<Cover Page>

ECE358: Computer Networks

Winter 2014

Project 2: Data Link Layers and ARQ Protocols

Date of submission:

Submitted by: Dhruv Chopra

Student ID: 20392338

Student name: Chopra,Dhruv

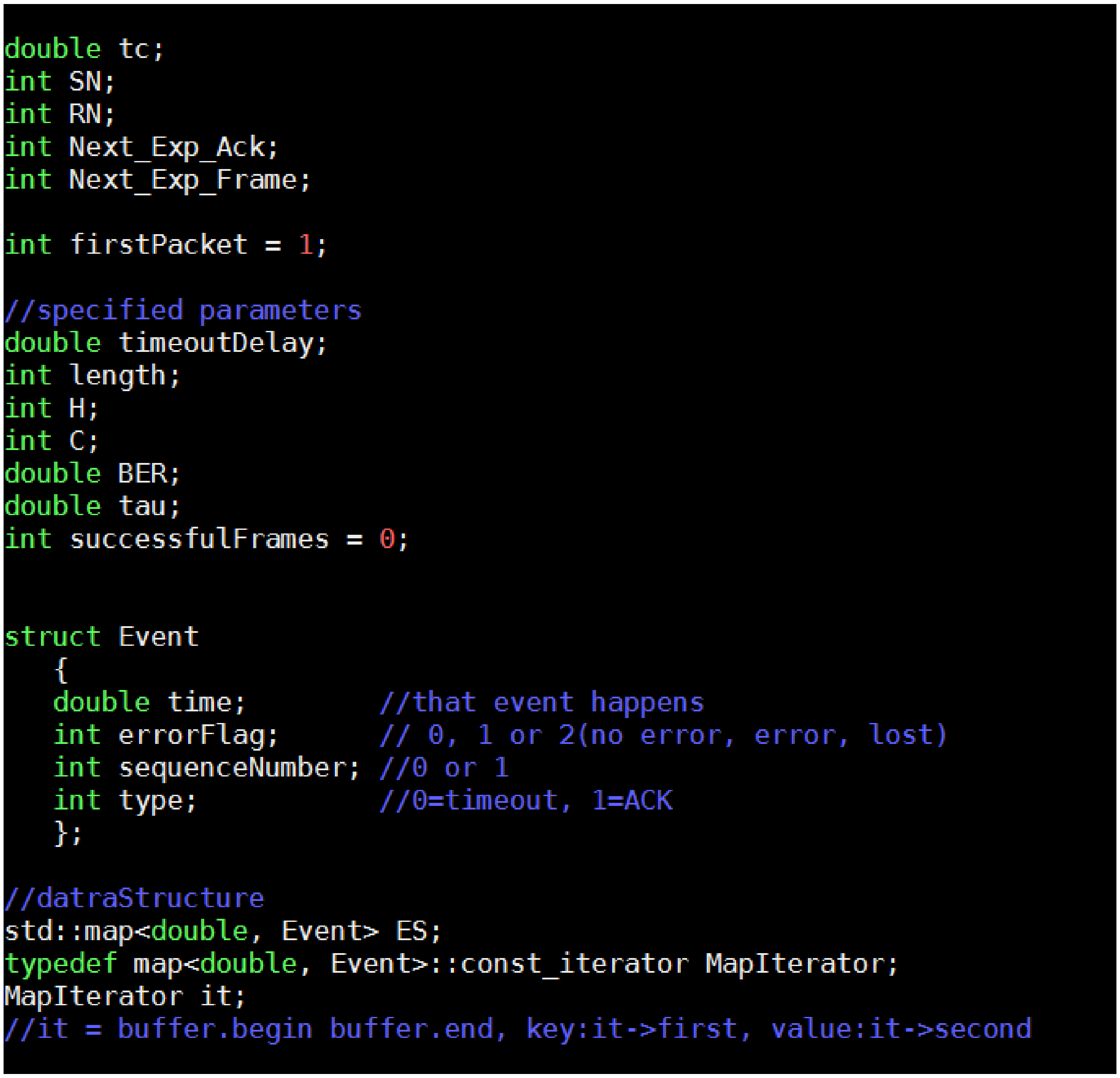
Waterloo Email address: dchopra@uwaterloo.ca

Marks received:

Marked by:

Question 1 - ABP Simulator

The basic ABP simulator was implemented using the dataflow diagram given by the lab instructor. Everything is implemented in one C++ file (abp.cpp). First some global variables are initialized as shown in the Figure 1 below.

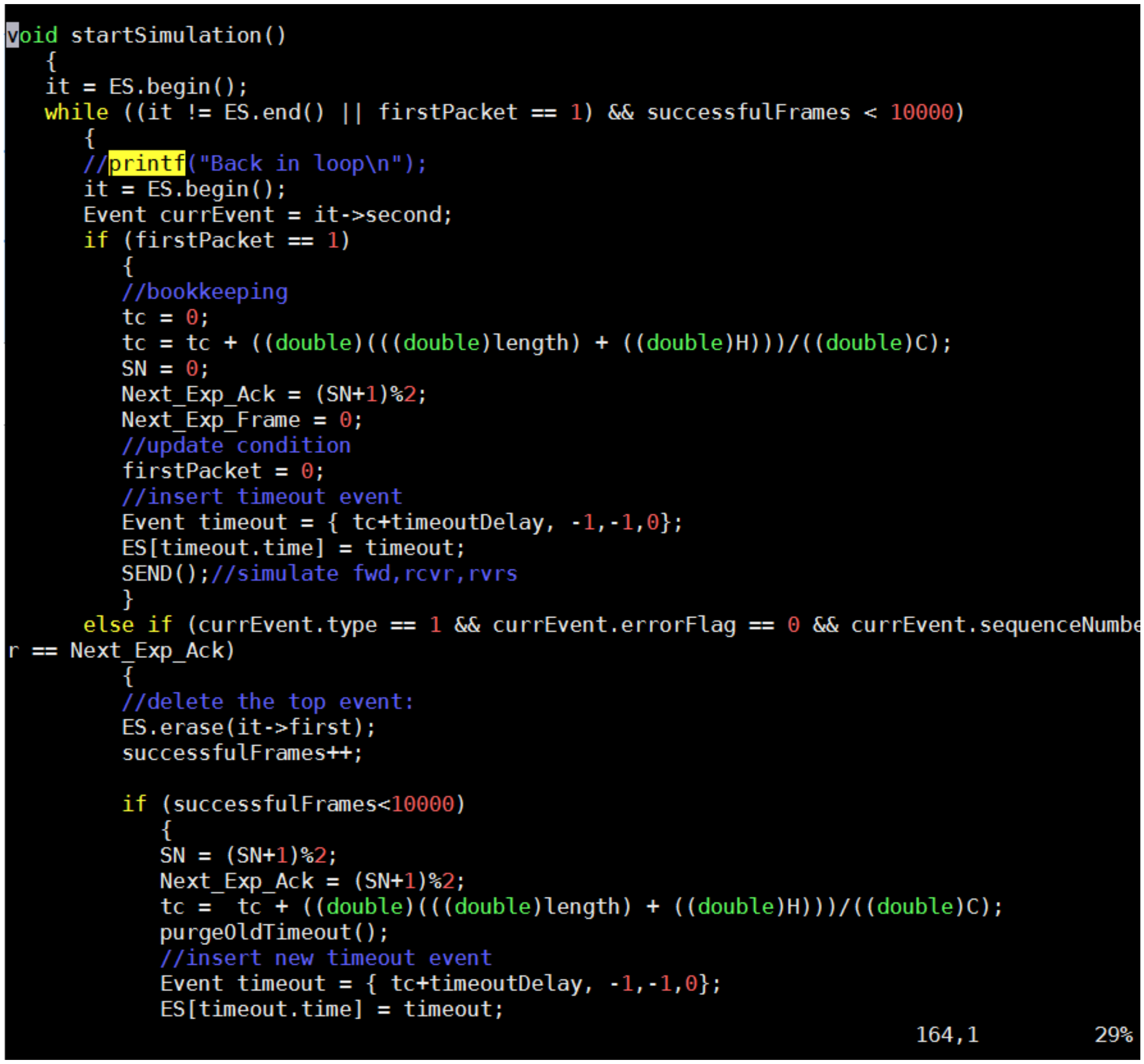


```
double tc;
int SN;
int RN;
int Next_Exp_Ack;
int Next_Exp_Frame;

int firstPacket = 1;

//specified parameters
double timeoutDelay;
int length;
int H;
int C;
double BER;
double tau;
int successfulFrames = 0;


struct Event
   {
   double time;        //that event happens
   int errorFlag;      // 0, 1 or 2(no error, error, lost)
   int sequenceNumber; //0 or 1
   int type;           //0=timeout, 1=ACK
   };

//datraStructure
std::map<double, Event> ES;
typedef map<double, Event>::const_iterator MapIterator;
MapIterator it;
//it = buffer.begin buffer.end, key:it->first, value:it->second
```

Figure 1: Initialization of global variables

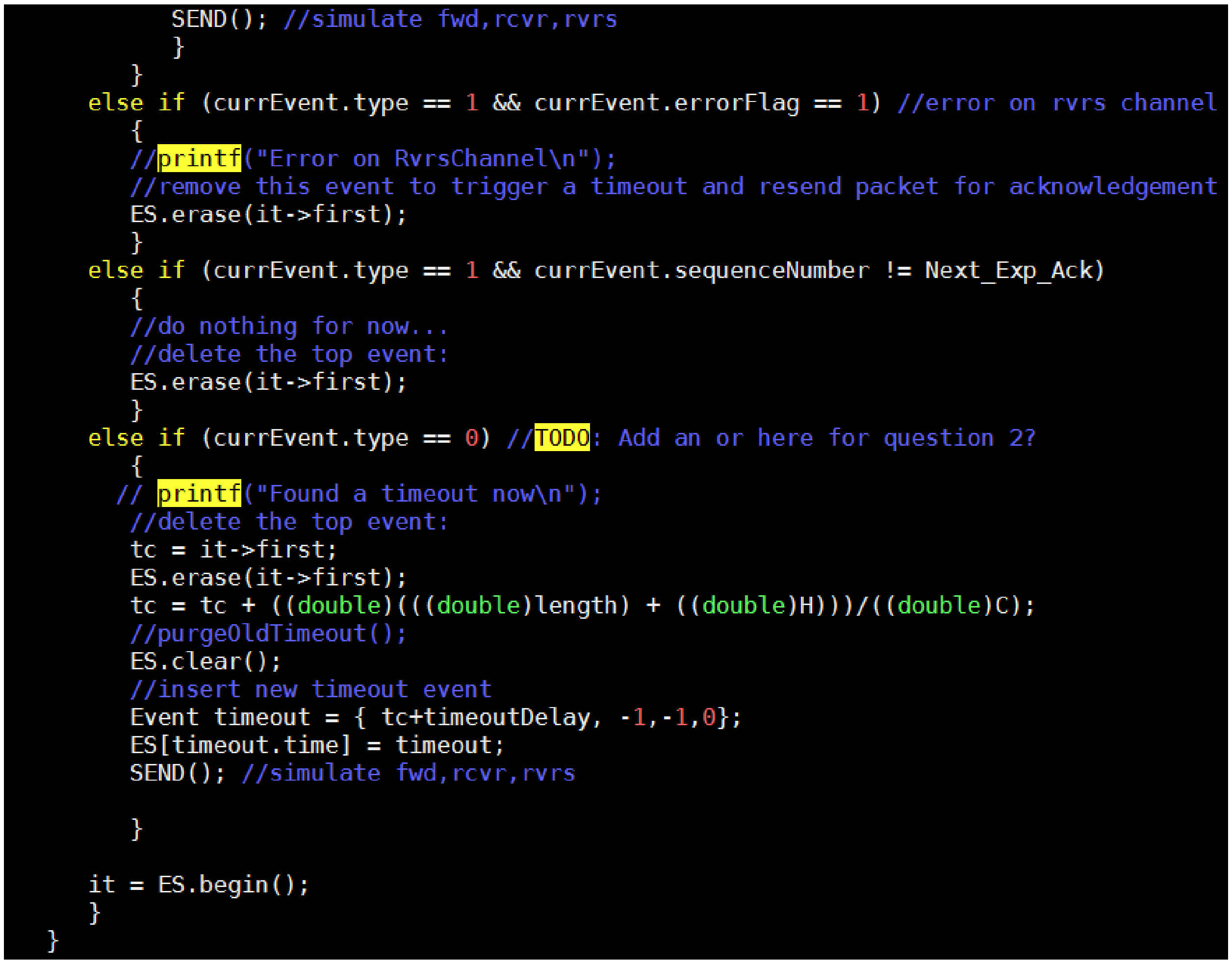
The event scheduler "ES" is a map which will sort values based on the key whenever a new value is inserted (Figure 1 above). The variable "firstPacket" is used to access an if condition when initializing the simulator and the variable "successfulFrames" is used to stop simulation once 10,000 frames have been successfully sent. The rest of the declaration in Figure 1 above do exactly as was specified in the lab manual.

Simulation is started using the "startSimulation()" method. Its implementation is shown in Figure 2 and 3 below.



```
void startSimulation()
   {
   it = ES.begin();
   while ((it != ES.end() || firstPacket == 1) && successfulFrames < 10000)
      {
      //printf("Back in loop\n");
      it = ES.begin();
      Event currEvent = it->second;
      if (firstPacket == 1)
         {
         //bookkeeping
         tc = 0;
         tc = tc + ((double)(((double)length) + ((double)H)))/((double)C);
         SN = 0;
         Next_Exp_Ack = (SN+1)%2;
         Next_Exp_Frame = 0;
         //update condition
         firstPacket = 0;
         //insert timeout event
         Event timeout = { tc+timeoutDelay, -1,-1,0};
         ES[timeout.time] = timeout;
         SEND();//simulate fwd,rcvr,rvrs
         }
      else if (currEvent.type == 1 && currEvent.errorFlag == 0 && currEvent.sequenceNumbe
r == Next_Exp_Ack)
         {
         //delete the top event:
         ES.erase(it->first);
         successfulFrames++;

         if (successfulFrames<10000)
            {
            SN = (SN+1)%2;
            Next_Exp_Ack = (SN+1)%2;
            tc =  tc + ((double)(((double)length) + ((double)H)))/((double)C);
            purgeOldTimeout();
            //insert new timeout event
            Event timeout = { tc+timeoutDelay, -1,-1,0};
            ES[timeout.time] = timeout;
                                                                   164,1          29%
```

Figure 2: First half of startSimulation() method



```
            SEND(); //simulate fwd,rcvr,rvrs
            }
         }
      else if (currEvent.type == 1 && currEvent.errorFlag == 1) //error on rvrs channel
         {
         //printf("Error on RvrsChannel\n");
         //remove this event to trigger a timeout and resend packet for acknowledgement
         ES.erase(it->first);
         }
      else if (currEvent.type == 1 && currEvent.sequenceNumber != Next_Exp_Ack)
         {
         //do nothing for now...
         //delete the top event:
         ES.erase(it->first);
         }
      else if (currEvent.type == 0) //TODO: Add an or here for question 2?
         {
        // printf("Found a timeout now\n");
         //delete the top event:
         tc = it->first;
         ES.erase(it->first);
         tc = tc + ((double)(((double)length) + ((double)H)))/((double)C);
         //purgeOldTimeout();
         ES.clear();
         //insert new timeout event
         Event timeout = { tc+timeoutDelay, -1,-1,0};
         ES[timeout.time] = timeout;
         SEND(); //simulate fwd,rcvr,rvrs

         }

      it = ES.begin();
      }
   }
```

Figure 3: Second half of startSimulation() method

This method consists of a while loop which is a if-else statement executing inside. This method will continue to execute as long as there is atleast one event in the queue or less than 10,000 frames have been transmitted.

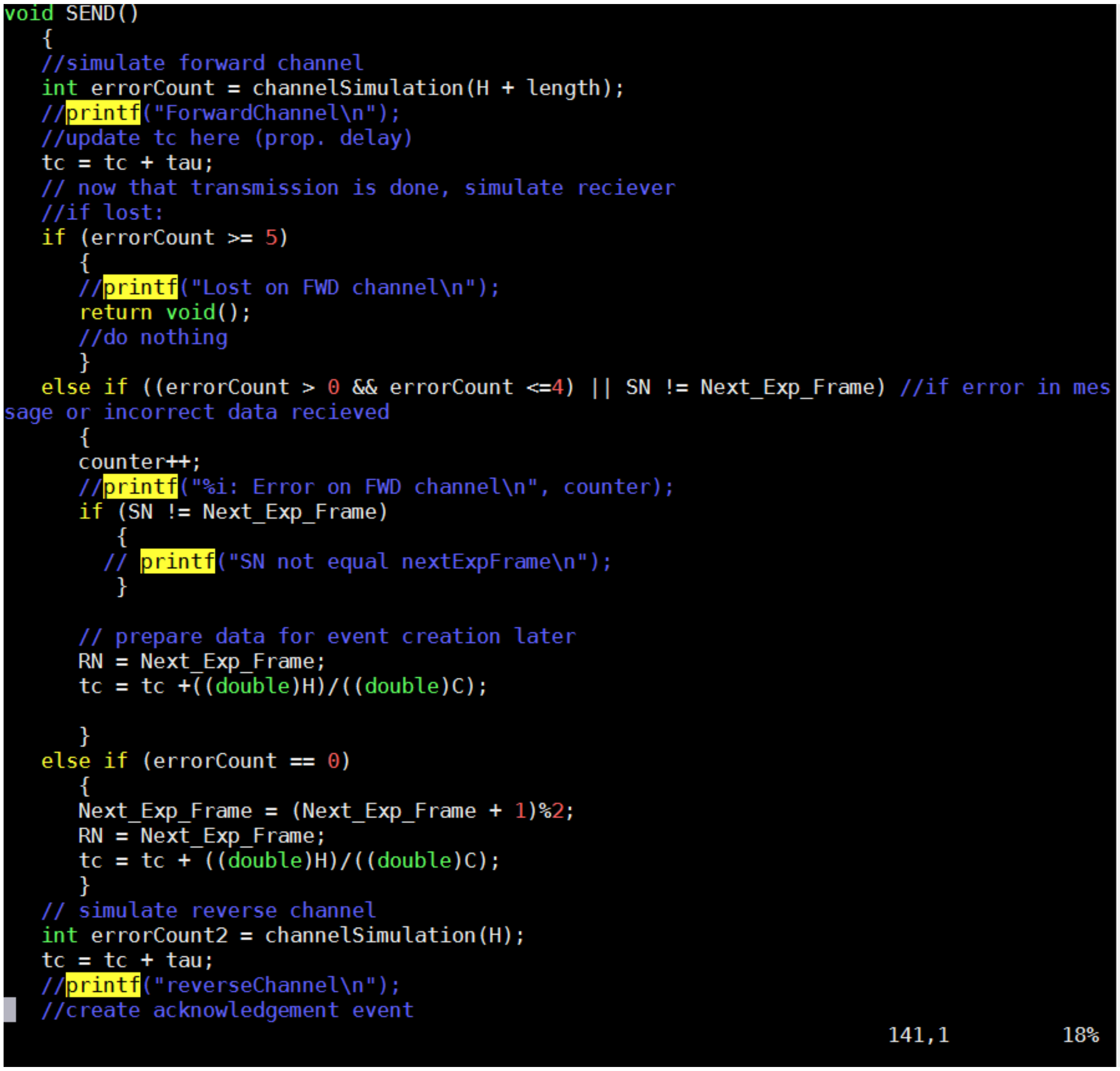
The first if condition will initialize the simulation (by check the firstPacket variable). If so then it will properly initialize all the variables and send the first packet.

The second if condition (the first if-else block) will check if the most recent event in the ES is an acknowledgement that we were expecting. If it is then it will update all the variables accordingly and transmit the next frame.

The next two if conditions check if the most recent event is an acknowledgement but is not a correct acknowledgement(acknowledgement has an error or is not the acknowledgement that the sender was expecting, respectively). In these two cases we do not take any action (except for removing the event from the ES) because we do not treat negative acknowledgements(NAKs) for this part of the lab.

Finally, if the event is a timeout, then the last if condition will account for it. In this case the ES is cleared and the same packet is sent again.

The next major portion of the code is the "SEND()" function. This function will actually simulate the transmission of the packet and the return of the acknowledgement event. It is shown in Figure 4 and 5 below.



```
void SEND()
   {
   //simulate forward channel
   int errorCount = channelSimulation(H + length);
   //printf("ForwardChannel\n");
   //update tc here (prop. delay)
   tc = tc + tau;
   // now that transmission is done, simulate reciever
   //if lost:
   if (errorCount >= 5)
      {
      //printf("Lost on FWD channel\n");
      return void();
      //do nothing
      }
   else if ((errorCount > 0 && errorCount <=4) || SN != Next_Exp_Frame) //if error in mes
sage or incorrect data recieved
      {
      counter++;
      //printf("%i: Error on FWD channel\n", counter);
      if (SN != Next_Exp_Frame)
         {
       // printf("SN not equal nextExpFrame\n");
         }

      // prepare data for event creation later
      RN = Next_Exp_Frame;
      tc = tc +((double)H)/((double)C);

      }
   else if (errorCount == 0)
      {
      Next_Exp_Frame = (Next_Exp_Frame + 1)%2;
      RN = Next_Exp_Frame;
      tc = tc + ((double)H)/((double)C);
      }
   // simulate reverse channel
   int errorCount2 = channelSimulation(H);
   tc = tc + tau;
   //printf("reverseChannel\n");
   //create acknowledgement event
                                                                  141,1          18%
```

Figure 4: First half of the SEND() function

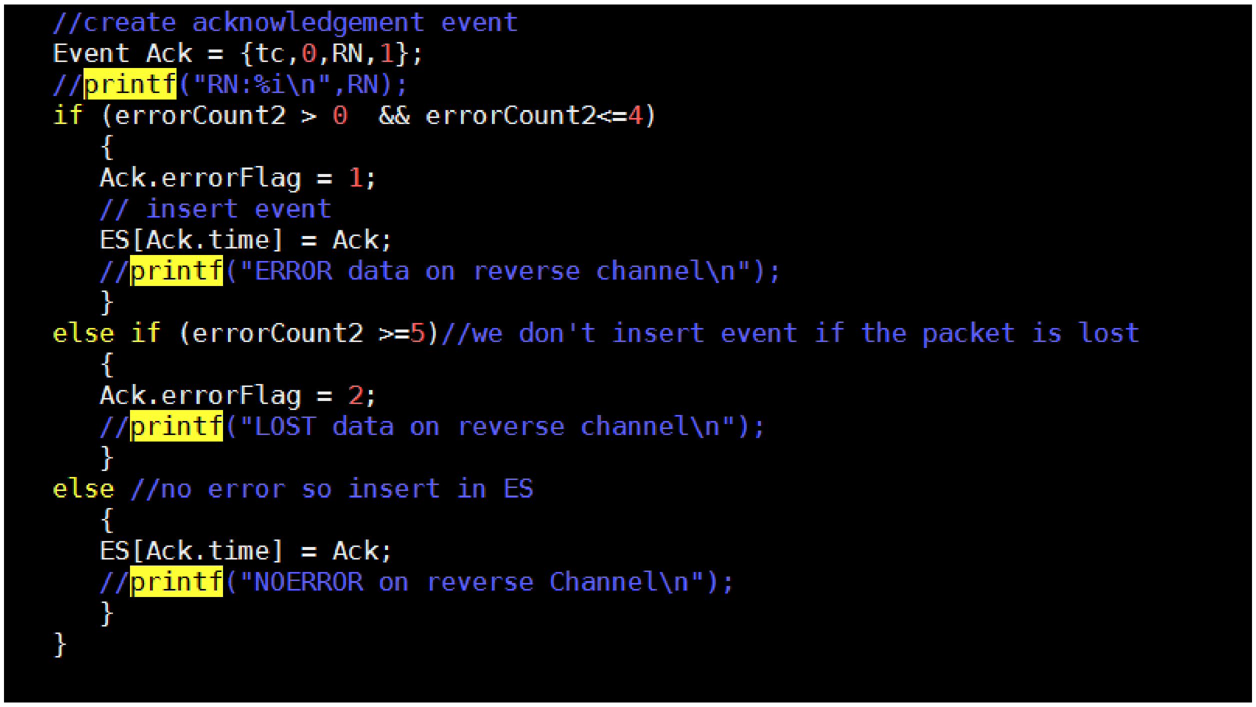

```
    //create acknowledgement event
    Event Ack = {tc,0,RN,1};
    //printf("RN:%i\n",RN);
    if (errorCount2 > 0  && errorCount2<=4)
       {
       Ack.errorFlag = 1;
       // insert event
       ES[Ack.time] = Ack;
       //printf("ERROR data on reverse channel\n");
       }
    else if (errorCount2 >=5)//we don't insert event if the packet is lost
       {
       Ack.errorFlag = 2;
       //printf("LOST data on reverse channel\n");
       }
    else //no error so insert in ES
       {
       ES[Ack.time] = Ack;
       //printf("NOERROR on reverse Channel\n");
       }
    }
```

Figure 5: Second half of the "SEND() function

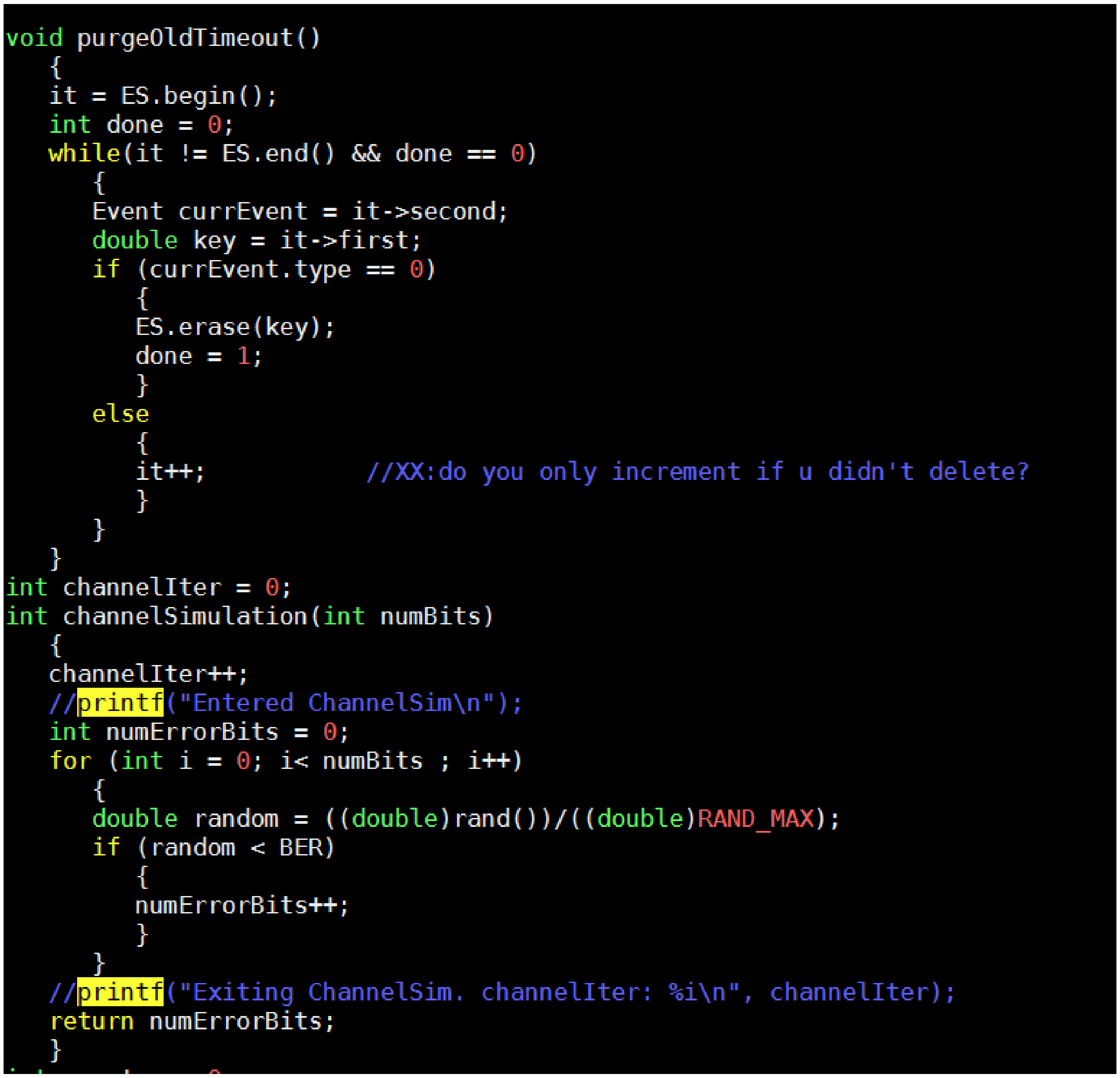
This method starts off by first simulating the forward channel (Figure 6) and then updating the time taken in the channel.

Then it will check if the packet was lost based on the channel simulation (more than 5 bit errors). If so then it will return void (we don't do anything if the packet is lost).

Otherwise it will check if the packet arrived with error or without error. If there are 0 bit errors then it arrived without error. Otherwise it arrived with errors. These two cases are shown by the subsequent if-else blocks below the lost packet check. If the packet arrived in error (or an unexpected packet arrived) then the sender will prepare to send the previous acknowledgement back. If it got the packet it was expecting without errors, then it will increment it's acknowledgement counter and send an updated acknowledgement back. In either case, the if conditions will prepare data to be sent back to the reverse channel.

After this, the reverse channel is simulated and the time is updated. Similar to before, we check if there was an error, if the acknowledgement was lost or if there was no error. In the case of an error, the errorFlag of the acknowledgement is updated and the event is queued in the ES. The same thing happens for the case when there are no errors except the errorFlag is not changed. Finally, if the packet is lost, then we do not queue the ack event. Eventually the timeout for this ACK will occur and the sender will have to resend the data and wait for another ACK of the same number.

Finally, there are two more things that need to be mentioned. First, if a previous timeout needs to be removed, then we use the purgeOldTimeout function which cycles through the ES and finds the one and only timeout in the ES and removes it. Secondly, we need to simulate the channel and if there are any bit errors in it. Both of these functions are shown in Figure 6 below:



```
void purgeOldTimeout()
   {
   it = ES.begin();
   int done = 0;
   while(it != ES.end() && done == 0)
      {
      Event currEvent = it->second;
      double key = it->first;
      if (currEvent.type == 0)
         {
         ES.erase(key);
         done = 1;
         }
      else
         {
         it++;           //XX:do you only increment if u didn't delete?
         }
      }
   }
int channelIter = 0;
int channelSimulation(int numBits)
   {
   channelIter++;
   //printf("Entered ChannelSim\n");
   int numErrorBits = 0;
   for (int i = 0; i< numBits ; i++)
      {
      double random = ((double)rand())/((double)RAND_MAX);
      if (random < BER)
         {
         numErrorBits++;
         }
      }
   //printf("Exiting ChannelSim. channelIter: %i\n", channelIter);
   return numErrorBits;
   }
```

Figure 6: purgeOldTimeout and channelSimulation methods

Data for 1.i and 1.ii

After simulation, the data for questions 1.i and 1.ii was obtained and is shown below:

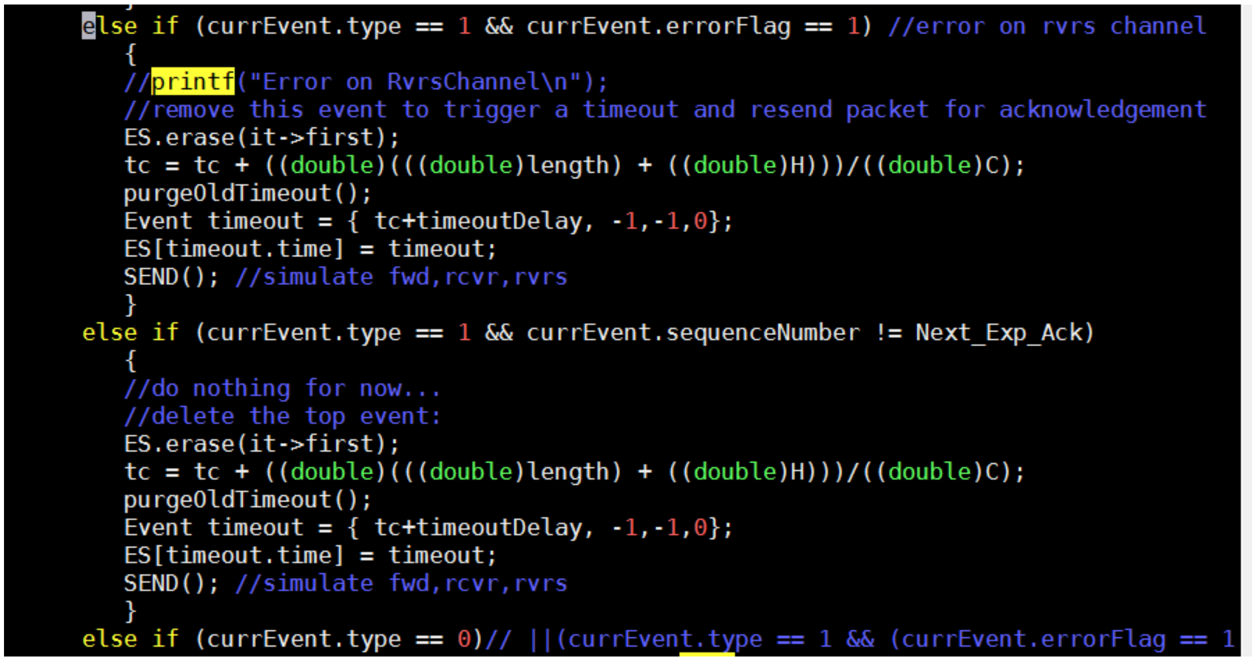
Delta/Tau	Tau=5ms			Tau=250ms		
	BER=0.0	BER=1e-5	BER=1e-4	BER=0.0	BER=1e-5	BER=1e-4
2.5	954441.3	820124.1	239986.4	23877.14	20527.37	5748.045
5	954441.3	735206.5	146812.6	23877.14	17872.89	3334.288
7.5	954441.3	671779.6	107010.1	23877.14	15821.24	2291.868
10	954441.3	598597.4	82710.01	23877.14	14110.7	1773.693
12.5	954441.3	554070.4	68063.47	23877.14	13006.8	1420.62

Table 1: Simulation Data for ABP

The data makes logical sense. When the error rate is zero we have constant throughput since none of the ACKs or frames fail to arrive on time. When there is an error rate, the throughput decreases as the error rate increases. Also, for a constant error rate, the throughput decreases as the timeout delay increases since the simulator must wait longer after a packet is lost to resend the data. Hence the total simulation time takes a big hit.

Question 2 ABP_NAK

For the ABP_NAK, a slight modification was made to the existing ABP code to get the NAK code to work. This is shown in Figure 7 below:



```
      else if (currEvent.type == 1 && currEvent.errorFlag == 1) //error on rvrs channel
         {
         //printf("Error on RvrsChannel\n");
         //remove this event to trigger a timeout and resend packet for acknowledgement
         ES.erase(it->first);
         tc = tc + ((double)(((double)length) + ((double)H)))/((double)C);
         purgeOldTimeout();
         Event timeout = { tc+timeoutDelay, -1,-1,0};
         ES[timeout.time] = timeout;
         SEND(); //simulate fwd,rcvr,rvrs
         }
      else if (currEvent.type == 1 && currEvent.sequenceNumber != Next_Exp_Ack)
         {
         //do nothing for now...
         //delete the top event:
         ES.erase(it->first);
         tc = tc + ((double)(((double)length) + ((double)H)))/((double)C);
         purgeOldTimeout();
         Event timeout = { tc+timeoutDelay, -1,-1,0};
         ES[timeout.time] = timeout;
         SEND(); //simulate fwd,rcvr,rvrs
         }
      else if (currEvent.type == 0)// ||(currEvent.type == 1 && (currEvent.errorFlag == 1
```

Figure 7: Changes made to ABP simulator to get ABP_NAK simulator

As mentioned in Question 1, the case when the sender receives an ACK in error or receives an unexpected ACK are ignored. These are negative acknowledgements(NAKs). For ABP_NAK, we do not ignore these cases. In both of these case we do the exact same thing. We resend the packet right away instead of waiting for a timeout. The changes to those if conditions are shown in Figure 7 above.

Data for 2.i and 2.ii

The combined data for both 2.i and 2.ii is shown in the table below:

Delta/Tau	Tau=5ms			Tau=250ms		
	BER=0.0	BER=1e-5	BER=1e-4	BER=0.0	BER=1e-5	BER=1e-4
2.5	954441.3	839586	276582.6	23877.14	20968.77	6709.265
5	954441.3	844115.4	270164.3	23877.14	21064.96	6756.236
7.5	954441.3	833136.6	269194.9	23877.14	21098.47	6599.856
10	954441.3	837009	262474	23877.14	20950.37	6508.93
12.5	954441.3	841956	261733.4	23877.14	21000.12	6487.328

Table 2: Simulation datra for ABP_NAK

An interesting result can be drawn from Table 2. The throughput does not drop or change by much as the timeout delay increases. This is in contrast to the regular ABP simulator. This makes sense because before we would wait for the timeout event to happen and then resend an frame. This would waste time since the system would be idle during this time. Now, we resend the frame right away. Hence time isn't wasted as much. It is also interesting to notice that as the error rate increases, the throughput still decreases. This is in agreement with our results from question 1.

Figures 8-13 show plots of the results between the two simulators

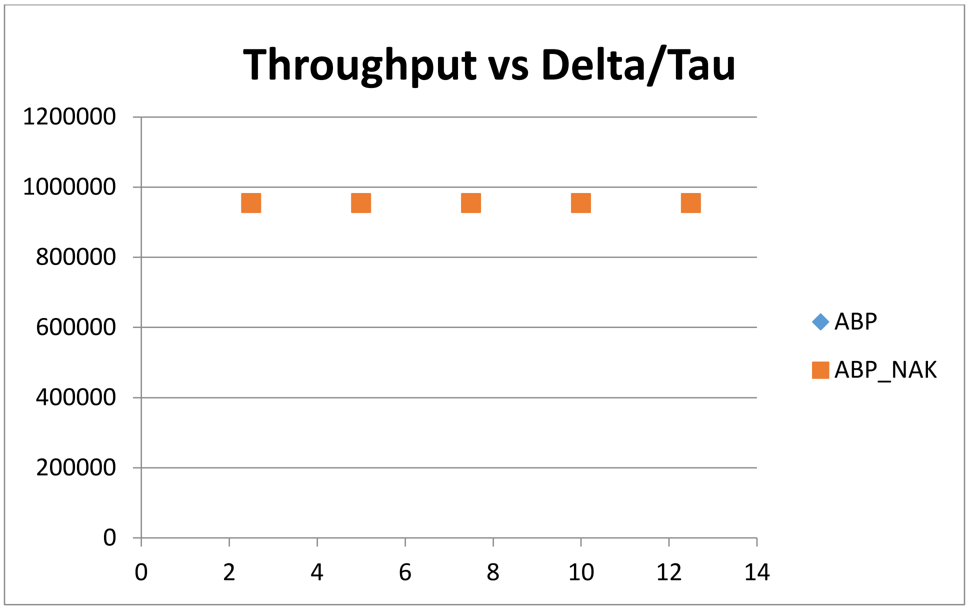


Figure 8: 2*tau = 10ms, BER = 0

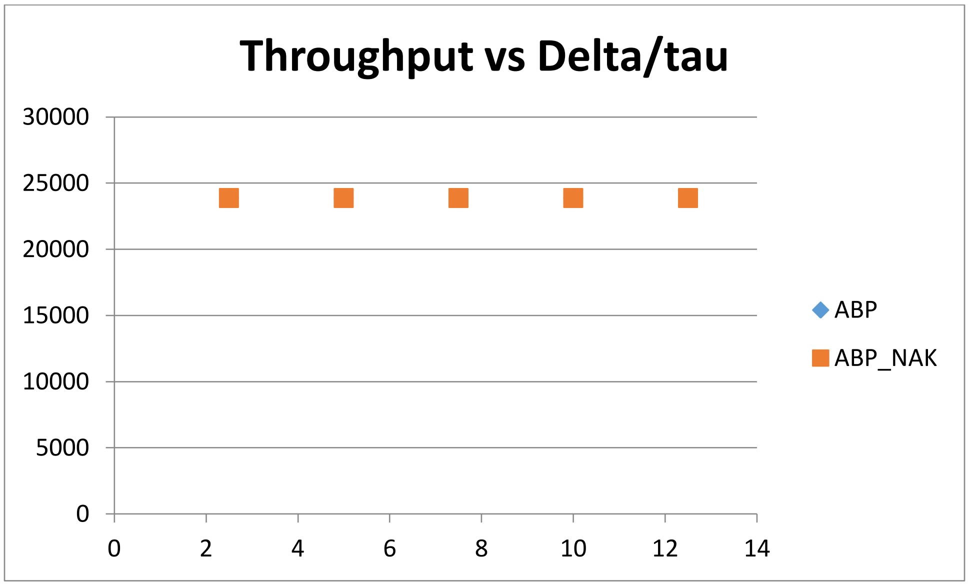


Figure 9: 2*tau = 500ms, BER = 0

As expected, throughput doesn't change between abp and abp_ank for a 0 bit error rate since the channel is 100% efficient and all acknowledgements and frames are delivered on time.

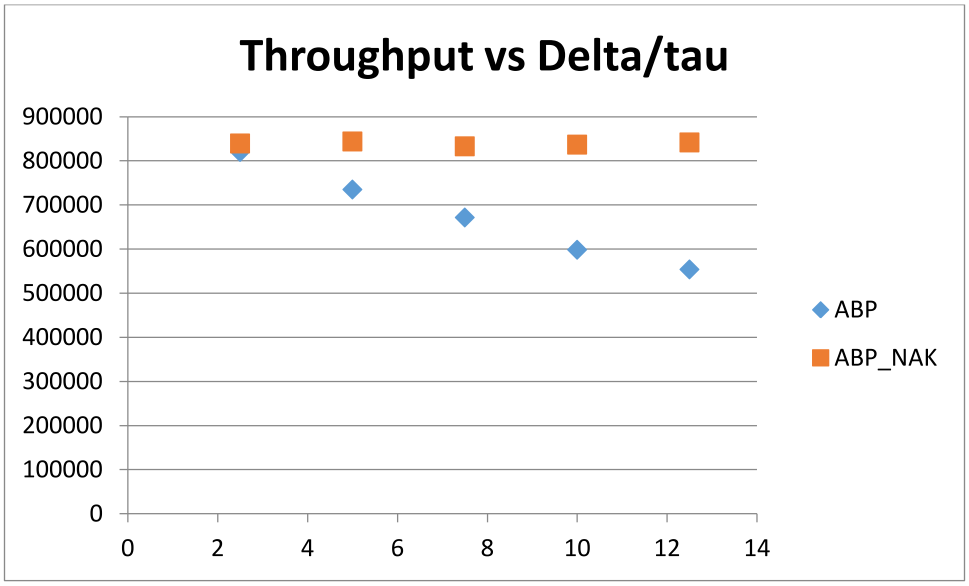


Figure 10: 2*tau = 10ms, BER = 1E-5

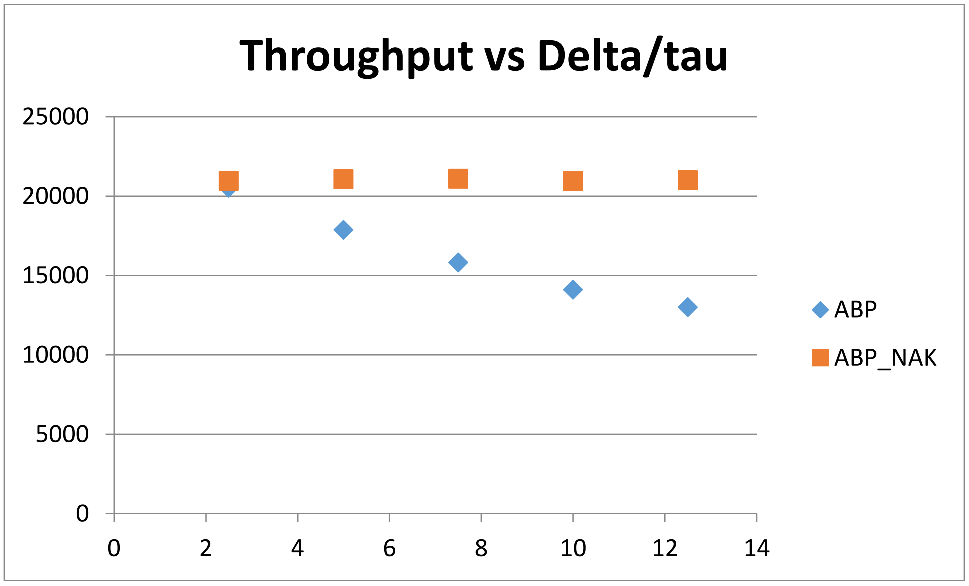


Figure 11: 2*tau = 250ms, BER = 1E-5

As expected, the throughput decreases for ABP and not for ABP_NAK as the timeout delay increases. This is in agreement with what was mentioned about the data before.

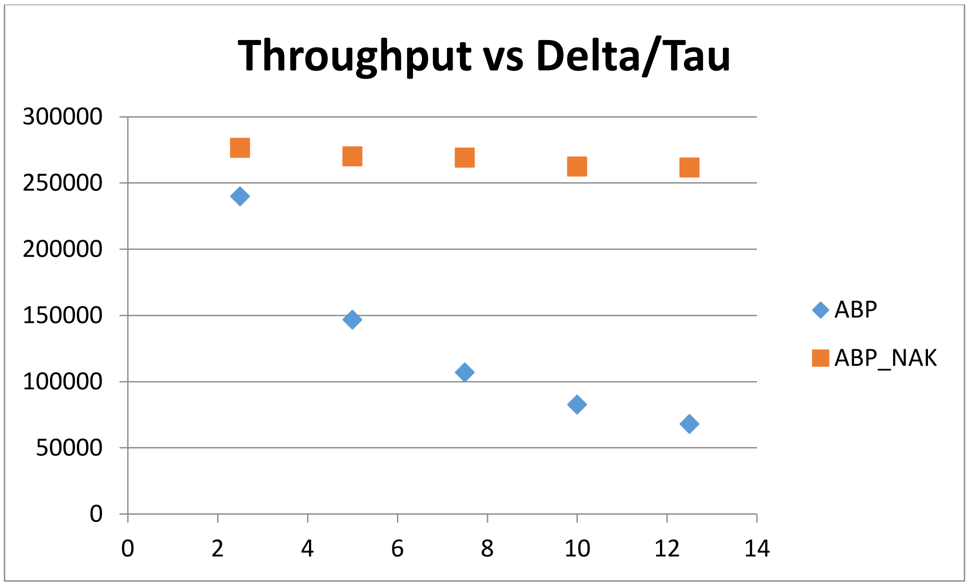


Figure 12: 2*tau = 10ms, BER = 1E-4

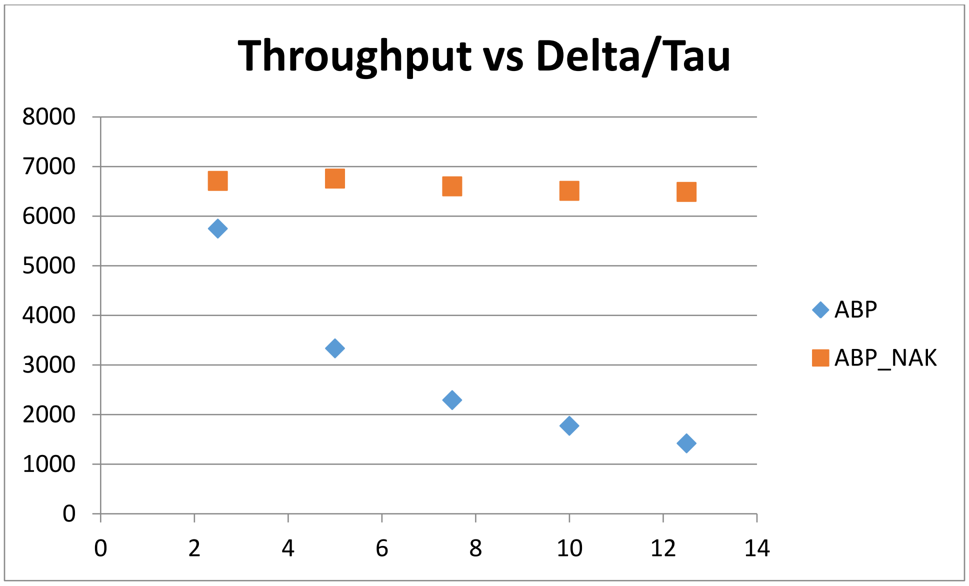
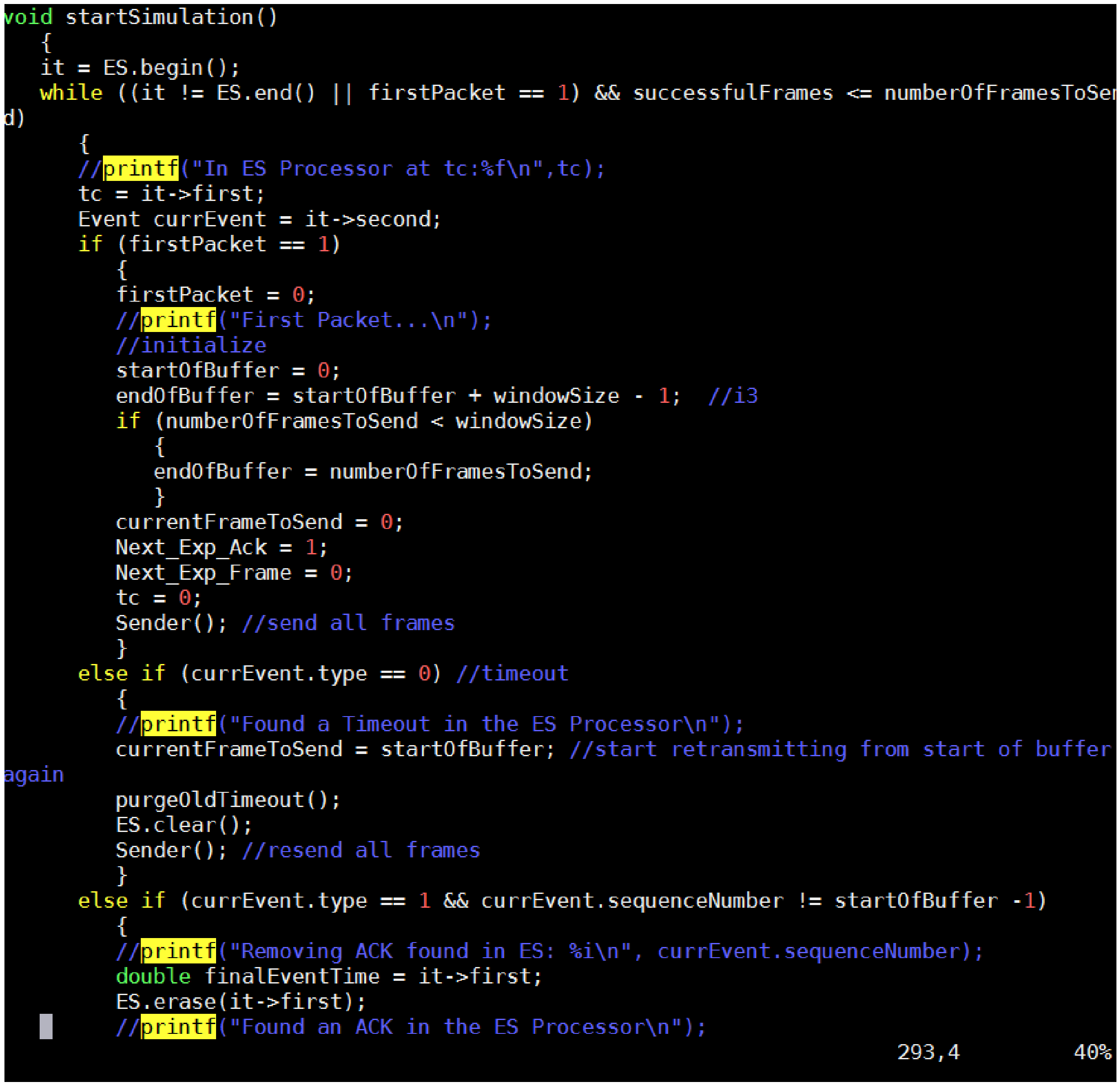


Figure 13: 2*tau = 250ms, BER = 1E-4

Figures 12 and 13 have an almost identical result as Figures 10 and 11. The only difference here is that the ABP simulator's throughput declines are a decreasing rate and looks to be converging to a value.

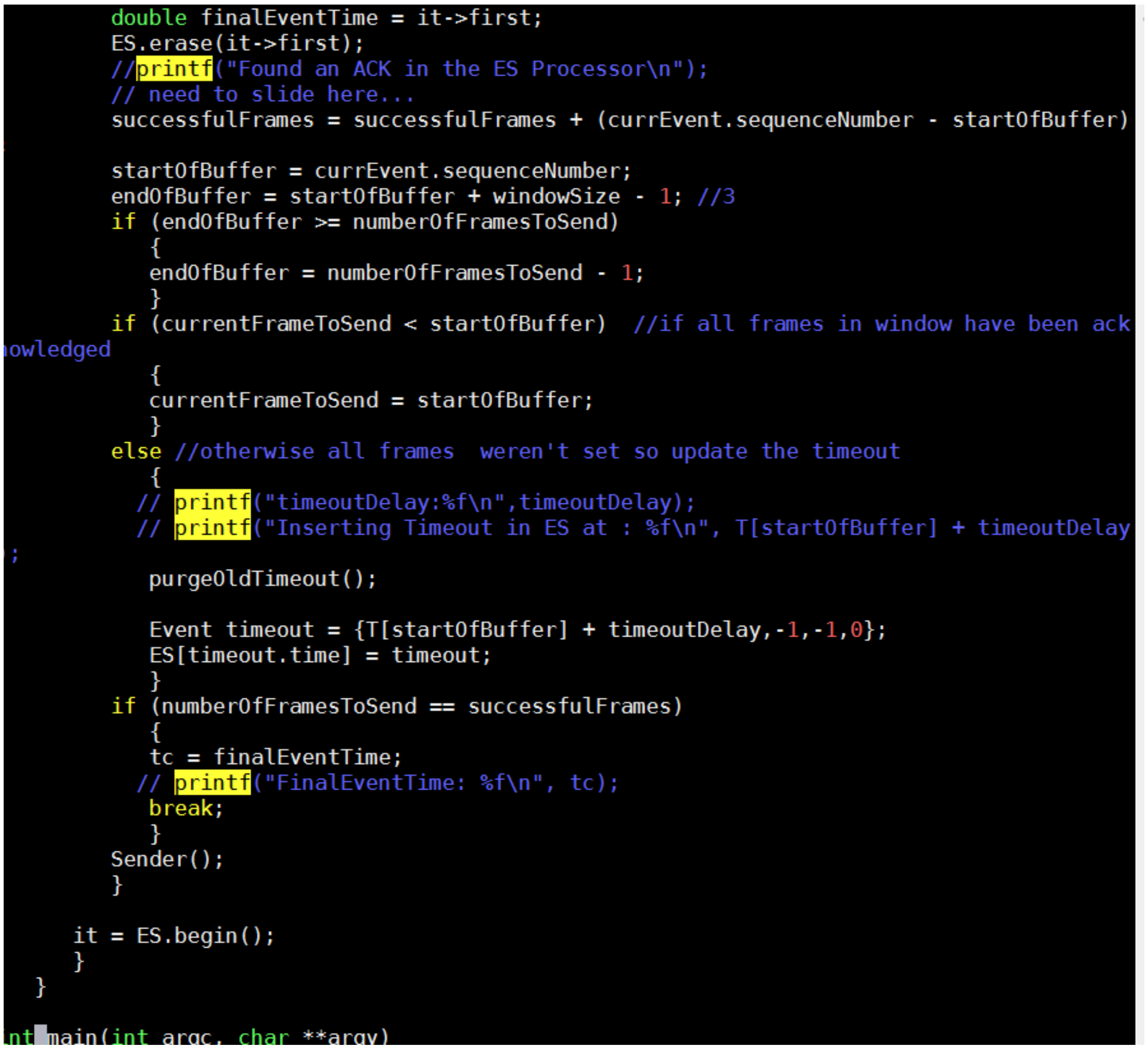
Question 3 GBN

The GBN function uses all the same tools as the previous two simulators and adds one more function: the Sender function. There are modifications made to the startSimulation function as well to account for multiple packets in flight. Figures 14 and 15 show the startSimulation function below.



```
void startSimulation()
   {
   it = ES.begin();
   while ((it != ES.end() || firstPacket == 1) && successfulFrames <= numberOfFramesToSe
d)
      {
      //printf("In ES Processor at tc:%f\n",tc);
      tc = it->first;
      Event currEvent = it->second;
      if (firstPacket == 1)
         {
         firstPacket = 0;
         //printf("First Packet...\n");
         //initialize
         startOfBuffer = 0;
         endOfBuffer = startOfBuffer + windowSize - 1;  //i3
         if (numberOfFramesToSend < windowSize)
            {
            endOfBuffer = numberOfFramesToSend;
            }
         currentFrameToSend = 0;
         Next_Exp_Ack = 1;
         Next_Exp_Frame = 0;
         tc = 0;
         Sender(); //send all frames
         }
      else if (currEvent.type == 0) //timeout
         {
         //printf("Found a Timeout in the ES Processor\n");
         currentFrameToSend = startOfBuffer; //start retransmitting from start of buffer
again
         purgeOldTimeout();
         ES.clear();
         Sender(); //resend all frames
         }
      else if (currEvent.type == 1 && currEvent.sequenceNumber != startOfBuffer -1)
         {
         //printf("Removing ACK found in ES: %i\n", currEvent.sequenceNumber);
         double finalEventTime = it->first;
         ES.erase(it->first);
         //printf("Found an ACK in the ES Processor\n");
                                                                       293,4          40%
```

Figure 14: PArt 1 of the startSimulation function

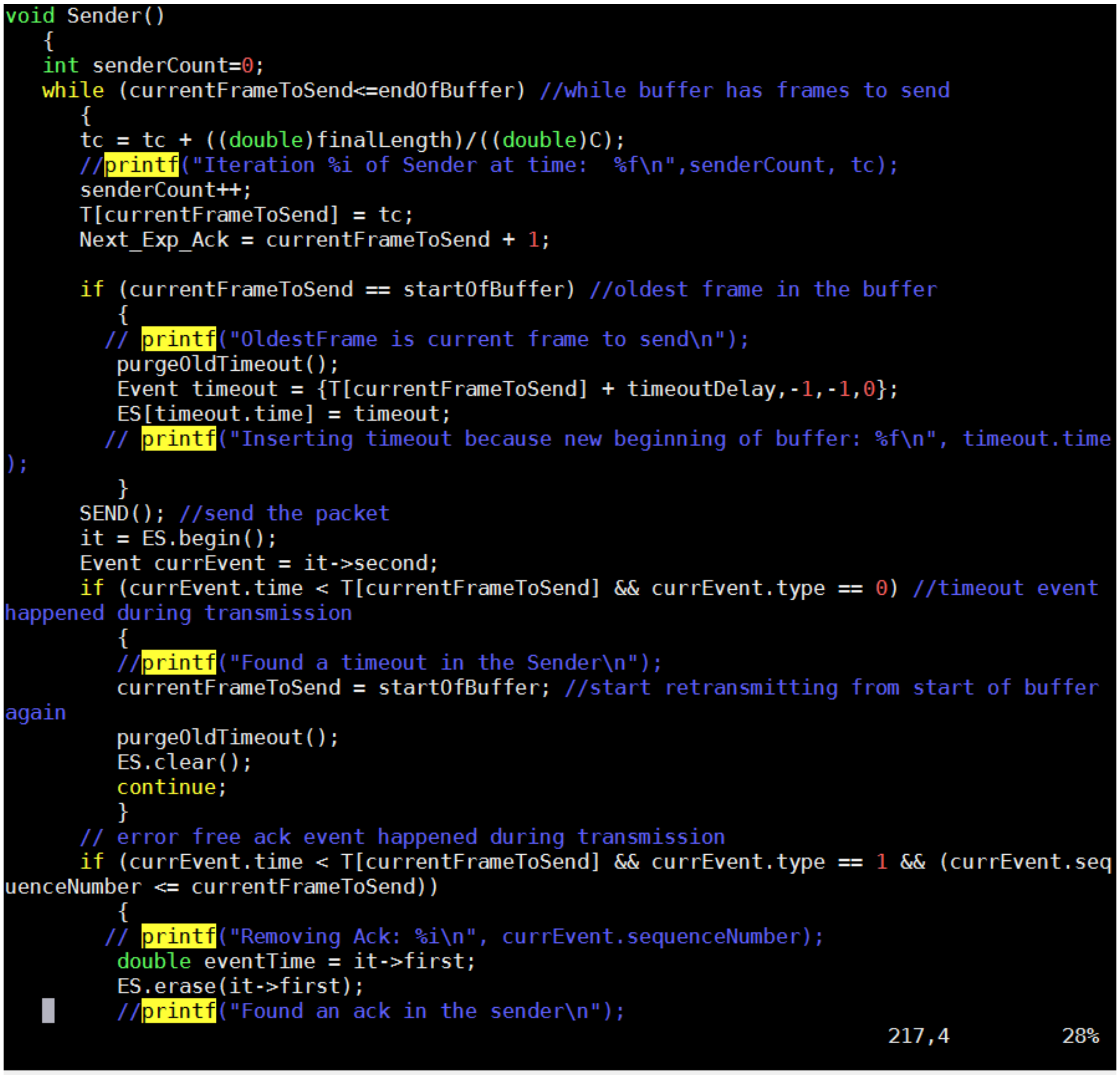


```
        double finalEventTime = it->first;
        ES.erase(it->first);
        //printf("Found an ACK in the ES Processor\n");
        // need to slide here...
        successfulFrames = successfulFrames + (currEvent.sequenceNumber - startOfBuffer)

        startOfBuffer = currEvent.sequenceNumber;
        endOfBuffer = startOfBuffer + windowSize - 1; //3
        if (endOfBuffer >= numberOfFramesToSend)
           {
           endOfBuffer = numberOfFramesToSend - 1;
           }
        if (currentFrameToSend < startOfBuffer)  //if all frames in window have been ack
owledged
           {
           currentFrameToSend = startOfBuffer;
           }
        else //otherwise all frames  weren't set so update the timeout
           {
          // printf("timeoutDelay:%f\n",timeoutDelay);
          // printf("Inserting Timeout in ES at : %f\n", T[startOfBuffer] + timeoutDelay
);

           purgeOldTimeout();

           Event timeout = {T[startOfBuffer] + timeoutDelay,-1,-1,0};
           ES[timeout.time] = timeout;
           }
        if (numberOfFramesToSend == successfulFrames)
           {
           tc = finalEventTime;
          // printf("FinalEventTime: %f\n", tc);
           break;
           }
        Sender();
        }

     it = ES.begin();
     }
  }

int main(int argc, char **argv)
```

Figure 15: Part 2 of the startSimulation function

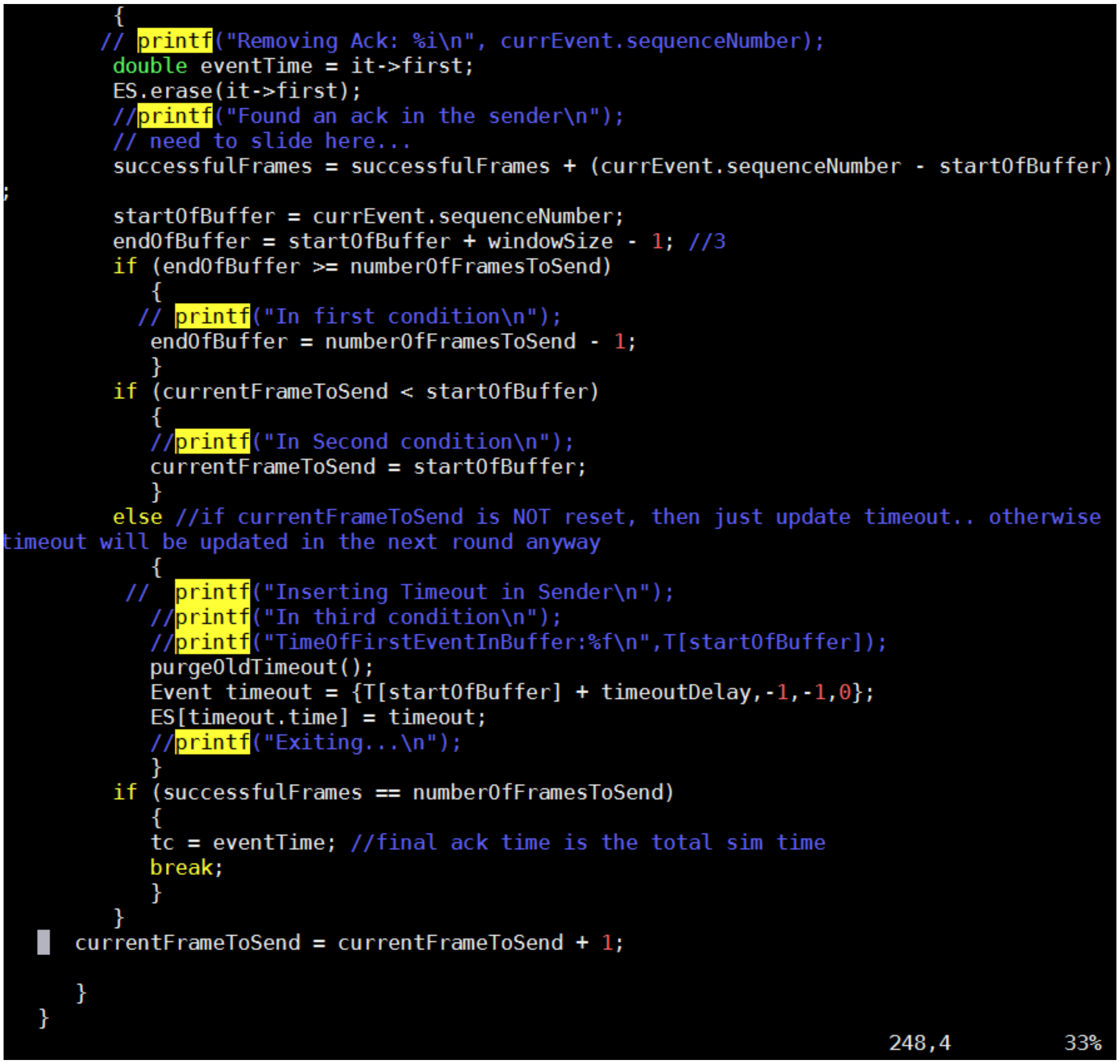
The key here is the window's start and end point is being tracked(to facilitate the sliding of the window) and the last SENT packet is also being tracked.

The sender() function code is shown in Figure 16 and 17 below:



```
void Sender()
   {
   int senderCount=0;
   while (currentFrameToSend<=endOfBuffer) //while buffer has frames to send
      {
      tc = tc + ((double)finalLength)/((double)C);
      //printf("Iteration %i of Sender at time:  %f\n",senderCount, tc);
      senderCount++;
      T[currentFrameToSend] = tc;
      Next_Exp_Ack = currentFrameToSend + 1;

      if (currentFrameToSend == startOfBuffer) //oldest frame in the buffer
         {
        // printf("OldestFrame is current frame to send\n");
         purgeOldTimeout();
         Event timeout = {T[currentFrameToSend] + timeoutDelay,-1,-1,0};
         ES[timeout.time] = timeout;
        // printf("Inserting timeout because new beginning of buffer: %f\n", timeout.time
);
         }
      SEND(); //send the packet
      it = ES.begin();
      Event currEvent = it->second;
      if (currEvent.time < T[currentFrameToSend] && currEvent.type == 0) //timeout event
happened during transmission
         {
         //printf("Found a timeout in the Sender\n");
         currentFrameToSend = startOfBuffer; //start retransmitting from start of buffer
again
         purgeOldTimeout();
         ES.clear();
         continue;
         }
      // error free ack event happened during transmission
      if (currEvent.time < T[currentFrameToSend] && currEvent.type == 1 && (currEvent.seq
uenceNumber <= currentFrameToSend))
         {
        // printf("Removing Ack: %i\n", currEvent.sequenceNumber);
         double eventTime = it->first;
         ES.erase(it->first);
         //printf("Found an ack in the sender\n");
                                                                      217,4          28%
```

Figure 16: Sender function part 1



```
         {
        // printf("Removing Ack: %i\n", currEvent.sequenceNumber);
         double eventTime = it->first;
         ES.erase(it->first);
         //printf("Found an ack in the sender\n");
         // need to slide here...
         successfulFrames = successfulFrames + (currEvent.sequenceNumber - startOfBuffer)
;

         startOfBuffer = currEvent.sequenceNumber;
         endOfBuffer = startOfBuffer + windowSize - 1; //3
         if (endOfBuffer >= numberOfFramesToSend)
            {
          // printf("In first condition\n");
            endOfBuffer = numberOfFramesToSend - 1;
            }
         if (currentFrameToSend < startOfBuffer)
            {
            //printf("In Second condition\n");
            currentFrameToSend = startOfBuffer;
            }
         else //if currentFrameToSend is NOT reset, then just update timeout.. otherwise
timeout will be updated in the next round anyway
            {
         //  printf("Inserting Timeout in Sender\n");
            //printf("In third condition\n");
            //printf("TimeOfFirstEventInBuffer:%f\n",T[startOfBuffer]);
            purgeOldTimeout();
            Event timeout = {T[startOfBuffer] + timeoutDelay,-1,-1,0};
            ES[timeout.time] = timeout;
            //printf("Exiting...\n");
            }
         if (successfulFrames == numberOfFramesToSend)
            {
            tc = eventTime; //final ack time is the total sim time
            break;
            }
         }
      currentFrameToSend = currentFrameToSend + 1;

      }
   }
                                                                  248,4          33%
```

Figure 17: Sender() function part 2

This function has the same if conditions as the startSimulation function except it will try to transmit the entire window while checking for events happening simultaneously.

Data for 3.i and 3.ii

Delta/Tau	Tau=5ms			Tau=250ms		
	BER=0.0	BER=1e-5	BER=1e-4	BER=0.0	BER=1e-5	BER=1e-4
2.5	3.82E+06	2.40E+06	301855	95508	63737.4	7615.07
5	3.82E+06	1.84E+06	169091	95508	45582.7	3923.15
7.5	3.82E+06	1.44E+06	118653	95508	34295.7	2567.43
10	3.82E+06	1.21E+06	90244.3	95508	26956.4	1924.34
12.5	3.82E+06	1.04E+06	73168.8	95508	23718.6	1576.78

Table 3

The results of this simulation follow all the same patterns as the original ABP simulator did in Question 1.

Figures 18-23 show the results of Question 3 compared with those of Question 1.

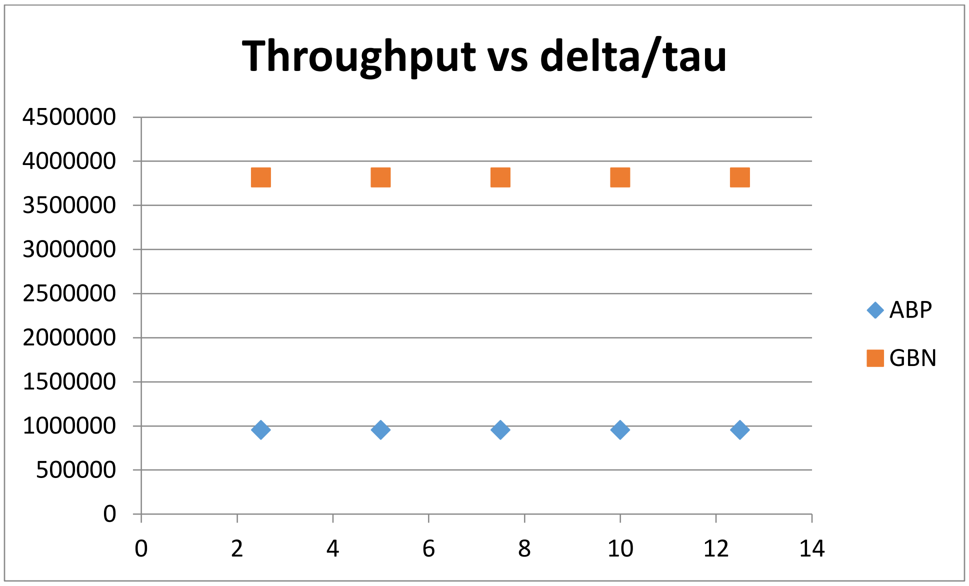


Figure 18: 2*tau = 10ms, BER=0

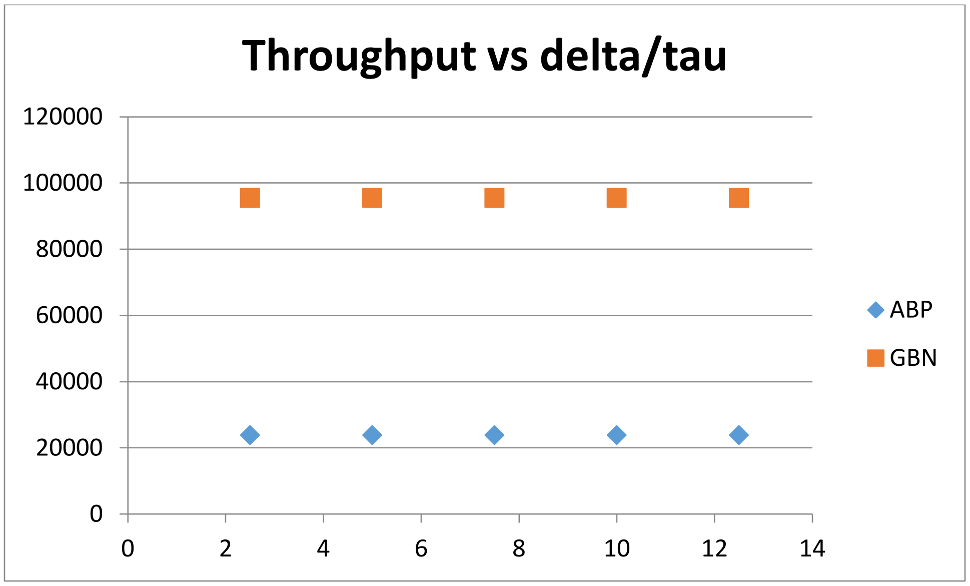


Figure 19: 2*tau = 250ms, BER=0

The pattern for 10ms and 250 ms remains consistent. The throughput of GBN is roughly 4 times as much as that of ABP. This makes sense since in ABP we can have at most one packet in flight and in GBN we can have 4. Since there is no error rate, we can deliver data 4 times as fast almost.

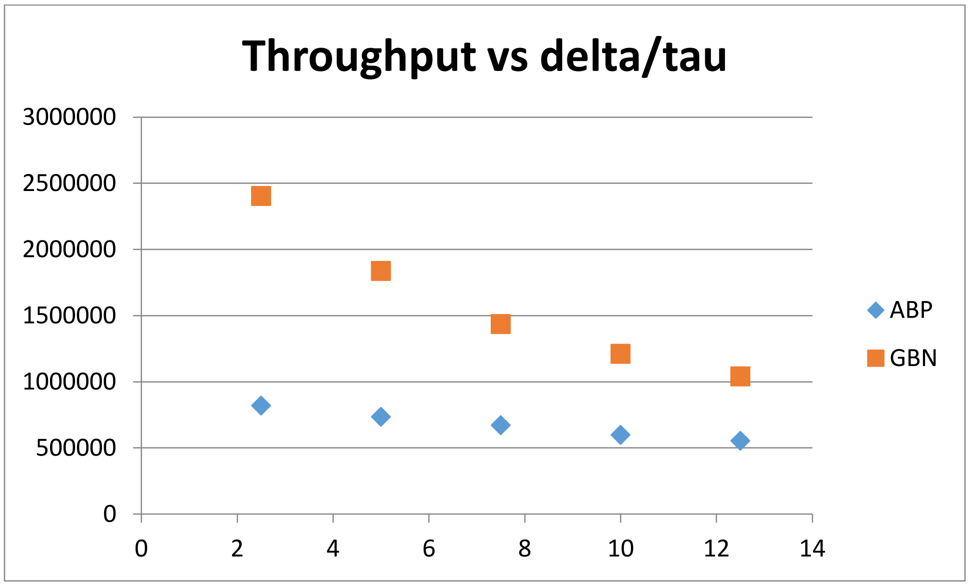


Figure 20: 2*tau = 10ms, BER=1e-5

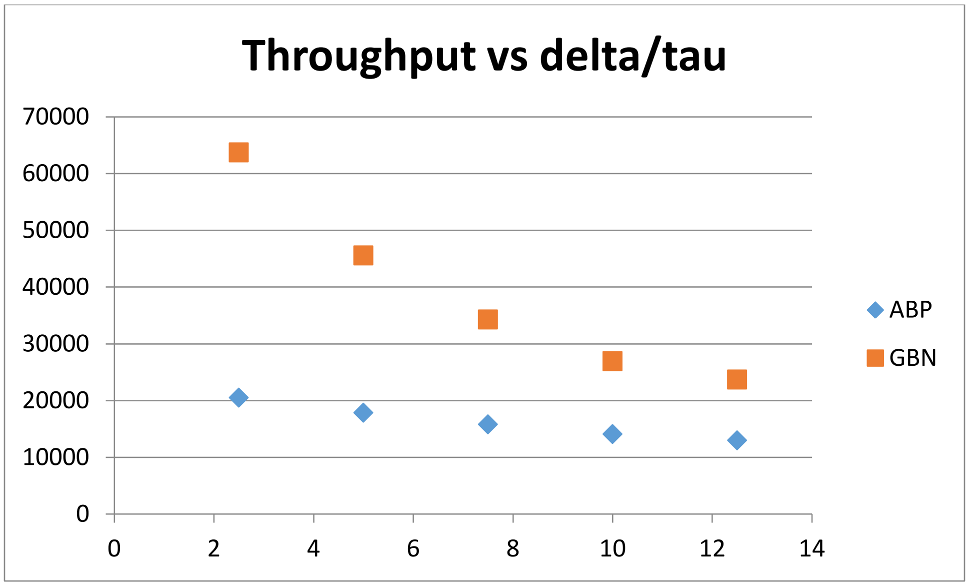


Figure 21: 2*tau = 250ms, BER=1e-5

Once again the pattern between 250ms and 10ms is the same. The throughput of GBN is almost 4 times as much for small timeout delays. However as the delay increases the throughput for both of them looks to converge. The GBN throughput falls sharply.

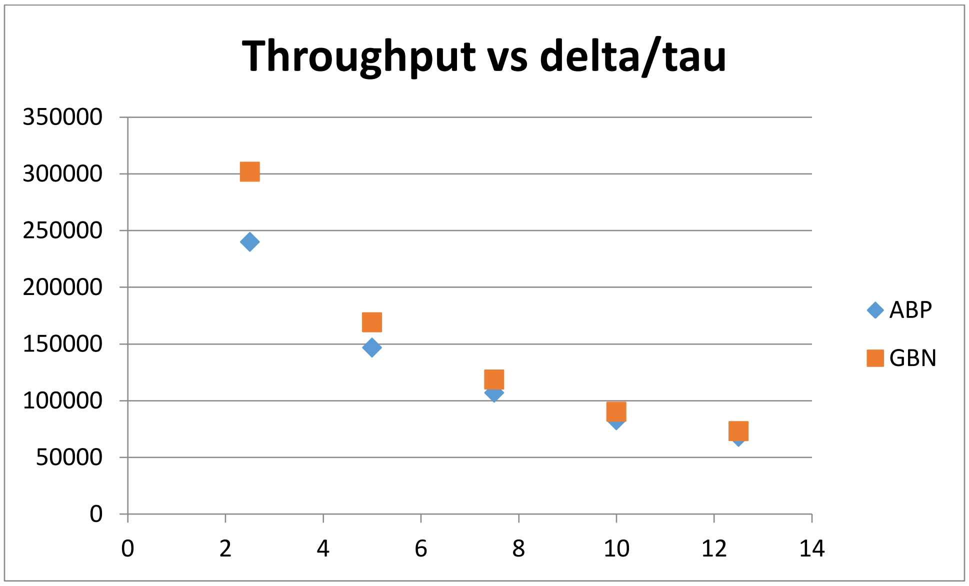


Figure 22: 2*tau = 10ms, BER=1e-4

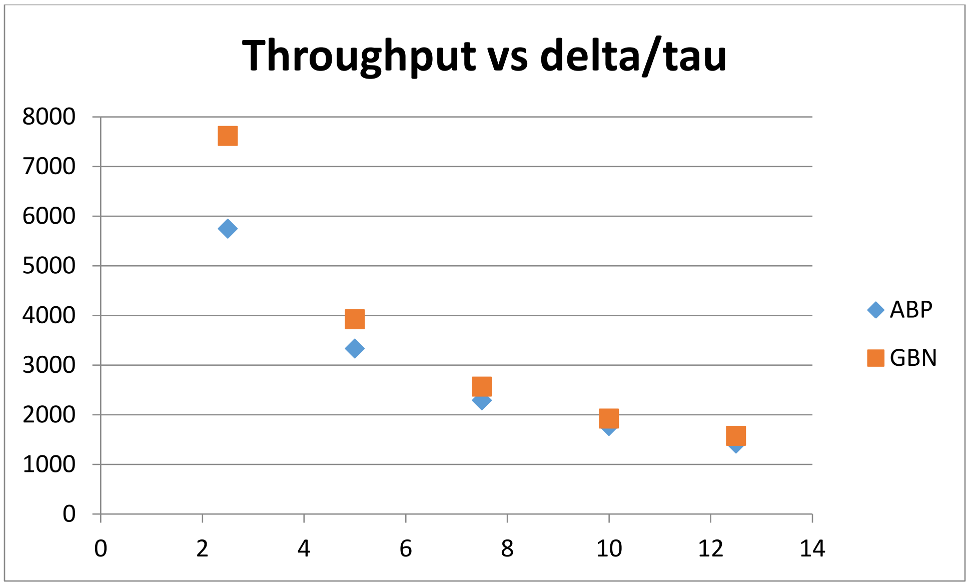


Figure 22: 2*tau = 250ms, BER=1e-4

Once again the pattern is the same for both the graphs and the throughputs are also similar for ABP and BER. The advantages of GBN are mitigated due to the high bit error rate(as evident from the graph).