Effects of various design issues on the performance of Go – Back – N sliding window protocol

Shyam S. Somasundaram
University of Minnesota, Twin Cities
somas009@umn.edu

Abstract – Go – Back – N sliding window protocol is an automatic repeat request (ARQ) protocols in which the sending window has a size of N and the receiving window has a size of 1. The sender sends frames present in the sliding window and does not wait for an acknowledgment. Once the sender receives an acknowledgment, the sending window slides to accommodate the next packet. Several factors affect the performance of the Go – Back – N protocol. Some factors and how they affect the performance of the protocol are discussed in this paper.

I. Introduction

Several factors affect the performance of the go back n sliding window protocol. Depending upon the application, varying these factors could have a great effect on the throughput as well as the simulation time of the protocol. The different factors are discussed in detail in the following sections.

II. Factors affecting performance

A. Packet size

Packet size is one of the major factors that affect the performance of the sliding window protocol. When the packet size increases the time taken to transfer a file of fixed size decreases until the packet size reaches a peak value. Once it crosses the peak value, increasing the packet size increases the time taken to transfer the file.

The following table and graph shows how different packet sizes affect the total time taken to transfer a file of size 5047 bytes.

Packetsize	Total transfer time (sec)
32	2.14
64	1.54
128	1.17
256	0.76
512	0.83
1024	1.68

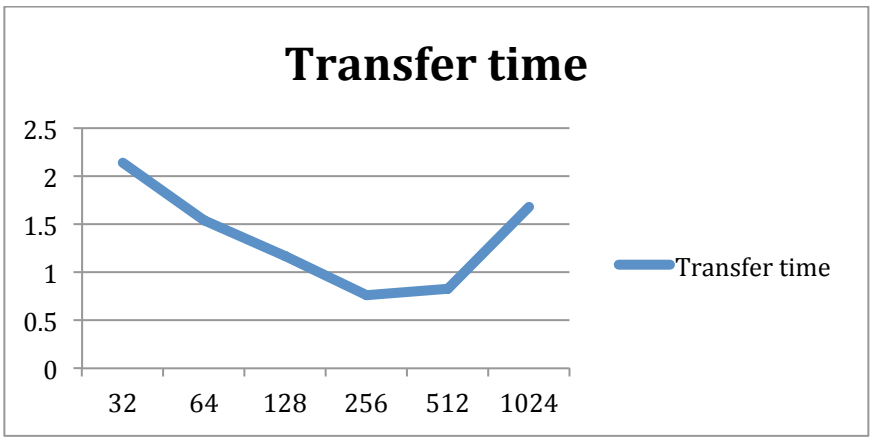

Fig. 1 Packet size vs Transfer time

From the graph we can see that the transfer time decrease as the packet size increase until it reach the lowest for packet size 256 bytes. After this point, the transfer time starts increasing once again with increase in packet size. This increase is attributed to the fact if the packet size is large, the last packet might contain a lot of free space and this free space is padded with junk characters before transmission.

B. Sending window size

Varying the sending window size affects the overall performance of the Go – Back – N protocol. When the size of the window size decreases, the time taken to transfer the files also decreases. This is attributed to the fact that, in the event of an error in the transmitted packet, the entire window needs to be transmitted. Hence keeping the size of the sending window to a reasonable size is essential.

The following table and graph shows the effect of varying the size of the sending window to transfer a file of file 5407 bytes and packet size of 64 bytes.

Sending window size	Total transfer time (sec)
85	1.54
65	0.98
45	0.74
25	0.62
5	0.63
3	0.58

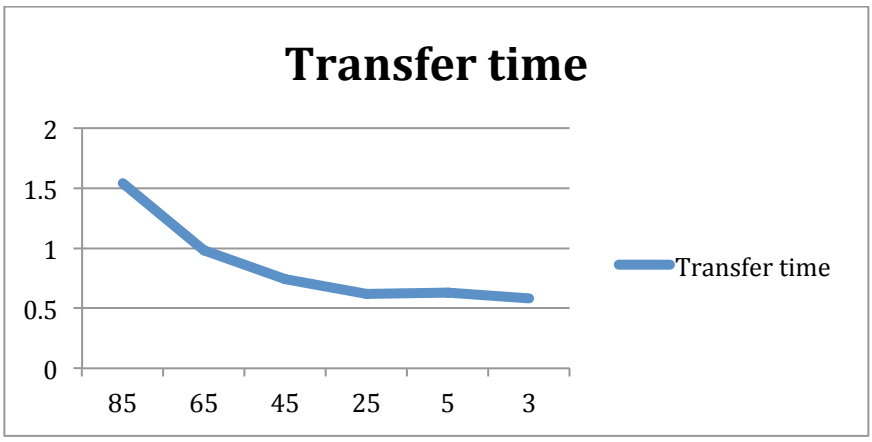

Fig 2. Sending window size vs Transfer time

From the graph we can see that by decreasing the sending window size, the time taken to transfer the file also decreases. The slight increase in value for the window size of 5 can be neglected due to the fact that this analysis was done a small file of size 5407 bytes.

C. Timeout interval

Timeout interval is the time period for which the sender waits to receive an acknowledgement before retransmitting the packet. If the timeout interval is too small, the sender does not give the receiver enough time to do the checksum verification and send an acknowledgement resulting in retransmitting the frame. On the other hand, if the timeout is too large, then the sender waits for a long time to retransmit resulting in increased transfer time.

The following table and graph shows the effect of varying the timeout interval in the go back n protocol for a file of size 5047 bytes, packet size 1024 bytes and one damaged packet.

Timeout interval (sec)	Total transfer time (sec)
1	7.99
2	11.71
3	16.74

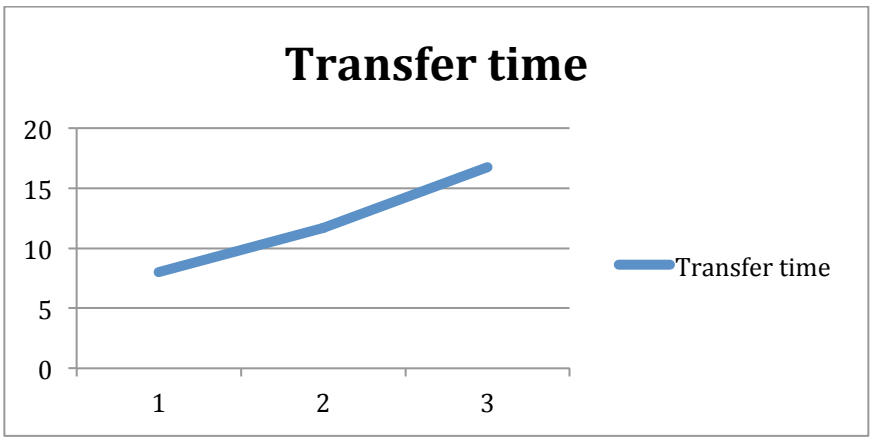

Fig 3. Timeout interval vs transfer time

From the graph we can clearly see how the transfer time increase with increase in timeout interval.

The following table shows how packetsize, windowsize affect the transfer time with respect to a timeout interval of 1sec.

Packetsize	Windowsize	Transfer time (sec)
1024	6	7.99
512	11	11.09
256	22	22.52

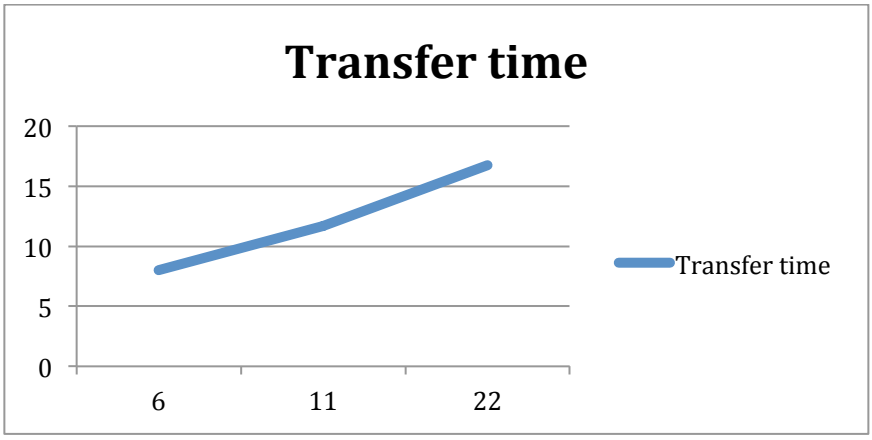

Fig 4. Transfer time vs Windowsize

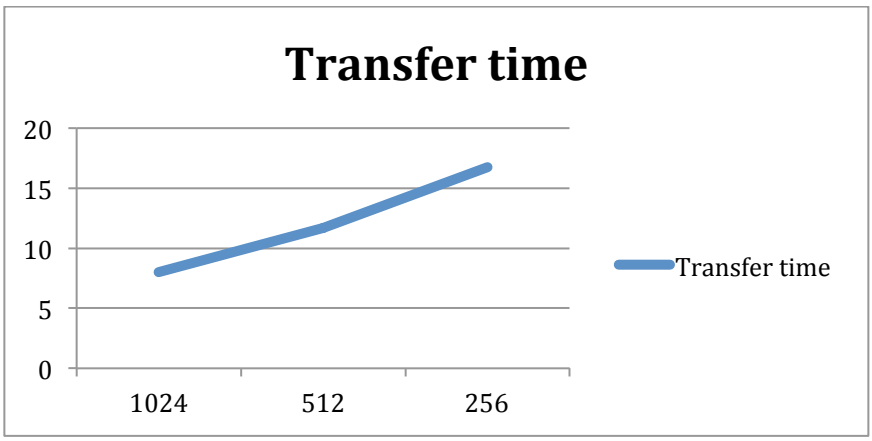

Fig 5. Transfer time vs Packetsize

From the above graphs, we can infer that the transfer time increases, when the packet size decreases or when the window size increases. This is due to the fact that, when the window size is large, more packets need to be transmitted. Hence it is advisable to choose a smaller timeout interval when the packet size is small or when the window size is large.

D. Static vs Dynamic timeout intervals

For transmitting packets of fixed size, a static timeout interval works well since the waiting time before retransmission is constant. However if the packet size is varying continuously, dynamic timeouts work better. This is because if the packet size is small and a large timeout interval is chosen in case of static timeout, then a lot of time is wasted waiting. Hence a dynamic timeout that is able to adjust the waiting time dynamically depending on the packet size is more suitable for varying packet sizes.

E. Implementing UNIX timeouts

The timeout is implemented using gettimeofday() function which is available in MacOSX and is available in <time.h> library. This function returns the current time of the day in microseconds. Once a frame is transmitted, the begin time is computed. Then a loop is started and an elapsed time is computed constantly. The loop keeps executing until the elapsed time reaches the timeout interval. If an acknowledgement is not received by then, the frame is retransmitted.

F. Sequence numbers

It is not advisable to use infinite sequence numbers since they can be reused inside a window.

The maximum sequence number needs to be at least greater than half the size of the sending window to avoid duplicate sequence numbers residing in the sending window. For example consider a sending window size of 4. If the maximum sequence number is 2 then the sending window looks like [0,1,2,0]. We have two packets with sequence

number 0. Hence we need to ensure the maximum sequence number is greater than half the size of the sending window.

It is also sufficient for the maximum sequence number to be equal to the window size - 1. Consider a window size of 4 and maximum sequence number of 3. Then the window looks like [0,1,2,3]. Before the next packet 0 enters the sliding window, the current packet 0 would have left the window. Hence it is sufficient for maximum sequence number to be equal to window size – 1.

G. Checksum algorithm for error detection

The checksum algorithm is for error detection due to its simplicity. It is easy to implement and hence is reasonable to use for error detection.

However several kinds of errors can get past the checksum algorithm. For instance, if some bytes are reordered, the checksum algorithm might not be able to detect it.

Depending upon the importance of the application or the data being transferred the error detection algorithm can be chosen. For more critical applications, more sophisticated error detection algorithms like Cyclic Redundancy Check (CRC) can be used. For simpler applications, the checksum algorithm is useful.

H. Frame format

It is sufficient to keep track of the data and the checksum. This is because, once we establish a connection with the receiver, all packets are going to be transmitted to the same destination and hence we need not keep track of destination address for each individual packet.

III. Conclusion

Based on this analysis it can be concluded that several factors affect the performance of the go back n protocol. Depending upon the application, these factors can be adjusted to achieve the maximum throughput or minimum transfer times.