Curtin University of Technology

School of Computing

Assignment Cover Sheet

(Please fill in all fields clearly and in upper-case letters)

Surname:	LALLY
Given Names:	BRENDAN
Student Number:	18407220
Unix Username:	
Unit Name:	COMPUTER COMMUNICATIONS
Unit Number:	CNCO2000
Lecturer's Name:	LING LI
Tutor's Name:	TIM PESKETT
Assignment Title/Number:	CC - SLIDING WINDOW ASSIGNMENT
Due Date:	23/05/16
Date Submitted:	23/05/16

I declare the above information to be true, complete and correct.

Except where clearly indicated in this assignment, I hereby declare this assignment is solely my own work and has not been submitted for assessment or feedback purposes in this or any other unit whether at Curtin University of Technology or at any other University/Educational Institution.

I understand there are severe penalties for cheating, collusion and copying and I have read and understood the terms and conditions listed in the Unit Outline for this Unit.

Signed:  Date: 23/05/16

Computer Communications Assignment 2016

'Implementation of the Sliding Window Protocol'

Brendan Lally

18407220

The implementation of the sliding window protocol using cnet was organised into 4 layers: the application layer, the network layer, the datalink layer and the physical layer. All of these layers play a role in delivering a message from one node to another. Each layer has two main functions that deal with the data as it is delivered down from the application layer to the physical layer, and up from the physical layer to the application layer.

Application Layer

The application layer generates a message and a destination address using the inbuilt cnet function, cnet_read_application. Once a message has been generated, it is sent down to the network layer along with the destination and source addresses and the message length. The read_application function determines the message and destination so there isn't much to implement in terms of the application layer.

The application layer also receives packets from the underlying network layer and writes them using the cnet function cnet_write_application.

Network Layer

The purpose of the network layer is to create packets of data from the information generated in the application layer and to determine the next link that these packets need to be sent on in order to reach their destination. The routing is a hugely important part of the protocol.

A static routing table is used to determine which link each node uses to send a frame from one node to another. This table was made using the topology given in the assignment specification (Figure 1).

	Australia	Indonesia	New Zealand	Fiji	Brunei	Malaysia	Singapore
Australia	0	1	2	3	1	1	1
Indonesia	1	0	1	1	2	3	4
New Zealand	1	1	0	1	1	1	1
Fiji	1	1	1	0	1	1	1
Brunei	1	1	1	1	0	1	1
Malaysia	1	1	1	1	1	0	2
Singapore	1	1	1	1	1	2	0

Figure 1. Static Routing Table. 0 represents the link between a node and itself. Values 1 – 4 represent the links that a frame must be sent on to reach each node from every other node.

On the other end, the network layer is also responsible for determining if an incoming packet is at the correct address or needs to be forwarded onwards to the next node in its path. Each packet is sent to a node based on the link found in the routing table. When each frame is sent, the sending node only knows which outgoing link to use in order to send a frame to its next address. It doesn't care if this next address is the destination address or not. It is up to the network layer of this node to determine if the packet is at the correct address or needs to be forwarded to another node. If it finds that the packet is not at the correct address, it looks up the routing table to determine which link to forward on, and sends the packet down to the datalink layer along with the link number to be sent on.

Datalink Layer

The datalink layer is where the packet created in the network layer is encased in a Frame and where the actual sliding window protocol is implemented. The key to the sliding window is that it is able to send multiple frames before receiving acknowledgments. To implement the sliding window, a queue of size equal to the maximum window size (a window size of 5 for the peripheral nodes and 8 for the switch nodes of Australia and Indonesia) was used to store frames that had been sent but not yet acknowledged. Each link from one node to another node had its own queue. To try and improve efficiency, each link also had an overflow buffer which was used to buffer frames that needed to be sent along the link but were unable to fit in the sliding window queue. The datalink layer adds frames to the sliding window queue until that queue is full before adding frames to the overflow buffer.

To stop the nodes from being overwhelmed, when a frame is created, the datalink layer checks if the queue is full and if it is then it prevents other nodes from generating new messages to that particular node.

The datalink layer is also responsible for receiving a frame from the physical layer and passing it up to the network layer. It firstly reads in the Frame before calculating the checksum and comparing it to the checksum value stored in the Frame. If the checksums do not match, the frame is discarded and will eventually timeout and be re-sent. On success, either two things will happen. If the frame is an

acknowledgement, the sequence number is compared to the expected number. On a match, the timer for that frame is stopped and the expected acknowledgment is incremented. The frame that has just been acknowledged is removed from the queue and if there are any frames waiting in the overflow buffer, they are added to the queue and sent. Each nodes queue is at all times, at most, the size of the window. The layer then re-enables the generation of new messages to that node. If the sequence numbers do not match then the frame is ignored. The other thing that can happen is if the incoming frame is a data frame. The sequence number is compared with the expected and if correct, will create an acknowledgment frame that is sent back along the same link. Once again, the frame is ignored if the two sequence numbers do not match and will eventually timeout.

Dealing with Timeouts and Re-transmissions

It was mentioned above that if the datalink layer receives acknowledgment or data frames that are out of order then they are ignored and will eventually timeout. Frames can timeout due to becoming lost and never reaching their destination or if the sending node doesn't receive an acknowledgment before the frames timer is up. If either of these occur then the frame will timeout, causing a timer event. The event is handled with the timeouts function. This implementation of sliding window uses the go-back-n idea where if a timeout occurs then that frame, along with all the subsequent frames that were sent, are re-sent. Once the initial timed-out frame has been re-sent, the timers for all of the subsequent frames are stopped to prevent them also timing out. They are then all re-sent across the same link. One issue here is that on re-transmissions, the expected frame number will not be set to receive the resent frames, causing them to all be ignored and eventually timeout. This causes a chain of timeouts effectively breaking the simulation.

To stop this, we use an assumption that if node A receives an acknowledgement frame from node B for say sequence number 4, then node B must have received frame 3 regardless of if node A explicitly received an acknowledgment. If this assumption is used then every time an acknowledgment is received other than the 'expected' one, all the frames between the expected and the received are assumed to have been successfully received and are then removed from the queue, allowing more frames to be added and sent.

Physical Layer

Similarly to the application layer, the physical layer is mostly implemented by cnet. It receives a frame from the datalink layer via the cnet function CNET_write_physical and CNET_write_physical_reliable. It then passes up a frame to the datalink layer using CNET_read_physical.

Data Structures

The sliding window implementation also required initially setting up data structures to store frames and keep track of important information such as the next frame to send, the expected data frame, the expected acknowledgment frame and timer information. Each of the frame counters were set up as an integer array, using the number of links as indexes for each node. Because every link of every node needed its own queue, an array of queues was set up for each node, indexed similarly to the counters mentioned above. A 2D array was used to store timers, based on a combination of the link and the sequence number.

These structures were set up once for each node at the beginning of the program and were dynamically updated throughout the running of the simulation.

Note: The dynamically allocated memory for each of these data structures is not explicitly free'd in the program as there is no static exit criteria. One option would be to have an Exit event that was handled by reboot_node which called a function that free'd all the allocations.