Lab 3 Part 1

1.

In " ALL_LINKS & ~(1<<arrived_on)". The ALL_LINKS refers to every link of the node. The (1<<arrived_on) refers to the link the packet is received from. The "~" means not. Therefore " ALL_LINKS & ~(1<<arrived_on)" refers to all links of a node minus the link the packet is received from.

2.

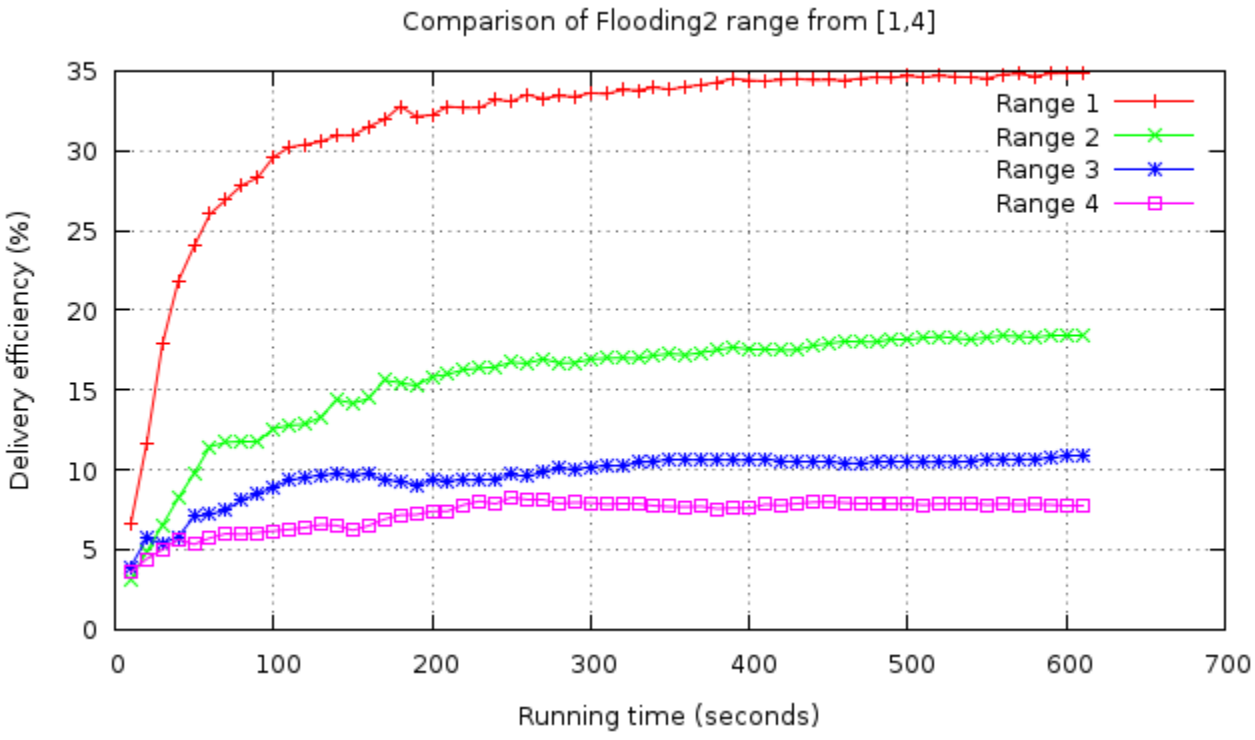
It needs to know the address of the remote node. This is initialized at the creation of the entry with a 0, but afterwards is set using NL_table[NL_table_size].address = address. This is used by the other function of nl_table to return the NL_table[t] address.

It needs to know the ackexpected or the packet sequence number to and from the node. This is initialized at the creation of the entry with a 0, which later on in the flooding2.c is incremented using the method of inc_NL_ackexpected. To determine what the value of it is at any given time it uses the method NL_ackexpected to return the NL_table[t].ackexpected.

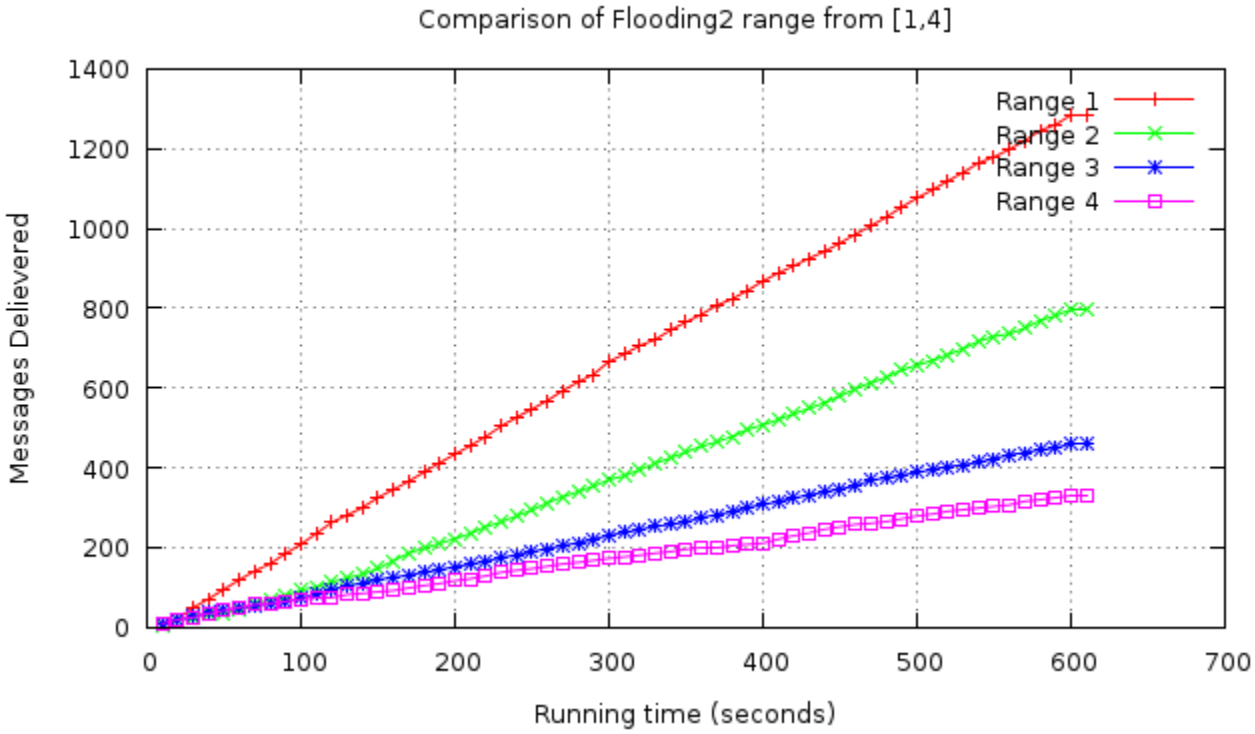
It needs to know the nextpakcettosend, which determines which packet to be sent next. This is initialized at the creation of the entry with a 0, which would increment the NL_table[t].nextpackettosend by 1 in flooding2.c.

It needs to know the packetexpected, which determines determines which packet it need to received. This is initialized at the creation of the entry with a 0, which is then later incremented by inc_NL_packetexpected or used to return seqno using NL_packetexpected.

3.



Based of the graph presented with Flooding2 at different range for MAXHOP [1,4], the efficiency for the lower end range is much higher than the higher ranges. Although that is the cases, this data is quite misleading as efficiency is calculated by total me ages generated by application layer divided by total frames exchanged between physical layer. Therefore with a higher MAXHOP it will have more frames exchanged physical layers as frames are being transfer across more nodes. Therefore with a lower MAXHOP it would be more efficiency than a higher one.



Based of the graph presented with Flooding2 at different range for MAXHOP [1,4], the amount of messages delivered for the lower end range is much higher than the higher ranges. Again this data is misleading this is due mainly to two major reason. One is that when you have more than 1 jump it takes longer for the node to know that this frame has reached the destination. Therefore there exist a delay that gets bigger as the amount of jump required gets bigger. Meaning that having more hops would reduce the amount of message delivered as we don't generate messages for nodes we did not get a response from. The second reason is that with 1 hop, those nodes with many neighbors that are 1 jump away has a less delay, therefore the source node is able to know that the other node has successfully received the frame. Therefore the delay that exist between sending the frame and receiving confirmation is small, allowing for more messages to be sent.

4

a) When the function was commented out, it resulted in a error. This error is created because the application layer does not wait for an ACK due to the fact that the function was responsible for waiting for an ACK. Therefore the sender node continues to sends frames across physical layers, due to too many frames being sent therefore resulting in too many being in queue. It could be the fact that the application layer is also taking in too many packets therefore resulting in the error due to too many in queue. There also the fact that the bandwidth is congested and therefore it becomes bottleneck.

b) When running the program with the unmodified program on a network of 20 nodes for 5 minutes. It did not present any bugs or error message. The program ran successfully. When using the

network with 40 nodes and running for 7 min also did not present any errors. This could be because the message generation rate is too small or the propagation delay is also too small. This would make it that each messages would go through each node too quickly therefore it does not create congestion/bottleneck.

5

a) If the node does not previously know the source address, it then has to update its routing table with this new source (see which link is more efficient and update that value and discard it if it less efficient)and the amount of hops to get to the destination address. If the destination address is unknown then the node will flood all links (minus the source link), if the destination address is known then it will send it on the most efficient link.

b) If the node knows the source address, then if the destination address is known then the node will then send it on the link that is the most efficient. If it does not know the destination address then the node will flood all the links (minus the source link).

c) If the node received the frame from a source it did not know before, then it would update its routing table. On the other hand if the node knows the source address, it would just send the frame on the desired link.

7

a) flooding2.c

Network Size	Probframeloss = 0	Probframeloss = 1
10	10 MIN (2055,1978)	10 MIN (99,19)
	30 MIN (6038,5964)	30 MIN (96, 12)
15	10 MIN (3063,2886)	10 MIN (223, 27)
	30 MIN (9288,9115)	30 MIN (224, 29)

a) lab3.c

Network Size	Probframeloss = 0	Probframeloss = 1
10	10 MIN (268,189)	10 MIN (402,328)
	30 MIN (235, 197)	30 MIN (1036, 965)
15	10 MIN (311,118)	10 MIN (470, 285)
	30 MIN (345,152)	30 MIN (602, 402)

Design Overview:

lab3.c:

- Handling the buffer via a structure that held <Packet destination, Packet, TimerID>
- 3 helper function:
  - updateTable(NL_PACKET packet, CnetAddr dest, CnetTimerID timerID) This updated the table. Either by adding to the table or replacing previous values in certain indexes.
  - getPkt (CnetTimerID timerID) This gets the packet from the structure using timerID
  - stopTimerID (CnetAddr dest) This stops the timer for the destination of the packet
- If the node receives the same packet again from the source after sending the ACK for it (I.E the ACK was not successfully sent back to source). It resends the ACK again
- Added timeout, as well as a new debug button 3 that stores the buffered packets of the node
- Added a EV_PERIODIC which would bring the content of DEBUG0 and DEBUG1 at set intervals

nl_table.c:

- Added a debug method (this just returns to the information from DEBUG0 to be printed in lab3.c)

Project Status:

Completed and works with Frameloss, as well as no-Frameloss. It has all the debugs as well as periodic event set. It can account for up to 32 host. The most difficult part of the assignment was ensuring that I saved each value in the structure properly.

Testing and Result:

To test to see if lab3.c was working properly I used the command “cnet LAB3” using AUSTRIALIA.map. Later on used N25 to test the code.

Testing 7, I used the commands “make test2”. By editing N10, N15 with “wan-probframeloss = 0 and =1”, changing the duration in Makefile as well as for “test2” by changing the file from flooding2.c to lab3.c.

Acknowledgement:

TA from lab
Eclass Hints
http://webdocs.cs.ualberta.ca/~c313/cnet-3.3.1-html/index.html