Steps For Using CRC

• Sender
  • When constructing the frame (header and payload), set CRC field to zero.
  • Convert frame to char array by calling convert_frame_to_char()
  • Append CRC into your frame by calling append_crc(). Append_crc() will call crc8() to compute the CRC remainder.
  • Send the resulting char array from frame + the crc remainder by appending to outgoing_frames_head_ptr.

• Receiver
  • Check for corruption when you receive a packet by calling is_corrupted() function.
  • If not corrupted, convert the char array to the frame. Otherwise, drop it.
Bitwise Operators in C

• X AND Y  -> X & Y
• X OR Y  -> X | Y
• X XOR Y  -> X ^ Y
• NOT X  -> ~X
• Shift X by Y bits to the left  -> X << Y
• Shift X by Y bits to the right  -> X >> Y

• Note: Bitwise operators have lower priority than comparison operators.
  • Exp. If we want to check whether the most significant bit of a byte is 0 or not.
  • If (byte & 0x80 == 0)
CRC 8 Computation

// Function returns the remainder from a CRC calculation on a char* array of length byte_len
char crc8(char* array, int array_len){
    // The most significant bit of the polynomial can be discarded in the computation, because:
    // (1) it is always 1
    // (2) it aligns with the next '1' of the dividend; the XOR result for this bit is always 0
    char poly =0x07; //00000111
    char crc = array[0];
    int i, j;
    for(i = 1; i < array_len; i++){
        char next_byte = ith byte of array;
        for(j = 7; j >= 0; j--){ // Start at most significant bit of next byte and work our way down
            if(crc’s most significant bit is 0){
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j); // get_bit(next_byte, j) returns the a bit in position j from next_byte
            } else{ // crc’s most significant bit is 1
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j);
            }
        }
        crc = crc XOR poly;
    }
    return crc;
}
Sender Buffer/Window

• Sender need to maintain window(buffer) while sending packets out
• The window is like this:
  • struct sendQ_slot {
    struct timeval* timeout;  // event associate with send timeout
    Frame frame;
  } sendQ[SWS];

• Timeout is of type struct timeval (declared in sys/time.h)
Receiver Buffer/Window

• Similarly, it is better for receiver to maintain a window too.
• Example:
  
  ```
  struct recvQ_slot {
    struct Frame_t* frame
  } recvQ[RWS]
  ```

• Why don’t we need a timeout here?
struct timeval in C

- struct timeval {
  time_t tv_sec; //seconds
  suseconds_t tv_usec; //microseconds
}

- To calculate the timeout, get the current time and add 0.1s to it.
  void calculate_timeout(struct timeval* timeout) {
    gettimeofday(timeout, NULL);
    timeout->tv_usecs += 100000;
    if (timeout->tv_usecs >= 1000000) {
      timeout->tv_usecs -= 1000000;
      timeout->tv_sec += 1;
    }
  }

- Take a look at time_val_usecdiff() in util.c
Fragmentation

• One way to implement fragmentation is to use the sender’s input command list.

• sender->input_cmd_list_head is a doubly linked list whose nodes are of type struct cmd (in common.h)

• Before popping off the head node from sender->input_cmd_list_head check if the message > your_payload_size.

• If yes, then split the head node into multiple nodes where each node contains part of the message, length <= your_payload_size

• You can also maintain your own data structure in sender_t to handle this.
Skeleton Code for Fragmentation

```c
void ll_split_head_if_necessary(LLnode **head_ptr, size_t cut_size) {
    //TODO: check if head is NULL
    LLnode* head = *head_ptr;
    Cmd* head_cmd = (Cmd*) head -> value;
    char* msg = head_cmd -> message;
    if(strlen(msg) < cut_size) {
        return;
    }
    else{
        size_t i;
        LLnode* curr, *next;
        Cmd* next_cmd;
        curr = head;
        for(i = cut_size; i < strlen(msg); i += cut_size) {
            //TODO: malloc next, next_cmd
            char* cmd_msg = (char*) malloc((cut_size + 1) * sizeof(char)); // One extra byte for NULL character
            memset(cmd_msg, 0, (cut_size + 1) * sizeof(char));
            strncpy(cmd_msg, msg + i, cut_size);
            //TODO: fill the next_cmd
            //TODO: fill the next_node and add it to the linked list
        }
        msg[cut_size] = '\0';
    }
}
```