CMPUT 313
Lab Assignment 2

Submitted By:
NEEL KUMAR
CCID: neel1

Part 1.

Network	Simulation Results (T_{sim} = …sec)			Analytical Results (using $\overline{Thr}_{with\_errors} = \frac{Thr_{error\_free}}{\bar{N}_r}$)	
	Messages correctly delivered to AL	KBytes correctly delivered to AL	Average throughput (msg/sec)	Estimated $\bar{N}_r$	Estimated $\overline{Thr}_{with\_errors}$ (msg/sec)
W20-TOPa	131	458	131/500 = **0.262**	**1** (no errors)	$Thr_{error\_free} = \frac{1}{\frac{L}{R} + 2 \times propdelay + messagerate} = \frac{1}{\frac{3500bytes}{56000bps} + 2 \times 1.5s + 0.3s} = \mathbf{0.297}$
W20-TOPb	74	525	74/500 = **0.148**	**1** (no errors)	$Thr_{error\_free} = \frac{1}{\frac{L}{R} + 2 \times propdelay + messagerate} = \frac{1}{\frac{7000bytes}{56000bps} + 2 \times 1.5s + 3s} = \mathbf{0.163}$
W20-TOPc	101	177	101/500 = **0.202**	$\bar{N}_r = \frac{1}{p_{success}} = \frac{1}{1 - p_{error}} = \frac{1}{1 - (\frac{1}{2^2})} = \mathbf{1.33}$	$Thr_{error\_free} = \frac{1}{\frac{L}{R} + 2 \times propdelay + messagerate} = \frac{1}{\frac{2500bytes}{56000bps} + 2 \times 1.5s + 0.3s} = 0.298$ $\overline{Thr}_{with\_errors} = \frac{Thr_{error\_free}}{\bar{N}_r} = \frac{0.298}{1.33} = \mathbf{0.224}$
W20-TOPd	61	101	61/500 = **0.122**	$\bar{N}_r = \frac{1}{p_{success}} = \frac{1}{1 - p_{error}} = \frac{1}{1 - (\frac{1}{2^2} + \frac{1}{2^2} - (\frac{1}{2^2} \times \frac{1}{2^2}))} = \mathbf{1.77}$	$Thr_{error\_free} = \frac{1}{\frac{L}{R} + 2 \times propdelay + messagerate} = \frac{1}{\frac{2500bytes}{56000bps} + 2 \times 1.5s + 0.3s} = 0.298$ $\overline{Thr}_{with\_errors} = \frac{Thr_{error\_free}}{\bar{N}_r} = \frac{0.298}{1.77} = \mathbf{0.168}$

CMPUT 313 Lab Assignment 2 – Report

Neel Kumar, CCID: neel1

Part I : A performance Analysis Study

In Part I, four different Cnet topology files were run using the `stopandwait.c` protocol. During a 500sec simulation, the following values were recorded: messages correctly delivered, Kbytes correctly delivered, average throughput.

The simulation results were compared with the corresponding theoretical values for throughput; the results along with calculations are summarized in the table above.

Topology files a and b have a probframeloss of zero, meaning no errors would be generated. This is shown in the simulation, b has a lower throughput because of the larger packet size and larger message rate.

Topology files c and d had identical parameters, except that both nodes in d could generate errors, as opposed to only one host generating errors in node c. As expected, the simulation with two nodes generating errors has a significantly lower throughput.

For all the topology files, the simulation throughputs correlate accordingly with the theoretical results, but the simulation throughputs are significantly lower. There could be a number of factors causing this, a few possibilities: not accounting for the code's delay in runtime, errors in bandwith speeds, delays in generating messages, other delays in application layer, etc.

Part II : Revising the Alternating Bit Protocol Implementation

Design Overview:

- A bi-directional alternating bit protocol for two nodes only
- Connection details for the two nodes are encapsulated in a struct
- The protocol handles errors and corrupt frames
- No "kind" field, instead "seq" and "ack" fields are used

Program Status:

The program works well. Some difficulties that were encountered in the development of the Part 2 protocol were: modifying the transmit_frame function and corresponding calls to accommodate for the newly implemented seq/ack fields, keeping track of the source and destination address, and creating an appropriate struct and initializing it at the correct moment.

Testing:

Multiple simulations were run to test the code. Furthermore, the protocol was modified from transmitting packets of sequence values 0/1, to values >0, showing that the protocol works for seq/ack values greater than zero. The Debug button was also useful in testing to ensure the right ackexpected, frameexpected, and nextframe were in place.

Part III : Reliable Data Transfer in a Path Network

Design Overview:

- A protocol that can reliably handle sending messages on a path network
- A node on the path can send to any other node, and intermediate nodes/routers forward the frames accordingly
- Similar to the previous protocol, timeouts/bad frames are handled, the seq/ack system is used instead of a "kind" field, and connection details are stored in a struct (but this time, due to multiple connections, an array containing conn structs is used)

Program Status:

The protocol was difficult to implement. Firstly, multiple "null" connections had to be stored in an array and initialized in the reboot_node event. After initialization, actual nodes on the path needed to be discovered; a newly discovered node needed to have its information initialized and stored in the conn struct array. Every time a data message is sent or received, the physical layer checks to see if it's aware of the source host's existence.

When a message is initially generated, the node doesn't know which link to send the message on. In this case, the node will randomly choose a link and see if an ack is received. If not, it will randomly send out again. When an ack is received, the node remembers this link, and will use it accordingly for further message transmissions.

Since multiple nodes are used in this path-network protocol, the nodes also need to behave accordingly if they receive a message that does not belong to them. (if the node's address does not match the frame's destination address). In such cases, the node sends the frame to a different link if available. If no other link is available (the node is an edge node), nothing happens, and the host will have to re-transmit in the other direction.

Additional details are pointed out in the code documentation.

Testing:

Multiple simulations were run to test the code. Various topology files were used, starting from a simple 2-node system, to a 7-node system with routers. Again, ascending sequence numbers were used to prove that the protocol can handle seq/ack larger than 0. Furthermore, the debug button was modified to show a table of all connections for that particular node, which was very useful in debugging.