Computer Networks Project Part 2

PERFORMANCE ANALYSIS MEASURING ROUND TRIP TIME (RTT) AND THROUGHPUT

Summary
[
In this report, we will do a performance analysis over TCP and UDP socket programming by measuring round trip time and throughput.]

Nurefşan
Sertbaş
040110078
sertbasn@itu.edu.tr

Introduction and Describing the Problem

The project has 2 main part as described in the project description. In this report, I explain the second part of the project; **Performance Analysis** measuring **Round Trip Time (RTT)** and **Throughput**. We choose Hangman as a game to implement so that it enables 2 players at the same time. Also, the written C code enables playing game between multi players and multi sessions.

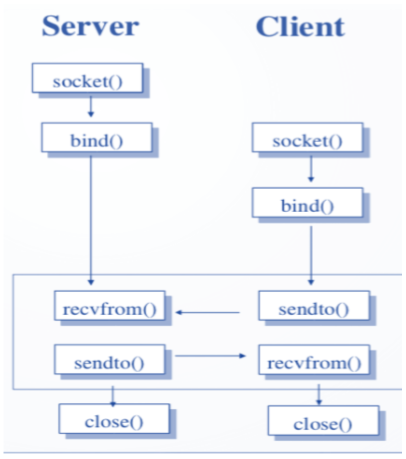
I will explain all the steps in sub sections with details as following.

Main Principle of the Program

UDP is not a connection oriented protocol. It means that there is no permanent connection like TCP. Also, there is no connect request from client or accept-listen operations at server side. UDP just bind to an address/port and waits for packets from all clients. At the server there is only one socket listening. When a packet is received we can find the sender of the packet by the packet's source IP and we can send back a response to the sender by using this IP.

Changes in Program to immigrate from TCP to UDP

Unlike TCP connection, there is no listen and accept operations at the server side as can be seen from the below figure so that we have to rewrite our code to satisfy this case.



At TCP part, when any request come to server server creates new socket and assigns that socket id as socked id of the requested client. In UDP there is no such a case so that we can distinguish two clients from each other by using their IP addresses.

In order to make this change we declare an array to hold address details of the clients at the server side as

```
int allclientadresses[CLIENTS]; //max 20 client is allowed
```

and in each client we have done the following assignments

```
struct userThread threadinfo;
```

```
threadinfo.sockfd = sockfd;
threadinfo.oyunId= 0;
threadinfo.userid=userindex;
allclientadresses[userindex]=ntohs(client_addr.sin_port);
userindex++;
```

Note that we have used threads in implementation so that after those operations we call pthread_create function to create pthread with assigned values.

Secondly, the send and recv functions are not used in UDP. Instead of them, we have used sendto and recvfrom functions. Their prototypes are given below.

```
ssize_t recvfrom(int s, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);

ssize_t sendto(int s, const void *buf, size_t len,
               int flags, const struct sockaddr *to,
               socklen_t tolen);
```

Measuring Round Trip Time

In order to measure the Round Trip Time (RTT) in TCP/UDP connection we have followed a simple procedure as stated below:

1→Read the current time with time()

2→Send a message to the server/client on an already opened socket

3→Wait the message back

4→ When the message is read, record the current time using time()

From the definition we can say that RTT is the difference between these two recorded times.

Measuring Throughput

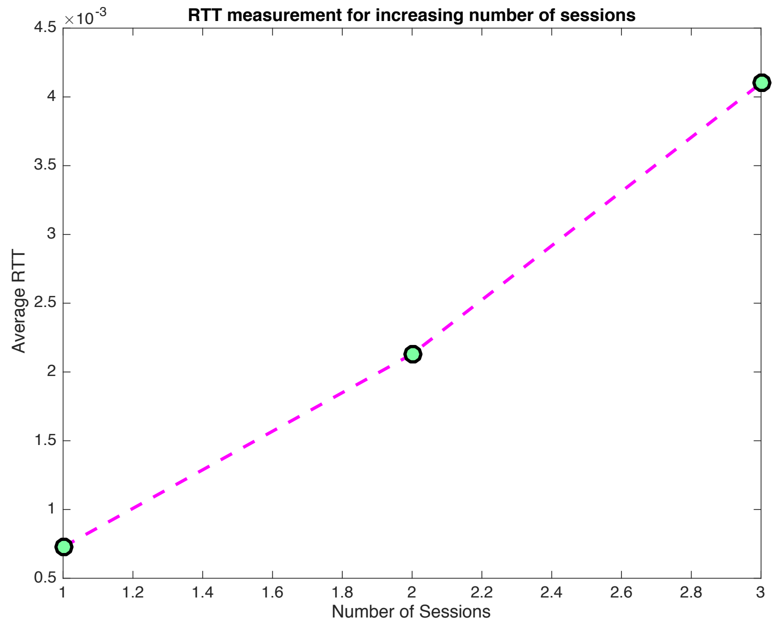
Throughput is significant parameter that shows the performance of the network. It is usually measured in bits per second (bit/s or bps) as we did.
In order to measure it we use the return values of send/recv functions to obtain how many bits are sent/received. Also, we use the similar approach in measuring RTT to measure the time that passed transferring those bits. Then, we obtained a ratio called throughput.

Note: Each move is not suitable to measure RTT in our game so that we cannot get their average we just measure move operations throughput and RTT to get resulted values.

Results

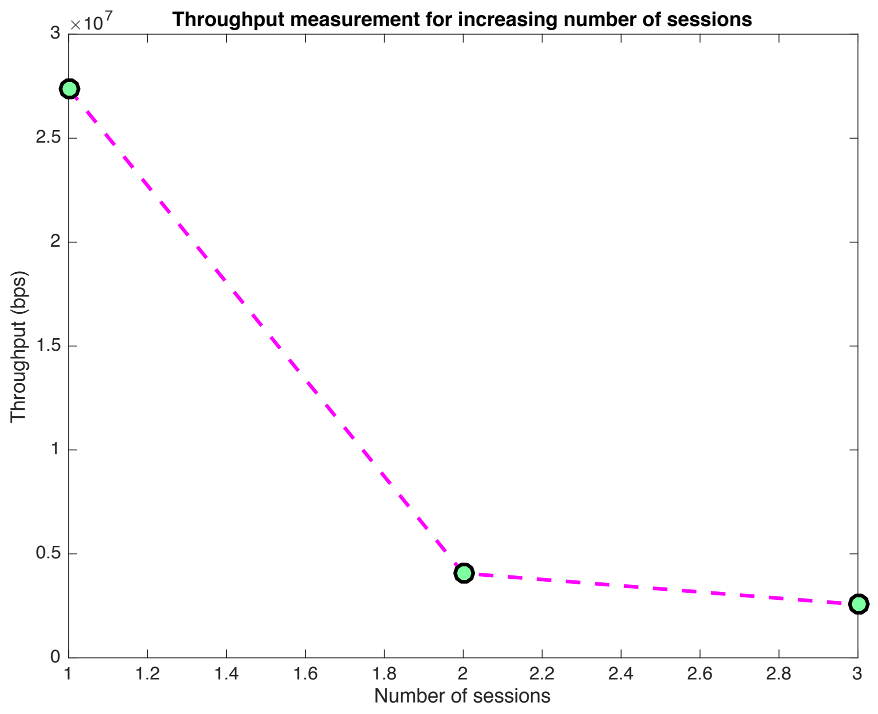
For an example game, we measured RTT for move operation in TCP as follows:

Trial index	Return time	Send time	RTT	Average RTT
1	0.004918	0.004269	6.4900e-04	7.2767e-04
2	0.004634	0.004030	6.0400e-04	
3	0.004178	0.003248	9.3000e-04	
1	0.005665	0.003078	0.0026	0.00213
2	0.005912	0.003885	0.0020	
3	0.005273	0.003440	0.0018	
1	0.007278	0.003140	0.0041	0.0041
2	0.007498	0.003350	0.0041	
3	0.007634	0.003468	0.0042	



We can say that in each time we send back approximately the same number of bits because we use move struct as a content of the message for this part. As a result, number of bits sent are the same but time that passed to sent those bits are increasing for increasing number of sessions so that network throughput is decreases.

Trial index	Number of bits	Transfer time	Throughput	Average Throughput(bps)
1	9840	0.003724-0.003140	1.6849e+07	2.738222666382462e+07
2	6560	0.003872-0.003611	2.5134e+07	
3	9840	0.004052-0.003807	4.0163e+07	
1	9840	0.005274-0.003036	4.3968e+06	4.067279907510817e+06
2	6560	0.005466-0.003271	2.9886e+06	
3	9840	0.005562-0.003519	4.8164e+06	
1	9840	0.006949-0.003494	2.8480e+06	2.567857214892233e+06
2	6560	0.007059-0.003736	1.9741e+06	
3	9840	0.007265-0.003850	2.8814e+06	


Throughput measurement for increasing number of sessions
×10^7
Throughput (bps)
Number of sessions