EE3204E Lab Assignment

Rohit Mukherjee

30/10/2013

Table of Contents

Introduction

The problem statement is:

Develop a UDP-based client-server socket program for transferring a large message. Here, the message transmitted from the client to server is read from a large file. The message is split into short data-units which are sent by using stop-and-wait flow control. Also, a data-unit sent could be damaged with some error probability. Verify if the file has been sent completely and correctly by comparing the received file with the original file. Measure the message transfer time and throughput for various sizes of data-units. Also, measure the performance for various error probabilities and also for the error-free scenario.

1) Use of C – sockets API call:

```
//Create Socket for Operation
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0)
{
    printf("Error in Creating Socket");
    exit(EXIT_FAILURE);
}
```

- Family specifies the protocol family (AF_INET for Internet, PF_INET for TCP/IP).
- Type specifies the type of service (SOCK_STREAM, SOCK_DGRAM).
- Protocol specifies the specific protocol (usually 0, which means the default).
- The socket() system call returns a socket descriptor (small integer) or -1 on error.
- socket() allocates resources needed for a communication endpoint - but it does not deal with endpoint addressing.

2) Initializing and using Bind Call (Only for server)

```
//Initialize Socket Address
my_address.sin_family = AF_INET;
my_address.sin_port = htons(MYUDP_PORT);
my_address.sin_addr.s_addr = INADDR_ANY;
bzero(&(my_address.sin_zero), 8);

//Bind Socket
if (bind(sockfd, (struct sockaddr *) &my_address, sizeof(struct sockaddr)) == -1) {
    printf("error in binding");
    exit(EXIT_FAILURE);
}
```

- The bind() system call is used to assign an address to an existing socket.int bind(int sockfd, const struct sockaddr *myaddr, int addrlen);bind returns 0 if successful or -1 on error.
- calling bind() assigns the address specified by the sockaddr structure to the socket descriptor
- You can give bind() a sockaddr_in structure - bind(mysock, (struct sockaddr*) &myaddr,sizeof(myaddr));
- Bind() call is used to bind the server to a well known port (in this case, MY_UDP_PORT)
- We don't use client to bind to a specified port because OS can assign it any available port number

3) Using sendto(…) and recvfrom(…) API calls

```
((n=recvfrom(sockfd, &received_message, DATALEN, 0, (struct sockaddr *)&client_address, &client_address_length)) > 0) {
datagram_count++;
printf("Received Datagram %d \n",datagram_count);
if(received_message[n-1] == '\0')
{
    end = 1;
    n--;
}
memcpy((buf+lseek), received_message, n);
lseek += n;

if ((sendto(sockfd, &acknowledgement, sizeof(acknowledgement), 0, (struct sockaddr *) &client_address, client_address_length))
{
    printf("Error in sending acknowledgement, n = %d\n",n);
    exit(EXIT_FAILURE);
}
else
    printf("Acknowledgement sent\n");
```

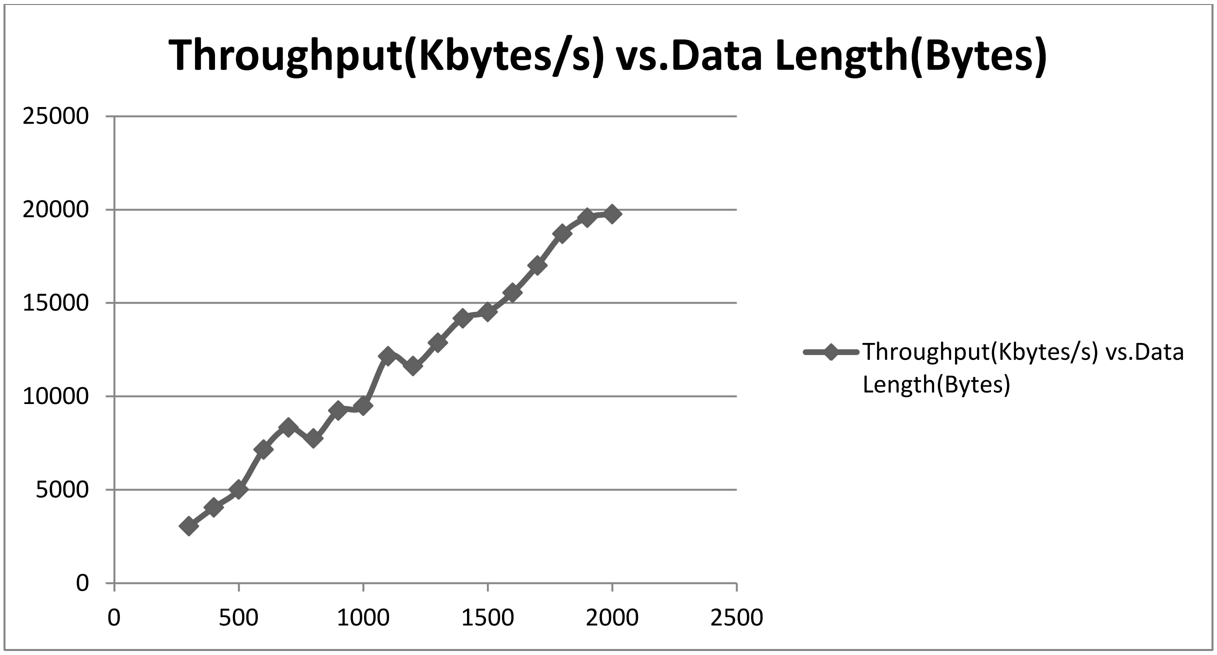
- Recvfrom(…) call receives packets on the MY_UDP_PORT and stores the client address (in packet header) inside client_address and length of client_address in client_address_length respectively
- Sendto(..) call sends the structure of type ack_so to the client_address

Results of Experiment

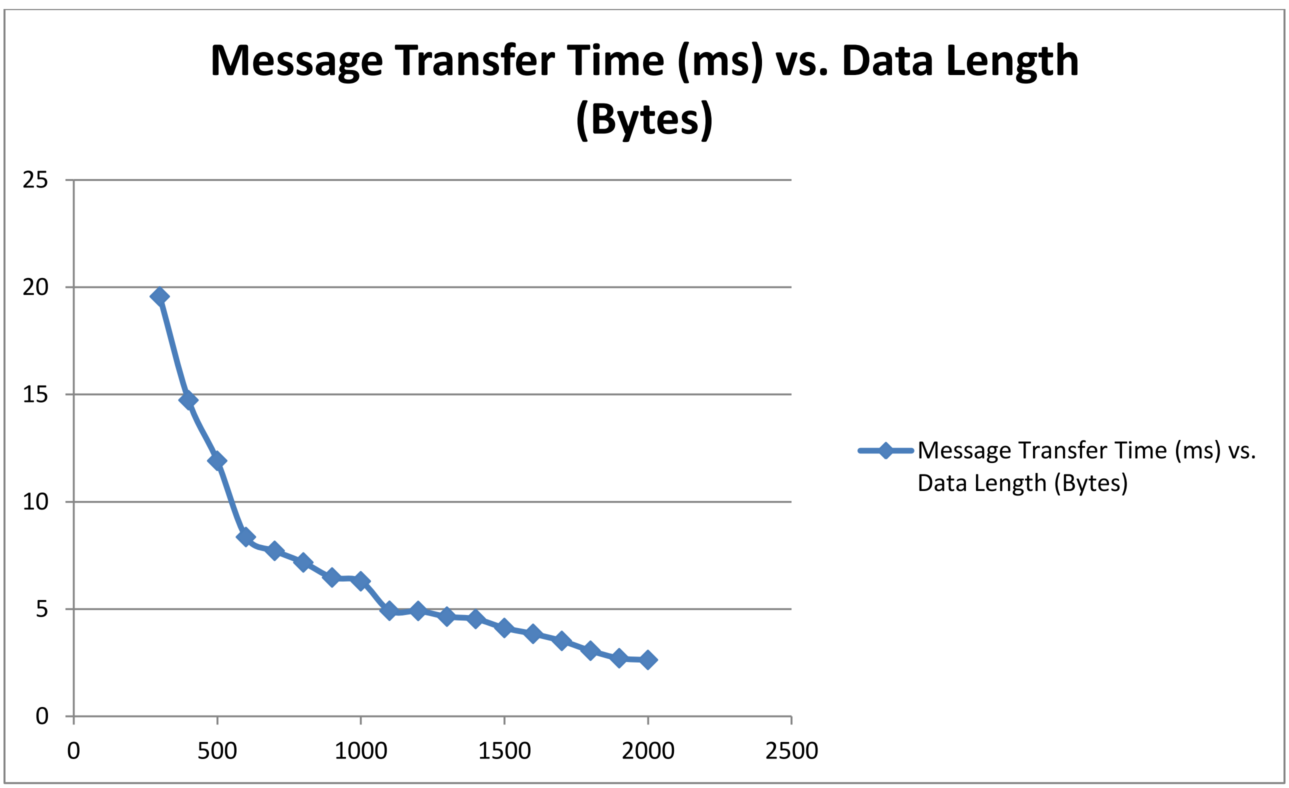
Varying Data Length and Keeping Frame Error Probability = 0

In the first experiment, I varied the DATALEN and from 300 to 2000 with zero frame error probability. The Message Transfer Times decreased and Throughput increased with larger message size. This is because fewer numbers of packets in total were sent through and this reduced the overhead required for framing the different packets.

Datalength	Packet Length	Throughput(Kbytes/s)	Message Transfer Time(ms)	Frame Error Probability (P)
300	308	3055.80835	19.567	No Frame Error Probability
400	408	4058.164795	14.734	
500	508	5021.246094	11.908	
600	608	7155.696777	8.356	
700	708	8339.330078	7.712	
800	808	7753.241699	7.17	
900	908	9243.004883	6.469	
1000	1008	9500	6.294	
1100	1108	12153.048828	4.92	
1200	1208	11628.354492	4.912	
1300	1308	12872.550781	4.645	
1400	1408	14180.268555	4.537	
1500	1508	14523.439453	4.117	
1600	1608	15550.844727	3.845	
1700	1708	17010.810547	3.515	
1800	1808	18708.697266	3.056	
1900	1908	19565.771484	2.707	
2000	2008	19760.943359	2.627	



Increasing the datalength leads to an increase in the Throughput due to reduction in framing overhead

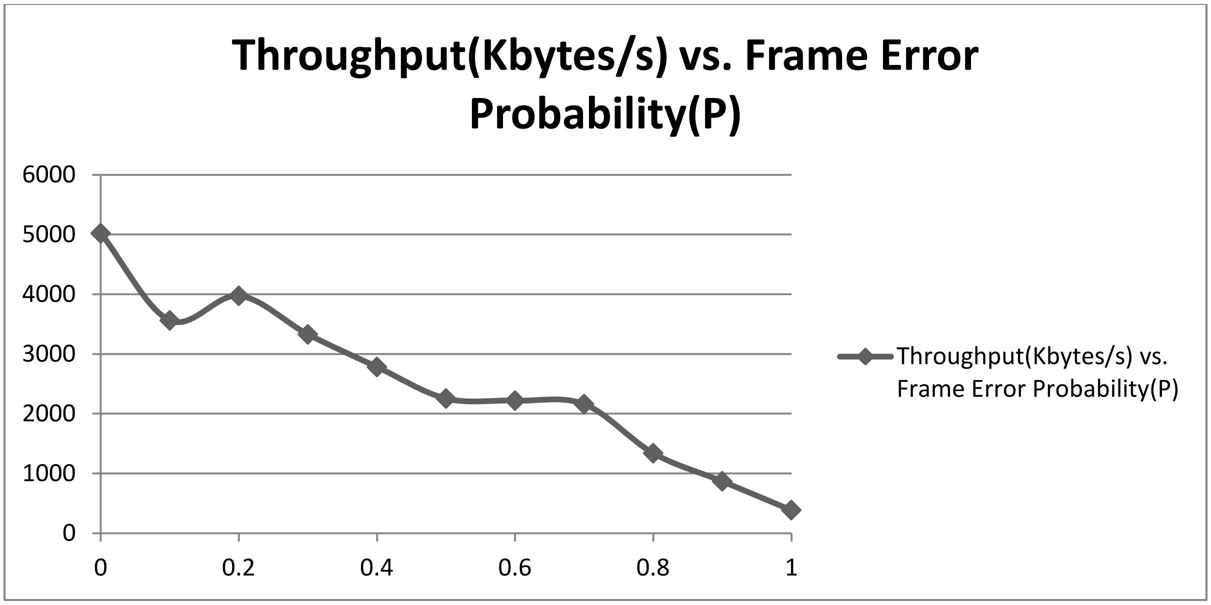


Increasing the datalength leads to a decrease in the Throughput due to reduction in framing overhead

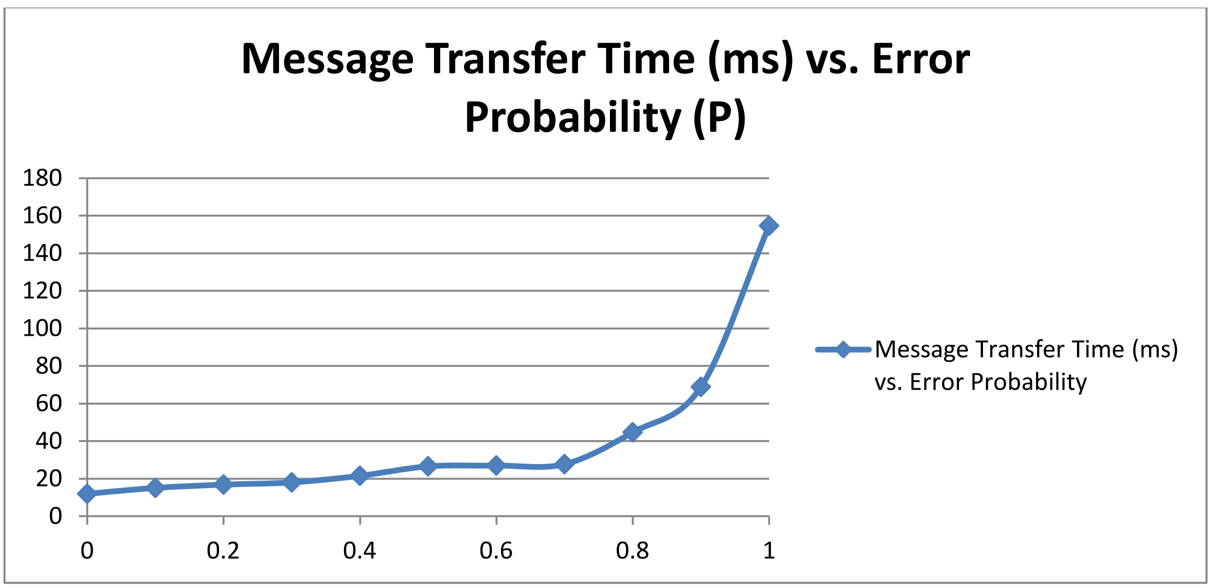
Varying Error Probability for Fixed Data Length = 500

With an increase in the probability of error, the number of retransmissions increased and therefore effective throughout decreased. Message transfer time increased as many bad packets were sent due to error probability so good packets had to be resent. Therefore, overall message transfer time increased and effective throughput decreased.

Datalength	Packet Length	Throughput(Kbytes/s)	Message Transfer Time(ms)	Frame Error Probability (P)
500	508	5021.246094	11.908	0
		3562.287842	15.046	0.1
		3974.013184	16.785	0.2
		3328.86084	17.962	0.3
		2783.270508	21.483	0.4
		2254.298096	26.524	0.5
		2219.734863	26.937	0.6
		2160.620117	27.674	0.7
		1339.388916	44.642	0.8
		868.743347	68.827	0.9
		386.756897	154.601	1



Increase in Frame Error Probability (P) leads to decrease in effective throughput



Increase in Frame Error Probability (P) leads to increase in Message Transfer Time

Conclusion

From the experiment it can be observed, that increasing packet size leads to lesser number of packets and therefore less overhead for framing each packet and hence effective throughput increases. It is also observed that increasing frame error probability leads to an increase in effective throughput.