

Practical time: THURSDAY 2:00pm

COMPUTER COMMUNICATIONS 200

CC200 Assignment Report

CONNOR BEARDSMORE - 15504319

5/1/2016

**Source Code Organization**

The source code is organized to reflect a typical layered network architecture. This architecture includes functionality for the transport, network and data link layer, with the application and physical layers being provided by the CNET simulator. The code is logically organized into the following primary subsections:

*Transport Layer*

The transport layer for this project did not require any major functionality and as such, the functions were essentially wrapper methods. The transport\_down() function reads a message from the application layer and passes it down to the network layer. The transport\_up() function performs the reverse.

*Network Layer*

The network layer was concerned primarily with routing. This section is responsible for encapsulating messages from the transport layer, determining their route and passing them down to the data link layer. Also required in this section was a function to forward an incoming frame that was destined for another node including rerouting this payload.

*Data Link Layer*

The data link layer is where the sliding window protocol is actually implemented. After encapsulation, frames are transmitted down to the physical layer. This section deals with handling sequence numbers, incoming acknowledgements and data frames. Error correction is also performed here utilizing checksums.

*Timeout Functions*

The setting of timers for outgoing frames is handled here. When timeouts occur, the timout\_link() functions handle the re-transmission of lost frames.

*Node Initialization*

When the simulation is started, all nodes initially run the reboot\_node() function. This function sets all relevant event handlers, initializes timers and enables the application layer to start generating messages.

*Miscellaneous Functions*

This segment contains all utility functions that did not belong specifically to any layer. These misc. functions include printing the window contents, incrementing sequence numbers and drawing frames within the CNET simulation itself.

*Header File*

Global variables were permitted to be used for the CNET network simulator and thus, the header file contains declarations for these variables. Structures for frames and variables for the routing table, node windows and node buffers are all contained within the *assignment.h* header file. Also included is function prototypes and pre-processor constants for all useful numbers.

**Design Issues**

Justify why sliding window was chosen over stop-and-wait. Also justify why circular buffer chosen over using a queue or list.

The sliding window implementation developed utilizes the basic Go-Back-N method, as demonstrated in the lecture slides. The window size represented by the global constant MAX\_SEQ can be modified, with all nodes having the same window size for simplicity. Extra traffic will inherently flow through the two switch nodes - Australia and Indonesia - and hence these nodes will be more congested than the remaining nodes. Every node has a separate window and overflow buffer for every link it has.

The sliding window protocol is responsible for storing all sent frames for every link until the receiver acknowledges receival of them. The overflow buffer is only used by nodes in the network that are required to forward frames. It stores frames that are required to be forwarded but cannot be sent at the moment due to a full window. The buffer size was chosen to be the same size as the window, to minimize the amount of unused memory space.

To keep track of each nodes current status, three status indicators were employed. Ackexpected contains the value of the next acknowledgement expected on each link of the node. Both the nextFrameToSend and frameExpected variables work in similar ways for each node.

The design issues faced for sending, receiving and re-transmitting frames are illustrated in the following section.

*Sending Frames*

After the application layer generates a message, the transport layer reads the message via CNET\_read\_application(). The transport layer then sends the message down to the network layer. The network layer encapsulates the message and utilizes the routing table to determine which route to send the message out on. A simple 2D array was used to store the static routing table for direct access to any route. Once the payload reaches the data link layer, it is added to the outgoing window and the window size is incremented by one. It's sequence number is determined using the nextFrameToSend value and the data link layer calls the transmit\_frame() function to send the frame over the physical layer. If the window is full at this stage, the message is added to the overflow buffer and the application layer is disabled for all destinations reached via that link. The overflow buffer is generally reserved for forwarded frames but this frame is the exception. The frame checksum and size is calculated within transmit\_frame() and the frame is written to the physical layer via CNET\_write\_physical().

*Receiving Frames*

*Data Frames*

When a data frame is received, the first check is to ensure that the checksum is correct. This ensures that the frame was not corrupted. If the checksum is incorrect the frame is dropped and the sender will eventually timeout and re-transmit the corrupted frame. The sequence number is then checked to ensure the frame's sequence number is the one expected via the frameExpected variable. If the number is less than expected, the frame is a duplicate. The duplicate frame is dropped and an acknowledgement for this duplicate sent to the sending node. If the sequence number is greater than the expected the frame is also dropped until the expected frame is received.

If the sequence number is equal to the number in frameExpected, the frame sequence number is the one expected and it is processed. An acknowledgement frame is sent and the frame is passed up to the network layer. Two situations can now occur. Either the frame is destined for the node it arrived on or it is destined elsewhere. If the frame is destined for the current node, the payload is unpacked and handed up to the transport layer. The transport layer will then write the message to the application layer and the message has successfully been received. If the frame is destined for another node, it must be forwarded on to its correct destination. The network layer reroutes the frame and passes it back down to the data link layer to send outwards again.

*Acknowledgement Frames*

When an acknowledgement frame arrives, the data link layer confirms that the sequence number is between the ackExpected value and the nextFrameToSend value. If this holds, all frames less than the acknowledgements sequence number and greater than ackExpected are implicitly received. Thus, all timers up to the acknowledgement received are stopped and the number of frames in the window is decreased accordingly. If the buffer is not empty, items are added to the window from the buffer and transmitted. The application layer is also re-enabled if the buffer is now empty. add little more here

*Frame Re-transmission*

If a timer timeout occurs before the appropriate acknowledgement frame is received, either the frame or the relevant acknowledgement frame failed to arrive. Thus, the frame is resent and the appropriate timer reset. The sequence number of the frame on re-transmission is left unchanged. This process will continue until the acknowledgement is successfully received for the frame and the sender is sure that the frame successfully arrived.

**Error Conditions**

There are numerous error conditions that the implemented protocol can successfully handle.

checksums

dupicate frames

**References**

Ling, Li. *"CNET Tips and Discussions."* Class lecture, Computer Communications from Curtin University, Perth, Australia, April 14 2016.

Ling, Li. *"Data Link Layer."* Class lecture, Computer Communications from Curtin University, Perth, Australia, March 7 2016.

McDonald, C. "The Cnet Network Simulator (v3.3.3)." The Cnet Network Simulator (v3.3.3). Accessed May 11, 2016.

http://www.csse.uwa.edu.au/cnet/.

Tanenbaum, Andrew S. *Computer Networks.* Upper Saddle River, NJ: Prentice Hall PTR, 1996.