ECE358: Computer Networks

Winter 2014

Project 2: Data Link Layers and ARQ Protocols

Date of submission: March 17, 2014

Submitted by:

Student ID: (daskinne) 20373565

Student Name: Davd Skinner

[daskinne@uwaterloo.ca](mailto:daskinne@uwaterloo.ca)

Marks received:

Marked by:

Table of Contents

Explanation ABP 3

Explanation ABP with NAK 4

Explanation GBN 4

Question 1 6

Question 2 6

Question 3 6

A1 Summary of Result Data 8

A2 Simulator Commands 9

Step 1) 9

A3 Summary of required plots (for the report) 10

Q1 and Q2 Together 10

Q1 and Q3 Together 13

# Explanation ABP

The ABP implementation required significant modifications from the lab 1 code. First, the following structures were defined.

enum EventType {

TIMEOUT, ACK

};

enum ErrorType {

NONE, ERROR, LOSS

};

struct simEvent {

EventType type;

double time;

int SN;

bool error;

} SimEvent;

These structures represent the events in the ES, as well as an enumeriation for the return value of the channel function.

The following member variables were defined to record the corresponding simulation properties.

int H, l;

double C, Delta, Tau, BER;0

list<simEvent> ES;

public:

/////////STATS/////

double tc;

double total\_time;

int num\_packets;

int num\_sent\_success;

int NEXT\_EXPECTED\_FRAME;

int NEXT\_EXPECTED\_ACK;

bool NACK;

Following the lab manual, beginning with the sen d\_packets function, which implements the starting point for the simulator. The sender is responsible for sending the packets and handling the sender logic (response to acknowledgements or timeout events). This is achieved by sending the first packet and entering a while loop that continues until the desired number of packets have been recorded as received. Inside of this loop, the sender gets the latest event from the scheduler. If this event is a timeout, the packet is resent, if the event is an acknowledgement one of two things happens. If the ack is positive, the next packet is sent and the next expected acknowledgement is incremented. Otherise, the ack was in error, and this sender does nothing, returning to check the scheduler which will have a timeout event.

The SEND function calls RECEIVE, which determines if the packet was positively received. If the packet is lost, SEND returns null. If the packet is to be acknowledged, SEND will return the corresponding ACK. This ACK is constructed by first recording the error state from RECEIVE (if the incoming packet was in error), and then checking if the ACK will be in error or lost by the channel by calling the channel function. If the ACK is lost, SEND returns null, otherwise it will add the event to the scheduler, in sorted order such that the sender will retrieve the ACK before the timeout event only in the case where it received prior to timeout.

The channel is implemented by running the bits\_in\_error function over the number of specified bits (l+h or h [if ack]). Bits\_in\_error is performed by iterating over the number of bits, and counting the number of UID[0,1] random variables less than the bit error rate specified.

Receive is implemented by calling the channel to determine the status of the frame, and subsequently checking the SN against the expected frame, and incrementing the number of received packets by one if it is the value expected. Otherwise the frame is ignored. If the frame is lost, or in error, the function returns the error type to the SEND function for generation of the appropriate ACK.

# Explanation ABP with NAK

ABP with NAK required only a slight modification of the ABP code. ABP with NAK represents an alternating bit protocol, in which ACKs which are received in error result in the retransmission of the previously unacknowledged SN. This alleviates the requirement for the sender to wait until timeout to verify that the packet was indeed not successfully transmitted to the receiver.

This sender logic is included below for convenience.

if (this->NACK) {

addTimeout(); //Add a new timeout

send\_packet();//retransmit

}

# Explanation GBN

To implement GBN the following structure was added to aid in the implementation of the buffer.

struct Packet {

int SN;

double timeout;

} SimPacket;

list<Packet> BUFFER;

The window (buffer) was implemented by a list of Packets as a member of the GBN simulator class. A window size parameter was added to the constructor to specify the size of window.

To implement the window the following functions were defined:

1. Rotate\_buffer

Rotate buffer, takes the SN of the packet being acknowledged. If a packet with the corresponding SN exists in the buffer, all packets from B[0] to B[N] are removed from the buffer (where N is the location of the packet with matching SN). This function further sends N new packets and adds them to the back of the buffer. The timeout value is updated at this point as well, since the head of the buffer has been modified.

1. Retransmit\_all

Retransmit all replaces the BUFFER list with a NEWBUFFER. This is achieved by iterating through the current buffer, and pushing packets with new timings to the NEWBUFFER. Each of these packets are sent, and a timeout event is added once the old BUFFER is replaced by the NEWBUFFER.

Further modifications were required in the sender and receiver.

For the sender, instead of sending a single packet at the beginning , we fill the entire buffer. Once this is complete, an event is read from the scheduler. Much like ABP the event is determined to be a timeout or an ACK. In the event of a timeout, the retransmit\_all helper function is called. In the event of a positive acknowledgement, the rotate\_buffer function is called. In the event of a NAK, no action is taken, though a sample implementation of NAK handling has been provided for your amusement.

Finally, for the receiver, we now check for sanity, that the SN is indeed in the buffer. The rest of the logic is as before, apart from the expected frame being modulo (window\_size+1).

# Question 1

i)

To begin, once the simulator code was established, the system was checked for stability. The ABP script was run with 10k and 20k packets respectively. The output was within the proposed margin of error in lab 1 of 5%.

ii)

The system was run with the corresponding bit error rates. The system was again checked for stability by running with 10k and 20k packets each.

# Question 2

The implementation of NAK was effected by including a Boolean flag argument in the constructor of the ABP simulator class. This flag enables a section of code in the sender which handles the NAK events by resending the packets. This section of code is included below for convenience.

if (this->NACK) {

addTimeout(); //Add a new timeout

send\_packet();//retransmit

}

Comparison of Results (with ABP no NAK)

Data and Plots are available in sections A1 and *A3::Q1 and Q2 Together* respectively.

# Question 3

I & ii)

The system was run with the corresponding bit error rates, and was again checked for stability by running with 10k and 20k packets each.

Comparison of Results (with ABP no NAK)

The corresponding plots can be found in *A3::Q1 and Q3 Together*.

# A1 Summary of Result Data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ABP** | 2Tau = 10ms | | | 2Tau = 500ms | | |
| Delta/Tau | BER=0 | BER=1e-5 | BER=1e-4 | BER=0 | BER=1e-5 | BER=1e-4 |
| 5 | 954479.95 | 936096.73 | 394350.88 | 23878.33 | 23402.55 | 9011.60 |
| 7.5 | 954479.95 | 928240.52 | 323600.79 | 23878.33 | 23129.01 | 7120.02 |
| 10 | 954479.95 | 918771.01 | 272662.03 | 23878.33 | 22924.09 | 5981.16 |
| 12.5 | 954479.95 | 914448.10 | 236622.03 | 23878.33 | 22923.15 | 5133.43 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ABP\_NAK** | 2Tau = 10ms | | | 2Tau = 500ms | | |
| Delta/Tau | BER=0 | BER=1e-5 | BER=1e-4 | BER=0 | BER=1e-5 | BER=1e-4 |
| 2.5 | 954479.95 | 948221.43 | 615834.00 | 23878.33 | 23717.04 | 15503.30 |
| 5 | 954479.95 | 947938.90 | 607386.13 | 23878.33 | 23702.92 | 15180.34 |
| 7.5 | 954479.95 | 947280.33 | 610868.92 | 23878.33 | 23700.56 | 15011.83 |
| 10 | 954479.95 | 947186.32 | 599561.85 | 23878.33 | 23672.37 | 14903.74 |
| 12.5 | 954479.95 | 946904.41 | 592250.32 | 23878.33 | 23691.16 | 14704.63 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **GBN** | 2Tau = 10ms | | | 2Tau = 500ms | | |
| Delta/Tau | BER=0 | BER=1e-5 | BER=1e-4 | BER=0 | BER=1e-5 | BER=1e-4 |
| 2.5 | 1602906.604 | 1591290.186 | 1044374.905 | 47527.31236 | 47140.75815 | 30795.89993 |
| 5 | 1602906.604 | 1585703.232 | 976736.0741 | 47527.31236 | 46390.90924 | 23142.32901 |
| 7.5 | 1602906.604 | 1573395.502 | 774092.7306 | 47527.31236 | 45774.91152 | 17360.72519 |
| 10 | 1602906.604 | 1544627.47 | 634232.9036 | 47527.31236 | 45307.71266 | 14050.83878 |
| 12.5 | 1602906.604 | 1529128.616 | 540868.3613 | 47527.31236 | 44049.00586 | 11877.15129 |

# A2 Simulator Commands

# Step 1)

make

Step 2) (Bake!)

./run\_ABP  
./run\_ABP\_NAK  
./run\_GBN

# A3 Summary of required plots (for the report)

## Q1 and Q2 Together

## Q1 and Q3 Together