CMPT 434
Computer Networks

Assignment Two

Due: Wednesday March 2nd, 6:00pm – late submissions will not be accepted.

Total Marks: 58

Submission Instructions: All assignment submissions must use the Moodle online submission system. Your submission should consist of a single compressed tar file that has been created using "tar -cvzf" on one of the Department's Linux machines. Your tar file should include, for Part A, all of your source files, a makefile, a readme file on how to run your code, and a single documentation file in either plain text or PDF format. Your tar file should also include a separate PDF file with your answers for Part B. With respect to the documentation file for Part A – think carefully about what the marker needs to know to evaluate what you have done. In addition to a description of what you have done, you should carefully describe any limitations that your solution has. If the marker discovers limitations that you have not described in your documentation, the marker will conclude that your testing was insufficient and you will be docked marks accordingly.

PART A

1. (*20 marks*) In this question you will implement the **go-back-n sliding window** protocol discussed in class, in C, for UDP-based communication from a **sender** to a **receiver**. Your sender should take as command line arguments the IP address of the receiver, the port number that the receiver will be using to receive data, the maximum sending window size, and the timeout value (in seconds) that should be used to trigger re-transmissions. The sender reads lines of text from `stdin` (for example, using `getline()`). Each line of text should have a header added to it that gives its sequence number, and added to a queue of messages to be sent to the receiver. For simplicity, assume that through some connection establishment protocol that you don't have to implement, the receiver and sender have agreed on the initial sequence number value that the sender will use (0). Each message should be sent to the receiver in a separate UDP segment, according to the rules of go-back-n sliding window.

   When your receiver gets a message from the sender, if it is the next in-order message (i.e., it has the sequence number following the sequence number of the last correctly received in-order message), your receiver should print it, together with its sequence number, to `stdout`. To simulate possible corruption or loss of the message, the receiver should then prompt for and read a line of text from `stdin`. If a line of text beginning with "Y" is input, your receiver should consider the message to have been correctly received; otherwise, your receiver should consider the message to have been corrupted. In the former case, your receiver should return an acknowledgement to the sender (using UDP). In either case, the receiver then does another receive call.

When your receiver gets a message from the sender, and it is neither the next in-order message nor a retransmission of the last correctly received in-order message, the message should be printed to `stdout` together with its sequence number, and an appropriate label indicating that the message was out-of-order. The receiver then does another receive call.

When your receiver gets a message from the sender, and it is a retransmission of the last correctly received in-order message, the message should be printed to `stdout` together with its sequence number, and an appropriate label. In this case (according to the rules of the go-back-n sliding window protocol), your receiver must return (again) an acknowledgement to the sender. The receiver then does another receive call.

Finally, note that although the above structure should allow testing of different scenarios with respect to patterns of message transmission and loss, it does not cover one aspect that could be an issue in real systems – corruption/loss of acknowledgements. Once you have got your code working according to the above structure, add a command line argument to your receiver that gives the probability of acknowledgement corruption/loss. Then, whenever you need to send an acknowledgement, with the specified probability don't actually send it.

Your code should work on the Department's Linux machines. You must use the proper socket API functions for UDP as described for example in Beej's guide to network programming. Note that with our lab configuration, only port numbers between 30000 and 40000 should be used.

Program defensively, checking for possible error conditions, and include informative comments in your code.

2. (*20 marks*) Modify your receiver and sender from question 1 to implement the **selective repeat sliding window** protocol discussed in class. Another command line argument will need to be added to your receiver, giving the value of r_{max} as defined in the lecture notes. The receiver should prompt for and read a line of text from `stdin` not just when the next in-order message is received, but also when any out-or-order message is received that the receiver should buffer (if received correctly). Buffered messages should be printed out (again), but not with prompting for input of another line of text, when they can be removed from the buffer because of earlier messages finally being received correctly.

PART B

1. (*2 marks*) Give the checksum for 10111010101 using CRC with generator 11011.

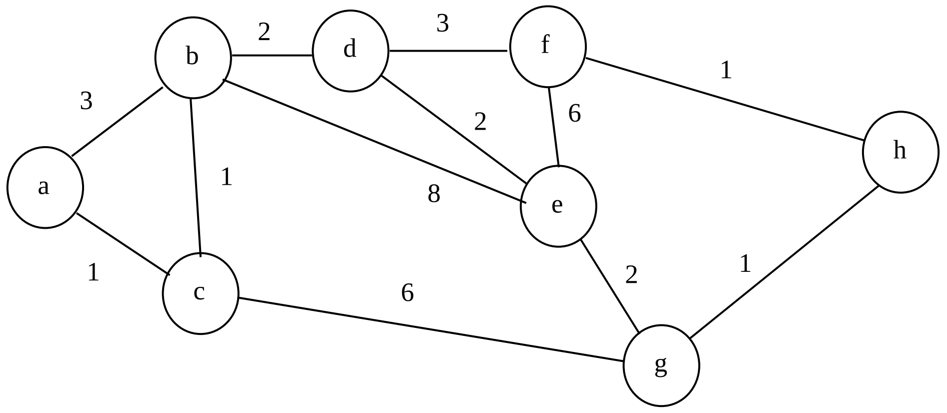
2. (*4 marks*) Give the ratio of the propagation delay τ to the frame transmission time t for each of the following wireless (i.e., use the speed of signal propagation through air)

networking scenarios (round each answer off to 2 significant figures), and state for each which of ALOHA or CSMA would be more appropriate:

(i) ≈ 30 meter distance between nodes, 100 Mbps data rate, 1 Kbit frames

(ii) ≈ 300 meter distance between nodes, 10 Gbps data rate, 4 Kbit frames.

3. Consider the following graph in which the nodes represent routers, the edges represent links, and the labels indicate the link costs.



(a) (*2 marks*) Show the shortest path length estimates after each step when Dijkstra's algorithm is applied to the above network with **node e** as the source.

(b) (*2 marks*) Suppose now that *distance vector* routing is used, and that each minute all nodes *simultaneously* send routing updates to their neighbours. Consider those updates relating to the path lengths to **node e**. Let $D^x(e)$ denote node *x*'s estimate of its shortest path length to node e. Each minute, each node *x* sends $D^x(e)$ to each of its neighbours, and simultaneously receives $D^y(e)$ from each of its neighbour nodes *y*. Node *x* will then update $D^x(e)$ to the minimum over all neighbours *y* of $D^y(e)$ plus the cost of the link between *x* and *y*. Supposing that link e-g fails, give a table showing the evolution of the $D^x(e)$ values for all nodes *x* (except e) until these values stabilize. For the first row of the table, use the shortest path lengths when there are no link failures. Note that because of the assumed synchronous operation, the $D^x(e)$ values in one row will reflect the current link costs together with the $D^y(e)$ values in the **previous** row.

4. Consider the graph of question 3, and suppose that routers c, d, f, and g have local hosts belonging to a multicast group being transmitted to by a host at router g.

(a) (*2 marks*) List the links in the minimum weight Steiner tree connecting all of the receiver routers and the source host router (c, d, f, and g).

(b) (*4 marks*) Suppose that a "core-based" (group-shared) tree approach is used for multicast with router a as the core (rendezvous point) router.

(i) List the links that would be used in the resulting multicast tree.

(ii) List the sequence of links that would be traversed by a packet transmitted to the group by a new source host (but not group member) at router e, assuming use of a **unidirectional** core-based tree.

(c) (*2 marks*) Assuming now a source-specific tree approach to multicast (such as MOSPF), list the links that would be used in the multicast tree for the source host at router g.