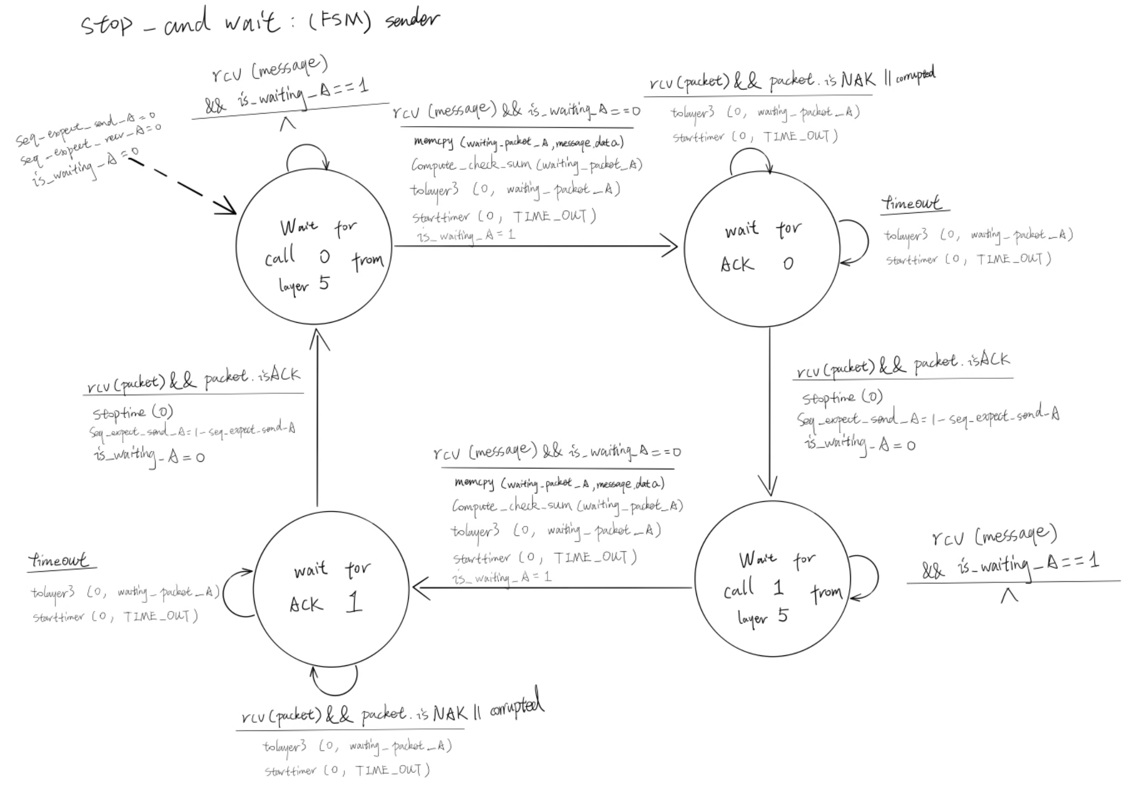
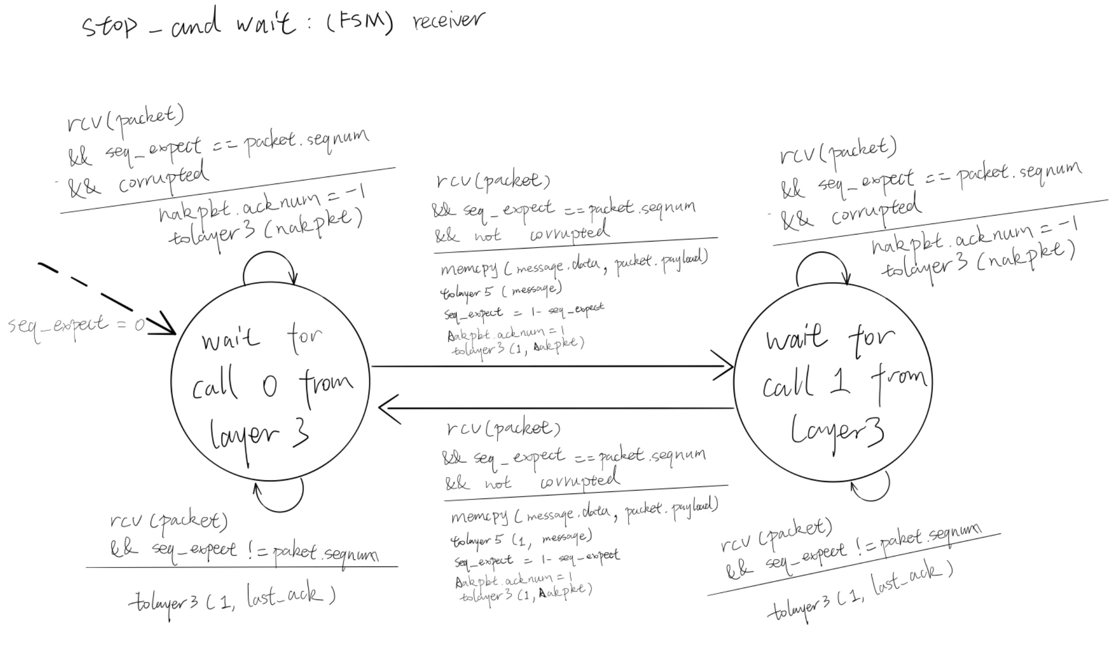
Name: Jie Zhou
CS-1652 Project2 Design Documentation

Stop and Wait Design Finite State Machine:



stop_and wait : (FSM) sender
seq_expect_send_A = 0
seq_expect_recv_A = 0
is_waiting_A = 0
rcv (message) && is_waiting_A == 1
Λ
Wait for call 0 from layer 5
rcv (message) && is_waiting_A == 0
memcpy (waiting_packet_A, message.data)
Compute_check_sum (waiting_packet_A)
tolayer3 (0, waiting_packet_A)
Starttimer (0, TIME_OUT)
is_waiting_A = 1
wait for ACK 0
rcv (packet) && packet.is NAK || corrupted
tolayer3 (0, waiting_packet_A)
Starttimer (0, TIME_OUT)
timeout
tolayer3 (0, waiting_packet_A)
Starttimer (0, TIME_OUT)
rcv (packet) && packet.isACK
Stoptime (0)
Seq_expect_send_A = 1 - seq_expect_send_A
is_waiting_A = 0
Wait for call 1 from layer 5
rcv (message) && is_waiting_A == 1
Λ
rcv (message) && is_waiting_A == 0
memcpy (waiting_packet_A, message.data)
Compute_check_sum (waiting_packet_A)
tolayer3 (0, waiting_packet_A)
Starttimer (0, TIME_OUT)
is_waiting_A = 1
wait for ACK 1
timeout
tolayer3 (0, waiting_packet_A)
Starttimer (0, TIME_OUT)
rcv (packet) && packet.is NAK || corrupted
tolayer3 (0, waiting_packet_A)
Starttimer (0, TIME_OUT)
rcv (packet) && packet.isACK
Stoptime (0)
Seq_expect_send_A = 1 - seq_expect_send_A
is_waiting_A = 0



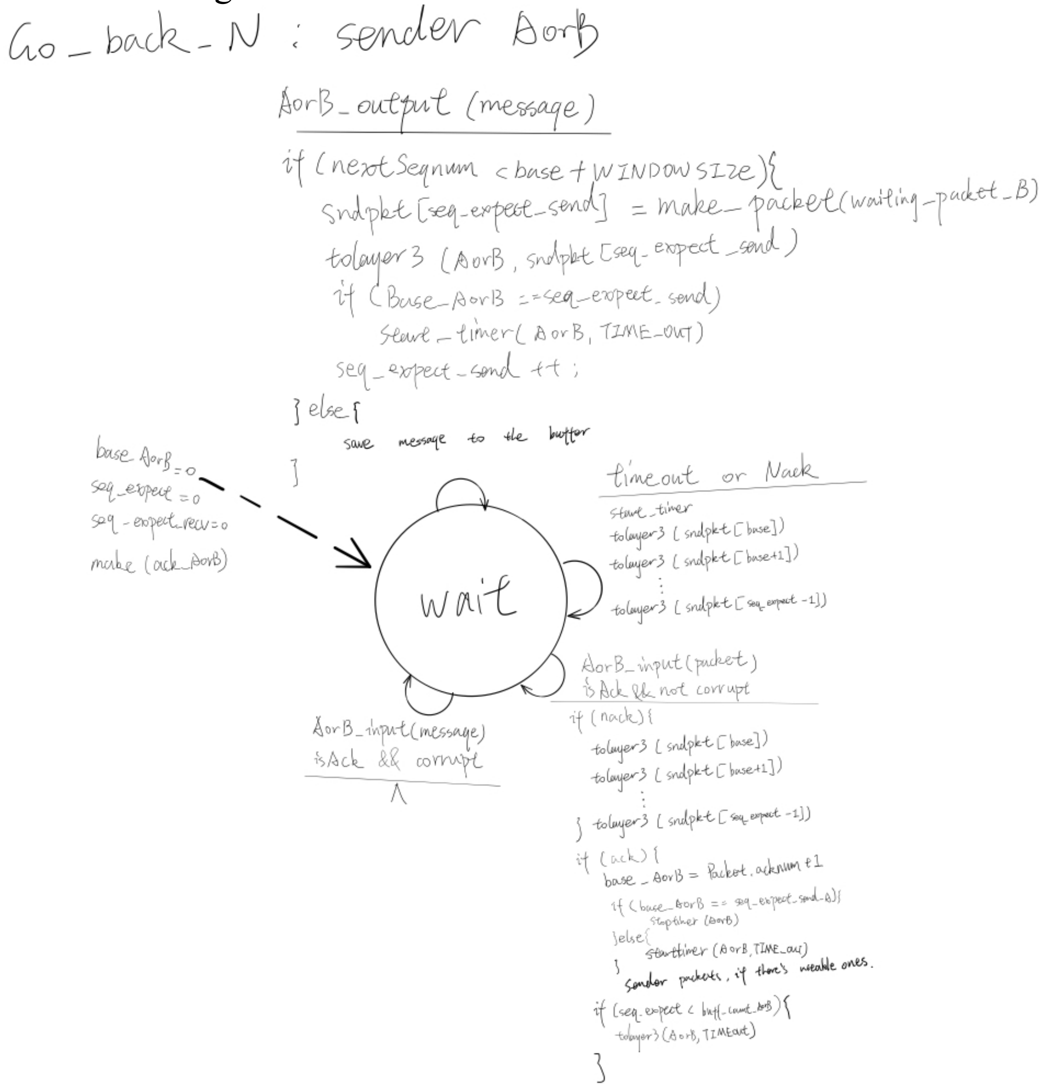
stop_and wait : (FSM) receiver
rcv (packet) && seq_expect == packet.seqnum && corrupted
nakpbt.acknum = -1
tolayer3 (nakpkt)
seq_expect = 0
wait for call 0 from layer 3
rcv (packet) && seq_expect == packet.seqnum && not corrupted
memcpy (message.data, packet.payload)
tolayer5 (message)
seq_expect = 1 - seq_expect
Aakpbt.acknum = 1
tolayer3 (1, Aakpkt)
wait for call 1 from Layer3
rcv (packet) && seq_expect == packet.seqnum && corrupted
nakpbt.acknum = -1
tolayer3 (nakpkt)
rcv (packet) && seq_expect == packet.seqnum && not corrupted
memcpy (message.data, packet.payload)
tolayer5 (1, message)
Seq_expect = 1 - seq_expect
Aakpbt.acknum = 1
tolayer3 (1, Aakpkt)
rcv (packet) && seq_expect != paket.seqnum
tolayer3 (1, last_ack)
rcv (packet) && seq_expect != paket.seqnum
tolayer3 (1, last_ack)

Description:

Since the stop_wait is unidirectional. The sender is A, the receiver is B. Stop and Wait Sender, must handle messages sent from layer5 and send the messages to layer3. The A_output in the project2_stop_wait.c handles that, whenever A_output is called, it will make a packet base of the message from layer5, send it to layer3, start the timer, and wait for a response from the receiver by setting the is_waiting_A to 1. The sender also need to take care of responses from the receiver, which is handle by A_input. Whenever a packet(Ack/Nack) is received by A, a Nack or a corrupted packet will cause the sender to send the previous packet again, a Ack will altering the bit number for the sequence number, causing the program move to the next state, and reset the is_wait_A to 0; So A_ouput can send a new packet to B. Additionally, the sender will resend the last packet if the previous sent packet is lost, causing a timeout.

The receiver, will be waiting packets from layer3, sending correct message to layer5, and send corresponding Ack/Nack back to the sender. Since B do not send any packets to A, so B_output is not utilized. B_input in project2_stop_wait.c handle all the variability of the receiver. And since the program rely on the sender to do timer control, B_timerinterrupt is not utilized too. Base on the design, the receiver will alternating state based on the received packet's sequence number. Upon receiving a packet, the receiver will send a Nack if the packet is corrupted, extract the message if the packet sequence number match the expected packet number and send it to layer5 if not corrupted and send a corresponding Ack back to the sender. Additionally, if a packet's sequence number does not match the expecting sequence number at the receiver, receiver will retransmit the last Ack back to the sender.

Go Back N Design Finite State Machine:



```
base_AorB = 0
seq_expect = 0
seq_expect_recv = 0
make (ack_AorB)
```

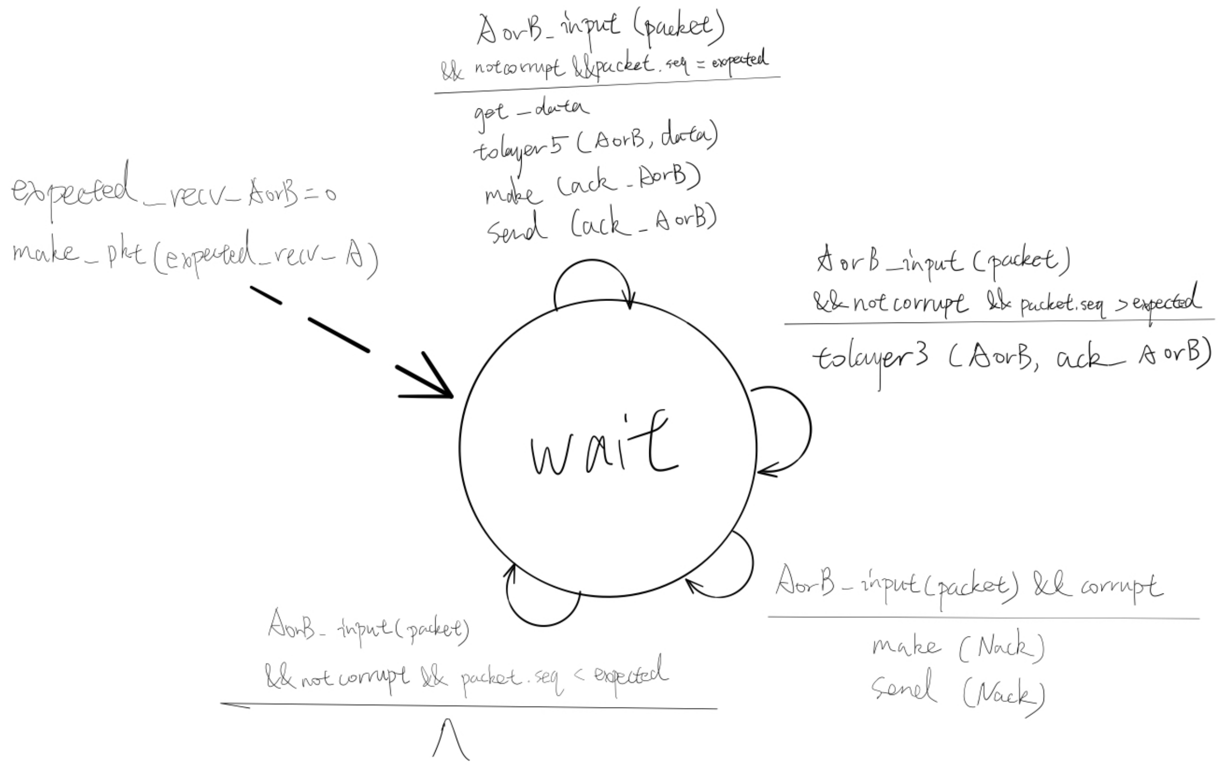
```
AorB_output (message)
_____________________
if (nextSeqnum < base + WINDOWSIZE){
    sndpkt[seq-expect-send] = make_packet(waiting_packet_B)
    tolayer3 (AorB, sndpkt[seq_expect_send)
     if (Base_AorB ==seq_expect_send)
         Start_timer( AorB, TIME_OUT)
     seq_expect_send ++;
} else {
    save message to the buffer
}
```

```
timeout or Nack
_____________________
Start_timer
tolayer3 ( sndpkt[base])
tolayer3 ( sndpkt[base+1])
      :
tolayer3 ( sndpkt[seq_expect -1])
```

```
AorB_input(packet)
isAck && not corrupt
_____________________
if (nack){
   tolayer3 ( sndpkt[base])
   tolayer3 ( sndpkt[base+1])
         :
}  tolayer3 ( sndpkt[seq_expect -1])
if (ack){
   base_AorB = Packet.acknum+1
    if (base_AorB == seq_expect_send_A){
          Stoptimer (AorB)
    }else{
          Starttimer (AorB, TIME_out)
    }
     Sender packets, if there's useable ones.
  if (seq_expect < buff_count_AorB){
     tolayer3(AorB, TIMEout)
  }
```

```
AorB_input(message)
isAck && corrupt
_____________________
Λ
```

Receiver(AorB)


AorB_input (packet)
&& notcorrupt &&packet.seq = expected
get_data
tolayer5 (AorB, data)
make (ack_AorB)
Send (ack_AorB)
expected_recv_AorB = 0
make_pkt (expected_recv_A)
AorB_input (packet)
&& not corrupt && packet.seq > expected
tolayer3 (AorB, ack_AorB)
wait
AorB_input(packet) && corrupt
make (Nack)
Send (Nack)
AorB_input(packet)
&& not corrupt && packet.seq < expected
Λ

Description:

The sender will make a packet base of the message sent from layer5 each time A_output is called, send the packet if it is < baseAorB +WINDOWSIE, print a message if the window is full.

The receiver always keeps track of the highest expected number of packet to receive. AorB_input is in charge of receiving Ack/Nack/Packets from the other side. If an Nack is received, as a sender, it will resend the packets start from the base to the expect_send_AorB. If an Ack is received, the base will increase to the packet.acknum + 1, the timer will be stopped if the base is equal to the expect_send_AorB; otherwise, the timer will be restarted because there’s still inflight packets waiting to be acknowledged. As a sender, upon receiving a Ack, the sender will check if there’s available packet from the buffer that can be sent after moving the base, and send out packets if possible and increase the expect_send_AorB accordingly. All Ack is cumulative, means all packets before has been received. If a corrupted Ack is received, simply ignore it. If received a packet, as a receiver, AorB_input will check the packet for corruption; if not corrupted, the receiver will extract the message from the packet and send it to layer5, and send corresponding Ack back to the sender. If an out of order packet is received, the receiver will simply send the latest correctly received ack. If the packet.acknum is less than the expected, the packet will be ignored. If a corrupted packet is received at the receiver, a Nack will be sent back to the sender. Upon timerinterrupt, the sender will resend packets from base up to the expected_send_AorB.