**GRS CS 655**

**Graduate Intro to Computer Networks**

**Programming Assignment 1**

Name: Haoxuan Sun

BUID: U58198360

Data: 10/6/2022

**How to run pa1part1 program?**

Run program ***EchoServer*** on csa1, csa2, csa3.bu.edu.

1. Enter the directory where these files are located.
2. On command line: ***javac EchoServer.java***
3. On command line: ***java EchoServer 58989***(any ports from 58000 to 58999)

Now the **EchoServer** is working and waiting for a client connection.

Run program ***EchoClient*** on local machine.

1. On command line: ***javac EchoClient.java***
2. On command line: ***java EchoClient ‘csa1.bu.edu’ 58989***(csa1 or csa2 or csa3 depends on what address you run server on, and the same port number the server is listening)

Now the ***EchoClient*** is connecting to the server.

Use Control + C to exit EchoClient program

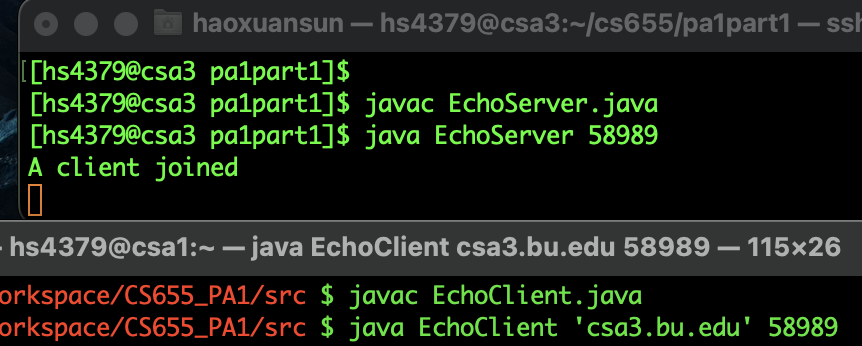


Figure Server and Client ready to communicate

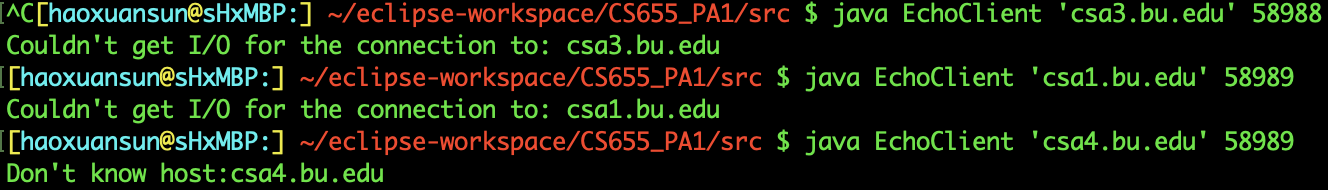
**pa1part1 tradeoff and extension**

Server program can be implemented to accept multi clients at the same time (multi thread).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Case no.** | **Address of Server** | **Port of Server** | **EchoClient argument [0]** | **EchoClient argument [1]** | **Terminal Output** | **Client Input** | **Echo Output** | **Expected?** |
| **1** | csa3.bu.edu | 58989 | csa3.bu.edu | 58989 | - | Hello | Hello | Yes |
| **2** | csa3.bu.edu | 58989 | csa3.bu.edu | 58989 | - | DLROW OLLEH | DLROW OLLEH | Yes |
| **3** | csa3.bu.edu | 58989 | csa3.bu.edu | 58988 | Couldn't get I/O for the connection to: csa3.bu.edu | - | - | Yes |
| **4** | csa3.bu.edu | 58989 | csa1.bu.edu | 58989 | Couldn't get I/O for the connection to: csa1.bu.edu | - | - | Yes |
| **5** | csa3.bu.edu | 58989 | csa4.bu.edu | 58989 | Don't know host:csa4.bu.edu | - | - | Yes |

**pa1part1 Testing Documents**

Table Testing Record

文本

描述已自动生成

Figure Test Case 3 ~ 5

Figure Test Case 1 ~ 2

5 tests are run to check the correctness of both server and client program. All the outcomes indicate that the program have achieved the design objectives. The first 2 cases are normal client input. The third case checks what if port number is inconsistent. The fourth case checks what if server address is incorrect. The fifth case check what if the server address in no real.

**How to run pa1part2 program?**

Run program ***Server*** on csa1, csa2, csa3.bu.edu.

0. Enter the directory where these files are located.

1. On command line: ***javac Server.java***
2. On command line: ***javac MultiThread.java***
3. On command line: ***java Server 58989***(any ports from 58000 to 58999)

Now the **Server** is working and waiting for a client connection.

Run program ***Client*** on local machine.

1. On command line: ***javac Client.java***
2. On command line: ***java Client ‘csa1.bu.edu’ 58989***(csa1 or csa2 or csa3 depends on what address you run server on, and the same port number the server is listening)

Now the ***Client*** is connecting to the server.

Use Control + C to exit EchoClient program

Output: Please enter measure type ('rtt' for RTT or 'tput' for throughput):

Input: 'rtt' or 'tput' (taking rtt as an example here)

Output: Please enter the number of measurement probes that the server should expect to receive:

Input: 20

Output: Please enter the number of bytes in the probe's payload:

Input: 1000

Output: Please enter the amount of time that the server should wait before echoing the message back to the client (in ms):

Input: 0

Output: ########### (Client sending and receiving logs)

Output: Mean RTT of 20 probes: 282 ms

CTP sending... t

CTP receiving... 200 OK: Closing Connection

- - - Client Close - - -

**Description**

The location of my client machine is at home. My laptop installs MacOS and home WiFi is used to access the network. The server part in whole part 2 assignment is done on csa1.bu.edu or ‘pcvm3-2.geni.uchicago.edu @ 50000’. For every averaged RTT or throughout, it is averaged by at least 3 measurements (detailed can be found later). And before the average is taken, 20 probes of a certain payload’s size is sent and received in every measurement. The line charts are produced using office software: Numbers. Both data file and line chart are provided in the attached data files.

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

Screenshot of a measured RTT (probe number = 20; message size = 1 byte; server delay = 0ms; measure no. 3)

**pa1part2 Testing Documents**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Case no.** | **Address of Server** | **Port of Server** | **Client argument [0]** | **Client argument [1]** | **Terminal Output** | **Client Input** | **Echo Output** | **Expected?** |
| **1** | csa1.bu.edu | 58989 | csa1.bu.edu | 58989 | - | Hello | Hello | Yes |
| **2** | csa1.bu.edu | 58989 | csa1.bu.edu | 58989 | - | DLROW OLLEH | DLROW OLLEH | Yes |
| **3** | csa1.bu.edu | 58989 | csa1.bu.edu | 58988 | Couldn't get I/O for the connection to: csa1.bu.edu | - | - | Yes |
| **4** | csa1.bu.edu | 58989 | csa3.bu.edu | 58989 | Couldn't get I/O for the connection to: csa3.bu.edu | - | - | Yes |
| **5** | csa1.bu.edu | 58989 | csa4.bu.edu | 58989 | Don't know host:csa4.bu.edu | - | - | Yes |

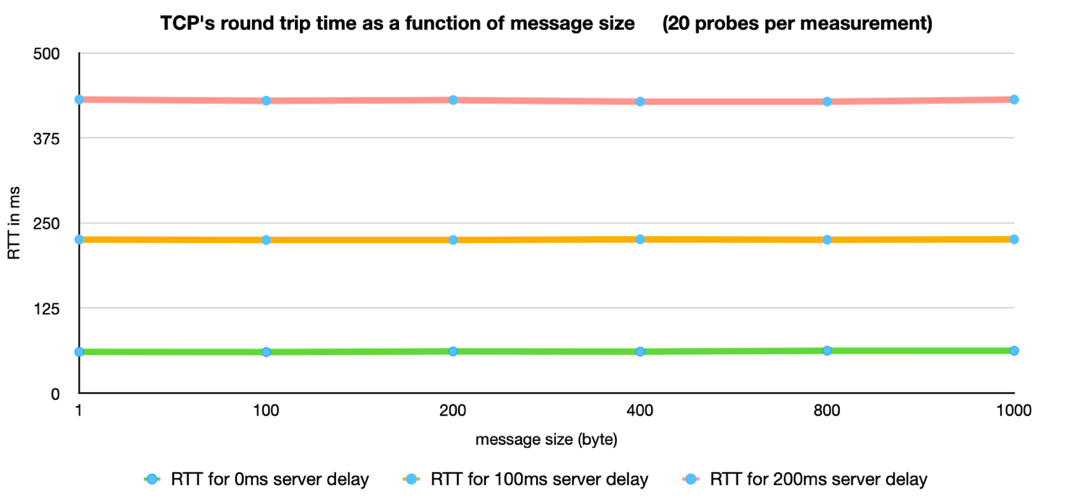
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Case no.** | **Get Measurement type** | **Get probe number** | **Get message size** | **Get server delay** | **Excepted output** | **output** | **Expected?** |
| 1 | “rt” | - | - | - | Ask again | Ask again | Yes |
| 2 | “rtt” | - | - | - | No error | No error | Yes |
| 3 | - | -1 | - | - | Ask again | Ask again | Yes |
| 4 | - | 0 | - | - | Ask again | Ask again | Yes |
| 5 | - | 1 | - | - | No error | No error | Yes |
| 6 | - | - | -1 | - | Ask again | Ask again | Yes |
| 7 | - | - | 0 | - | Ask again | Ask again | Yes |
| 8 | - | - | 1 | - | No error | No error | Yes |
| 9 | - | - | - | -1 | Ask again | Ask again | Yes |
| 10 | - | - | - | 0 | Ask again | Ask again | Yes |
| 11 | - | - | - | 1 | No error | No error | Yes |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Terminal Input** | **Excepted output** | **output** | **Expected?** |
| 1 | java Client csa1.bu.edu | Wrong Input Format! 'java Client <host name> <port number>' | Wrong Input Format! 'java Client <host name> <port number>' | Yes |
| 2 | Java Server | Wrong Input Format! 'java Server <port number>' | Wrong Input Format! 'java Server <port number>' | Yes |

**Summary of Results**

The first two figures show the result obtained through running experiment on csa1.bu.edu.  
Figure 1 shows how the size of message affect TCP’s RTT. In the graph, each RTT value is averaged by 3 measured RTT and each measurement contains 20 probes (See attached file csa1\_data\_file for detailed dataset).

Figure 1 TCP's RTT as a function of message size (20 probe per measurement)



We can find that, no matter what the size of the message is, the RTT for a group of TCP remains the same (for a fixed server delay). The RTT for size 1 byte is almost the same as the RTT for size 1000 bytes when delay is 0ms. When change the server delay, the previous statement still stands. The RTT for size 1 byte is almost the same as the RTT for size 1000 bytes when delay is 100ms and 200ms.

However, if we focus on how RTT changes when set different server delay while keeping message size unchanged, we’ll find that the bigger server delay is, the bigger RTT is. It is easy to understand because server delay is to emulate paths with propagation delays. Longer path causes bigger RTT.

The difference between average RTT and 2 times of server delay are about 61ms, 25ms and 30ms respectively when server delay are 0ms, 100ms and 200ms.

Figure 2 TCP throughput as a function of message size (20 probes per measurement)

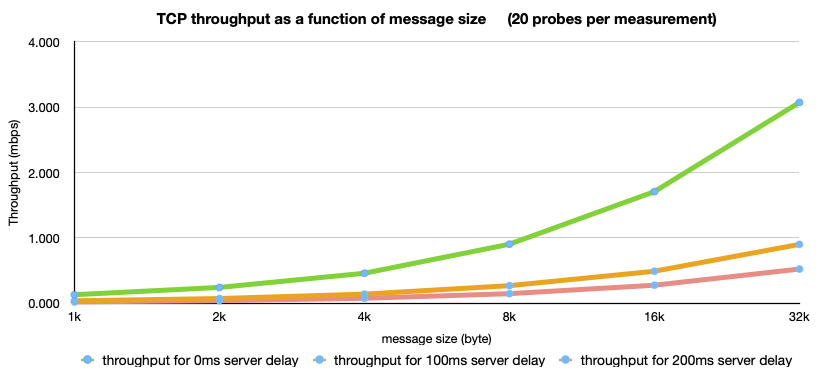
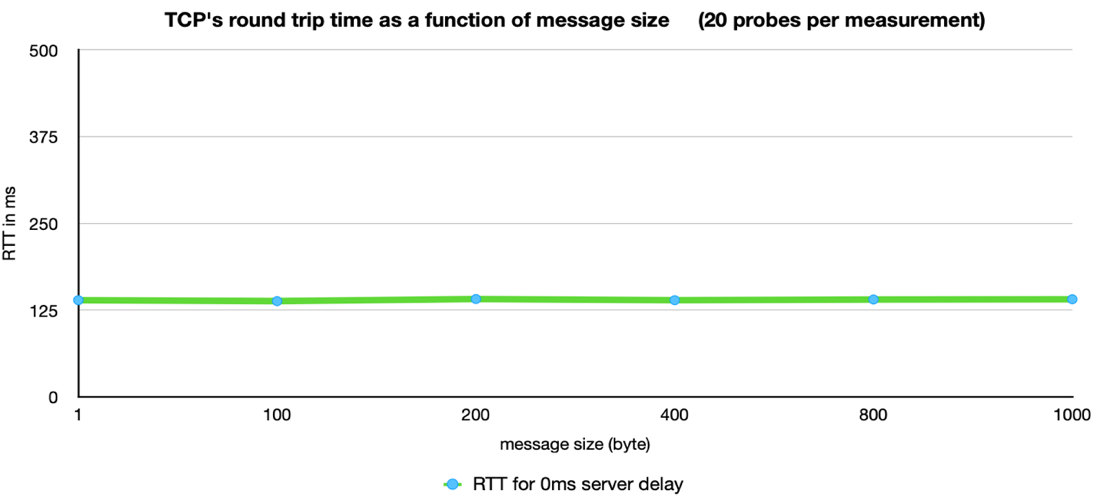


Figure 2 shows how the size of message affect TCP’s throughput (megabits per second). In the graph, each throughput value is averaged by 3 measured throughput and each measurement contains 20 probes (See attached file csa1\_data\_file for detailed dataset).  
We can see that, as message size increases, the throughout is also increasing, and throughout grows a little slower than the times of message grows. For example, when server delay is 0ms, throughout of 1000 byte is 0.128, throughout of 2000 byte is 0.240 (twice of 0.128 is 0.256); throughout of 4000 byte is 0.464 (twice of 0.240 is 0.480).  
If we focus on how throughout changes when set different server delay while keeping message size unchanged, we’ll find that the bigger server delay is, the smaller throughout is. It is also easy to understand because server delay is like a traffic jam. More time jammed on road, less average speed a vehicle has.

The following two figures show the result obtained through running experiment on ‘pcvm3-2.geni.uchicago.edu @ 50000’.

In the figure 3, each RTT value is averaged by 3 measured throughput and each measurement contains 20 probes (See attached file UChicago\_data\_file for detailed dataset).  
Like figure 1, figure 3 shows how the size of message affect TCP’s RTT and the server delay is only required to be 0ms. The TCP trend conclusion still stands. The RTT remains the same (almost) no matter how the size of message varies. The RTT on this server is much bigger that the RTT in figure 1, the physical location of the server and the length of path for messages travel from local machine to server could be the reason.

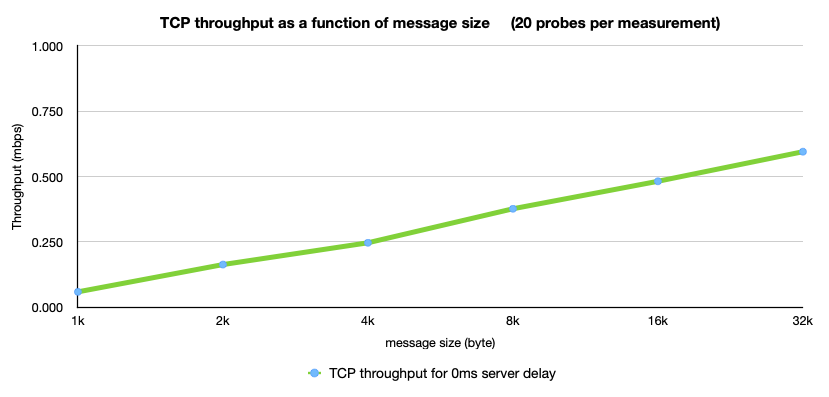
Figure 3 TCP's RTT as a function of message size (20 probes per measurement)



In the figure 4, each RTT value is averaged by 5 measured throughput and each measurement contains 20 probes (See attached file UChicago\_data\_file for detailed dataset).

Like figure 2, figure 4 shows how the size of message affect TCP’s throughput (megabits per second) and the server delay is only required to be 0ms. The graph shows the similar trend. As message size increases, the throughout is also increasing, but this time, throughout grows much slower than the times of message grows, and it is growing slower and slower. For example, throughout of 4000 byte is 0.246 mbps, throughout of 8000 byte is 0.374 mbps (message size doubles but throughout increases 55%); throughout of 16000 byte is 0.489 (message size doubles but throughout increases 31%).

Figure 4 TCP throughput as a function of message size (20 probes per measurement)



***csa1\_data\_file***

***图片包含 图形用户界面

描述已自动生成图表, 折线图

描述已自动生成***

***UChicago\_data\_file***

***图形用户界面

描述已自动生成图表, 折线图

描述已自动生成***