CMPUT313 Lab2 Report

Zhengyao Zhang 1583946

- Objectives

The main objective of this programming assignment is to obtain some extent of knowledge and experience of the API of application layer of cnet, and also to improve the understanding of stop-and-wait protocol itself.

For this assignment, my expectations are having better understanding on the interaction between application layer and transport layer, and also getting some insights and ideas of tree structure network and connection encapsulation within that network.

- Part1

Network	Simulation Results ($T_{sim} = \cdots$ sec.)			Analytical Results (using $\overline{Thr}_{with\_errors} = \frac{Thr_{error\_free}}{\overline{N_r}}$)	
	Messages correctly delivered to AL	KBytes correctly delivered to AL	Average throughput (msg/sec)	Estimated $\overline{N_r}$	Estimated $\overline{Thr}_{with\_errors}$ (msg/sec)
W21-TOPa	107	428	0.1783	1	0.1783
W21-TOPb	66	528	0.11	1	0.11
W21-TOPc	37	335.409	0.0617	1.4459	0.0380
W21-TOPd	16	115.427	0.0267	2.1522	0.0124

For the purpose of obtaining required data and filling the table, I examined the result of running the stop-and-wait protocol with topology file 'W21-TOPa', 'W21-TOPb', 'W21-TOPc' and 'W21-TOPd' respectively.

The simulation results can be easily recorded via the global statistics and the node info from the GUI, while the analytical results requires understanding of the protocol and the info from topology files. To be more specific, the Estimated $\overline{N_r}$ (the average number of times a pkt is transmitted by the sender) can be calculated by the formula $\frac{Frames\_transmitted+Frames\_lost}{Frames\_transmitted}$, and the Estimated $\overline{Thr}_{with\_errors}$ can be calculated by the given formula also.

The results from the simulation and calculation are reasonable and acceptable. The data reveals that the performance of topology files 'W21-TOPa' and 'W21-TOPb' are better than 'W21-TOPc' and 'W21-TOPd', and that is indeed true because the attribute of probframeloss of the first two topology files are 0, while that of the latter two files are 1. Also, the comparison between latter two files indicate the one with only one side probframeloss will have better performance than the one with two side probframeloss.

- Part2

- Design Overview

• The idea of the designed protocol is based on the example stop-and-wait protocol.
• The revision is mainly focused on the encapsulation of connections.

• The related info of specific connection is stored in a defined CONN struct.
• The related info of specific connection will be printed to the interface of each node within the cnet GUI.
• The protocol assumes there will be only one connection for each node (i.e $nodeinfo.nlinks == 1$).

- Program Status

The program is finished and completed at this point, as it follows the instruction and fulfilling the requirements of the lab specification.

The difficulty of this part is relatively lower compared with part3, and the problems I encountered are most about understanding of the specification itself. At first I was thinking there will be multiple connections for each node (i.e $nodeinfo.nlinks \geq 1$) and I was confusing the difference between part2 and part3, as I was thinking both of them require a discovery process for finding their neighbours. After noticing that part2 is just a preparation stage for the purpose of completing part3, I ensure the focus of part2 is on the connection encapsulation, and the implementation is not tricky if understanding example stop-and-wait protocol finely.

- Testing

The testing of this protocol uses given topology file 'W21-TOPa', 'W21-TOPb', 'W21-TOPc' and 'W21-TOPd'. 'W21-TOPa' and 'W21-TOPb' tests the situation which has no packet loss and packet corruption, while W21-TOPc' and 'W21-TOPd' tests the situation which has possibility of losing or corrupting packet.

The protocol compiles and executes successfully on all of the input topology file, no error detected.

- Part3

- Design Overview

• This protocol is a revised version of the protocol from part2.
• The revision of the protocol mainly focus on the concurrency and encapsulation of multiple reliable connections.
• The related info of specific connection is stored in a defined CONN struct.
• The related info of specific connection will be printed to the interface of each node within the cnet GUI.
• Two hosts x and y are connected with a simple path(no routers within the path), or with a certain number of routers rely in the path.
• Only message exchange between hosts, not routers.
• Host will first discover its neighbours, and then establishing connections.

- Program Status

The program is finished at this point as it can finely executed within all scenarios, but there is one little issue left, which is there will be one error received message for each host in GUI. Although it seems have no effect on the overall correctness of execution, I still have no clue why that is happening

The were several difficulties I encountered as I implementing the protocol. First of all, I encountered a phenomenon that the checksum of each frame never matched its stored checksum, and I finally found out it is because I did not initialize checksum of each frame to zero before I writing it into the physical layer, and the uninitialized attribute of frame made the computed checksum being random and incorrect. The second difficulty is the one that I did not figure out a solution for, which is that the error received field for each hots in GUI will always be 1, I think it might be some issues with the application layer enable and disable mechanism but I am not sure yet at this point.

- Testing

The testing of part3 is done with my written topology file 'TREE', I changed the probframeloss and probframecorrupt parameters several times for the purpose of checking my protocol performance under different circumstances.

- Acknowledgements

The design and implementation of the protocols in part2 and part3, and the answers of part1 are all completed individually with no external help.

I do use some resources in Eclass(e.g four input files for part2) and consult to cnet official documentation(e.g checking the definition of APIs).

Part2 and Part3 are based on the idea of stop-and-wait protocol provided on the cnet official website.

References:

Eclass CMPUT313 Website: https://eclass.srv.ualberta.ca/course/view.php?id=67469
Cnet Official Website: https://www.csse.uwa.edu.au/cnet/index.php