

Project\_Report

Computer Networks CSE335

Submitted by:

Gasser Tarek 16P6024 CESS

Shehab Ahmed 16P6014 CESS

Hazem Hamada 16P3100 CESS

Submitted to:

DR. Ayman Bahaa

**table\_of\_contents**

INTRODUCTION

ALTERNATING\_BIT\_PROTOCOL:

* FUNCTIONS:
* TEST\_CASES:

GO\_BACK\_N\_PROTOCOL:

* FUNCTIONS:
* TEST\_CASES:

**INTRODUCTION**

* In this project, we wrote the sending and receiving transport-level code for implementing a simple reliable data transfer protocol.
* There are two versions of this project, the Alternating-Bit-Protocol version and the Go-Back-N version.

**ALTERNATING\_BIT\_PROTOCOL**

**FUNCTIONS:**

1. A\_output():

* where message is a structure of type msg, containing data to be sent to the B-side. This routine will be called whenever the upper layer at the sending side (A) has a message to send.
* The algorithm flow:

First the sender A waits for acknowledgement from the receiver, then set the packet sequence number and set the packet acknowledgement number. Then take the message from the application layer and put it in a packet, and calculate the packet check sum, then send the packet and wait for acknowledgement.

* Code:

A\_output(message) struct msg message;

{

if (waiting\_on\_ack)

{

printf("new message arrived at A: can't send there is still a packet in transit\n");

return;

}

struct pkt packet;

packet.seqnum = sender\_sequence\_bit;

packet.acknum = 0; // no need for it for the sender

memcpy(packet.payload, message.data, sizeof(packet.payload));

packet.checksum = check\_sum(&packet);

prev\_packet = packet;

waiting\_on\_ack = 1;

printf("sending a packet with sequence: %d from A\n", packet.seqnum);

print\_packet(&packet);

tolayer3(0, packet);

starttimer(0, TIMER\_INCREMENT);

}

1. A\_input():

* where packet is a structure of type pkt. This routine will be called whenever a packet sent from the B-side (i.e., as a result of a tolayer3() being done by a B-side procedure) arrives at the A-side. packet is the (possibly corrupted) packet sent from the B-side.
* Algorithm flow:

First check the check sum validity and the packet acknowledgment number. If true, stop the timer and then toggle the sender sequence bit.

* Code:

A\_input(packet) struct pkt packet;

{

if (validate\_checksum(&packet) && packet.acknum == sender\_sequence\_bit)

{

printf("ack for packet sequence: %d received from B\n", packet.acknum);

stoptimer(0);

waiting\_on\_ack = 0;

sender\_sequence\_bit ^= 0x1;

}

else

printf("nack or corrupted ack received at A, waiting for time out to resend\n");

}

1. A\_timerinterrupt():

* This routine will be called when A's timer expires (thus generating a timer interrupt).
* Algorithm flow:

Send the packet on the network from A to B, then start the timer.

* Code:

A\_timerinterrupt()

{

printf("time out resending packet with sequence: %d to B\n", prev\_packet.seqnum);

tolayer3(0, prev\_packet);

starttimer(0, TIMER\_INCREMENT);

}

1. A\_init():

* This routine will be called once, before any of your other A-side routines are called.
* Algorithm flow:

Set the sender sequence bit to zero.

* Code:

A\_init()

{

sender\_sequence\_bit = 0;

waiting\_on\_ack = 0;

}

1. B\_input(packet):

* Where the packet is a structure of type pkt. This routine will be called whenever a packet sent from the A-side to the B-side.
* Algorithm flow:

First validate the packet check sum and sequence number. If true ,print the packet and send to A the packet payload, then calculate the acknowledgment check sum and send the acknowledgment to A. if the packet check sum and sequence number are not valid, send negative acknowledgment to A.

* Code:

B\_input(packet) struct pkt packet;

{

if ((validate\_checksum(&packet)) && packet.seqnum == receiver\_sequence\_bit)

{

printf("correct packet with sequence: %d recevied at B sending ack\n", packet.seqnum);

print\_packet(&packet);

tolayer5(1, packet.payload);

struct pkt ack = {0, receiver\_sequence\_bit, 0, {0}};

ack.checksum = check\_sum(&ack);

tolayer3(1, ack);

receiver\_sequence\_bit ^= 0x1;

}

else

{

printf("corrupt or duplicate packet with sequence: %d recevied at B, sending nack\n", packet.seqnum);

print\_packet(&packet);

struct pkt nack = {0, (receiver\_sequence\_bit ^ 0x1), 0, {0}};

nack.checksum = check\_sum(&nack);

tolayer3(1, nack);

}

}

1. B\_init():

* This routine will be called once, before any of the other B-side routines are called. It is used for initialization.
* Algorithm flow:

Set the receiver sequence bit to zero.

* Code:

B\_init()

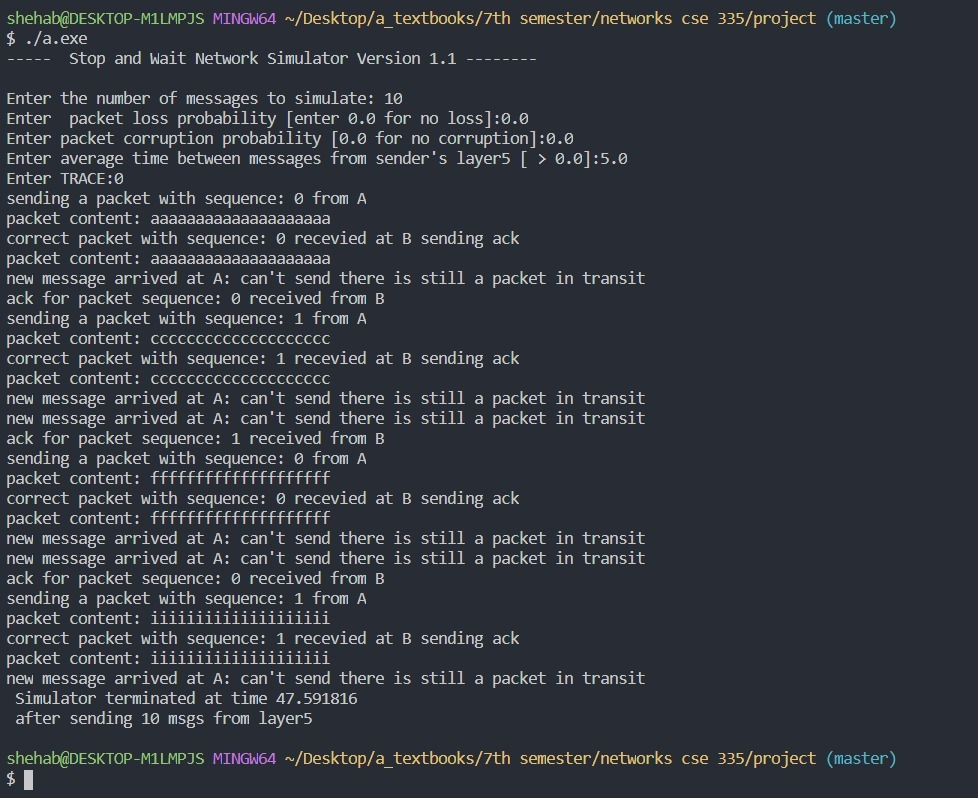
{

receiver\_sequence\_bit = 0;

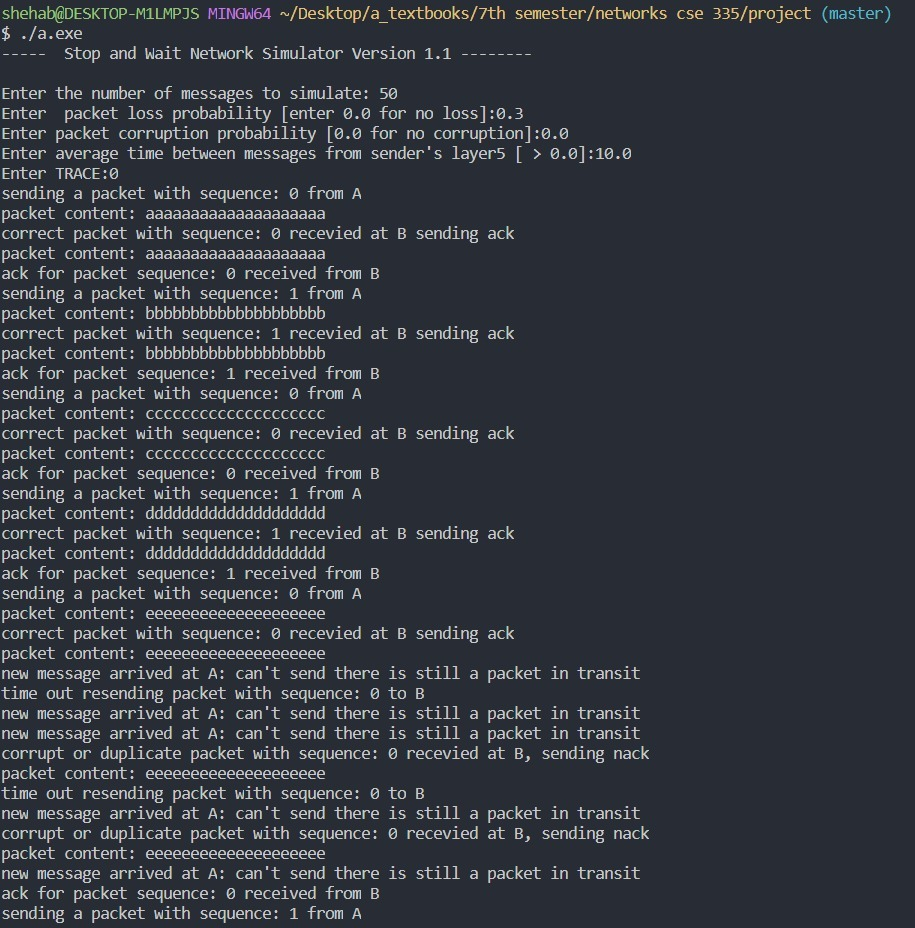
}

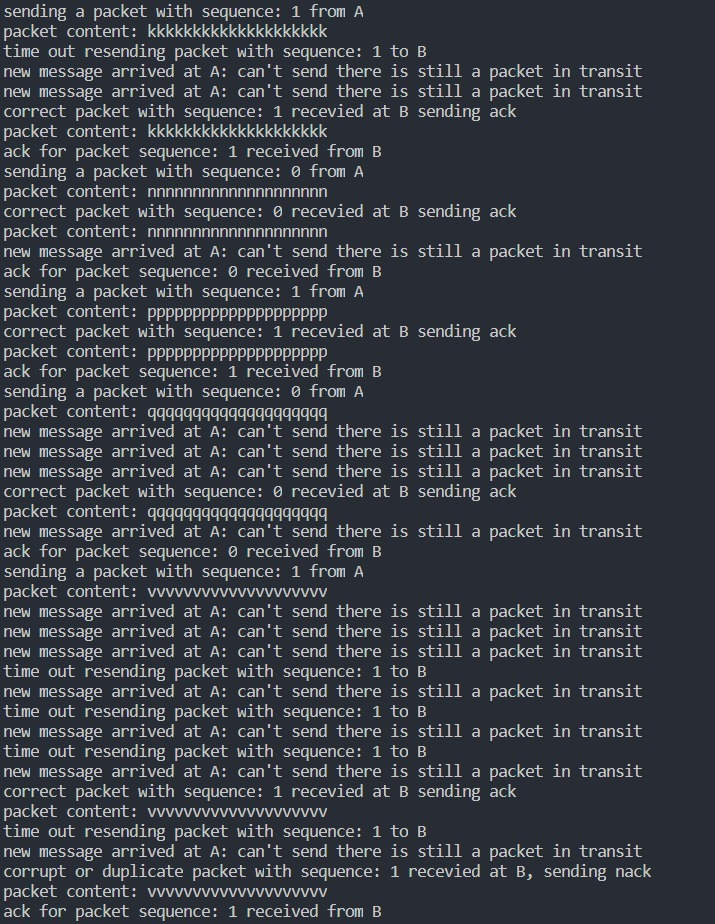
**TEST\_ CASES**

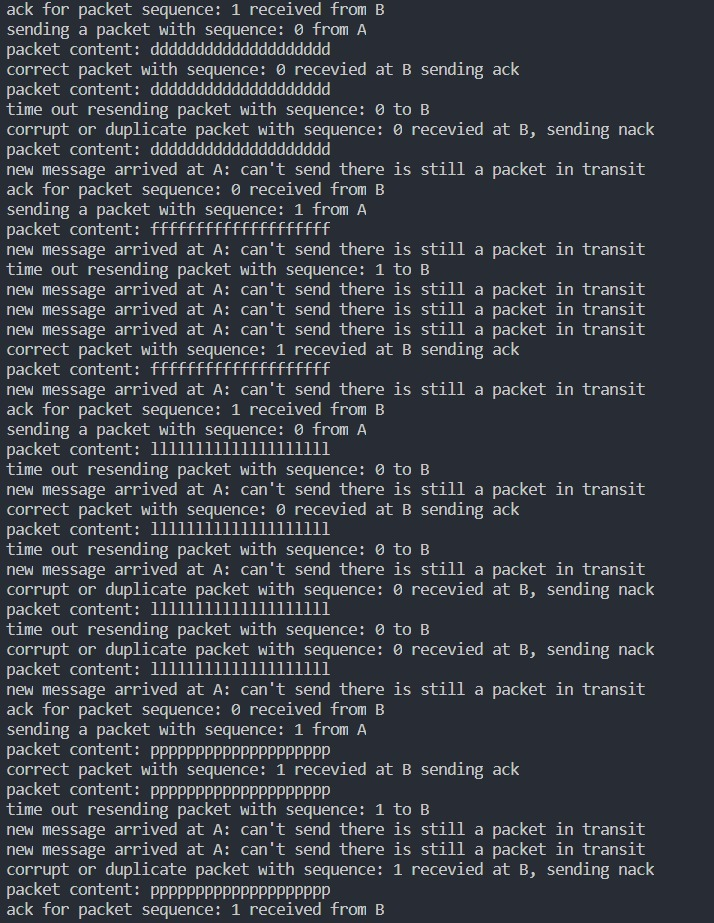
Test\_Case\_A:

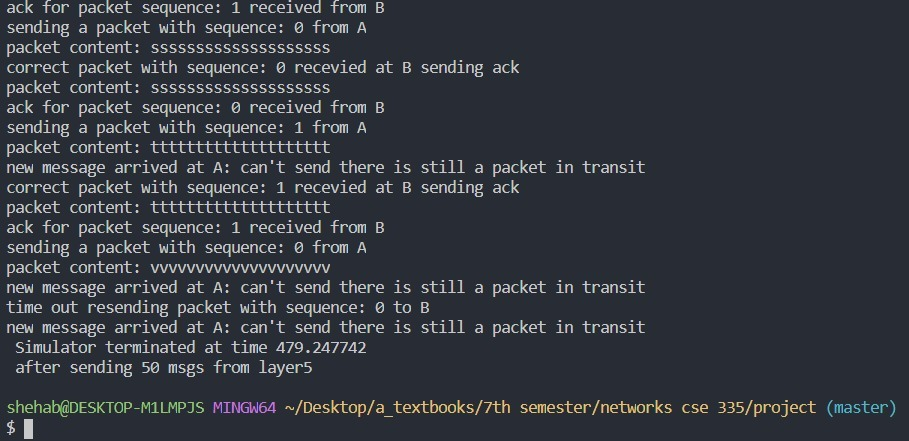


Test\_Case\_B:

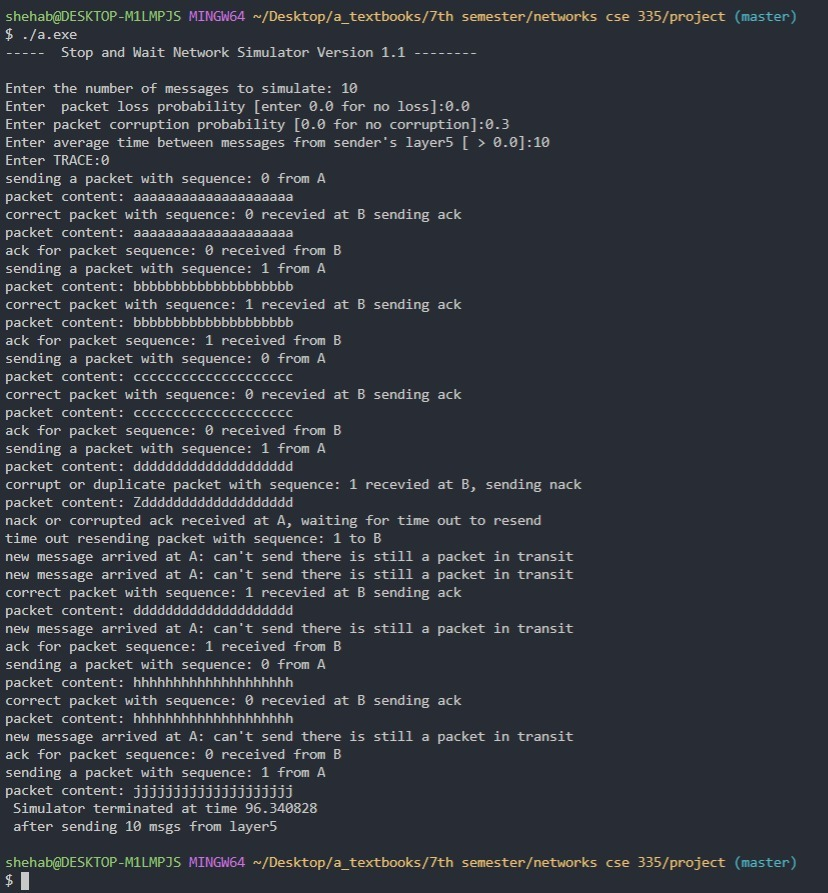








Test\_Case\_C:



**GO\_BACK\_N\_PROTOCOL**

**FUNCTIONS:**

1. A\_output():

* Where message is a structure of type msg, containing data to be sent to the B-side. This routine will be called whenever the upper layer at the sending side (A) has a message to send.
* Algorithm flow:

First the sender A set packet sequence number and sends the packet then sends the next sequence number packet without receiving an ACK signal from the receiver.

* Code:

A\_output(message) struct msg message;

{

if (!(buffer.capacity > 0)) {

printf("new message arrived but droped because buffer is full\n");

return;

}

struct pkt packet;

packet.seqnum = buffer.tail;

packet.acknum = 0;

memcpy(packet.payload, message.data, sizeof(packet.payload));

packet.checksum = check\_sum(&packet);

buffer.packets[buffer.tail] = packet;

buffer.tail = (buffer.tail + 1) % BUFFER\_SIZE;

buffer.capacity--;

int case1 = (buffer.nextseqnum >= buffer.base && buffer.nextseqnum < (buffer.base + WINDOW\_SIZE)) ? 1 : 0;

int case2 = (buffer.nextseqnum < buffer.base && buffer.nextseqnum < (buffer.base + WINDOW\_SIZE) % BUFFER\_SIZE) ? 1 : 0;

if (case1 || case2) {

printf("sending a packet with sequence: %d from A ", buffer.packets[buffer.nextseqnum].seqnum);

print\_packet(&(buffer.packets[buffer.nextseqnum]));

tolayer3(0, buffer.packets[buffer.nextseqnum]);

if (buffer.base == buffer.nextseqnum) {

starttimer(0, TIMER\_INCREMENT);

}

buffer.nextseqnum = (buffer.nextseqnum + 1) % BUFFER\_SIZE;

}

else {

printf("new message arrived but buffred because unacked packets exceed window\n");

}

}

1. A\_input(packet):

* Where packet is a structure of type pkt. This routine will be called whenever a packet sent from the B-side (i.e., as a result of a tolayer3() being done by a B-side procedure) arrives at the A-side. packet is the (possibly corrupted) packet sent from the B-side.
* Algorithm flow:

First check the check sum validity and the packet acknowledgment number,then If no more packets left to send stop the timer.

* Code:

A\_input(packet) struct pkt packet;

{

if (validate\_checksum(&packet) && (buffer.base <= packet.acknum) ||

( (packet.acknum < buffer.base) && (packet.acknum <= (buffer.base+WINDOW\_SIZE)%BUFFER\_SIZE) ))

{

printf("ack for packet sequence: %d received from B\n", packet.acknum);

int acked\_packets = (packet.acknum < buffer.base) ? (packet.acknum + BUFFER\_SIZE - buffer.base + 1) : (buffer.base - packet.acknum + 1);

buffer.capacity += acked\_packets;

buffer.base = (packet.acknum + 1) % BUFFER\_SIZE;

update\_window();

if (buffer.base == buffer.nextseqnum) {

stoptimer(0);

}

else {

stoptimer(0);

starttimer(0, TIMER\_INCREMENT);

}

}

else {

printf("recived corrupted or already acknowledge ack: %d at A waiting for time out to resend\n", packet.acknum);

}

}

1. A\_timerinterrupt():

* This routine will be called when A's timer expires (thus generating a timer interrupt).
* Algorithm flow:

resend the timed-out packet, then restart the timer.

* Code:

A\_timerinterrupt()

{

printf("time out resending packet from base sequence: %d to B\n", buffer.base);

starttimer(0, TIMER\_INCREMENT);

resend\_window();

}

1. A\_init():

* This routine will be called once, before any of your other A-side routines are called.
* Algorithm flow:

Initializes the buffer and sequence number.

* Code:

A\_init()

{

buffer.base = 0;

buffer.nextseqnum = 0;

buffer.capacity = 256;

buffer.tail = 0;

}

1. B\_input(packet):

* Where the packet is a structure of type pkt. This routine will be called whenever a packet sent from the A-side to the B-side.
* Algorithm flow:

First validate the packet check sum and EXPECTED sequence number. If true, print the packet and send the packet payload, then calculate the acknowledgment check sum and send the acknowledgment to A. if the packet check sum and sequence number are not valid, send last ACKed sequence.

* Code:

B\_input(packet) struct pkt packet;

{

if ((validate\_checksum(&packet)) && packet.seqnum == expectedseqnum)

{

printf("correct packet with sequence: %d recevied at B sending ack ", packet.seqnum);

print\_packet(&packet);

tolayer5(1, packet.payload);

struct pkt ack = { 0, expectedseqnum, 0,{ 0 } };

ack.checksum = check\_sum(&ack);

tolayer3(1, ack);

expectedseqnum = (expectedseqnum + 1) % BUFFER\_SIZE;

}

else

{

printf("corrupt or unexpected packet with sequence: %d recevied at B, sending last acked sequence: %d ", packet.seqnum, expectedseqnum - 1);

print\_packet(&packet);

struct pkt nack = { 0, (expectedseqnum - 1), 0,{ 0 } };

nack.checksum = check\_sum(&nack);

tolayer3(1, nack);

}

}

1. B\_init():

* This routine will be called once, before any of the other B-side routines are called. It is used for initialization.
* Algorithm flow:

Initialize expected sequence number.

* Code:

B\_init()

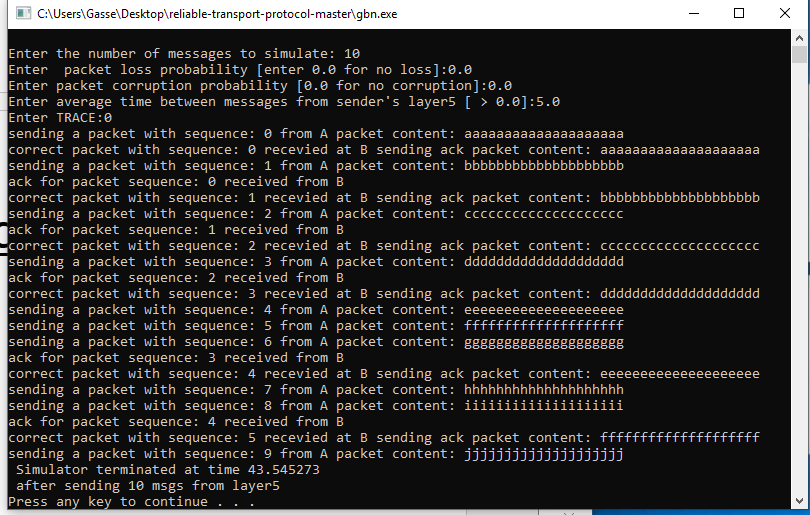
{

int expectedseqnum = 0;

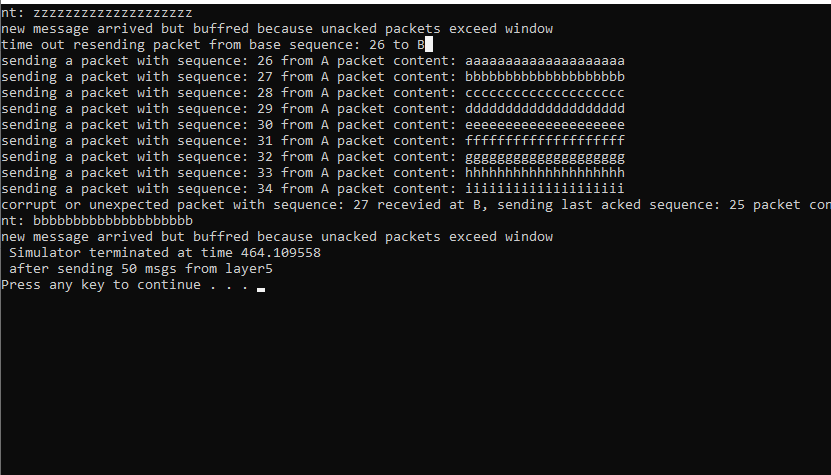
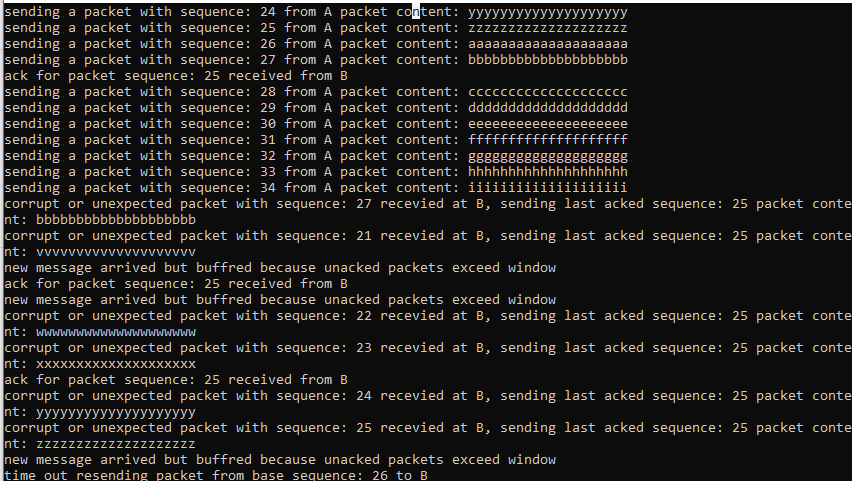
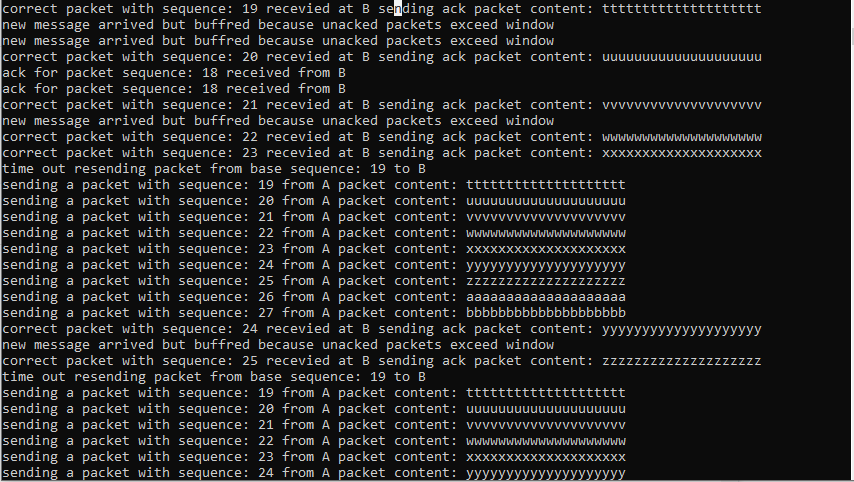
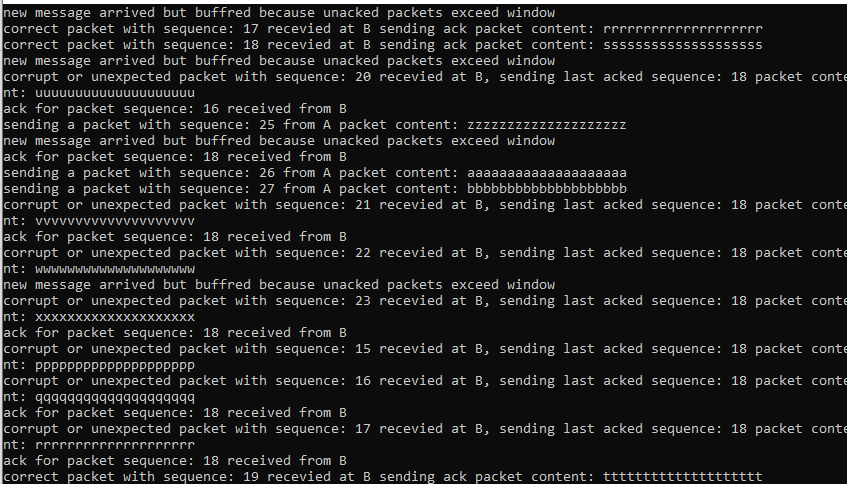
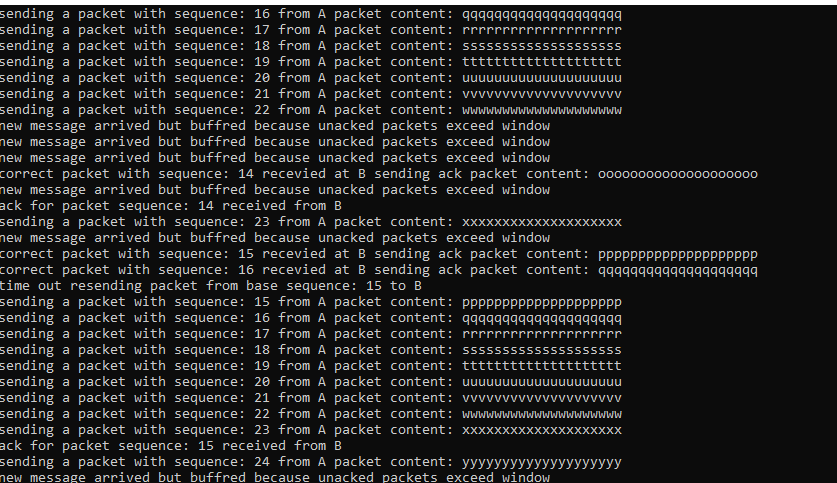
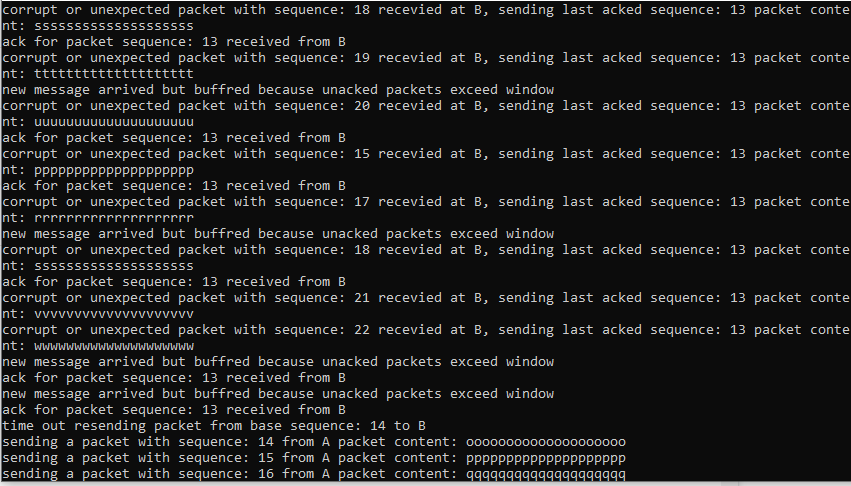
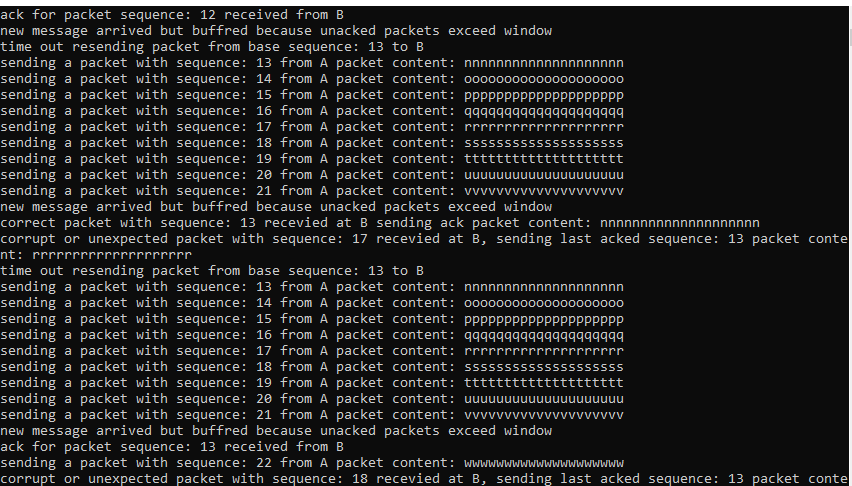
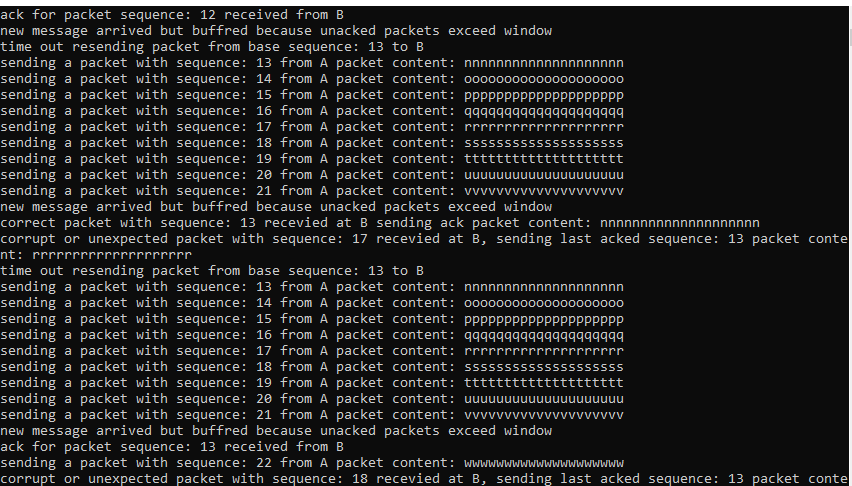
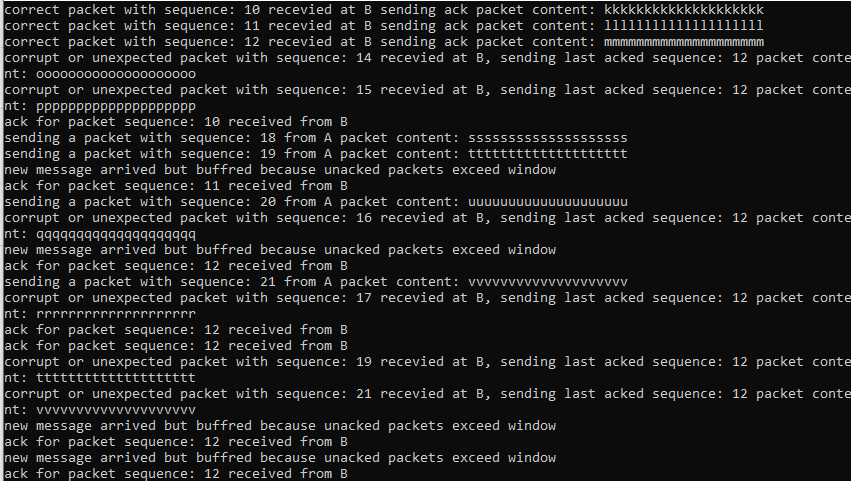
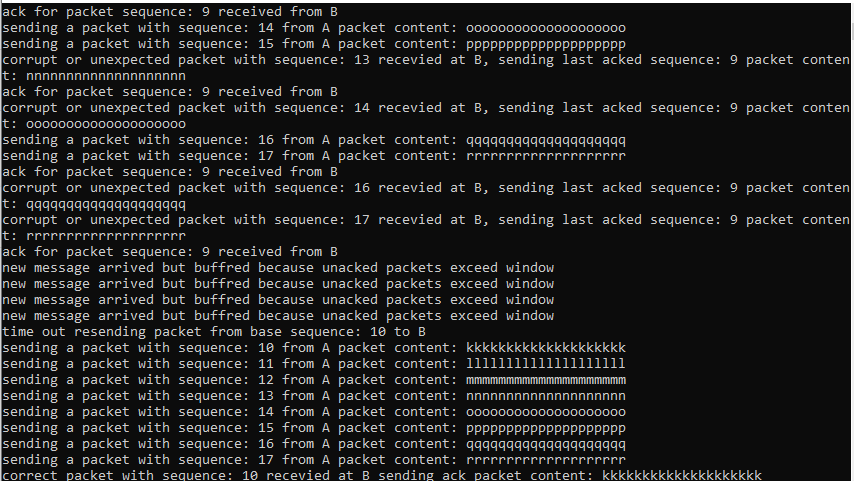
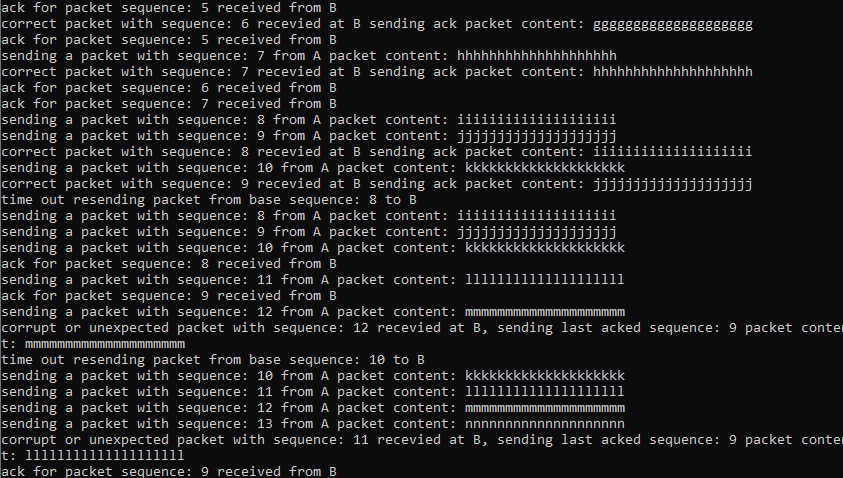
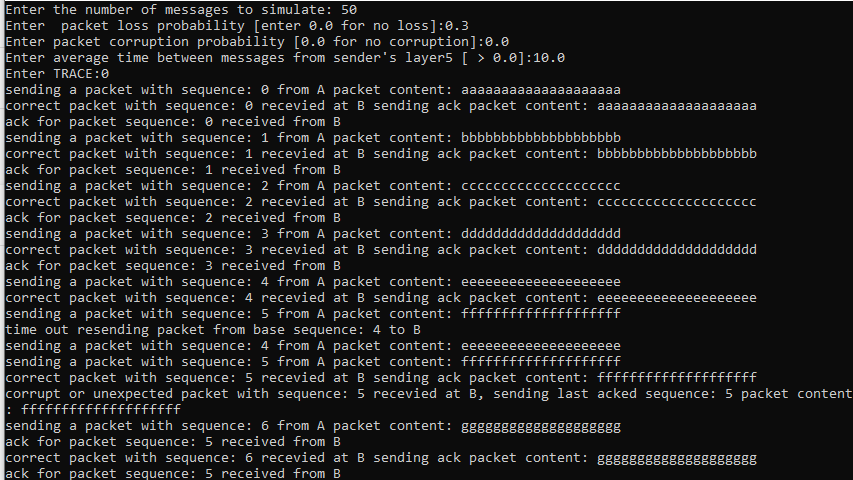
}

**TEST\_ CASES**

Test\_Case\_A:



Test\_Case\_B:



Test\_Case\_C:

