Curtin University

5/1/2016

Computer Communications Assignment Report

CONNOR BEARDSMORE - 15504319

Practical time: THURSDAY 2:00pm

COMPUTER COMMUNICATIONS 200

Source Code Organization

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

Transport Layer

The transport layer for this project does not require any major functionality and as such, the associated 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 primary role of the network layer is routing. Hence, this section is responsible for encapsulating messages from the transport layer, determining their route and passing them down to the data link layer via *network_down().* Also included in this section is the *forward_frame()* function to forward an incoming frame that was destined for another node including rerouting the payload.

Data Link Layer

The data link layer is where the sliding window protocol is essentially implemented. After encapsulation, frames are transmitted down to the physical layer in the *datalink_down()* function. This section deals with handling sequence numbers, incoming acknowledgements and incoming data frames. Error correction is also performed here utilizing checksum functionality provided by the CNET API.

Timeout Functions

The setting of timers for outgoing frames is handled within this subsection via *set_timer().* When timeouts occur, the *timout_link()* functions handle the re-transmission of lost or corrupted 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 to all other nodes.

Miscellaneous Functions

This segment contains all utility functions that do not belong specifically to any layer. These miscellaneous functions include printing the window contents, incrementing sequence numbers and drawing frames within the CNET simulation itself. The *draw_frame()* function and it's role is described within the CNET documentation included (McDonald 2016).

Header File

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

Design Issues

The stop-and-wait flow control protocol implemented within the data link layer is inefficient if propagation time is greater than the time to send a frame. Efficiency can be improved by allowing multiple frames to propagate along the line simultaneously (Tanenbaum 1996). This ideology is the basis of the sliding window protocol.

The sliding window implementation developed utilizes the simple Go-Back-N method, as demonstrated in the lecture slides referenced. The window size represented by the global constant MAX_SEQ can be modified as required, with all nodes having the same window size, primarily for simplicity. Extra traffic will inherently flow through the two switch nodes - Australia and Indonesia - and hence these nodes will typically 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 generally only used by nodes in the network that are required to forward frames. The buffer stores frames that are required to be forwarded but cannot be sent currently 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. An array implemented as a circular buffer was employed for the window for a simplicity, but a queue data structure could also be employed.

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, specifying the sequence numbers of the next outgoing and incoming frames respectively.

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 payload and utilizes the routing table to determine which link to send a frame on to reach the desired destination. 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 first frame is an 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 frame for this duplicate sent to the sending node. If the sequence number is greater than the expected sequence number, 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 via *CNET_write_application()* 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 using the *forward_frame()* function 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.

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. As a result, the frame is resent and the appropriate timer is 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 can confirm that the frame successfully arrived. An alternative method involves sending a *reject* acknowledgement upon determining a frame error. The decision to not utilize this approach was made to maximize system throughput and for pure simplicity.

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.