that I have implemented the algorithm looks for an update with a lower cost path in the network. If calculates the routing table correctly, but when implementing the functionality to allow links to go up and down, my method cannot be used. I need to keep a memory of the received tables, so that if a cost goes higher this can be taken into account. Thus, my method only works if the cost decreases.

The algorithm I used is shown below.

```
iterate through the given table
    if the address is me, then drop out
        iterate through the routing table
            if the address of the given table is in the routing table
                then we have found the address and drop out

            if the address isn't in the routing table
                create a new table entry because we haven't seen this update before
                    iterate through the routing table
                        if the given table's previous address is the routing table
                            calculate new cost from the updated table and dropout
                    add the new entry into the routing table
                    increment number nodes

            else:
            iterate through the routing table
                calculate the new cost from the updated table
                    if the new cost is less than the original cost
                        update the cost in the routing table to the new cost
                        update the previous address to the new via node
```

Description of Implementation

The task was split up into three main parts. These were to find the hosts on the network that we connected to you, send them your routing table and finally run Dijkstra's shortest path algorithm to complete the least cost routing table.

In order to run Dijkstra's algorithm, the network topology must be know first. For each node, we know how many links we have from the nodeinfo.nlinks field. For my implementation, I have chosen to add two NL_TYPEs, these being NL_HELLO and NL_REPLY. Both types are on kind NL_ROUTE. I have included these cases into the switch statement within the 'up_to_routing' method. When the node is first booted, it sends NL_HELLO packets to all its neighbours. The payload of these packets is empty. This is done by simply running through a for loop from 1 to nlinks, send it it out on each link. This starts from 1 because 0 is the loopback and we already know we are on the network. Also we don't want to add ourselves to the routing table.

I created a method called 'send_ping' which takes in a pointer to a ping packet, the link to send on and the kind of packet. This creates a NL_PACKET, with the NL_PING packet encapsulated into the msg field. This method can be used for sending both hello and reply packets. For hello packets, the NL_TYPE is set to NL_HELLO and the link is 0. If the link is 0 that means send out on all links. For reply packets, NL_TYPE is NL_REPLY, and the link is the same link that the hello was received on. This method is useful because it can handle two scenarios without having to replicate code.

If a hello packet is received by a host, a NL_REPLY packet is sent back with the payload of the message being the receiving node's address, using the send_ping method. This packet is then send down on the link that it came in on. If more time was available, a simple improvement could be made to this. This would be to add the host that sent you the hello packet into your link_table because obviously the host and link is up. This would allow the host to not send out hello packets on links already in the table. The advantage of this would mean less packets in the network and thus a faster set-up time. The only change to the way packets are sent is that hello packets as well as reply packets would have to send the source address in the msg field of the NL_PING.

If a reply packet is received by a host, it will add the destination and nexthop link to the 'link_table'. The source address is retrieved from the payload of the packet and cast as a 'CnetAddr'. The link_table is an array of the hosts that are connected to this particular host. This has a destination and the nexthop link cost for each host connected to it (at this stage).

If more time was available, I would make the storage of the routing table and the link stable dynamic, rather than the static arrays at the moment. Currently they are both set to size 20, but this would need to be changed for a real-world application. This would require methods to resize the array and copy all the data over to the new one. Another option is to use a linked list, being dynamic by its very nature. This would require extra coding that is out of the scope if this project.

We can find out if we have received replies from the hosts connected to us because i have the integer elements which stores the number of elements in the 'link_table'. To tell the other host on the network the information that i have i flood the network with my NL_SEND_PACKET struct. This contains a sequence number and an array of NL_TABLEs. The sequence number is for controlled flooding. Each node maintains an array of NL_SEND_TABLEs that holds the sequence number and the table. When a node receives a routing table update, it check that the sequence number given is equal to or greater than the sequence number we are looking for next. If this condition is not met, the packet is dropped.

When a routing table update is received, there are three possibilities. These are shown below:

1. Startup condition. Here I add all nodes that are connected to me into the routing table.
2. Duplicate condition. Here the entry is already in the table. I check to see if the route is a lower cost than the one i currently have. If so then update table and forward to other hosts. If the cost is greater, then drop the packet.
3. If a link goes down and comes back up, the cost of the link is set to infinity and it is routed around.

Once a routing table update is received, Dijkstra's shortest path algorithm is performed. To be able to perform Dijkstra's, information about the whole network must be know first. Information must be kept about the state of the network so that if a link fails, a route can be found around the link that is down. Dijkstra's involves finding the least cost path between two nodes. Each time through, more links are added to the list and the least cost path is added to another list. Therefore, the least cost is then calculated between the hosts if run multiple times.

Given more time i would've been able to complete the functionality to allow links to go up and down. I implemented Dijkstra's algorithm the best I could, but I was not possible to add the required functionality. I shall describe how I would implement this functionality given more time. When a link goes down, I would get the neighbours to broadcast their new routing, that includes a cost of infinity for the link that has gone down. The other nodes then receive the routing table update and calculate their new routing table which will route around the link. When the link comes back up, the nodes at the end of the link again broadcast their modified routing tables, with the link with the specified cost. This then forces all the other nodes to re-calculate their routing tables.

I have implemented the getNextHop method by using recursion. The base case is that the address we are looking for is a neighbour of ours. If this is the case, then the node we are looking for is in my link_table. Thus I search through the link_table for that neighbour and get the link that that link is on. This link is returned. If the address we are looking for is not a neighbour of ours, then getNextHop() is called again, but now with the previous address from the routing table. This is done by searching through the table for the given destination address and getting the via address. Thus, given a node's address, we can work backwards using the via address until we hit a neighbour. The getNextHop() method is commented out because i am getting an infinite loop, where it keeps calling getNextHop() on itself. This obviously causes the program to crash. This is a very strange problem because it should never be called with me as the destination because it is called within the up_to_routing() method only when the packet is not meant for us. Also the base case of the recursion is when the previous address is a neighbour of ours, so it should break out of the recursion and return the number of the link to send the packet on to get to us. The code is there, and I'm pretty confident that it is mostly correct.

I have implemented the show_table method. When the button is pressed on the cnet user interface, this sends an event which is caught and calls the show_table method. I have implemented the method so that it prints out the table of neighbours and the routing table, formatted so it is legible.

Description of Structs

I have created a new NL_PING type in routing.h. This is the hollow shell of the struct that is encapsulated into a NL_PACKET type. The only field it has is a source address that is needed for replying to ping packets. This source address is used so that i can add the address into my link table and the initial routing table. This is not needed for hello packets because I used them purely to get the host to respond with their address.

I have created an NL_SEND_TABLE struct in routing.h which I use to send out routing tables to other hosts on the network. This has the source address, the sequence number, the number of elements in the NL_SEND_TABLE array and an array of NL_SEND_TABLEs. The source address is used when running Dijkstra's because then we don't have to look up the address that it came from in the routing table. The sequence number is used for controlled flooding. This struct is created from other structs in the NL_REPLY case of the up_to_routing() method, and flooded out to all other hosts on the network.

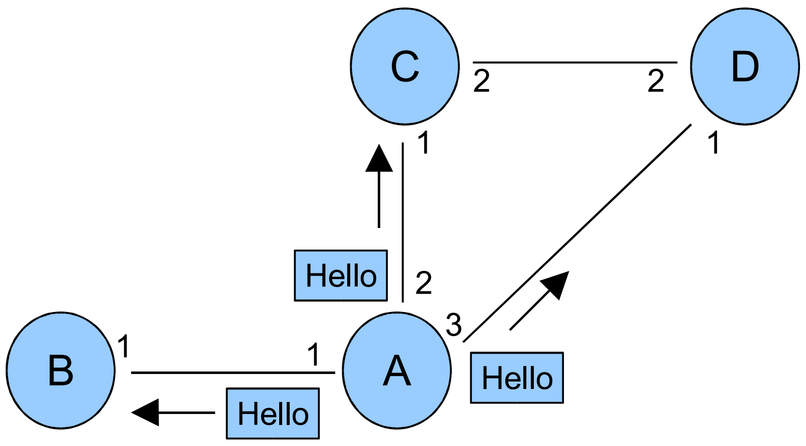
I have not modified the other structs in routing.h.

David Harris   COMP306 – Project 2b
300069566

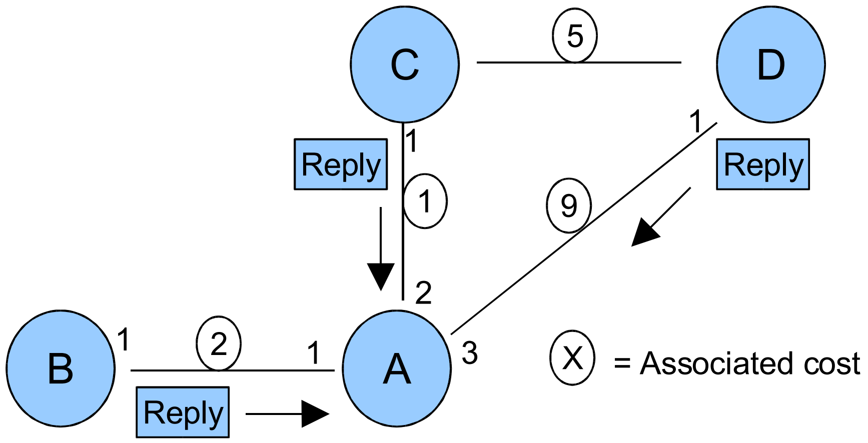
Diagrams of my Algorithm in Practice

Bear in mind that these processes are running concurrently on all nodes, not just node A. I have only shown A for simplicity.

1. Firstly, the node must find out all the other hosts on the network. Starting at A, hello packets are sent out to discover neighbours.



2. Next, the packets are received by the hosts and replys are sent back to A

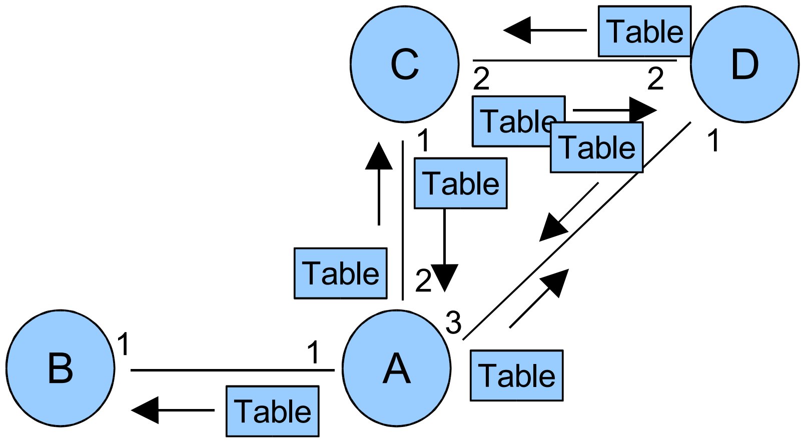


3. A link_table is established at A which has entries for B, C and D, and their destination and nexthop link.

This table is shown below:

Address	Nexthop link
B	1
C	2
D	3

4. Once all the nodes have been found, we flood the network with the entries of the table.



5. Now that all the nodes know the network topology, Dijkstra's shortest path algorithm is run on each node.

6. After Dijkstra's algorithm, the routing table is produced, showing how to get to each node with an associated cost. The routing table for A will be:

Address	Via	Cost
B	B	2
C	C	16
D	D	9

7. Links can go up and down. This has to change the routing table, and so Dijkstra's algorithm is run again on the network. For example if the link between A and C goes down, the routing table will have to change and route around it. This is shown below.

Address	Via	Cost
B	B	2
C	D	1
D	C	6

8. If the link is restored, the routing table must be updated accordingly.

David Harris COMP306 – Project 2b
300069566

Testing

As mentioned above, my implementation does not handle links going up and down. This is because the way that I implemented Dijkstra's algorithm does not allow me to do so. I tried to fix it, but I ran out of time. Below is the tests that I ran. All tests we performed with the standard NewZealand.top topology file, but with some of the link costs changed. The modified topology file is attached.

Test 1: Discovering neighbour nodes and learn their network addresses

This was a simple check to see if the nodes collected the correct data about their neighbours. When the test was run, the neighbours table was found to be correct. Hamilton's neighbour table is shown below:

```
Neighbours Table (with 4 neighbours)
Address Nexthop Cost
--------------------
  0       1       9
  3       2       2
  4       3       2
  5       4       8
```

The test gave the expected results.

Test 1: Calculating the routing table

This was a check to see if Dijkstra's algorithm was correct. The table must list the least cost path between two nodes, giving the correct via node.
The test was run, and the routing table was found to be correct. Hamilton's routing table is shown below:

```
Routing Table - 6
Address  Via    Cost
--------------------
  0       3       7
  3       2       2
  4       2       2
  5       2       8
  1       3       5
  6       3       6
```

The test gave the correct results because it shown that there is a least cost path to New Plymouth running down through Taupo and Wellington to New Plymouth because the cost is 7 rather than 9 running direct to New Plymouth.

There are some anomalies I found during my testing of the routing table. Sometimes it prints the correct routing table as above, but usually it fills in in the costs as the initial value of an integer (-846731578). I have tried to find the source of this error but have failed to do so. Sometimes it didn't correctly compute the correct shortest cost path.