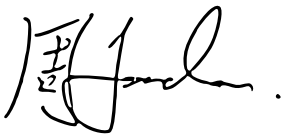
Computer Communications CNCO2000
C-NET Assignment

Jordan Pinglin Chou
18348691

May 2016

I declare that this is my work.



1 Source Code Organisation

1.0.1 sliding_window.c

The source code in this file is organised into functions that simulate a protocol stack. At the top of the file is where you will find the **draw_frame** function. This is a function written by Chris McDonald that allows you to look at packets within a link. After that you will find that the source code is organised in a way that shows the logical order when you go down a protocol stack, beginning with the compulsory **reboot_node** function.

```
1 void application_down_to_network(CnetEvent ev, CnetTimerID timer, CnetData data)
2 void network_down_to_datalink(Frame frame)
3 void datalink_down_to_physical_transmit(Frame frame, int link)
4 void datalink_down_to_physical_forward(Frame frame, int link)
5 void datalink_down_to_physical_ack(int dest, int link, int sequence)
```

After the "going down" functions, the functions then "go up" the protocol stack, imitating what flow of execution of when a message is received.

```
1 void physical_up_to_datalink(CnetEvent ev, CnetTimerID timer, CnetData data)
2 void datalink_up_to_network(Frame frame, int link, int length)
3 void datalink_up_to_network_ack(Frame frame, int link)
4 void network_up_to_application(Frame frame, int link)
```

After the protocol functions we can find the timer functions. A separate function is given for each timer on a link (maximum of 4 links in our topology).

```
1 void set_timer(Frame frame, int link, int sequence)
2 void timeout_link_1(CnetEvent ev, CnetTimerID timer, CnetData data)
3 void timeout_link_2(CnetEvent ev, CnetTimerID timer, CnetData data)
4 void timeout_link_3(CnetEvent ev, CnetTimerID timer, CnetData data)
5 void timeout_link_4(CnetEvent ev, CnetTimerID timer, CnetData data)
```

1.0.2 sliding_window.h

This file defines the structs and prototypes of functions used in **sliding_window.c**. This is also where the definition and initialisation of the static global variables (global to each node) are located. Also located in this file are the pre-processor constants and macros used in the assignment. The decision to place the variables in the header file was made in order to increase the readability and cleanness of the code.

1.0.3 ASSIGNMENT

The topology file for the assignment. It defines which nodes are linked to which as well as global attributes for all nodes. Contains the seven nodes declared as hosts.

1.0.4 Log Files & Miscellaneous

Stored in `~/CC200/assignment/` are nine more files:

1. **Australia.log**: Output log file for Australia node
2. **Fiji.log**: Output log file for Fiji node
3. **NewZealand.log**: Output log file for NewZealand node
4. **Indonesia.log**: Output log file for Indonesia node
5. **Singapore.log**: Output log file for Singapore node
6. **Malaysia.log**: Output log file for Malaysia node
7. **Brunei.log**: Output log file for Brunei node
8. **README**: Basic documentation of the files and how to run and comile the application

2 Design Issues/Implementation

NOTE: The implementation of sliding window was done using Go-Back-N, with a window size of 3

2.1 Sending

2.1.1 DATA

A message is generated by the application layer of a host. The message is then encapsulated into the appropriate data structure. In the simulated network layer the link to send the message on is determined from the routing table (constructed from the links and positions of the hosts). The checksum is calculated using **CNET_ccitt** and placed into the message and the message is then passed onto the data-link layer and placed into the buffer for the corresponding link. If the buffer is full then the generation of messages will be temporarily halted until the buffer decreases in size (either from acknowledgements or timeouts). The data-link layer will then "write" the message to the physical layer using **CNET_write_physical()**.

2.1.2 ACKNOWLEDGEMENTS

When a message has been received (and is not out of sequence or corrupt) an acknowledgement is sent. The **datalink_down_to_physical_ack** function is called with the destination, source, link and sequence of the message that arrived.Given those arguments the function then creates a message with the type ACK. The checksum of the message is then set using the **CNET_ccitt()** function. The message is then written to the physical layer for transmission using **CNET_write_physical()**.

2.2 Receiving

2.2.1 DATA

When a message is read from the physical layer (using **CNET_read_physical()**) it is then passed onto the data-link layer. In the data-link layer the checksum of the message is then checked to see if the transmission occurred properly. If the checksums do not match up then the message is discarded, else the sequence number of the message is checked. If the sequence number of the message has already been received the message is dropped. If the sequence number is out of order then the message is also dropped. If the sequence number is fine then the message is passed onto the network layer. At the network layer a check is done to make sure that the message has arrived at the correct destination. If the destination of the message is not for the current host then the message is forwarded onto the correct link.

2.2.2 ACKNOWLEDGEMENTS

When a message is read from the physical layer with a type of ACK it means that it is an acknowledgement. It is first passed to the datalink layer and the timer for the message is stopped using the **CNET_stop_timer()** function then checksum of the message is checked, if the message transmitted properly then sequence numbers are checked against the already transmitted data messages. If the sequence number is greater than than the previously acknowledged message this implicitly acknowledges the previously sent messages and the window is adjusted accordingly. If the node has any messages in the store-forward buffer then these are forwarded on.

2.3 Re-transmitting

2.3.1 Corrupted Frames

After a message has been received and is passed onto the data-link layer (using **physical_up_to_datalink** and **datalink_up_to_network** the checksum of the message is first checked. If the checksum does not match up then the message is discarded. In this case an acknowledgement is not sent. The timer at the sending node for the message should timeout and the message should be retransmitted. The messages that are retransmitted depend on the received ACK's the sending node has previously received. It will re-transmit messages after the last ACK'd message up until the window size is full.

2.3.2 Forwarding

When a message is passed from the data-link layer to the network layer the destination of the message is checked. If the destination of the message is not the current node then the message is given back the the data-link layer to forward to the correct node. When the message is given to the data-link layer the link given is obtained from the routing table to ensure that the message will arrive at the correct node. If the window is currently full for the forwarding node then the message is stored in a store-forward buffer until the window shrinks. As soon as the window shrinks (through an ACK received) the message will be transmitted from the buffer.

2.3.3 Out of order sequence numbers

In the event where a message is corrupted or lost, subsequent messages to the same link/node will be received out of order. If a message received does not have the correct sequence number then it is ignored and the expected sequence number is printed out. The message's timer will expire and a timeout event will occur resulting in the message being resent, hopefully in the correct order.

References

[1] David J. Wetherall Andrew S. Tanenbaum. *Computer Networks*. 5th ed. Pearson, 2011, pp. 226–239. ISBN: 9780132126953.

[2] A. Prof Ling Li. *Assignment Tips Slides*. Curtin University, 2016.

[3] A. Prof Ling Li. *Module 3: Data Link Layer*. Curtin University, 2016.

[4] Chris McDonald. *The cnet network simulator (v3.3.1)*. URL: `http://www.csse.uwa.edu.au/cnet/`.