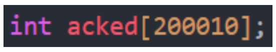
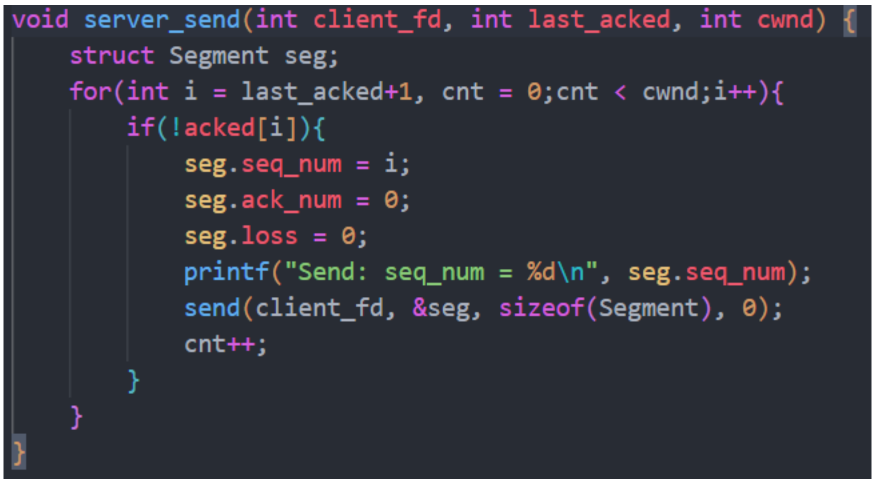
1. How does the server send the packet with the correct sequence number to client?



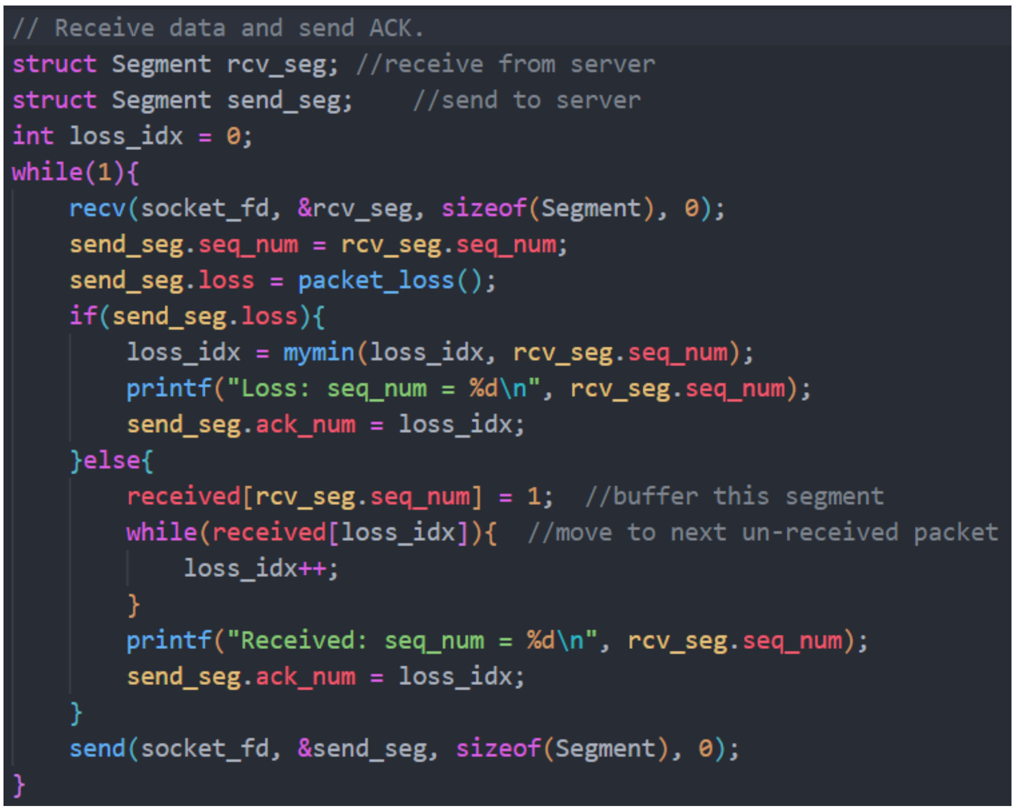
```
int acked[200010];
```



```
void server_send(int client_fd, int last_acked, int cwnd) {
    struct Segment seg;
    for(int i = last_acked+1, cnt = 0;cnt < cwnd;i++){
        if(!acked[i]){
            seg.seq_num = i;
            seg.ack_num = 0;
            seg.loss = 0;
            printf("Send: seq_num = %d\n", seg.seq_num);
            send(client_fd, &seg, sizeof(Segment), 0);
            cnt++;
        }
    }
}
```

維護一個 acked 陣列，當對方有正確收到 packet with seq number i 時，就將 acked[i]標記為 1。從 last acked packet 的下一個 packet 開始傳送，總共需要傳送 cwnd 個 packet，每送出一個 packet 時就讓 counter+1。在送的時候需注意編號 i 的 packet 有沒有被 acked 過，如果有 acked 過就代表對方有正確收到，只需傳送尚未被 acked 的 packet 就好。

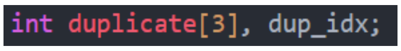
2. How to simulate packet loss?



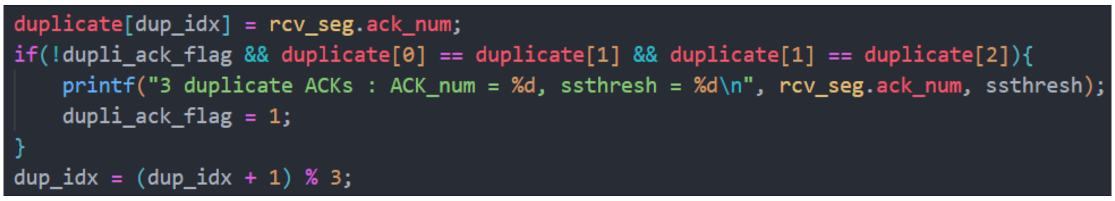
```
// Receive data and send ACK.
struct Segment rcv_seg; //receive from server
struct Segment send_seg;    //send to server
int loss_idx = 0;
while(1){
    recv(socket_fd, &rcv_seg, sizeof(Segment), 0);
    send_seg.seq_num = rcv_seg.seq_num;
    send_seg.loss = packet_loss();
    if(send_seg.loss){
        loss_idx = mymin(loss_idx, rcv_seg.seq_num);
        printf("Loss: seq_num = %d\n", rcv_seg.seq_num);
        send_seg.ack_num = loss_idx;
    }else{
        received[rcv_seg.seq_num] = 1;  //buffer this segment
        while(received[loss_idx]){  //move to next un-received packet
            loss_idx++;
        }
        printf("Received: seq_num = %d\n", rcv_seg.seq_num);
        send_seg.ack_num = loss_idx;
    }
    send(socket_fd, &send_seg, sizeof(Segment), 0);
}
```

當 client.c (segment 接收端)接收來自 server.c 的 packet 時，會使用 header.c 定義的 packet_loss()來決定這個 packet 會不會 loss，之後就根據 loss 的值去進行不同的操作，待 send_seg 的值都設定完成後，就會將 send_seg 傳給 server.c。

3. How to detect 3-duplicate ACKs?



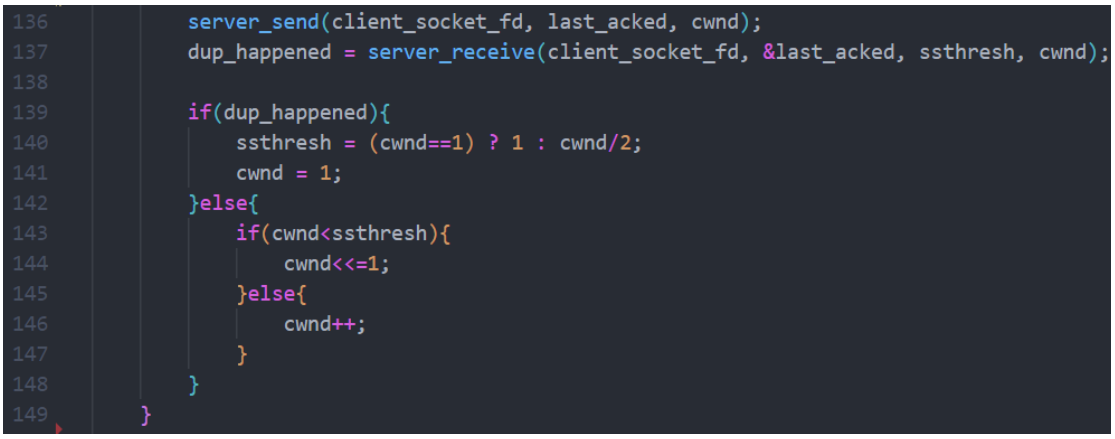
```
int duplicate[3], dup_idx;
```



```
duplicate[dup_idx] = rcv_seg.ack_num;
if(!dupli_ack_flag && duplicate[0] == duplicate[1] && duplicate[1] == duplicate[2]){
    printf("3 duplicate ACKs : ACK_num = %d, ssthresh = %d\n", rcv_seg.ack_num, ssthresh);
    dupli_ack_flag = 1;
}
dup_idx = (dup_idx + 1) % 3;
```

我利用一個長度為 3 的陣列 (duplicate[3]) 以及該陣列的 iterator (dup_idx)去檢查最新收到的 3 個 packet，它們的 ack num 有沒有重複。每次收到新 packet 的時候，會將該 packet 的 ack num 放進 dup_idx 所指到的那個位置，然後再檢查陣列內 3 個元素是否相同即可，同時讓 iterator 移至下一個位置，如果 iterator 移到陣列末端(dup_idx==3)，就從頭開始(dup_idx==0)。

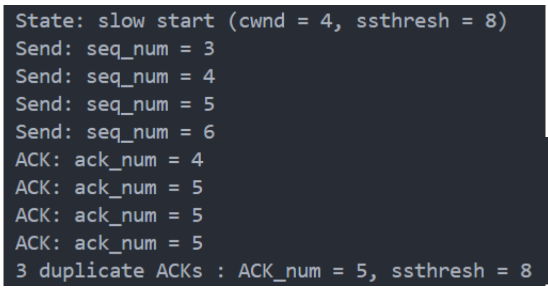
4. How to update cwnd and ssthresh?



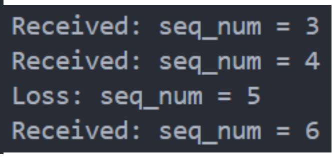
```
        server_send(client_socket_fd, last_acked, cwnd);
        dup_happened = server_receive(client_socket_fd, &last_acked, ssthresh, cwnd);

        if(dup_happened){
            ssthresh = (cwnd==1) ? 1 : cwnd/2;
            cwnd = 1;
        }else{
            if(cwnd<ssthresh){
                cwnd<<=1;
            }else{
                cwnd++;
            }
        }
    }
```

因為在 server_receive()中會檢查有沒有發生 3 duplicate ACKs 的情況，因此我讓 server_receive()回傳是否發生 3 duplicate ACKs，如果發生的話就更新 ssthresh, cwnd，沒發生的話就根據現在的 state 去更新 cwnd。

5. Packet loss 截圖

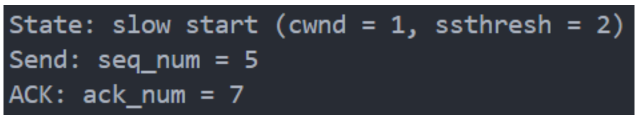


```
State: slow start (cwnd = 4, ssthresh = 8)
Send: seq_num = 3
Send: seq_num = 4
Send: seq_num = 5
Send: seq_num = 6
ACK: ack_num = 4
ACK: ack_num = 5
ACK: ack_num = 5
ACK: ack_num = 5
3 duplicate ACKs : ACK_num = 5, ssthresh = 8
```

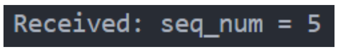


```
Received: seq_num = 3
Received: seq_num = 4
Loss: seq_num = 5
Received: seq_num = 6
```

6. Retransmission 截圖



```
State: slow start (cwnd = 1, ssthresh = 2)
Send: seq_num = 5
ACK: ack_num = 7
```



```
Received: seq_num = 5
```