Wireshark Lab 10

ALTERNATING BIT AND GO-BACK-N NETWORK EMULATOR

Suraj Durgesht

(16EC06)

Instructor: Dr. Rajendra Hegadi

#include <stdio.h>#include <stdlib.h>

#include <time.h>

#include <string.h>

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALTERNATING BIT AND GO-BACK-N NETWORK EMULATOR:

VERSION 1.1 J.F.Kurose

Revised 1.2 D.R.Figueiredo This code should be used for PA2, unidirectional data transfer

protocols (from A to B). Network properties:

- one way network delay averages some number of 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).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\* 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 (using the message struct) 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];

};

void starttimer(int, float);

void stoptimer(int);

void tolayer3(int,struct pkt);

void tolayer5(int, struct msg);

int check\_checksum(struct pkt);

int generate\_checksum(struct pkt);

int flip\_number(int);

/\*\*\*\*\*\*\*\*\* STUDENTS WRITE THE NEXT SEVEN ROUTINES \*\*\*\*\*\*\*\*\*/#define A 0

#define B 1

#define TIMER 20

#define MESSAGE\_SIZE 20

int A\_STATE = 0;

int B\_STATE = 0;int count = 0;int ACK = 0;

int SEQ = 0;

int prev\_sequence\_number;

struct msg prev\_message;

struct pkt prev\_packet;

struct pkt B\_prev\_packet;

/\* called from layer 5, passed the data to be sent to other side. Return a 1 if

data is accepted for transmission, negative number otherwise \*/

int A\_output(message)

struct msg message;

{

if(A\_STATE){

printf("\nA will get ACK : %d",ACK);

return -1;

}else{ struct pkt packet;

strcpy(packet.payload,message.data);

packet.seqnum = SEQ;

packet.acknum = flip\_number(ACK);

packet.checksum = generate\_checksum(packet); prev\_packet = packet;

A\_STATE = flip\_number(A\_STATE);

SEQ = flip\_number(SEQ); printf("\nA sends packet details:\nSeq no: %d\nChecksum: %d\nData: %s\nA expected of ACK:%d",packet.seqnum,packet.checksum,packet.payload,ACK);

tolayer3(A, packet);

starttimer(A,TIMER);

return 1;

}

}void A\_input(packet)

struct pkt packet;

{

printf("\nA has received ACK packet details:\nACK: %d\nCurrent expected ACK: %d",packet.acknum,ACK);

if(packet.acknum == ACK && check\_checksum(packet)){

stoptimer(A); struct msg message;

strcpy(message.data,packet.payload); ACK = flip\_number(ACK);

A\_STATE = flip\_number(A\_STATE);

tolayer5(A,message);

}

}/\* called when A's timer goes off \*/

void A\_timerinterrupt()

{

printf("\nA has timeout, so retransmit the packet.\n");

tolayer3(A,prev\_packet);

starttimer(A,TIMER);

} void A\_init()

{

A\_STATE = 0;

ACK = 0;

SEQ = 0;

}/\* used to check the checksum \*/

int check\_checksum(struct pkt package){

int i;

int sum = (package.seqnum + package.acknum);

for(i = 0; i < MESSAGE\_SIZE; i++){

sum += (int)package.payload[i];

}

return (sum == package.checksum);

}

/\* used to generate checksum \*/

int generate\_checksum(struct pkt package){

int i;

int sum = (package.seqnum + package.acknum);

for(i = 0; i < MESSAGE\_SIZE; i++){

sum += (int)package.payload[i];

}

return sum;

}

/\* used to alternate the number\*/

int flip\_number(int number){

if(number == 0) return 1;

else return 0;

}/\* called from layer 3, when a packet arrives for layer 4 at B\*/

void B\_input(packet)

struct pkt packet;

{

printf("\nB received packet Details:\nSeq no: %d\nChecksum: %d\nData: %s\nACK B is expected:%d",packet.seqnum,packet.checksum,packet.payload,ACK);

if(check\_checksum(packet) && packet.seqnum == B\_STATE){

struct msg message;

strcpy(message.data,packet.payload);

B\_STATE = flip\_number(B\_STATE); struct pkt ack\_packet;

ack\_packet.acknum = packet.seqnum;

ack\_packet.checksum = generate\_checksum(ack\_packet); B\_prev\_packet = ack\_packet; printf("\nB is sending the ACK"); count++;

tolayer5(B,message);

tolayer3(B, ack\_packet);

stoptimer(B);

starttimer(B,TIMER);

}else{printf("\nB is expecting the next:%d ",count);

}}/\* called when B's timer goes off \*/

void B\_timerinterrupt()

{

printf("\nB has timeout, so retransmit the packet.");

tolayer3(B,B\_prev\_packet);

starttimer(B,TIMER);

}/\* the following rouytine will be called once (only) before any other \*/

/\* entity B routines are called. You can use it to do any initialization \*/

void B\_init()

{

B\_STATE = 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 definitely should not have to modify anything.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/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 \*//\* use for bidirectional transfer of data \*/

#define BIDIRECTIONAL 0 /\* possible events: \*/

#define TIMER\_INTERRUPT 0

#define FROM\_LAYER5 1

#define FROM\_LAYER3 2#define A 0

#define B 1

int TRACE = 1; /\* for my debugging \*/

int nsim = 0; /\* number of messages from layer 5 to 4 so far \*/

int nsimmax = 0; /\* number of msgs to generate, then stop \*/

float simul\_time = 0.000; /\* global simulation simul\_time \*/

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\*/

int randseed; /\* random number seed \*/

/\* use only for biderectional data transfer \*/

int B\_output(message) /\* need be completed only for extra credit \*/

struct msg message;

{ return 0;

}/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EVENT LIST ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Event list manipulation routines \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void insertevent(p)

struct event \*p;

{

struct event \*q,\*qold; if (TRACE>2) {

printf(" INSERTEVENT: time is %lf\n",simul\_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;

}

}

}void printevlist()

{

struct event \*q;

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 \*/

void stoptimer(AorB)

int AorB; /\* A or B is trying to stop timer \*/

{

struct event \*q; if (TRACE>2)

printf(" STOP TIMER: stopping timer at %f\n",simul\_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");

}void starttimer(AorB,increment)

int AorB; /\* A or B is trying to stop timer \*/

float increment;

{ struct event \*q;

struct event \*evptr; if (TRACE>2)

printf(" START TIMER: starting timer at %f\n",simul\_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 = simul\_time + increment;

evptr->evtype = TIMER\_INTERRUPT;

evptr->eventity = AorB;

insertevent(evptr);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TOLAYER3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double random\_number() { // generate a uniform random number in the interval [0,1)

return (double)1.0\*rand()/(RAND\_MAX+1.0);

}void init\_random(unsigned int randseed) {

// initialize the random number generator

if (!randseed) {

srand((unsigned int)time(NULL));

} else

srand(randseed);

}void tolayer3(AorB,packet)

int AorB; /\* A or B is trying to stop timer \*/

struct pkt packet;

{

struct pkt \*mypktptr;

struct event \*evptr,\*q;

float lastime, x;

int i; ntolayer3++; /\* simulate losses: \*/

if (random\_number() < 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 = simul\_time;

for (q=evlist; q!=NULL ; q = q->next)

if ( (q->evtype==FROM\_LAYER3 && q->eventity==evptr->eventity) )

lastime = q->evtime;

evptr->evtime = lastime + 1.0 + 9.0\*random\_number(); /\* simulate corruption: \*/

if (random\_number() < corruptprob) {

ncorrupt++;

if ( (x = random\_number()) < .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);

} void tolayer5(AorB, msgReceived)

int AorB;

struct msg msgReceived;

{

int i;

if (TRACE>2) {

printf(" TOLAYER5: data received: ");

for (i=0; i<20; i++)

printf("%c",msgReceived.data[i]);

printf("\n");

}}/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EVENT HANDLINE ROUTINES \*\*\*\*\*\*\*/

/\* The next set of routines handle the event list \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/void generate\_next\_arrival(int entity)

{

double x;

struct event \*evptr; if (TRACE>2)

printf(" GENERATE NEXT ARRIVAL: creating new arrival\n"); x = lambda\*random\_number()\*2; /\* x is uniform on [0,2\*lambda] \*/

/\* having mean of lambda \*/

evptr = (struct event \*)malloc(sizeof(struct event));

evptr->evtime = simul\_time + x;

evptr->evtype = FROM\_LAYER5; if (entity)

evptr->eventity = entity;

else {

if (BIDIRECTIONAL && (random\_number()>0.5) )

evptr->eventity = B;

else

evptr->eventity = A;

}

insertevent(evptr);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INITIALIZATION ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Read input from user and initalize parameters \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init()

{

int i;

float sum, avg; 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 a seed for the random number generator [0 will provide a random seed]: ");

//scanf("%d",&randseed);

printf("Enter TRACE [0,1,2,3]: ");

scanf("%d",&TRACE); /\* init random number generator \*/

init\_random(randseed); sum = 0.0; /\* test random number generator for students \*/

for (i=0; i<1000; i++)

sum=sum+random\_number(); /\* 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 random\_number() in the emulator code. Sorry. \n");

exit(0);

}

ntolayer3 = 0;

nlost = 0;

ncorrupt = 0;

simul\_time=0.0; /\* initialize simul\_time to 0.0 \*/

generate\_next\_arrival(0); /\* initialize event list \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Main simulation loop and handling of events \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

struct event \*eventptr;

struct msg msg2give;

struct pkt pkt2give;

int i,j;

/\* initialize our data structures and read parameters \*/

init();

/\* call the user's init functions \*/

A\_init();

B\_init();

/\* loop forever... \*/

while (1) {

eventptr = evlist; /\* get next event to simulate \*/

if (eventptr==NULL) {

printf("INTERNAL PANIC: Event list is empty! This should NOT have happened.\n");

break;

}

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);

}

simul\_time = eventptr->evtime; /\* update simul\_time to next event time \*/

if (nsim==nsimmax)

break; /\* all done with simulation \*/

if (eventptr->evtype == FROM\_LAYER5 ) {

/\* 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("\nMAINLOOP: data given to student: ");

for (i=0; i<20; i++)

printf("%c", msg2give.data[i]);

printf("\n");

}

if (eventptr->eventity == A)

j = A\_output(msg2give);

else

j = B\_output(msg2give);

if (j < 0) {

if (TRACE>=1)

printf("\nMAINLOOP: data NOT accepted by layer 4 (student code)\n");

/\* set up future arrival for the same entity\*/

generate\_next\_arrival(eventptr->eventity); } else {

nsim++;

if (TRACE>=1)

printf("\nMAINLOOP: data accepted by layer 4 (student code)\n");

/\* set up future arrival \*/

generate\_next\_arrival(0);

}

}

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");

break;

}

free(eventptr);

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

return 0;

}





