CSE 123 – Project 1 Design Document

Daniel Byun (A10800886)
s4byun@ucsd.edu

Frame

Each frame (64 bytes) consists of:

1. **char receiver_addr** - Receiver address (2 bytes)
2. **char sender_addr** - Sender address (2 bytes)
3. **char data** - Payload (58 bytes)
4. **unsigned char seqnum** - Sequence number (1 byte)
5. **char crc** - CRC (1 byte)

Sender

Each sender contains:

1. **int send_id** - Sender ID
2. **int seqnum** - Sequence number
3. **int LFS** - LFS
4. **int LAR** - LAR
5. **LLnode * send_q_head –** Buffer for un-ACKed sent frames

Every sent frame has its own timeout value, so both the frame and its timeout are first stored in **struct sendQ_slot**, and then that struct is appended in the sent buffer (linked list, the head of which is tracked by **send_q_head**).

General flow:

1. Sender checks if there are any frames in the sent buffer that are waiting to be sent out, finds the one that times out the soonest, then sends it.

2. Sender checks if there are any incoming ACKs. When an ACK is received from the receiver, the sender first makes sure that the received ACK is LAR + 1. If true, the frame that corresponds to the ACK is removed from the sent buffer. Otherwise, nothing happens to the buffer, meaning that frame will be sent out again.

3. Sender takes more input from command line to send to receiver, and if the message is longer than 57 bytes (58th byte is reserved for null character), then the message is split up into smaller size messages, such that no message is greater than 57 bytes. As long as there is space within the window, which is checked with the length of sent buffer, sender will continue to queue these messages via **outgoing_frames_head_ptr**. If the window is full, sender simply waits until some in-order ACKs are received to pop **sendQ_slots** off the sent buffer via **send_q_head**.

4. Sender goes through the sent buffer to make sure there are no frames whose timeout field is either not initialized or already expired. Sender then updates such frames' timeout fields, and gets them ready to be sent out.

5. Sender sends all the frames queued via **outgoing_frames_head_ptr**.

**Sender does not buffer out-of-order ACKs. It only deals with cumulative ACKs.
Extra functions in sender:

1. **void ll_split_head(LLnode **, int)** – Splits outgoing messages that are too long to fit in one frame.
2. **int is_next_ack(Sender *, int)** – Checks if the int parameter corresponds to the next expected ACK.
3. **int send_q_size(Sender *)** – Counts how many **sendQ_slots** are in the sent buffer.

<u>**Receiver**</u>

Each receiver contains:

1. **int recv_id** – Receiver ID
2. **int NFE** – NFE
3. **struct Frame* recv_q[8]** – Buffer for frames received out of order.

General flow:

1. Receiver receives a frame if there is space in the buffer.
2. Receiver checks if the frame's sequence number is within the receiving window. If not, receiver simply throws away the frame and sends an ACK to the sender to let it know that this frame was already received long time ago.
3. Receiver checks if the frame's sequence number matches the NFE. If not, it buffers the message. Go to step 1.
4. If the frame's sequence number matches the NFE, receiver receives the message, and updates the NFE. Go to step 5.
5. Receiver iterates through the buffer to see if there is frame whose sequence number matches the next NFE. If true, pop that frame off the buffer, receive it, and update the NFE. Repeat step 5 until this is false.

Extra functions in receiver:

1. **int is_valid(Receiver *, int)** – Checks if the int parameter is within the current receiving window
2. **int recv_q_size(Receiver *)** – Counts how many frames are currently in the receiver buffer
3. **int is_frame_in_buffer(Receiver *, int)** – Returns 1 if frame with sequence number **int** is in the buffer. Returns 0 otherwise.