Lab #3

Objectives:

The objective of this lab was to get more familiar with the flooding protocols in which messages are delivered and routed based on a set of 3 algorithms. The intention was to understand how these algorithms worked and further to understand how to make some modifications to them and integrate stop and wait algorithm in each node to allow packets that have not been delivered to be resent.

Part 1:

Q1.

The flood2 function takes a packet, packet length, and an integer representing links to send the packet out on. The flooding2 sends the packet it received out on all links except the link the packet was received on.

ALL_LINKS is originally defined as a signed int with a value of -1. This means all the bits are initially set to 1.
The memory looks something like the following 11111111 11111111 11111111 11111111

This value is bitwise anded together with ~(1<<arrived_on)

arrived_on is the link number the packet came in on. The 1 is shifted by this number of times left. So now we have something that may look like this if arrived on == 2
(00000000 00000000 00000000 00000001 << 2)
Which is equivalent to:
00000000 00000000 00000000 00000100
We then do a bitwise not and end up with:
11111111 11111111 11111111 11111011

Now this value is bitwise anded together with 11111111 11111111 11111111 11111111 from the ALL_LINKS variable and we get:
11111111 11111111 111111111 11111011

This is the value passed into the flood2 function.
The flood 2 function then iterates through each bit, if the bit is set to 1, then the packet is sent to the link corresponding to the link value in the for loop. otherwise that link is skipped and the algorithm proceeds to the next bit.

On a basic level the statement simply prevents the packet from being sent on the same link it was received as per flooding2.

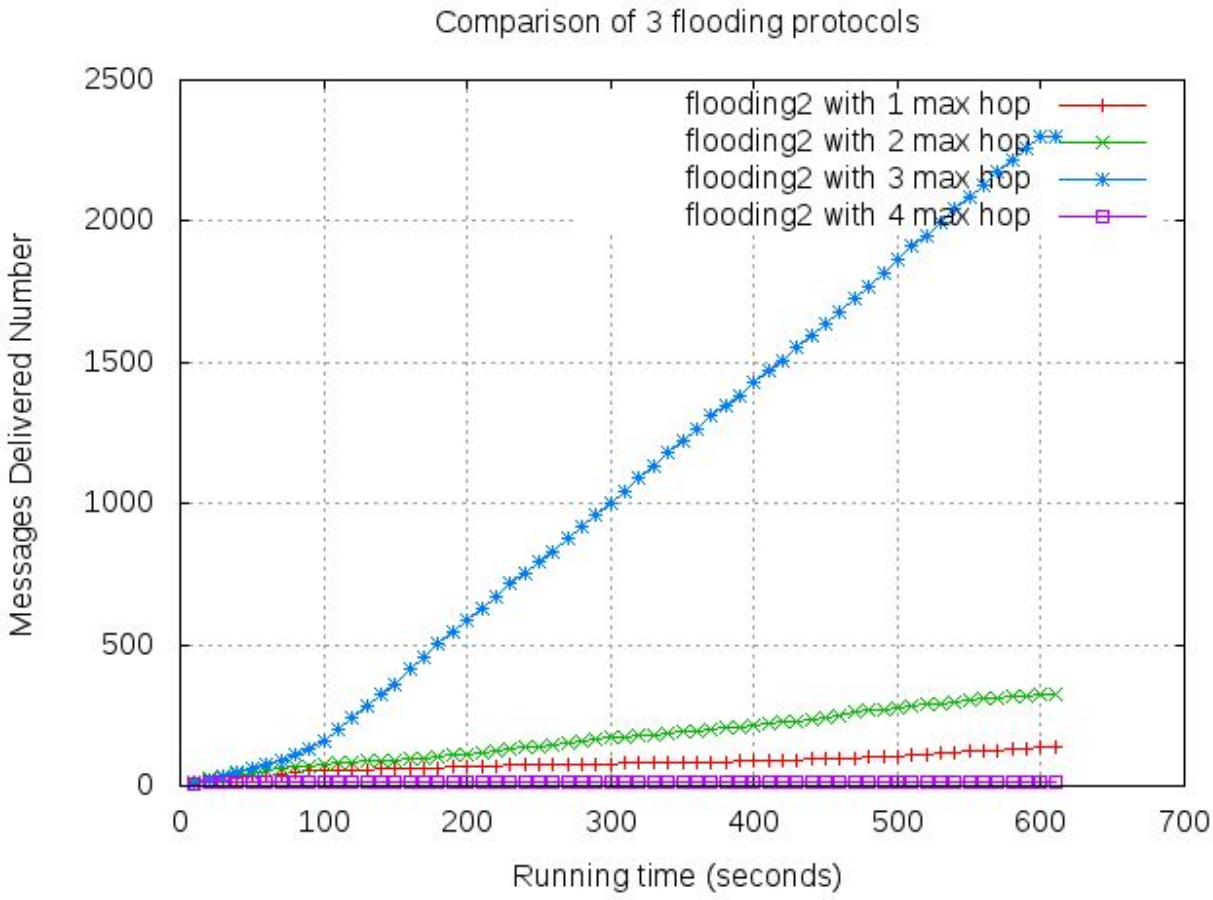
Q2

The standard stop and wait variables are stored in a NLTABLE struct in the nl_table.c file. The initial value of the variables for each node in the struct is zero except for the address of the node which is set appropriately. These values are then incremented appropriately on packet receipt/generation. The values are initialized when a find_address is performed and an existing struct holding values for that source address does not exists. At this point the NLTABLE struct is initialized as described above and placed in an array of other such structs.

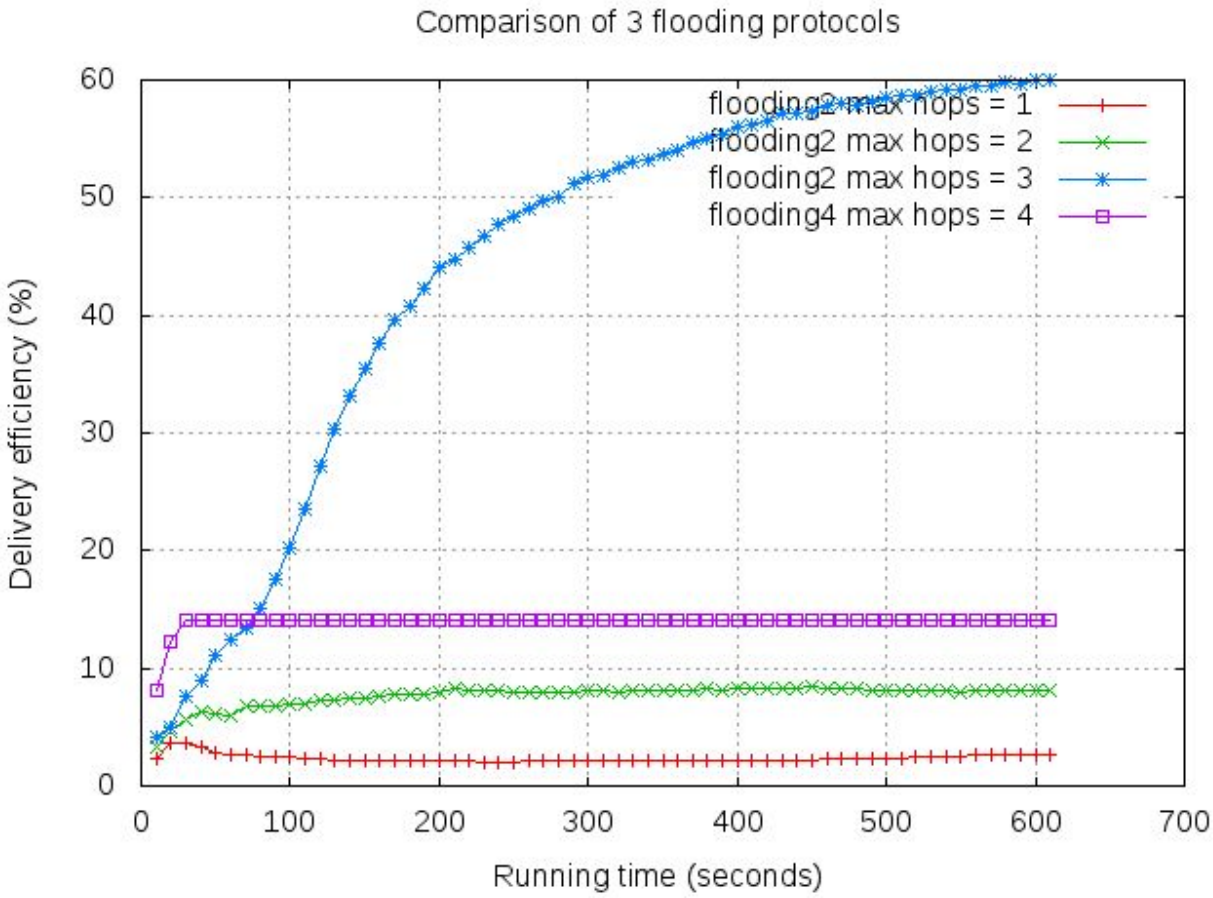
Q3

To gauge performance I have used the messages delivered stat and the efficiency stat. Efficiency is the total size of all application layer generated messages over all physical layer exchanged messages. Messages delivered is a simple statistic showing the number of messages that get to there destination.

As max hops is increased the number of messages delivered increases dramatically as can be seen in the graph below. This makes intuitive sense. If we view each node as a filter a message must pass through, increasing the number of filters a message can hit before being discarded increases the chances the message will get to its final destination.



With respect to delivery efficiency we see a similar trend as the chart below demonstrates. As the number of max hops increases we get an increasing level of efficiency. This makes intuitive sense since we are comparing physically generated messages to message delivered. Since increasing the hop count increases the chance a message gets delivered and has no effect on the number of physical messages generated the ratio of the two values must tend toward higher efficiency (100%).



Q4

a)

When running the 8 node topology with CNET_disable_application commented out an error occurs. The error occurs when CNET_write_phsyical is called. This is because when we comment out CNET_disable_application the nodes stop waiting for acks before sending more packets. This means the network becomes congested and can't properly tolerate the throughput. Therefore when this state happens and an attempt is made to write to the physical layer the function down_to_datalink is too busy to accept the message.

b)

With the 20 node topology the same error did not occur. This is because the nodes still wait for acks to send further messages. This prevents the physical layer from being overwhelmed (being

called when the too_busy flag is set). However, it is still possible that the same error could occur if at any one moment the physical layer is full and cannot accept further packets. This could happen for a number of reasons, for example increased message production rates at each source.

Q5

Answer 2 things for each: Source address seen/nnot seen, and if seen, is it better that what you have already.

In the beginning a node has no routing information. Upon receipt of a packet, it records the number of hops that packete took to get to it, and stores that information in the routing table.

Only replaces if new, or less than the hops it previously took a packet to go from a source to you.

So at first no pathways are known at all, and all packets are flooded. Upon receiving a packet, the number of hops it took to get from that source to you is stored in NL table. If there is a NL_table entry on this source we now know how to send quickly to this node without flooding the entire network. As this information builds up the strategy gets smarter and smarter and uses less flooding2 method and more direct routing. It only replaces information in the NL table if it receives a message from a node already in the NL_table array but that has travelled a shorter distance than it did previously.

a)

If the node has not received a frame from this source address it creates a new entry in its NLTABLE array. This entry holds the number of hops to the source and the source address. Since this is the fastest way we know to get to the source from our location this becomes our default path until we receive a message from that same source again which is discussed in part b.

b)

If we receive a message from a source we have already received from a check is performed. If the minhops to the source address is smaller than what we have stored in our NLTABLE array, we replace the minhops value with the new minhops. This allows the system to slowly evolve as new, shorter paths are discovered from a source to itself.

c)

If we receive a message we check to see the next hop in the NLTABLE entry and use this value to forward the packet.

Part 2:

Design Overview:
I attempted developing a design based on the hints provided on the course website.

- Create a buffer at each node.
- Place the most recently sent message in the buffer.
- Create variables required to administer the resending of data.
- Create a timer so that unacked messages can be resent.
- Upon timeout resend unacked messages.

Project Status:
At first I was unable to understand the requirements of the stop and wait integration. I as also very confused as to how flooding 3 found its best path. Hearing that the algorithm learned was quite confusing. However a helpful TA explained that in the beginning a node send all packets to all other nodes much like flooding2. As it receives packets it essentially creates an accounting system holding the next hop to each node it received packets from.

I was also quite confused with some of the variable setup as I'm probably not as familiar with C as I should be. After declaring the variable in one file I was unable to access it from other files so I had to learn more about the extern declaration. In the end I was unable to get a working system. I think mostly this was due to integrating some of the new information that was presented in the last lab. If I had a few more days I likely could have figured it out.

As of now the program does not work.

Testing and Results:
I tried testing the system based on the suggested method in part 2. When testing I kept having issues with a crashing program. I was unable to perform all tests without a crashing program.

Acknowledgements:
Helpful TAs