Computer Networks

CSE 335

Project Fall 2019

Submitted by:

Engy Samy Salah
(16P3004)
Mayar Wessam Nour
(16P3008)
Yara Hossam Mohamed
(16P3002)

Part (1): Go Back N:

1- Description of each function, what tasks are performed:

1)interrupt_multitimer:
We call this function after the first timer goes off or was closed

2) start_multitimer:
It starts a timer for a packet
We make a bound check and a warning saying we can't create more than a certain number of timers. Finally if there isn't any timer running, we start the timer right now otherwise it adds this timer into the queue.

3) stop_multitimer:
It stops the first timer and we make a bound check if there isn't any timers running and if so a Warning saying that we are trying to stop a timer isn't running. If there are, it stops the first timer and if there is more than one timer, it runs them right now.

4) push:
This function puts a message in the queue

5) initialize:
This following routine will be called only once before any other

6) timerinterrupt
This is called when A's timer goes off

7) A_output
It is called from layer 5 and pass the data that is going to be sent to other side

8) A_input
It is called from layer 3, when a packet arrives for layer 4

9) A_init
Entity A routines are called. We use it to do any initialization.
It calls function initialize

10)B_input
It is called from layer 3, when a packet arrives for layer 4 at B

11) B_timerinterrupt

It is called when B's timer goes off

12) B_init

Entity B routines are called. We use it to do any initialization
It calls function initialize

13) init:

This initializes the simulator
It makes the user enter the following:

- Enter the number of messages to simulate
- Enter packet loss probability
- Enter packet corruption probability
- Enter average time between messages from sender's layer5
- Enter trace number

14) jimsrand:

Return a float in range [0,1]. It is used to isolate all random number generation in one location.
It assumes that the system-supplied rand() function return an int in therange [0,mmm]

15) generate_next_arrival:

Creating a new arrival

16) insertevent:

Inserts event to the list

17) printevlist

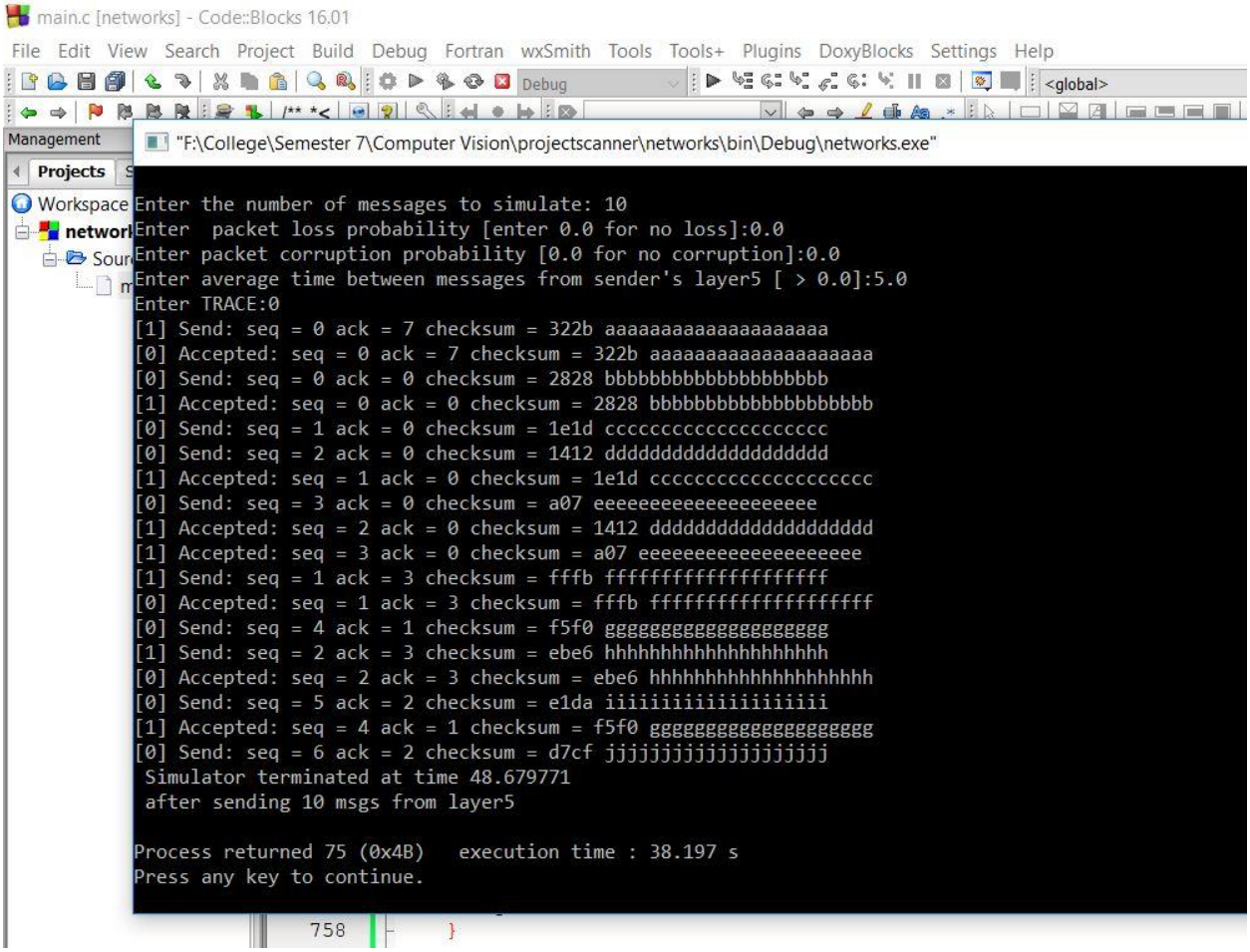
Prints the event list

18) stoptimer

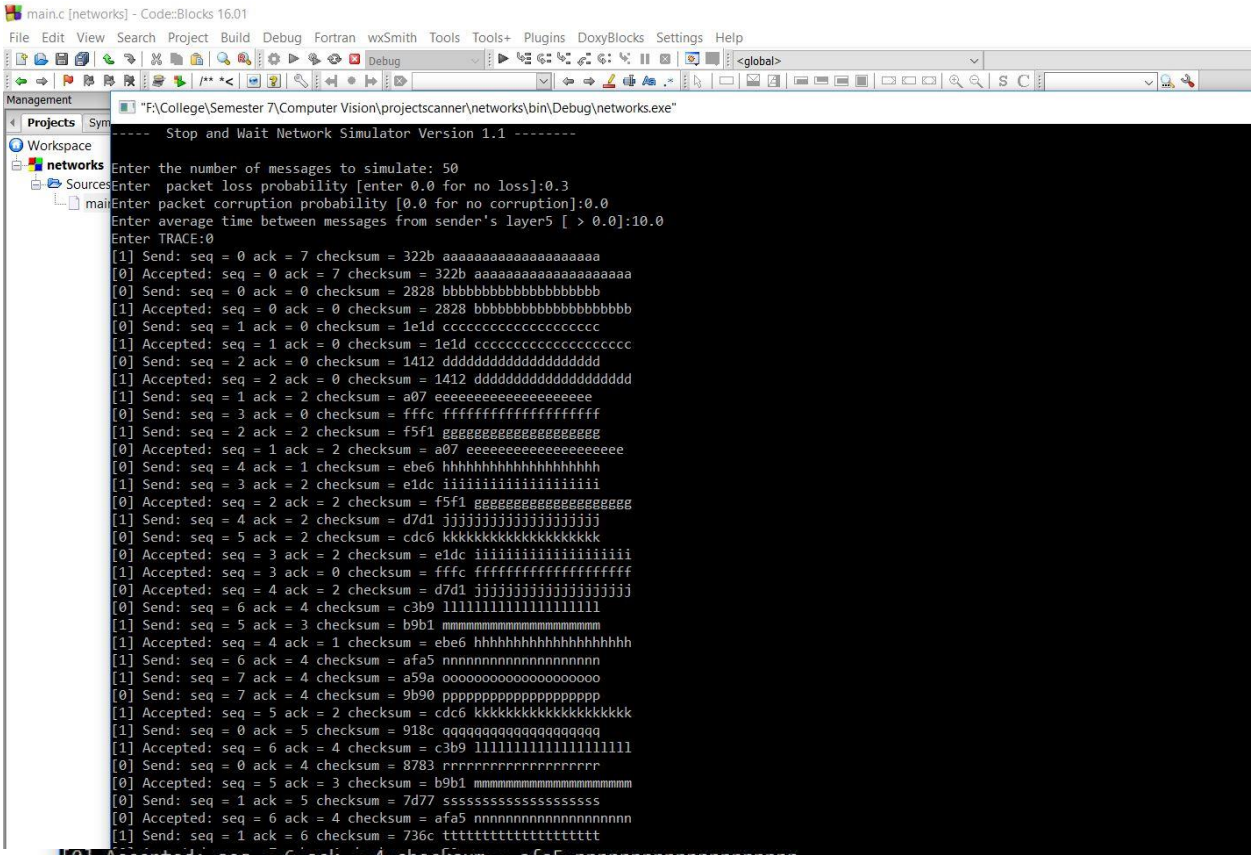
It is called to stop previously started timer. When A or B is trying to stop timer

2-OUTPUT SCREENSHOTS:

1)



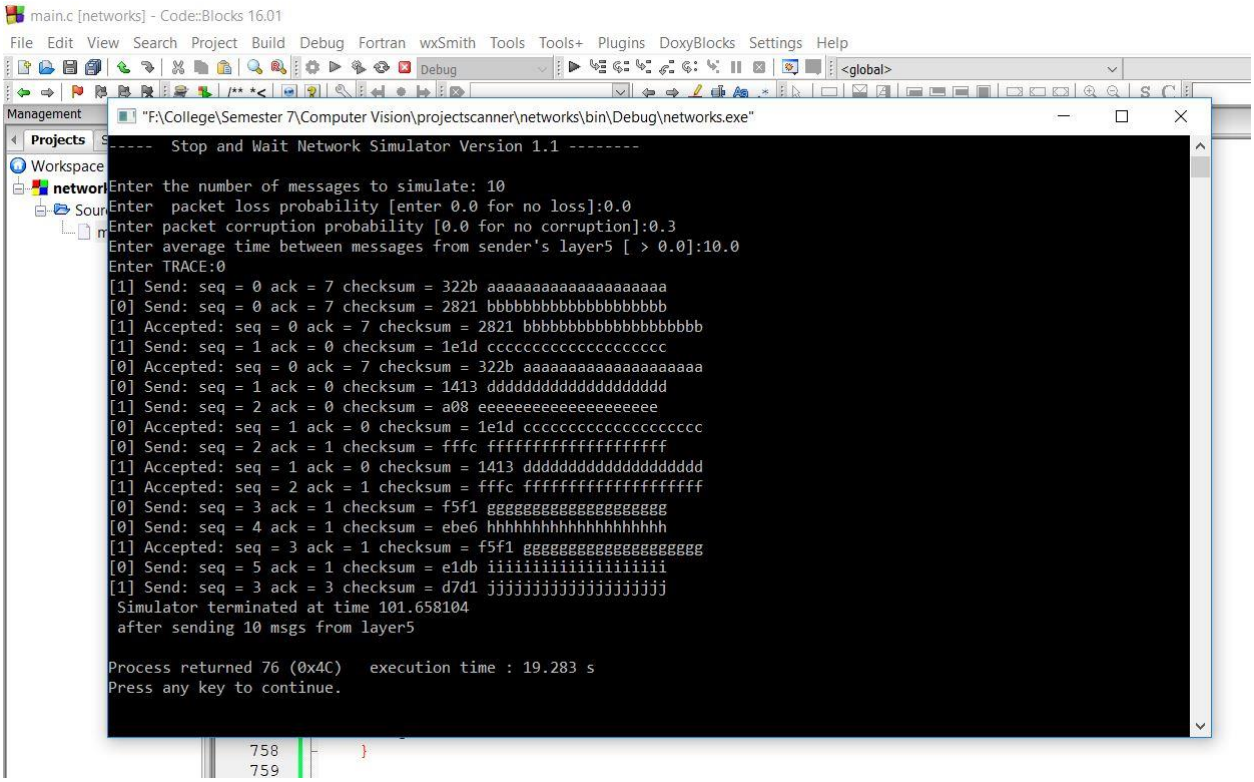
2)



[0] Accepted: seq = 6 ack = 4 checksum = afa5 nnnnnnnnnnnnnnnnnnn
[1] Send: seq = 1 ack = 6 checksum = 736c ttttttttttttttttttt
[0] Accepted: seq = 7 ack = 4 checksum = a59a ooooooooooooooooooo
[0] Send: seq = 2 ack = 7 checksum = 6960 uuuuuuuuuuuuuuuuuuu
[0] Accepted: seq = 0 ack = 5 checksum = 918c qqqqqqqqqqqqqqqqqqqq
[1] Send: seq = 2 ack = 6 checksum = 5f57 vvvvvvvvvvvvvvvvvvvv
[1] Send: seq = 3 ack = 6 checksum = 554c wwwwwwwwwwwwwwwwwww
[0] Send: seq = 3 ack = 0 checksum = 4b48 xxxxxxxxxxxxxxxxxxxx
[0] Send: seq = 4 ack = 0 checksum = 413d yyyyyyyyyyyyyyyyyyyy
[1] Send: seq = 4 ack = 6 checksum = 372d zzzzzzzzzzzzzzzzzzz
[0] Send: seq = 5 ack = 0 checksum = 2823 bbbbbbbbbbbbbbbbbbb
[1] Accepted: seq = 7 ack = 4 checksum = 9b90 pppppppppppppppppppp
[1] Accepted: seq = 0 ack = 4 checksum = 8783 rrrrrrrrrrrrrrrrrrr
[1] Accepted: seq = 1 ack = 5 checksum = 7d77 sssssssssssssssssss
[0] Accepted: seq = 1 ack = 6 checksum = 736c ttttttttttttttttttt
[1] Accepted: seq = 2 ack = 7 checksum = 6960 uuuuuuuuuuuuuuuuuuu
[0] Accepted: seq = 2 ack = 6 checksum = 5f57 vvvvvvvvvvvvvvvvvvvv
[0] Accepted: seq = 3 ack = 6 checksum = 554c wwwwwwwwwwwwwwwwwww
[0] Accepted: seq = 4 ack = 6 checksum = 372d zzzzzzzzzzzzzzzzzzz
[1] Accepted: seq = 3 ack = 0 checksum = 4b48 xxxxxxxxxxxxxxxxxxxx
Simulator terminated at time 405.668488
after sending 50 msgs from layer5

Process returned 76 (0x4C) execution time : 43.204 s
Press any key to continue.

3)



3- The C source files

```
#include
<stdio.h>
            /* ******************************************************************
             ALTERNATING BIT AND GO-BACK-N NETWORK EMULATOR: VERSION 1.1  J.F.Kurose
               This code should be used for PA2, unidirectional or bidirectional
               data transfer protocols (from A to B. Bidirectional transfer of data
               is for extra credit and is not required).  Network properties:
               - one way network delay averages five time units (longer if there
                 are other messages in the channel for GBN), but can be larger
               - packets can be corrupted (either the header or the data portion)
                 or lost, according to user-defined probabilities
               - packets will be delivered in the order in which they were sent
                 (although some can be lost).
            *******************************************************************/
            #define BIDIRECTIONAL 1    /* change to 1 if you're doing extra credit */
            /* and write a routine called B_output */
            /* a "msg" is the data unit passed from layer 5 (teachers code) to layer  */
            /* 4 (students' code).  It contains the data (characters) to be delivered */
```

```
/* to layer 5 via the students transport level protocol entities.         */
struct msg {
       char data[20];
};
/* a packet is the data unit passed from layer 4 (students code) to layer */
/* 3 (teachers code).  Note the pre-defined packet structure, which all   */
/* students must follow. */
struct pkt {
       int seqnum;
       int acknum;
       int checksum;
       char payload[20];
};
/********* STUDENTS WRITE THE NEXT SEVEN ROUTINES *********/
#define DEBUG 1
#define POINT 2
#define MAX_SEQ 7
#define MAX_WINDOW (MAX_SEQ+1)
#define MAX_BUF 50
#define TIME_OUT 16.0
#define INC(a) ((a+1)%MAX_WINDOW)
#define DEC(a) ((a+MAX_WINDOW-1)%MAX_WINDOW)
int expect_ack[POINT];
int expect_send[POINT];
int packet_in_buffer[POINT];
int expect_recv[POINT];
struct pkt window_buffer[POINT][MAX_WINDOW];
/* print packet content */
print_packet(AorB, action, packet)
char *action;
int AorB;
struct pkt packet;
{
       printf("[%d] %s: ", AorB, action);
       printf("seq = %d ack = %d checksum = %x ", packet.seqnum, packet.acknum,
packet.checksum);
       int i;
       for (i = 0; i < 20; i++)
              putchar(packet.payload[i]);
       putchar('\n');
}
/* compute checksum */
int compute_check_sum(packet)
```

```
struct pkt packet;
{
	int sum = 0, i = 0;
	sum = packet.checksum;
	sum += packet.seqnum;
	sum += packet.acknum;
	sum = (sum >> 16) + (sum & 0xffff);
	for (i = 0; i < 20; i += 2) {
		sum += (packet.payload[i] << 8) + packet.payload[i+1];
		sum = (sum >> 16) + (sum & 0xffff);
	}
	sum = (~sum) & 0xffff;
	return sum;
}
/* check if a <= b < c circularly */
int between(a, b, c)
int a, b, c;
{
	if ((a <= b && b < c)
		|| (c < a && a <= b)
		|| (b < c && c < a))
		return 1;
	else
		return 0;
}
/* construct a packet */
struct pkt construct_packet(AorB, message)
int AorB;
struct msg message;
{
	struct pkt packet;
	memcpy(packet.payload, message.data, sizeof(message.data));
	packet.seqnum = expect_send[AorB];
	packet.acknum = DEC(expect_recv[AorB]);
	packet.checksum = 0;
	packet.checksum = compute_check_sum(packet);
	window_buffer[AorB][expect_send[AorB]] = packet;
	expect_send[AorB] = INC(expect_send[AorB]);
	packet_in_buffer[AorB]++;
	return packet;
}
/* multitimer:
 * start a timer for each packet using one timer
```

```
 */
float timers_expire[POINT][MAX_WINDOW];
int timers_seqs[POINT][MAX_WINDOW];
int timers_seq[POINT] = {0, 0};
int timers_running[POINT] = {0, 0};
int timers_head[POINT] = {0, 0};
int timers_tail[POINT] = {0, 0};
float time = 0.0;
/* call this function after the first timer goes off or was be closed */
interrupt_multitimer(AorB)
int AorB;
{
	timers_running[AorB] = 0;
}
/* start a timer for a packet */
start_multitimer(AorB, seqnum)
int AorB, seqnum;
{
	/* bound check */
	if (timers_head[AorB] == timers_tail[AorB] + 1) {
		printf("Warning: you can't create more than %d timers.\n",
MAX_WINDOW);
		return;
	}
	if (timers_running[AorB] == 0) {	/* if timers isn't running, start the
timer right now */
		timers_running[AorB] = 1;
		timers_seq[AorB] = seqnum;
		starttimer(AorB, TIME_OUT);
	} else {					/* else, add this timer into
the queue */
		timers_expire[AorB][timers_tail[AorB]] = time + TIME_OUT;
		timers_seqs[AorB][timers_tail[AorB]] = seqnum;
		timers_tail[AorB] = INC(timers_tail[AorB]);
	}
}
/* stop the first timer */
stop_multitimer(AorB, seqnum)
int AorB, seqnum;
{
	/* bound check */
	if (timers_running[AorB] == 0) {
		printf("Warning: you are trying to stop a timer isn't running.\n");
```

```
                return;
        }
        /* stop the first timer */
        stoptimer(AorB);
        timers_running[AorB] = 0;
        /* if there is more timer, run it right now */
        if (timers_head[AorB] != timers_tail[AorB]) {
                timers_running[AorB] = 1;
                float increment = timers_expire[AorB][timers_head[AorB]] - time;
                timers_seq[AorB] = timers_seqs[AorB][timers_head[AorB]];
                timers_head[AorB] = INC(timers_head[AorB]);
                starttimer(AorB, increment);
        }
}
/* queue:
 * when message is out of the sender's window, put the messge in queue
 */
int queue_head[POINT] = {0, 0};
int queue_tail[POINT] = {0, 0};
struct msg queue_buffer[POINT][MAX_BUF];
/* check if queue is empty */
#define empty(AorB) (queue_head[AorB] == queue_tail[AorB])
/* put message in queue */
push(AorB, message)
int AorB;
struct msg message;
{
        /* bound check */
        if (queue_head[AorB] == queue_tail[AorB] + 1) {
                printf("Warning: there is no avaliable space in queue.\n");
                return;
        }
        queue_buffer[AorB][queue_tail[AorB]] = message;
        queue_tail[AorB] = INC(queue_tail[AorB]);
}
/* get messsage out of queue */
struct msg pop(AorB)
int AorB;
{
        /* bound check */
        if (empty(AorB)) {
                printf("Warning: no packet in queue.\n");
                return;
```

```
        }
        struct msg message = queue_buffer[AorB][queue_head[AorB]];
        queue_head[AorB] = INC(queue_head[AorB]);
        return message;
}
/* called from layer 5, passed the data to be sent to other side */
output(AorB, message)
int AorB;
struct msg message;
{
        /* check if msg is in the window */
        if (packet_in_buffer[AorB] < MAX_WINDOW) {
                /* construct a packet */
                struct pkt packet = construct_packet(AorB, message);
                tolayer3(AorB, packet);
                start_multitimer(AorB, packet.seqnum);
                /* debug output */
                if (DEBUG)
                        print_packet(AorB, "Send", packet);
        } else {
                push(AorB, message);
        }
}
/* called from layer 3, when a packet arrives for layer 4 */
input(AorB, packet)
int AorB;
struct pkt packet;
{
        /*      if (DEBUG)
                print_packet("Recieved", packet);*/
        if (compute_check_sum(packet) == 0 && expect_recv[AorB] == packet.seqnum) {
                /* pass data to layer5 */
                struct msg message;
                memcpy(message.data, packet.payload, sizeof(packet.payload));
                tolayer5(AorB, message);
                expect_recv[AorB] = INC(expect_recv[AorB]);
                /* release ACKed packet */
                while (between(expect_ack[AorB], packet.acknum, expect_send[AorB]))
{
/*                      if (DEBUG)
                                print_packet(AorB, "Acknowledged",
window_buffer[AorB][expect_ack[AorB]]);*/
                        expect_ack[AorB] = INC(expect_ack[AorB]);
```

```
                        packet_in_buffer[AorB]--;
                        stop_multitimer(AorB, expect_ack[AorB]);
                }
                /* add new packet from queue */
                while (packet_in_buffer[AorB] < MAX_WINDOW && !empty(AorB)) {
                        struct msg message = pop(AorB);
                        struct pkt packet = construct_packet(AorB, message);
                        tolayer3(AorB, packet);
                        start_multitimer(AorB, packet.seqnum);
                        /* debug output */
                        if (DEBUG)
                                print_packet(AorB, "Send", AorB, packet);
                }
                /* debug output */
                if (DEBUG)
                        print_packet(AorB, "Accepted", packet);
        }
}
/* the following routine will be called once (only) before any other */
initialize(AorB)
int AorB;
{
        packet_in_buffer[AorB] = 0;
        expect_send[AorB] = 0;
        expect_ack[AorB] = 0;
        expect_recv[AorB] = 0;
}
/* called when A's timer goes off */
timerinterrupt(AorB)
int AorB;
{
        interrupt_multitimer(AorB);
        int seqnum;
        for (seqnum = expect_ack[AorB]; seqnum != expect_send[AorB]; seqnum =
INC(seqnum)) {
                if (seqnum != expect_ack[AorB])
                        stop_multitimer(AorB, seqnum);
                tolayer3(AorB, window_buffer[AorB][seqnum]);
                start_multitimer(AorB, seqnum);
/*              if (DEBUG)
                        print_packet(AorB, "Timeout retransmit",
window_buffer[AorB][seqnum]);*/
        }
```

```
}
/* called from layer 5, passed the data to be sent to other side */
A_output(message)
struct msg message;
{
       output(0, message);
}
B_output(message)  /* need be completed only for extra credit */
struct msg message;
{
       output(1, message);
}
/* called from layer 3, when a packet arrives for layer 4 */
A_input(packet)
struct pkt packet;
{
       input(0, packet);
}
/* called when A's timer goes off */
A_timerinterrupt()
{
       timerinterrupt(0);
}
/* the following routine will be called once (only) before any other */
/* entity A routines are called. You can use it to do any initialization */
A_init()
{
       initialize(0);
}
/* Note that with simplex transfer from a-to-B, there is no B_output() */
/* called from layer 3, when a packet arrives for layer 4 at B*/
B_input(packet)
struct pkt packet;
{
       input(1, packet);
}
/* called when B's timer goes off */
B_timerinterrupt()
{
       timerinterrupt(1);
}
/* the following rouytine will be called once (only) before any other */
/* entity B routines are called. You can use it to do any initialization */
```

```
B_init()
{
       initialize(1);
}
/****************************************************************
***************** NETWORK EMULATION CODE STARTS BELOW ***********
The code below emulates the layer 3 and below network environment:
  - emulates the tranmission and delivery (possibly with bit-level corruption
    and packet loss) of packets across the layer 3/4 interface
  - handles the starting/stopping of a timer, and generates timer
    interrupts (resulting in calling students timer handler).
  - generates message to be sent (passed from later 5 to 4)
THERE IS NOT REASON THAT ANY STUDENT SHOULD HAVE TO READ OR UNDERSTAND
THE CODE BELOW.  YOU SHOLD NOT TOUCH, OR REFERENCE (in your code) ANY
OF THE DATA STRUCTURES BELOW.  If you're interested in how I designed
the emulator, you're welcome to look at the code - but again, you should have
to, and you defeinitely should not have to modify
****************************************************************/
struct event {
       float evtime;           /* event time */
       int evtype;             /* event type code */
       int eventity;           /* entity where event occurs */
       struct pkt *pktptr;     /* ptr to packet (if any) assoc w/ this event */
       struct event *prev;
       struct event *next;
};
struct event *evlist = NULL;   /* the event list */
/* possible events: */
#define  TIMER_INTERRUPT 0
#define  FROM_LAYER5     1
#define  FROM_LAYER3     2
#define  OFF             0
#define  ON              1
#define   A    0
#define   B    1
int TRACE = 1;             /* for my debugging */
int nsim = 0;              /* number of messages from 5 to 4 so far */
int nsimmax = 0;           /* number of msgs to generate, then stop */
/*float time = 0.000;*/
float lossprob;            /* probability that a packet is dropped  */
float corruptprob;         /* probability that one bit is packet is flipped */
float lambda;              /* arrival rate of messages from layer 5 */
int   ntolayer3;           /* number sent into layer 3 */
```

```
int   nlost;              /* number lost in media */
int ncorrupt;             /* number corrupted by media*/
main()
{
        struct event *eventptr;
        struct msg  msg2give;
        struct pkt  pkt2give;
        int i, j;
        char c;
        init();
        A_init();
        B_init();
        while (1) {
                eventptr = evlist;            /* get next event to simulate */
                if (eventptr == NULL)
                        goto terminate;
                evlist = evlist->next;        /* remove this event from event list
*/
                if (evlist != NULL)
                        evlist->prev = NULL;
                if (TRACE >= 2) {
                        printf("\nEVENT time: %f,", eventptr->evtime);
                        printf("  type: %d", eventptr->evtype);
                        if (eventptr->evtype == 0)
                                printf(", timerinterrupt  ");
                        else if (eventptr->evtype == 1)
                                printf(", fromlayer5 ");
                        else
                                printf(", fromlayer3 ");
                        printf(" entity: %d\n", eventptr->eventity);
                }
                time = eventptr->evtime;        /* update time to next event time
*/
                if (nsim == nsimmax)
                        break;                        /* all done with simulation */
                if (eventptr->evtype == FROM_LAYER5 ) {
                        generate_next_arrival();   /* set up future arrival */
                        /* fill in msg to give with string of same letter */
                        j = nsim % 26;
                        for (i = 0; i < 20; i++)
                                msg2give.data[i] = 97 + j;
                        if (TRACE > 2) {
                                printf("          MAINLOOP: data given to student:
```

```
");
                        for (i = 0; i < 20; i++)
                                printf("%c", msg2give.data[i]);
                        printf("\n");
                }
                nsim++;
                if (eventptr->eventity == A)
                        A_output(msg2give);
                else
                        B_output(msg2give);
        }
        else if (eventptr->evtype ==  FROM_LAYER3) {
                pkt2give.seqnum = eventptr->pktptr->seqnum;
                pkt2give.acknum = eventptr->pktptr->acknum;
                pkt2give.checksum = eventptr->pktptr->checksum;
                for (i = 0; i < 20; i++)
                        pkt2give.payload[i] = eventptr->pktptr->payload[i];
                if (eventptr->eventity == A)     /* deliver packet by
calling */
                        A_input(pkt2give);            /* appropriate entity
*/
                else
                        B_input(pkt2give);
                free(eventptr->pktptr);          /* free the memory for
packet */
        }
        else if (eventptr->evtype ==  TIMER_INTERRUPT) {
                if (eventptr->eventity == A)
                        A_timerinterrupt();
                else
                        B_timerinterrupt();
        }
        else  {
                printf("INTERNAL PANIC: unknown event type \n");
        }
        free(eventptr);
    }
terminate:
    printf(" Simulator terminated at time %f\n after sending %d msgs from
layer5\n", time, nsim);
}
init()                          /* initialize the simulator */
{
```

```
        int i;
        float sum, avg;
        float jimsrand();
        printf("-----  Stop and Wait Network Simulator Version 1.1 -------- \n\n");
        printf("Enter the number of messages to simulate: ");
        scanf("%d", &nsimmax);
        printf("Enter  packet loss probability [enter 0.0 for no loss]:");
        scanf("%f", &lossprob);
        printf("Enter packet corruption probability [0.0 for no corruption]:");
        scanf("%f", &corruptprob);
        printf("Enter average time between messages from sender's layer5 [ >
0.0]:");
        scanf("%f", &lambda);
        printf("Enter TRACE:");
        scanf("%d", &TRACE);
        srand(9999);              /* init random number generator */
        sum = 0.0;                /* test random number generator for students */
        for (i = 0; i < 1000; i++)
                sum = sum + jimsrand(); /* jimsrand() should be uniform in [0,1] */
        avg = sum / 1000.0;
        if (avg < 0.25 || avg > 0.75) {
                printf("It is likely that random number generation on your
machine\n" );
                printf("is different from what this emulator expects.  Please
take\n");
                printf("a look at the routine jimsrand() in the emulator code.
Sorry. \n");
                exit(0);
        }
        ntolayer3 = 0;
        nlost = 0;
        ncorrupt = 0;
        time = 0.0;                   /* initialize time to 0.0 */
        generate_next_arrival();     /* initialize event list */
}
/*****************************************************************/
/* jimsrand(): return a float in range [0,1].  The routine below is used to */
/* isolate all random number generation in one location.  We assume that the*/
/* system-supplied rand() function return an int in therange [0,mmm]        */
/*****************************************************************/
float jimsrand()
{
        double mmm = 32767;             /* largest int  - MACHINE DEPENDENT!!!!!!!!
```

```
*/
       float x;                    /* individual students may need to change mmm */
       x = rand() / mmm;           /* x should be uniform in [0,1] */
       return (x);
}
/******************** EVENT HANDLINE ROUTINES *******/
/*  The next set of routines handle the event list   */
/*****************************************************/
generate_next_arrival()
{
       double x, log(), ceil();
       struct event *evptr;
       char *malloc();
       float ttime;
       int tempint;
       if (TRACE > 2)
              printf("          GENERATE NEXT ARRIVAL: creating new arrival\n");
       x = lambda * jimsrand() * 2; /* x is uniform on [0,2*lambda] */
       /* having mean of lambda        */
       evptr = (struct event *)malloc(sizeof(struct event));
       evptr->evtime =  time + x;
       evptr->evtype =  FROM_LAYER5;
       if (BIDIRECTIONAL && (jimsrand() > 0.5) )
              evptr->eventity = B;
       else
              evptr->eventity = A;
       insertevent(evptr);
}
insertevent(p)
struct event *p;
{
       struct event *q, *qold;
       if (TRACE > 2) {
              printf("            INSERTEVENT: time is %lf\n", time);
              printf("            INSERTEVENT: future time will be %lf\n", p-
>evtime);
       }
       q = evlist;     /* q points to header of list in which p struct inserted */
       if (q == NULL) { /* list is empty */
              evlist = p;
              p->next = NULL;
              p->prev = NULL;
       }
```

```
        else {
                for (qold = q; q != NULL && p->evtime > q->evtime; q = q->next)
                        qold = q;
                if (q == NULL) { /* end of list */
                        qold->next = p;
                        p->prev = qold;
                        p->next = NULL;
                }
                else if (q == evlist) { /* front of list */
                        p->next = evlist;
                        p->prev = NULL;
                        p->next->prev = p;
                        evlist = p;
                }
                else {     /* middle of list */
                        p->next = q;
                        p->prev = q->prev;
                        q->prev->next = p;
                        q->prev = p;
                }
        }
}
printevlist()
{
        struct event *q;
        int i;
        printf("--------------\nEvent List Follows:\n");
        for (q = evlist; q != NULL; q = q->next) {
                printf("Event time: %f, type: %d entity: %d\n", q->evtime, q-
>evtype, q->eventity);
        }
        printf("--------------\n");
}
/********************** Student-callable ROUTINES **********************/
/* called by students routine to cancel a previously-started timer */
stoptimer(AorB)
int AorB;  /* A or B is trying to stop timer */
{
        struct event *q, *qold;
        if (TRACE > 2)
                printf("          STOP TIMER: stopping timer at %f\n", time);
        /* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next)  */
        for (q = evlist; q != NULL ; q = q->next)
```

```
                if ( (q->evtype == TIMER_INTERRUPT  && q->eventity == AorB) ) {
                        /* remove this event */
                        if (q->next == NULL && q->prev == NULL)
                                evlist = NULL;       /* remove first and only event
on list */
                        else if (q->next == NULL) /* end of list - there is one in
front */
                                q->prev->next = NULL;
                        else if (q == evlist) { /* front of list - there must be
event after */
                                q->next->prev = NULL;
                                evlist = q->next;
                        }
                        else {     /* middle of list */
                                q->next->prev = q->prev;
                                q->prev->next =  q->next;
                        }
                        free(q);
                        return;
                }
        printf("Warning: unable to cancel your timer. It wasn't running.\n");
}
starttimer(AorB, increment)
int AorB;  /* A or B is trying to stop timer */
float increment;
{
        struct event *q;
        struct event *evptr;
        char *malloc();
        if (TRACE > 2)
                printf("          START TIMER: starting timer at %f\n", time);
        /* be nice: check to see if timer is already started, if so, then  warn */
        /* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next)  */
        for (q = evlist; q != NULL ; q = q->next)
                if ( (q->evtype == TIMER_INTERRUPT  && q->eventity == AorB) ) {
                        printf("Warning: attempt to start a timer that is already
started\n");
                        return;
                }
        /* create future event for when timer goes off */
        evptr = (struct event *)malloc(sizeof(struct event));
        evptr->evtime =  time + increment;
        evptr->evtype =  TIMER_INTERRUPT;
```

```
        evptr->eventity = AorB;
        insertevent(evptr);
}
/************************ TOLAYER3 ***************/
tolayer3(AorB, packet)
int AorB;  /* A or B is trying to stop timer */
struct pkt packet;
{
        struct pkt *mypktptr;
        struct event *evptr, *q;
        char *malloc();
        float lastime, x, jimsrand();
        int i;
        ntolayer3++;
        /* simulate losses: */
        if (jimsrand() < lossprob)  {
                nlost++;
                if (TRACE > 0)
                        printf("          TOLAYER3: packet being lost\n");
                return;
        }
        /* make a copy of the packet student just gave me since he/she may decide
*/
        /* to do something with the packet after we return back to him/her */
        mypktptr = (struct pkt *)malloc(sizeof(struct pkt));
        mypktptr->seqnum = packet.seqnum;
        mypktptr->acknum = packet.acknum;
        mypktptr->checksum = packet.checksum;
        for (i = 0; i < 20; i++)
                mypktptr->payload[i] = packet.payload[i];
        if (TRACE > 2)  {
                printf("          TOLAYER3: seq: %d, ack %d, check: %d ", mypktptr-
>seqnum,
                        mypktptr->acknum,  mypktptr->checksum);
                for (i = 0; i < 20; i++)
                        printf("%c", mypktptr->payload[i]);
                printf("\n");
        }
        /* create future event for arrival of packet at the other side */
        evptr = (struct event *)malloc(sizeof(struct event));
        evptr->evtype =  FROM_LAYER3;   /* packet will pop out from layer3 */
        evptr->eventity = (AorB + 1) % 2; /* event occurs at other entity */
        evptr->pktptr = mypktptr;       /* save ptr to my copy of packet */
```

```
        /* finally, compute the arrival time of packet at the other end.
           medium can not reorder, so make sure packet arrives between 1 and 10
           time units after the latest arrival time of packets
           currently in the medium on their way to the destination */
        lastime = time;
        /* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next) */
        for (q = evlist; q != NULL ; q = q->next)
                if ( (q->evtype == FROM_LAYER3  && q->eventity == evptr->eventity)
)
                        lastime = q->evtime;
        evptr->evtime =  lastime + 1 + 9 * jimsrand();
        /* simulate corruption: */
        if (jimsrand() < corruptprob)  {
                ncorrupt++;
                if ( (x = jimsrand()) < .75)
                        mypktptr->payload[0] = 'Z'; /* corrupt payload */
                else if (x < .875)
                        mypktptr->seqnum = 999999;
                else
                        mypktptr->acknum = 999999;
                if (TRACE > 0)
                        printf("          TOLAYER3: packet being corrupted\n");
        }
        if (TRACE > 2)
                printf("          TOLAYER3: scheduling arrival on other side\n");
        insertevent(evptr);
}
tolayer5(AorB, datasent)
int AorB;
char datasent[20];
{
        int i;
        if (TRACE > 2) {
                printf("          TOLAYER5: data received: ");
                for (i = 0; i < 20; i++)
                        printf("%c", datasent[i]);
                printf("\n");
        }
}
```

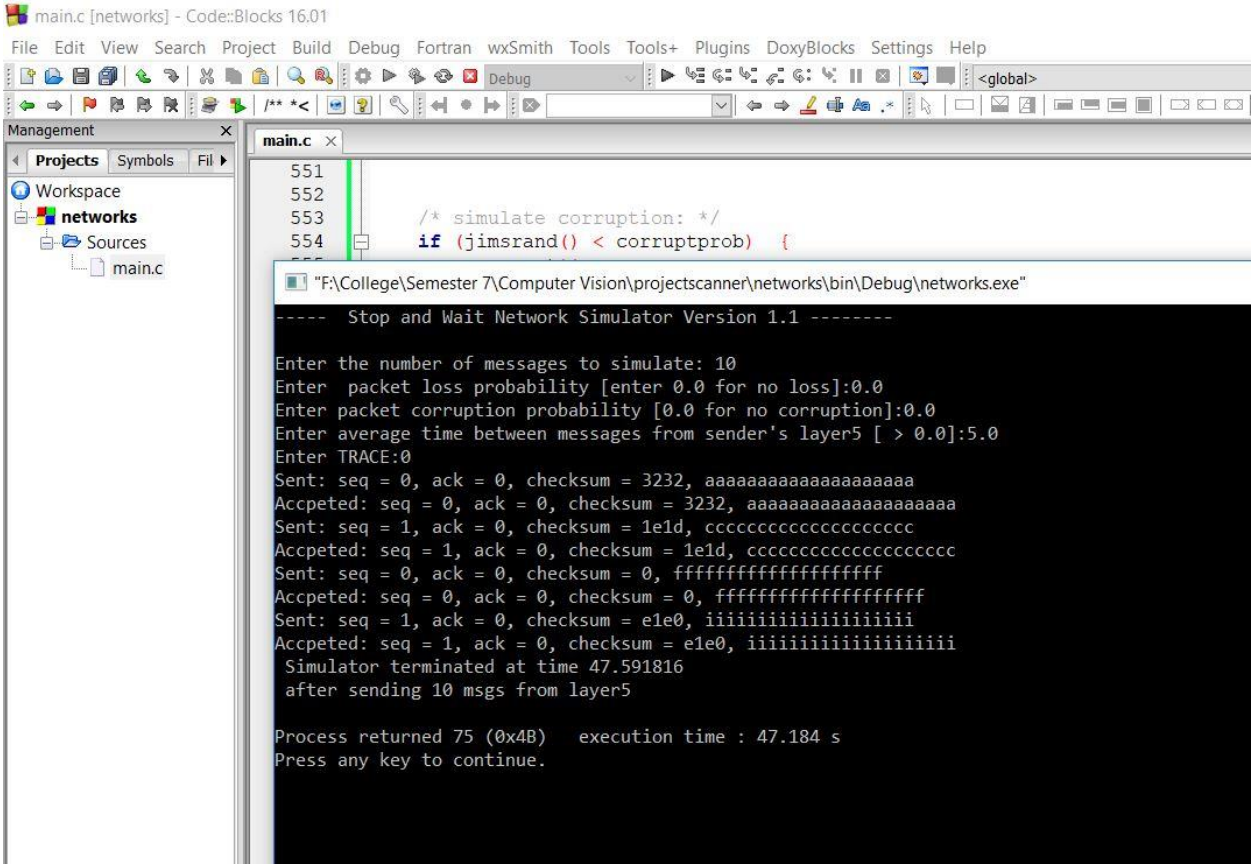
Part (2): Alternating bit protocol:

1- Description of each function, what tasks are performed:

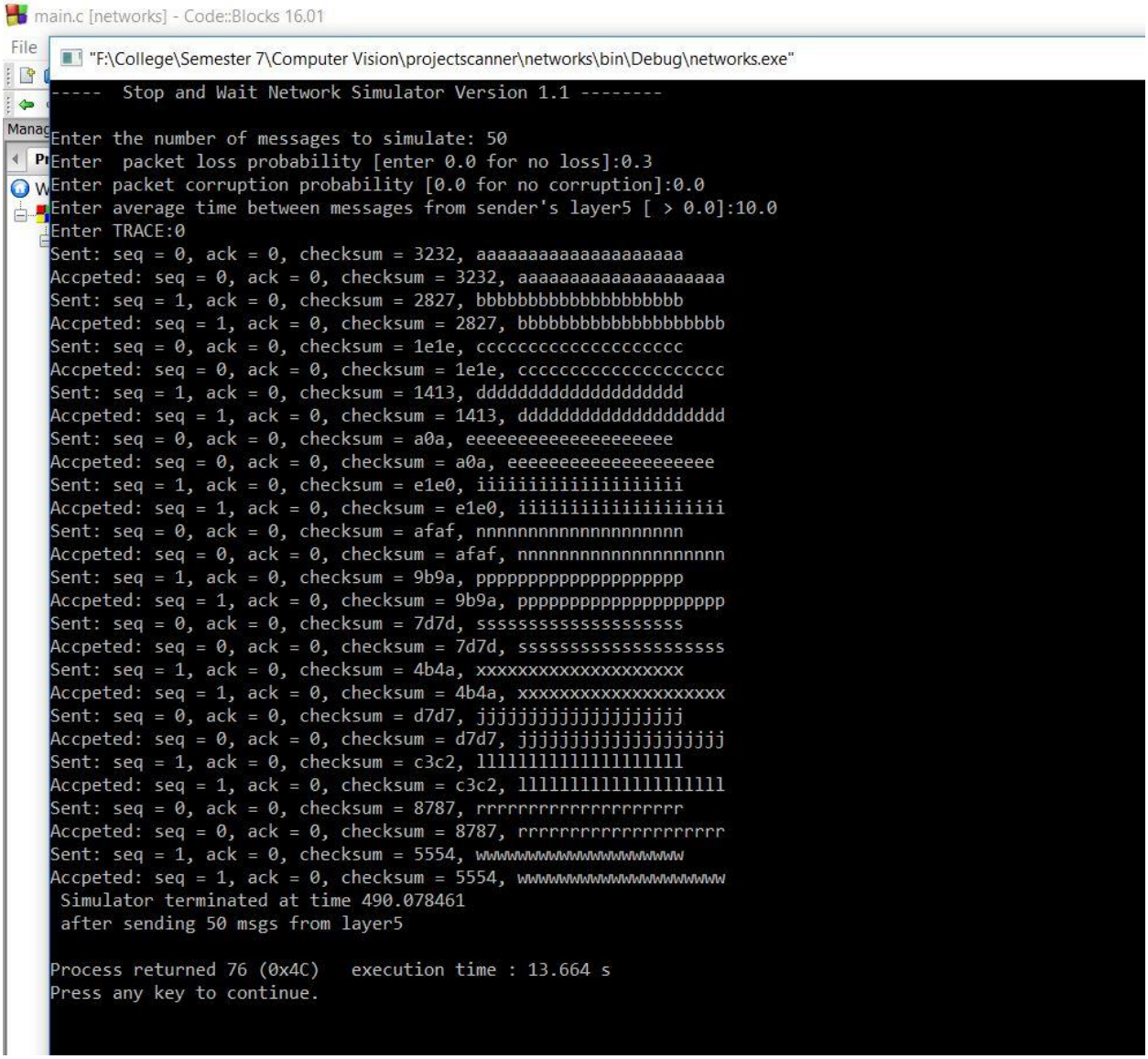
- ***A_init()*** :this function for entity A routines are called. You can use it to do any initialization
- ***print_pkt(action, packet):*** this function is responsible for Printing the payload by passing the 2 parameters to it
- ***compute_check_sum(packet):*** this function is responsible for computing checksum by passing 1 parameter to it
- ***A_timerinterrupt()***:this function is called when A's timer goes off
- ***B_timerinterrupt():***called when B's timer goes off
- ***B_init():*** entity B routines are called. You can use it to do any initialization
- ***init():*** a function used for initializing the simulator
- ***jimsrand()*** :and return a float variable float in range [0,1]and it have to be uniformed in [0,1] and could be used in simulating corruptions and is called while initializing the simulator
- ***printevlist():*** function responsible for printing Event List
- ***generate_next_arrival()***: set up future arrival or initializing event list

2-OUTPUT SCREENSHOTS:

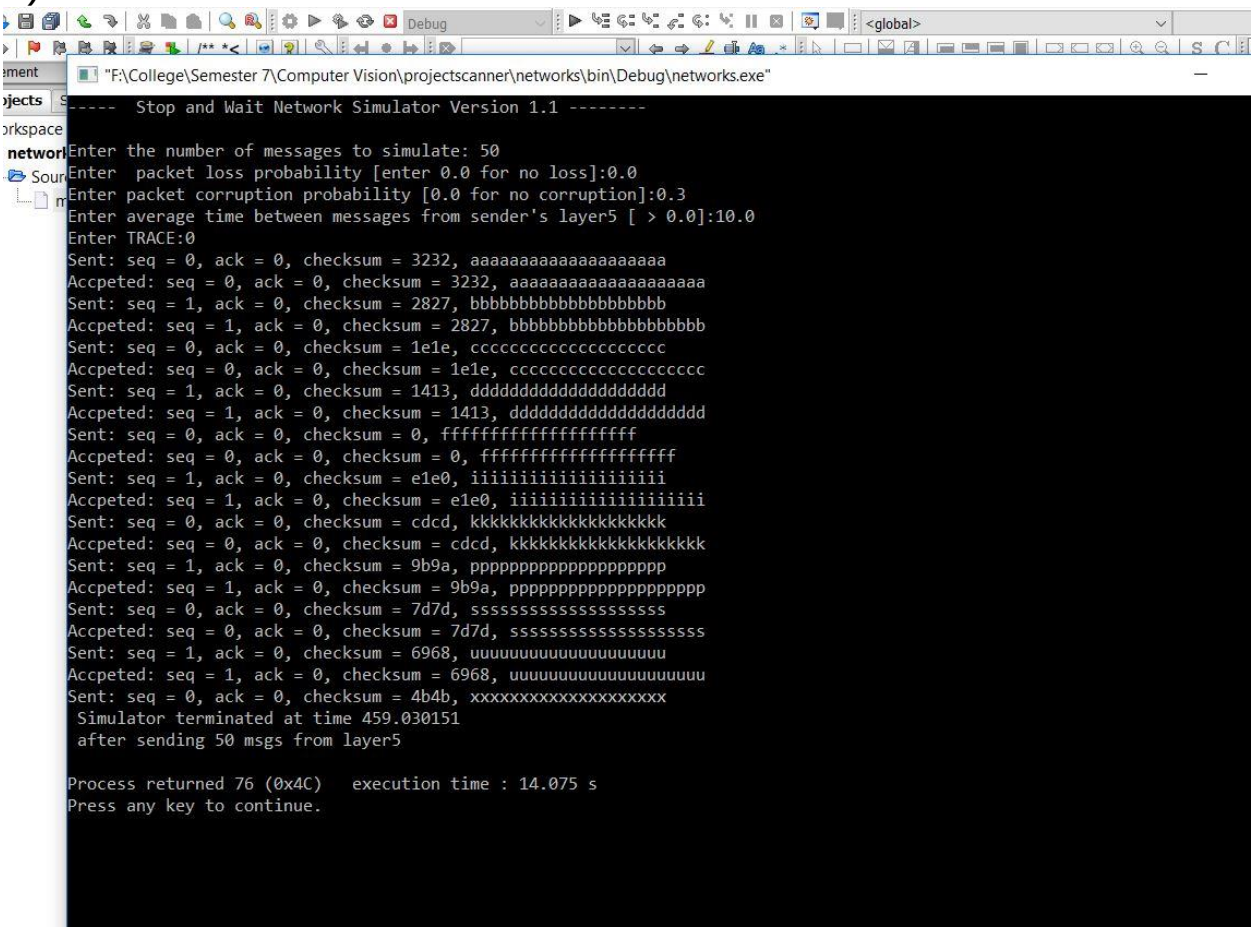
1)



2)


main.c [networks] - Code::Blocks 16.01
File
"F:\College\Semester 7\Computer Vision\projectscanner\networks\bin\Debug\networks.exe"
----- Stop and Wait Network Simulator Version 1.1 --------
Enter the number of messages to simulate: 50
Enter packet loss probability [enter 0.0 for no loss]:0.3
Enter packet corruption probability [0.0 for no corruption]:0.0
Enter average time between messages from sender's layer5 [> 0.0]:10.0
Enter TRACE:0
Sent: seq = 0, ack = 0, checksum = 3232, aaaaaaaaaaaaaaaaaaa
Accpeted: seq = 0, ack = 0, checksum = 3232, aaaaaaaaaaaaaaaaaaa
Sent: seq = 1, ack = 0, checksum = 2827, bbbbbbbbbbbbbbbbbbb
Accpeted: seq = 1, ack = 0, checksum = 2827, bbbbbbbbbbbbbbbbbbb
Sent: seq = 0, ack = 0, checksum = 1e1e, ccccccccccccccccccc
Accpeted: seq = 0, ack = 0, checksum = 1e1e, ccccccccccccccccccc
Sent: seq = 1, ack = 0, checksum = 1413, ddddddddddddddddddd
Accpeted: seq = 1, ack = 0, checksum = 1413, ddddddddddddddddddd
Sent: seq = 0, ack = 0, checksum = a0a, eeeeeeeeeeeeeeeeeee
Accpeted: seq = 0, ack = 0, checksum = a0a, eeeeeeeeeeeeeeeeeee
Sent: seq = 1, ack = 0, checksum = e1e0, iiiiiiiiiiiiiiiiiii
Accpeted: seq = 1, ack = 0, checksum = e1e0, iiiiiiiiiiiiiiiiiii
Sent: seq = 0, ack = 0, checksum = afaf, nnnnnnnnnnnnnnnnnn
Accpeted: seq = 0, ack = 0, checksum = afaf, nnnnnnnnnnnnnnnnnn
Sent: seq = 1, ack = 0, checksum = 9b9a, pppppppppppppppppp
Accpeted: seq = 1, ack = 0, checksum = 9b9a, pppppppppppppppppp
Sent: seq = 0, ack = 0, checksum = 7d7d, sssssssssssssssssss
Accpeted: seq = 0, ack = 0, checksum = 7d7d, sssssssssssssssssss
Sent: seq = 1, ack = 0, checksum = 4b4a, xxxxxxxxxxxxxxxxxxx
Accpeted: seq = 1, ack = 0, checksum = 4b4a, xxxxxxxxxxxxxxxxxxx
Sent: seq = 0, ack = 0, checksum = d7d7, jjjjjjjjjjjjjjjjjjj
Accpeted: seq = 0, ack = 0, checksum = d7d7, jjjjjjjjjjjjjjjjjjj
Sent: seq = 1, ack = 0, checksum = c3c2, lllllllllllllllllll
Accpeted: seq = 1, ack = 0, checksum = c3c2, lllllllllllllllllll
Sent: seq = 0, ack = 0, checksum = 8787, rrrrrrrrrrrrrrrrrrr
Accpeted: seq = 0, ack = 0, checksum = 8787, rrrrrrrrrrrrrrrrrrr
Sent: seq = 1, ack = 0, checksum = 5554, wwwwwwwwwwwwwwwwwww
Accpeted: seq = 1, ack = 0, checksum = 5554, wwwwwwwwwwwwwwwwwww
Simulator terminated at time 490.078461
after sending 50 msgs from layer5
Process returned 76 (0x4C) execution time : 13.664 s
Press any key to continue.

3)


"F:\College\Semester 7\Computer Vision\projectscanner\networks\bin\Debug\networks.exe"
----- Stop and Wait Network Simulator Version 1.1 --------
Enter the number of messages to simulate: 50
Enter packet loss probability [enter 0.0 for no loss]:0.0
Enter packet corruption probability [0.0 for no corruption]:0.3
Enter average time between messages from sender's layer5 [> 0.0]:10.0
Enter TRACE:0
Sent: seq = 0, ack = 0, checksum = 3232, aaaaaaaaaaaaaaaaaa
Accpeted: seq = 0, ack = 0, checksum = 3232, aaaaaaaaaaaaaaaaaa
Sent: seq = 1, ack = 0, checksum = 2827, bbbbbbbbbbbbbbbbbb
Accpeted: seq = 1, ack = 0, checksum = 2827, bbbbbbbbbbbbbbbbbb
Sent: seq = 0, ack = 0, checksum = 1e1e, cccccccccccccccccc
Accpeted: seq = 0, ack = 0, checksum = 1e1e, cccccccccccccccccc
Sent: seq = 1, ack = 0, checksum = 1413, ddddddddddddddddddd
Accpeted: seq = 1, ack = 0, checksum = 1413, ddddddddddddddddddd
Sent: seq = 0, ack = 0, checksum = 0, ffffffffffffffffff
Accpeted: seq = 0, ack = 0, checksum = 0, ffffffffffffffffff
Sent: seq = 1, ack = 0, checksum = e1e0, iiiiiiiiiiiiiiiiii
Accpeted: seq = 1, ack = 0, checksum = e1e0, iiiiiiiiiiiiiiiiii
Sent: seq = 0, ack = 0, checksum = cdcd, kkkkkkkkkkkkkkkkkk
Accpeted: seq = 0, ack = 0, checksum = cdcd, kkkkkkkkkkkkkkkkkk
Sent: seq = 1, ack = 0, checksum = 9b9a, pppppppppppppppppp
Accpeted: seq = 1, ack = 0, checksum = 9b9a, pppppppppppppppppp
Sent: seq = 0, ack = 0, checksum = 7d7d, ssssssssssssssssss
Accpeted: seq = 0, ack = 0, checksum = 7d7d, ssssssssssssssssss
Sent: seq = 1, ack = 0, checksum = 6968, uuuuuuuuuuuuuuuuuu
Accpeted: seq = 1, ack = 0, checksum = 6968, uuuuuuuuuuuuuuuuuu
Sent: seq = 0, ack = 0, checksum = 4b4b, xxxxxxxxxxxxxxxxxxxx
Simulator terminated at time 459.030151
after sending 50 msgs from layer5
Process returned 76 (0x4C) execution time : 14.075 s
Press any key to continue.

3- The C source files

```
#include <stdio.h>

#define BIDIRECTIONAL 0    /* change to 1 if you're doing extra credit */
/* and write a routine called B_output */

/* a "msg" is the data unit passed from layer 5 (teachers code) to layer  */
/* 4 (students' code).  It contains the data (characters) to be delivered */
/* to layer 5 via the students transport level protocol entities.         */
struct msg {
        char data[20];
};

/* a packet is the data unit passed from layer 4 (students code) to layer */
/* 3 (teachers code).  Note the pre-defined packet structure, which all   */
/* students must follow. */
struct pkt {
        int seqnum;
        int acknum;
        int checksum;
        char payload[20];
};

/*** STUDENTS WRITE THE NEXT SEVEN ROUTINES ***/

#define TIME_OUT 24.0
#define DEBUG 1

int seq_expect_send;  /* Next sequence number of A */
int seq_expect_recv;  /* Next sequence number of B */
int is_waiting;                 /* Whether side A is waiting */
struct pkt waiting_packet;      /* Packet hold in A */

/* Print payload */
print_pkt(action, packet)
char *action;
struct pkt packet;
{
        printf("%s: ", action);
        printf("seq = %d, ack = %d, checksum = %x, ", packet.seqnum,
packet.acknum, packet.checksum);
        int i;
        for (i = 0; i < 20; i++)
                putchar(packet.payload[i]);
        putchar('\n');
```

```
}

/* Compute checksum */
int compute_check_sum(packet)
struct pkt packet;
{
        int sum = 0, i = 0;
        sum = packet.checksum;
        sum += packet.seqnum;
        sum += packet.acknum;
        sum = (sum >> 16) + (sum & 0xffff);
        for (i = 0; i < 20; i += 2) {
                sum += (packet.payload[i] << 8) + packet.payload[i+1];
                sum = (sum >> 16) + (sum & 0xffff);
        }
        sum = (~sum) & 0xffff;
        return sum;
}

/* called from layer 5, passed the data to be sent to other side */
A_output(message)
struct msg message;
{
        /* If A is waiting, ignore the message */
        if (is_waiting)
                return;
        /* Send packet to B side */
        memcpy(waiting_packet.payload, message.data, sizeof(message.data));
        waiting_packet.seqnum = seq_expect_send;
        waiting_packet.checksum = 0;
        waiting_packet.checksum = compute_check_sum(waiting_packet);
        tolayer3(0, waiting_packet);
        starttimer(0, TIME_OUT);
        is_waiting = 1;
        /* Debug output */
        if (DEBUG)
                print_pkt("Sent", waiting_packet);
}

B_output(message)  /* need be completed only for extra credit */
struct msg message;
{

}

/* called from layer 3, when a packet arrives for layer 4 */
```

```
A_input(packet)
struct pkt packet;
{
        stoptimer(0);
        if (packet.acknum == seq_expect_send) {    /* ACK */
                seq_expect_send = 1 - seq_expect_send;
                is_waiting = 0;
        } else if (packet.acknum == -1) {              /* NAK */
                tolayer3(0, waiting_packet);
                starttimer(0, TIME_OUT);
        }
}

/* called when A's timer goes off */
A_timerinterrupt()
{
        tolayer3(0, waiting_packet);
        starttimer(0, TIME_OUT);
}

/* the following routine will be called once (only) before any other */
/* entity A routines are called. You can use it to do any initialization */
A_init()
{
        seq_expect_send = 0;
        is_waiting = 0;
}


/* Note that with simplex transfer from a-to-B, there is no B_output() */

/* called from layer 3, when a packet arrives for layer 4 at B*/
B_input(packet)
struct pkt packet;
{
        if (packet.seqnum == seq_expect_recv) {
                /* If corruption occurs, send NAK */
                if (compute_check_sum(packet)) {
                        struct pkt nakpkt;
                        nakpkt.acknum = -1;
                        tolayer3(1, nakpkt);
                        return;
                }
                /* Pass data to layer5 */
                struct msg message;
                memcpy(message.data, packet.payload, sizeof(packet.payload));
```

```
                tolayer5(1, message);
                seq_expect_recv = 1 - seq_expect_recv;
                /* Debug output */
                if (DEBUG)
                        print_pkt("Accpeted", packet);
        }
        /* Send ACK to A side */
        struct pkt ackpkt;
        ackpkt.acknum = packet.seqnum;
        tolayer3(1, ackpkt);
}

/* called when B's timer goes off */
B_timerinterrupt()
{

}

/* the following rouytine will be called once (only) before any other */
/* entity B routines are called. You can use it to do any initialization */
B_init()
{
        seq_expect_recv = 0;
}


/************************
****** NETWORK EMULATION CODE STARTS BELOW ****
The code below emulates the layer 3 and below network environment:
  - emulates the tranmission and delivery (possibly with bit-level corruption
    and packet loss) of packets across the layer 3/4 interface
  - handles the starting/stopping of a timer, and generates timer
    interrupts (resulting in calling students timer handler).
  - generates message to be sent (passed from later 5 to 4)
THERE IS NOT REASON THAT ANY STUDENT SHOULD HAVE TO READ OR
UNDERSTAND
THE CODE BELOW.  YOU SHOLD NOT TOUCH, OR REFERENCE (in your code)
ANY
OF THE DATA STRUCTURES BELOW.  If you're interested in how I designed
the emulator, you're welcome to look at the code - but again, you should have
to, and you defeinitely should not have to modify
************************/

struct event {
        float evtime;           /* event time */
        int evtype;             /* event type code */
```

```
        int eventity;          /* entity where event occurs */
        struct pkt pktptr;     / ptr to packet (if any) assoc w/ this event */
        struct event *prev;
        struct event *next;
};
struct event evlist = NULL;   / the event list */

/* possible events: */
#define  TIMER_INTERRUPT 0
#define  FROM_LAYER5     1
#define  FROM_LAYER3     2

#define  OFF             0
#define  ON              1
#define   A    0
#define   B    1



int TRACE = 1;             /* for my debugging */
int nsim = 0;             /* number of messages from 5 to 4 so far */
int nsimmax = 0;           /* number of msgs to generate, then stop */
float time = 0.000;
float lossprob;           /* probability that a packet is dropped  */
float corruptprob;        /* probability that one bit is packet is flipped */
float lambda;             /* arrival rate of messages from layer 5 */
int   ntolayer3;          /* number sent into layer 3 */
int   nlost;             /* number lost in media */
int ncorrupt;             /* number corrupted by media*/

main()
{
        struct event *eventptr;
        struct msg  msg2give;
        struct pkt  pkt2give;

        int i, j;
        char c;

        init();
        A_init();
        B_init();

        while (1) {
                eventptr = evlist;          /* get next event to simulate */
                if (eventptr == NULL)
```

```
        goto terminate;
evlist = evlist->next;        /* remove this event from event list */
if (evlist != NULL)
        evlist->prev = NULL;
if (TRACE >= 2) {
        printf("\nEVENT time: %f,", eventptr->evtime);
        printf("  type: %d", eventptr->evtype);
        if (eventptr->evtype == 0)
                printf(", timerinterrupt  ");
        else if (eventptr->evtype == 1)
                printf(", fromlayer5 ");
        else
                printf(", fromlayer3 ");
        printf(" entity: %d\n", eventptr->eventity);
}
time = eventptr->evtime;        /* update time to next event time */
if (nsim == nsimmax)
        break;                  /* all done with simulation */
if (eventptr->evtype == FROM_LAYER5 ) {
        generate_next_arrival();   /* set up future arrival */
        /* fill in msg to give with string of same letter */
        j = nsim % 26;
        for (i = 0; i < 20; i++)
                msg2give.data[i] = 97 + j;
        if (TRACE > 2) {
                printf("        MAINLOOP: data given to student: ");
                for (i = 0; i < 20; i++)
                        printf("%c", msg2give.data[i]);
                printf("\n");
        }
        nsim++;
        if (eventptr->eventity == A)
                A_output(msg2give);
        else
                B_output(msg2give);
}
else if (eventptr->evtype ==  FROM_LAYER3) {
        pkt2give.seqnum = eventptr->pktptr->seqnum;
        pkt2give.acknum = eventptr->pktptr->acknum;
        pkt2give.checksum = eventptr->pktptr->checksum;
        for (i = 0; i < 20; i++)
                pkt2give.payload[i] = eventptr->pktptr->payload[i];
        if (eventptr->eventity == A)     /* deliver packet by calling */
                A_input(pkt2give);          /* appropriate entity */
        else
                B_input(pkt2give);
```

```
                free(eventptr->pktptr);         /* free the memory for packet */
        }
        else if (eventptr->evtype ==  TIMER_INTERRUPT) {
                if (eventptr->eventity == A)
                        A_timerinterrupt();
                else
                        B_timerinterrupt();
        }
        else  {
                printf("INTERNAL PANIC: unknown event type \n");
        }
        free(eventptr);
    }

terminate:
    printf(" Simulator terminated at time %f\n after sending %d msgs from
layer5\n", time, nsim);
}



init()                  /* initialize the simulator */
{
    int i;
    float sum, avg;
    float jimsrand();


    printf("-----  Stop and Wait Network Simulator Version 1.1 -------- \n\n");
    printf("Enter the number of messages to simulate: ");
    scanf("%d", &nsimmax);
    printf("Enter  packet loss probability [enter 0.0 for no loss]:");
    scanf("%f", &lossprob);
    printf("Enter packet corruption probability [0.0 for no corruption]:");
    scanf("%f", &corruptprob);
    printf("Enter average time between messages from sender's layer5 [ > 0.0]:");
    scanf("%f", &lambda);
    printf("Enter TRACE:");
    scanf("%d", &TRACE);

    srand(9999);              /* init random number generator */
    sum = 0.0;                /* test random number generator for students */
    for (i = 0; i < 1000; i++)
        sum = sum + jimsrand(); /* jimsrand() should be uniform in [0,1] */
    avg = sum / 1000.0;
    if (avg < 0.25 || avg > 0.75) {
```

```
                printf("It is likely that random number generation on your machine\n"
);
                printf("is different from what this emulator expects.  Please take\n");
                printf("a look at the routine jimsrand() in the emulator code. Sorry.
\n");
                exit(0);
        }

        ntolayer3 = 0;
        nlost = 0;
        ncorrupt = 0;

        time = 0.0;             /* initialize time to 0.0 */
        generate_next_arrival();     /* initialize event list */
}

/***************************/
/* jimsrand(): return a float in range [0,1].  The routine below is used to */
/* isolate all random number generation in one location.  We assume that the*/
/* system-supplied rand() function return an int in therange [0,mmm]        */
/***************************/
float jimsrand()
{
        double mmm = 32767;           /* largest int  - MACHINE
DEPENDENT!!!!!!!!   */
        float x;                /* individual students may need to change mmm */
        x = rand() / mmm;         /* x should be uniform in [0,1] */
        return (x);
}

/******* EVENT HANDLINE ROUTINES ***/
/*  The next set of routines handle the event list   */
/*********************/

generate_next_arrival()
{
        double x, log(), ceil();
        struct event *evptr;
        char *malloc();
        float ttime;
        int tempint;

        if (TRACE > 2)
                printf("        GENERATE NEXT ARRIVAL: creating new arrival\n");

        x = lambda * jimsrand() * 2; /* x is uniform on [0,2*lambda] */
```

```
        /* having mean of lambda        */
        evptr = (struct event *)malloc(sizeof(struct event));
        evptr->evtime =  time + x;
        evptr->evtype =  FROM_LAYER5;
        if (BIDIRECTIONAL && (jimsrand() > 0.5) )
                evptr->eventity = B;
        else
                evptr->eventity = A;
        insertevent(evptr);
}


insertevent(p)
struct event *p;
{
        struct event *q, *qold;

        if (TRACE > 2) {
                printf("        INSERTEVENT: time is %lf\n", time);
                printf("        INSERTEVENT: future time will be %lf\n", p->evtime);
        }
        q = evlist;     /* q points to header of list in which p struct inserted */
        if (q == NULL) { /* list is empty */
                evlist = p;
                p->next = NULL;
                p->prev = NULL;
        }
        else {
                for (qold = q; q != NULL && p->evtime > q->evtime; q = q->next)
                        qold = q;
                if (q == NULL) { /* end of list */
                        qold->next = p;
                        p->prev = qold;
                        p->next = NULL;
                }
                else if (q == evlist) { /* front of list */
                        p->next = evlist;
                        p->prev = NULL;
                        p->next->prev = p;
                        evlist = p;
                }
                else {     /* middle of list */
                        p->next = q;
                        p->prev = q->prev;
                        q->prev->next = p;
                        q->prev = p;
```

```
                }
        }
}

printevlist()
{
        struct event *q;
        int i;
        printf("--------------\nEvent List Follows:\n");
        for (q = evlist; q != NULL; q = q->next) {
                printf("Event time: %f, type: %d entity: %d\n", q->evtime, q->evtype, q-
>eventity);
        }
        printf("--------------\n");
}



/******** Student-callable ROUTINES *********/

/* called by students routine to cancel a previously-started timer */
stoptimer(AorB)
int AorB;  /* A or B is trying to stop timer */
{
        struct event *q, *qold;

        if (TRACE > 2)
                printf("        STOP TIMER: stopping timer at %f\n", time);
        /* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next)  */
        for (q = evlist; q != NULL ; q = q->next)
                if ( (q->evtype == TIMER_INTERRUPT  && q->eventity == AorB) ) {
                        /* remove this event */
                        if (q->next == NULL && q->prev == NULL)
                                evlist = NULL;      /* remove first and only event on list */
                        else if (q->next == NULL) /* end of list - there is one in front */
                                q->prev->next = NULL;
                        else if (q == evlist) { /* front of list - there must be event after */
                                q->next->prev = NULL;
                                evlist = q->next;
                        }
                        else {    /* middle of list */
                                q->next->prev = q->prev;
                                q->prev->next =  q->next;
                        }
                        free(q);
                        return;
```

```
                }
        printf("Warning: unable to cancel your timer. It wasn't running.\n");
}


starttimer(AorB, increment)
int AorB;  /* A or B is trying to stop timer */
float increment;
{

        struct event *q;
        struct event *evptr;
        char *malloc();

        if (TRACE > 2)
                printf("        START TIMER: starting timer at %f\n", time);
        /* be nice: check to see if timer is already started, if so, then  warn */
        /* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next)  */
        for (q = evlist; q != NULL ; q = q->next)
                if ( (q->evtype == TIMER_INTERRUPT  && q->eventity == AorB) ) {
                        printf("Warning: attempt to start a timer that is already
started\n");
                        return;
                }

        /* create future event for when timer goes off */
        evptr = (struct event *)malloc(sizeof(struct event));
        evptr->evtime =  time + increment;
        evptr->evtype =  TIMER_INTERRUPT;
        evptr->eventity = AorB;
        insertevent(evptr);
}


/********** TOLAYER3 ******/
tolayer3(AorB, packet)
int AorB;  /* A or B is trying to stop timer */
struct pkt packet;
{
        struct pkt *mypktptr;
        struct event *evptr, *q;
        char *malloc();
        float lastime, x, jimsrand();
        int i;
```

```
        ntolayer3++;

        /* simulate losses: */
        if (jimsrand() < lossprob)  {
                nlost++;
                if (TRACE > 0)
                        printf("        TOLAYER3: packet being lost\n");
                return;
        }

        /* make a copy of the packet student just gave me since he/she may decide */
        /* to do something with the packet after we return back to him/her */
        mypktptr = (struct pkt *)malloc(sizeof(struct pkt));
        mypktptr->seqnum = packet.seqnum;
        mypktptr->acknum = packet.acknum;
        mypktptr->checksum = packet.checksum;
        for (i = 0; i < 20; i++)
                mypktptr->payload[i] = packet.payload[i];
        if (TRACE > 2)  {
                printf("        TOLAYER3: seq: %d, ack %d, check: %d ", mypktptr-
>seqnum,
                     mypktptr->acknum,  mypktptr->checksum);
                for (i = 0; i < 20; i++)
                        printf("%c", mypktptr->payload[i]);
                printf("\n");
        }

        /* create future event for arrival of packet at the other side */
        evptr = (struct event *)malloc(sizeof(struct event));
        evptr->evtype =  FROM_LAYER3;   /* packet will pop out from layer3 */
        evptr->eventity = (AorB + 1) % 2; /* event occurs at other entity */
        evptr->pktptr = mypktptr;       /* save ptr to my copy of packet */
        /* finally, compute the arrival time of packet at the other end.
          medium can not reorder, so make sure packet arrives between 1 and 10
          time units after the latest arrival time of packets
          currently in the medium on their way to the destination */
        lastime = time;
        /* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next) */
        for (q = evlist; q != NULL ; q = q->next)
                if ( (q->evtype == FROM_LAYER3  && q->eventity == evptr->eventity)
)
                        lastime = q->evtime;
        evptr->evtime =  lastime + 1 + 9 * jimsrand();
```

```
        /* simulate corruption: */
        if (jimsrand() < corruptprob)  {
                ncorrupt++;
                if ( (x = jimsrand()) < .75)
                        mypktptr->payload[0] = 'Z'; /* corrupt payload */
                else if (x < .875)
                        mypktptr->seqnum = 999999;
                else
                        mypktptr->acknum = 999999;
                if (TRACE > 0)
                        printf("        TOLAYER3: packet being corrupted\n");
        }

        if (TRACE > 2)
                printf("        TOLAYER3: scheduling arrival on other side\n");
        insertevent(evptr);
}

tolayer5(AorB, datasent)
int AorB;
char datasent[20];
{
        int i;
        if (TRACE > 2) {
                printf("        TOLAYER5: data received: ");
                for (i = 0; i < 20; i++)
                        printf("%c", datasent[i]);
                printf("\n");
        }

}
```