DEVELOPING A HYPOTHETICAL RELIABLE DATA TRANSFER PROTOCOL AND A PERFORMANCE ANALYSIS

GUNADASA M.M.G.S.M

E/09/417

August 3, 2013

INTRODUCTION

This report will introduce a *hypothetical reliable data transfer protocol* implemented using **UDP** as its transport layer protocol and its implementation using *c* language. A performance study for the developed data transfer protocol will be done to identify the potentials and weakness of the protocol. Further improvements will be suggested at the end of this report to improve the potential of the protocol.

HYPOTHETICAL RELIABLE DATA TRANSFER PROTOCOL

The *hypothetical reliable data transfer protocol* introduced in here is developed based on the **ARQ** protocol for file transfer using **UDP** as its transport layer protocol. In this protocol a window size of W will be used. Sending program will break the large file to be send, in to small data chunks and add the sequence number for each and every one. Sending function will transmit the packets equal to the window size then wait for the acknowledgement of the receiver. In the receiving par, when a packet is received receiver program will check the packets for the errors. If there are any error it will discard the packet and not only that it will discard rest of the packets corresponding to the current window. Then it will wait until sender transmit all the packets corresponding to the current window and after that receiver will send a negative acknowledgement indicating the sequence number of the packet which identified as error packet. If there are no errors in any packet corresponding to the current window, receiver will send a positive acknowledgement indicating that there were no errors in previous window.

if the sender gets an negative acknowledgement from the receiver with sequence number j it will start the transmit from the packet with same sequence number.

IMPLEMENTATION

To represent a *packet* and an *acknowledgement* when implementing the *hypothetical reliable data transfer protocol* tow different data structures are used.
Data structure to represent a *packet* goes as follows.

```
struct packet
{
char  data[DATALEN];
int  seq_num;
int  window_end;
int  no_of_packets;
int  packet_number;
};
```

In this data structure the *data* field is used to store the actual data coming from the file. Fields *seq num* and *packet number* are used to store the sequence number and the packet number respectively. Sequence number is rotating from n to m. This n and m can be changed. Once the sender start to transmit packets, it will count the number of packets it transmit. Corresponding count for a packet will be insert in to the field

packet number. This will be used when re transmitting error packets.
Data structure represent an *acknowledgement* is goes follows.

```
struct ack_so
{
uint8_t ack_type ;
uint8_t error_seq_no;
uint8_t error_packet;
uint8_t len;
};
```

The field *ack type* will signal whether it is a positive acknowledgement of a negative acknowledgement. If this field is set to 0 the acknowledgement will be a positive acknowledgement else it will be a negative acknowledgement. Sequence number of the packet with error and the packet number of that packet will be held with the fields *error seq no* and *error packet* respectively.

IMPLEMENTATION OF THE SENDER PROGRAM (CLIENT)

Client program gets the server address as a command line argument when it is executing. Client program will make a **UDP** socket and it will bind the socket with the server address. After connecting with the server the transmitting function will be called. Transmitter function is declared as follows.
*void client(FILE *fp, int socket id, struct sockaddr *addres, int addres len);*

Once this function is called by the *main* function, the size of the file pointed by the file pointer **fp* will be calculated. *malloc()* function will be used to allocate memory to hold the file in the memory. *fread()* is used to copy the file to the memory. Number of packets about to send is calculated by using the file size.
number of packets to be send = ceil(file size/length of a packet)

Number of packets to be send will be sent to the receiver before starting to send the actual file. Length of the packet should be count before start copying data from the file for each and every packet. Normal length of a packet (is stored in the field called *packet length*) is defined as *DATALEN* in the header file *headsock*. If remaining amount of data of the file is less than the amount defined in *DATALEN packet length* will be equal to the remaining amount of data of the file. Then the data is copied to the packet using *memcpy()* function. Starting point to copy data form the file will be *file buffer + current index*. *file buffer* points the starting point of the file and the *current index* is increased by the *packet length* each time new packet it created.
current index = current index + packet length.
After copying the data to the packet rest of fields are modified. Sequence number for a packet will be generated as follows.

```
seq_num = (packet_count % MAX_SEQ_NO) + 1;
```

This will give *seq num* goes from one to *MAX SEQ NO* in a repetitive manner. *packet count* is continuously increased until file transfer complete. After number of packets send equal to the window size in the last packet of the window the field called *window end* will be set to 1. For rest of the packets in the window this field will be remain 0. This field signal the receiver that end of the window has occurred. Once all packets corresponds to a window is send out sender will wait for the acknowledgement of the receiver.
If the acknowledgement from the receiver is a positive acknowledgement sender will continue the transmission and it will repeat the same procure. If a negative acknowledgement received from the receiver *current index* will be roll back to the starting point of the packet which generate the error. Pseudo code for the process goes follows.

```
packets = number of packets /*total number of packets
* send up to now*/
int count;
while(packets >0){
if(error packet no == packets%MAX SEQ NO){
return count;
}else{
packet--;
count++;
}

indx = count * DATALEN;
```

current index will be reduce from the amount calculated from the above method.
current index = current index - index
number of packets will also edited to the value that hold by the *error packet* in the negative acknowledgement send by the receiver. After rolling back transmission of the file will be same as described above. This procedure will be continue until total file is send out.
Once the file transmission is complete sender will generate a special packet with out any data. *seq num* field will be set to -1 and *window end* field is set to 1 in this special packet. This combination of the *seq num* and *window end* will signal the receiver that the file transmission has ended.

IMPLEMENTATION OF THE RECEIVER PROGRAM (SERVER)

Receiver program takes no terminal arguments when it is executing it will make a **UDP** socket and it will continue listening to the socket. When first packet received from the sender server start to cont the time. This is useful to the performance analysis of the protocol. After receiving a packet server will check the packet for errors if there are no any errors data comes from the packet will be copied to a buffer. Receiver will wait for the window end signal from the sender. This signal is generated when the *window*

end field in a up coming packet is set to 1. Once the receiver got the signal it will send an acknowledgement to the sender indicating that there were no error occurred during the packet transmission of last window. When sending positive acknowledgement *ack type* field will be set to 0.
If there are any errors in an incoming packet receiver will store the values in fields *seq num* and *packet count.* Then the rest of the packet in the corresponding window will be discarded. Even though the receiver encounter an error in a packet it will wait until the *window end* field in an incoming packet to be 1 to send the negative acknowledgement. When sending negative acknowledgement *ack type* field will be set to 0 . *error seq no* and *error packet no* will be filed with the data stored when the error packet is detected. This procedure will be continue until the end of the file transmission signal send by the sender.
When the sender signals end of the file transmission (if incoming packet has the following data in the corresponding fields *seq no = -1* and *window end = 1*) receiver will stop listening to the socket and it will copy the data in the buffer to the file. After copying the data in to the file the time taken to the file transfer will be calculated.

SIMULATING ERROR PACKETS

Since these two programs running in two *terminals* there will be no bit errors in the packets that transmitted by this programs. To simulate the error packets accordion to a given error ratio following method is used.
When initiative packet received it contain no data but the meta-data (*number of packets to be received*) about the file to be send. Once this initial packet received informations in the packet are saved. When sender start to send the packets with actual data a random number will be generated between 0 and the square root value of the number of packets to be received. The random number is tested against the *error ratio* in the following manner.

```
random_number <= sq_root * error_ratio
```

If above condition is true current packet received will be considered as a packet with error. The situation will be handled as described in the above.

PERFORMANCE ANALYSIS

Performance analysis of the developed protocol was conducted as stated below.
MAX SEQ NUM kept in a constant value through out the performance analysis. *error ratio* was changed from 0.0 to 0.5 with step size of 0.1 while keeping the *window size* unchanged. Time taken to transfer the file was recorded and the throughput was calculated as follows
throughput = total time taken to transmit the file / file size
Then the *window size* changed from 1 to 5 while repeating above procedure with every window size.
This performance analysis was conducted with two files. First one was done with a small file, Next one was done with a file grater than ten times for the first file.

Collected data are shown in here.

Table 1: Transmission speed (*byte/milliseconds*) with varying the error ratio and window size for a small file

window cont	0.0	0.1	0.2	0.3	0.4	0.5
1	835	759	376	323	181	149
2	942	907	498	340	309	160
3	945	945	839	264	197	141
4	873	950	436	270	192	124
5	1130	885	488	221	151	94

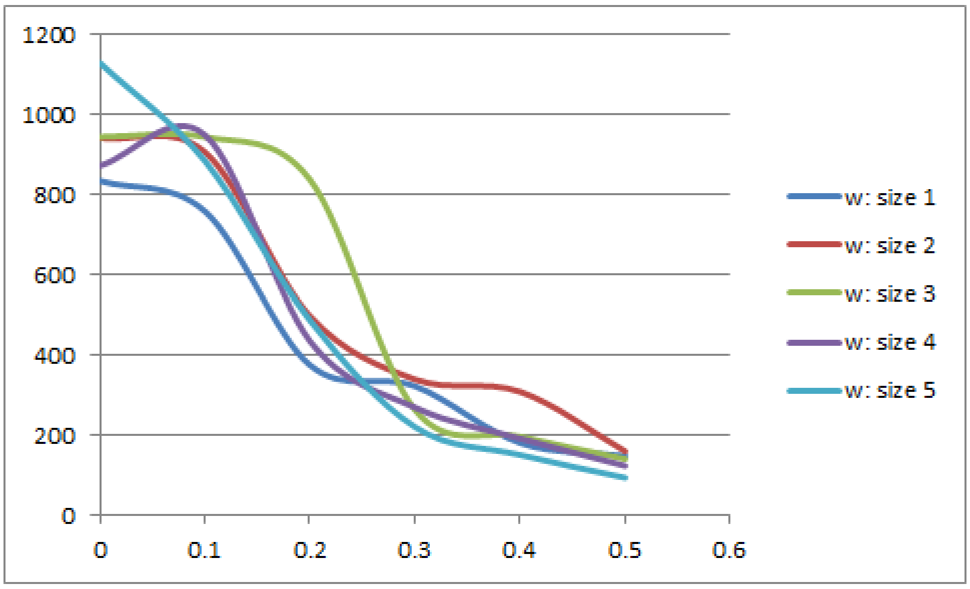

Figure 1: Transmission speed (*byte/milliseconds*) with varying the error ratio and window size for a small file

Table 2: Transmission speed (*byte/milliseconds*) with varying the error ratio and window size for a large file

window cont	0.0	0.1	0.2	0.3	0.4	0.5
1	255	235	210	179	149	120
2	404	313	259	212	167	120
3	459	341	271	204	155	107
4	487	364	259	192	142	103
5	540	351	258	166	122	87

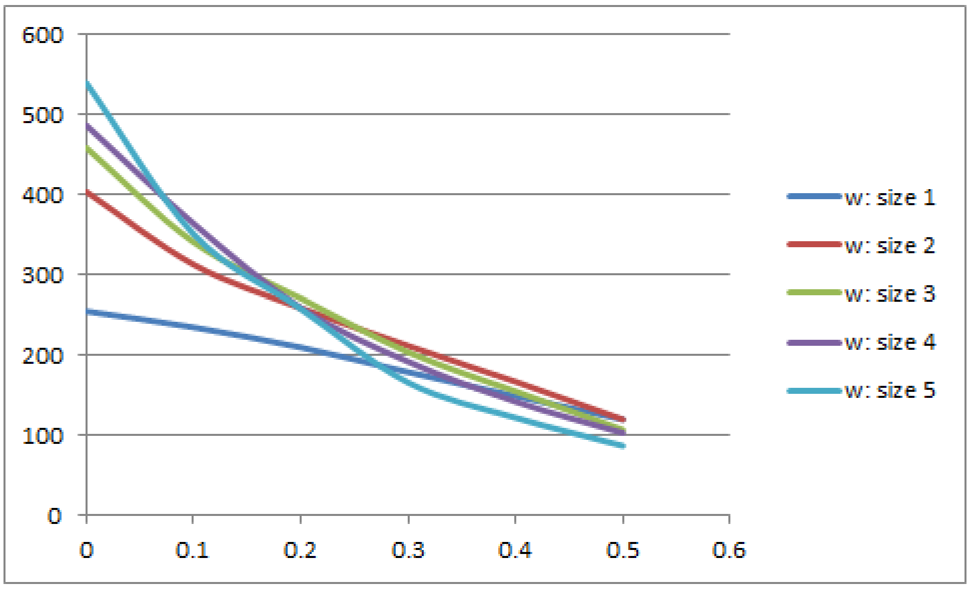

Figure 2: Transmission speed (*byte/milliseconds*) with varying the error ratio and window size for a large file

DISCUSSION

According to the graphs obtained from the performance analysis it can be directly seen that the most suitable window size for the *hypothetical reliable data transfer protocol* is highly depends on the error ratio of the network which it is going to be implemented. For the networks with low error ratio it is good to have relatively high window size while relatively low window size is suitable for the networks with high error ratio.
When there is a high error ratio in the network, number of error packets generated for a file transfer goes high. If the first packet in a window get an error whole window must be transmitted in the next step. If the error ratio is high retransmission of the same window may be fails. Time used for transmit the packet with out the errors will be a waste. In this case if the window size is high the time getting wasted is also high while the time waste will be relatively low if the window size is small.

CONCLUSION

Performance analysis can be concluded as follows

Table 3: Conclusion of the performance analysis

window size	low error ratio	high error ratio
low window size	not good	good
high window size	good	not good

IMPROVEMENTS

Further improvements can be done to the developed *hypothetical reliable data transfer protocol* to improve its performances. One of the major drawback of the developed protocol was is discard the rest of the packets in a window once it founds an error in an incoming packet. Because of this approach packet with out errors are getting discard and they needed to be retransmit in the next window. This will highly affect to the performance of the protocol. This issue can be overcome by discarding only the packet with errors, while rest of the packets in window are saved. The packet which encountered with errors is retransmitted at the beginning of the next window rest of the window will be started from the previous window stopped.

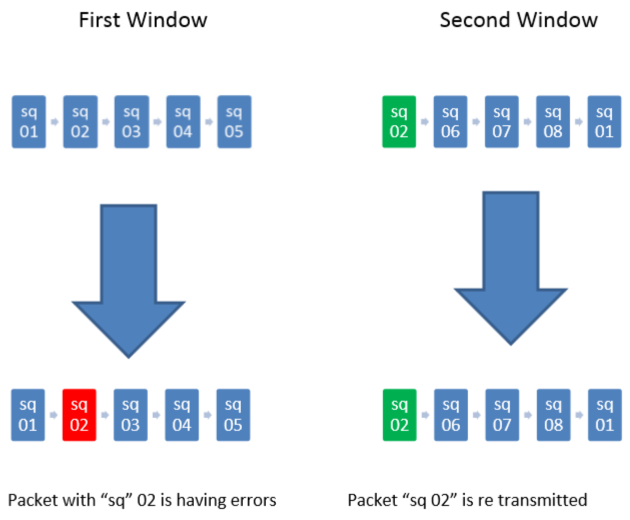

Figure 3: Improved version of *hypothetical reliable data transfer protocol*

PERFORMANCE ANALYSIS FOR IMPROVED PROTOCOL

A performance analysis can be done to the improved implementation of the *hypothetical reliable data transfer protocol* as done earlier.
Data from the performance analysis can be stated as follows.

Table 4: Transmission speed ($byte/milliseconds$) with varying the error ratio and window size for a small file with improved protocol

window cont	0.0	0.1	0.2	0.3	0.4	0.5
1	785	351	283	329	178	196
2	921	1056	892	486	529	779
3	900	1055	937	437	885	865
4	1105	1091	985	1040	821	947
5	1167	1132	918	1045	1093	976

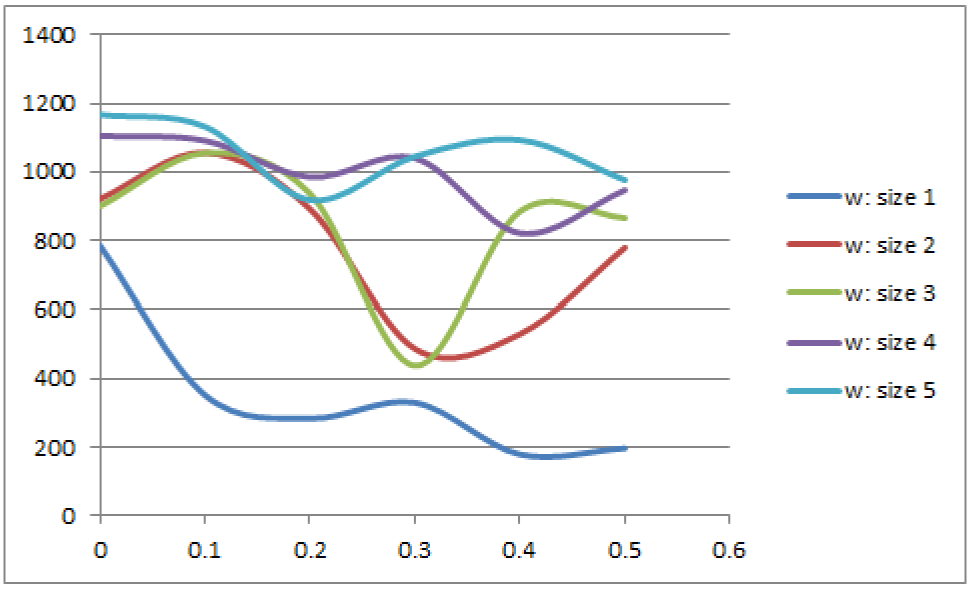

Figure 4: Transmission speed ($byte/milliseconds$) with varying the error ratio and window size for a small file with improved protocol

Table 5: Transmission speed ($byte/milliseconds$) with varying the error ratio and window size for a large file with improved protocol

window cont	0.0	0.1	0.2	0.3	0.4	0.5
1	275	256	222	202	163	133
2	415	385	337	318	280	247
3	501	468	426	391	354	342
4	580	515	455	440	419	416
5	644	572	527	502	462	456

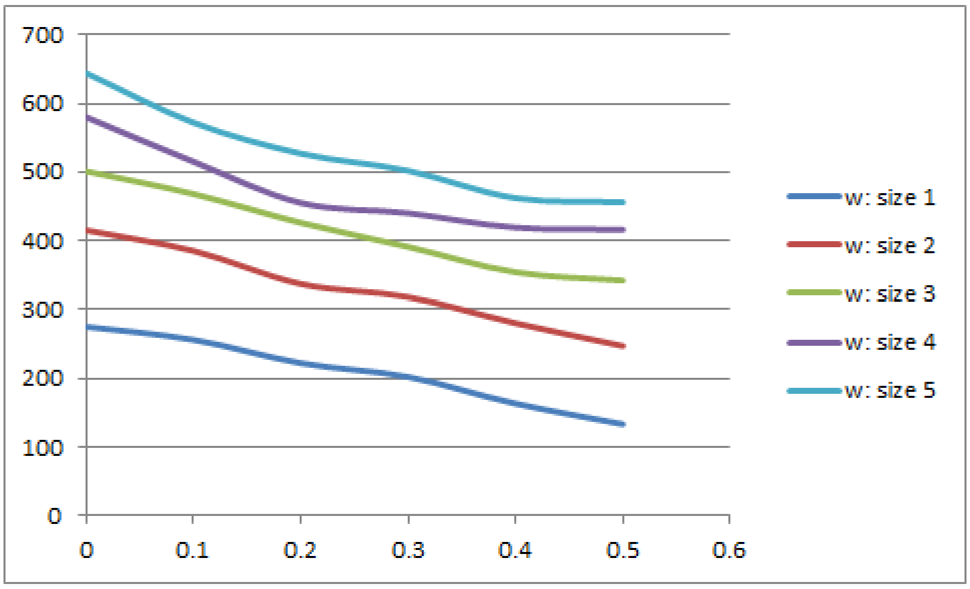

Figure 5: Transmission speed (*byte/milliseconds*) with varying the error ratio and window size for a large file with improved protocol

COMPARISON BETWEEN TWO IMPLEMENTATION

When comparing two implementations for the *hypothetical reliable data transfer protocol* it is clearly visible that performances of these two implementations are very similar when they are tested with a low error ratio. When the error ratio is increased improved implementation shows high performances rather than the first implementation of the protocol. This improved version is similar to the widely used implementation in real world called *selective repeat* method. This method will increase the performances by decrease the unwanted retransmission of packets. There for this modified method is more suitable for real applications rather than the first implementation.