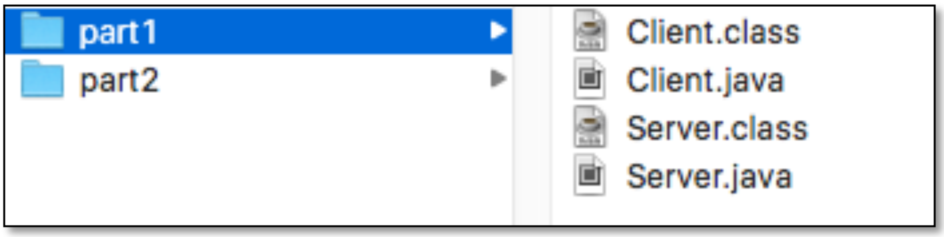
Report Document of CS655 PA1

Part1

Directory:



Compilation instruction:

Client.java and Server.java are included in a package called "part1", they can be compiled with command:

```
java Client.java
java Server.java
```

When we want to run the program, cd to the upper directory, and use:

```
java part1.Client
java part1.Server
```

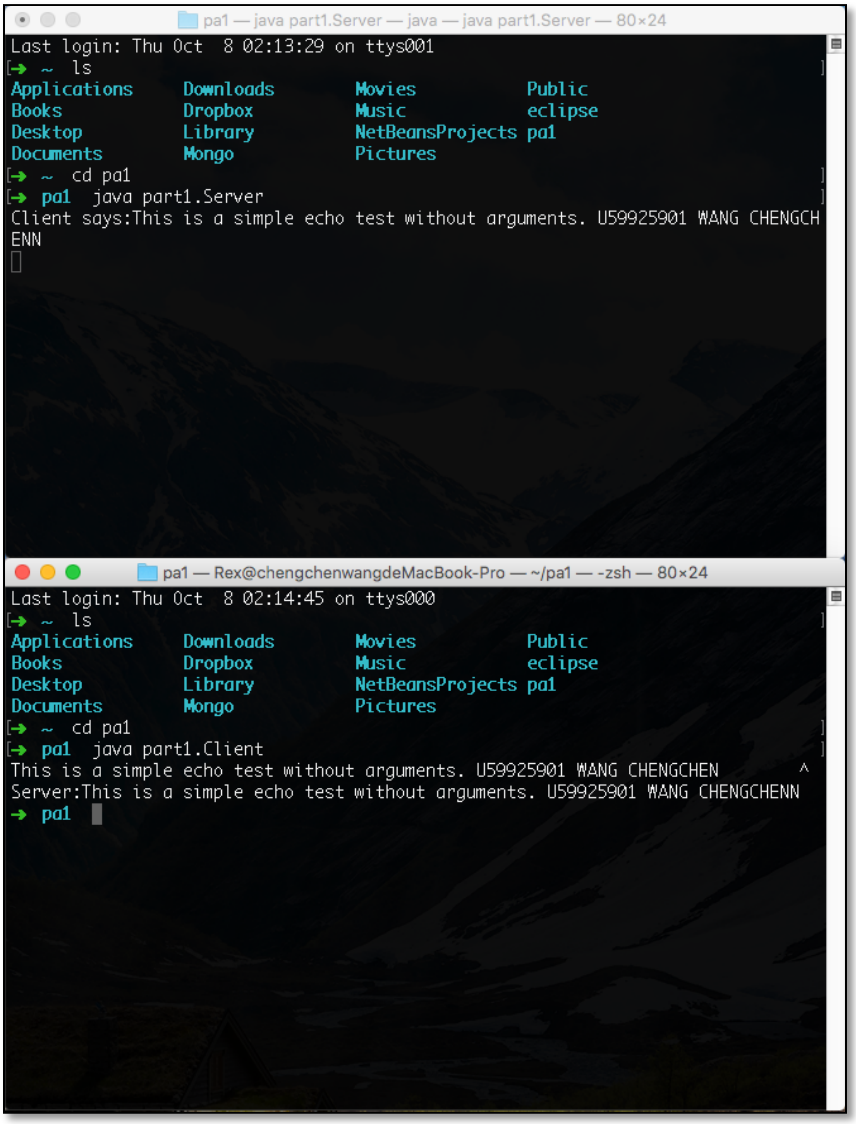
Argument Specification:

When using client, and user wants to specify Server IP address and port number, just run the client with command arguments like (use whitespace between IP and port):

```
java part1.Client 192.168.1.1 58669
```

Testing Results:

(1) Run client and server both locally without specifying IP and Port:

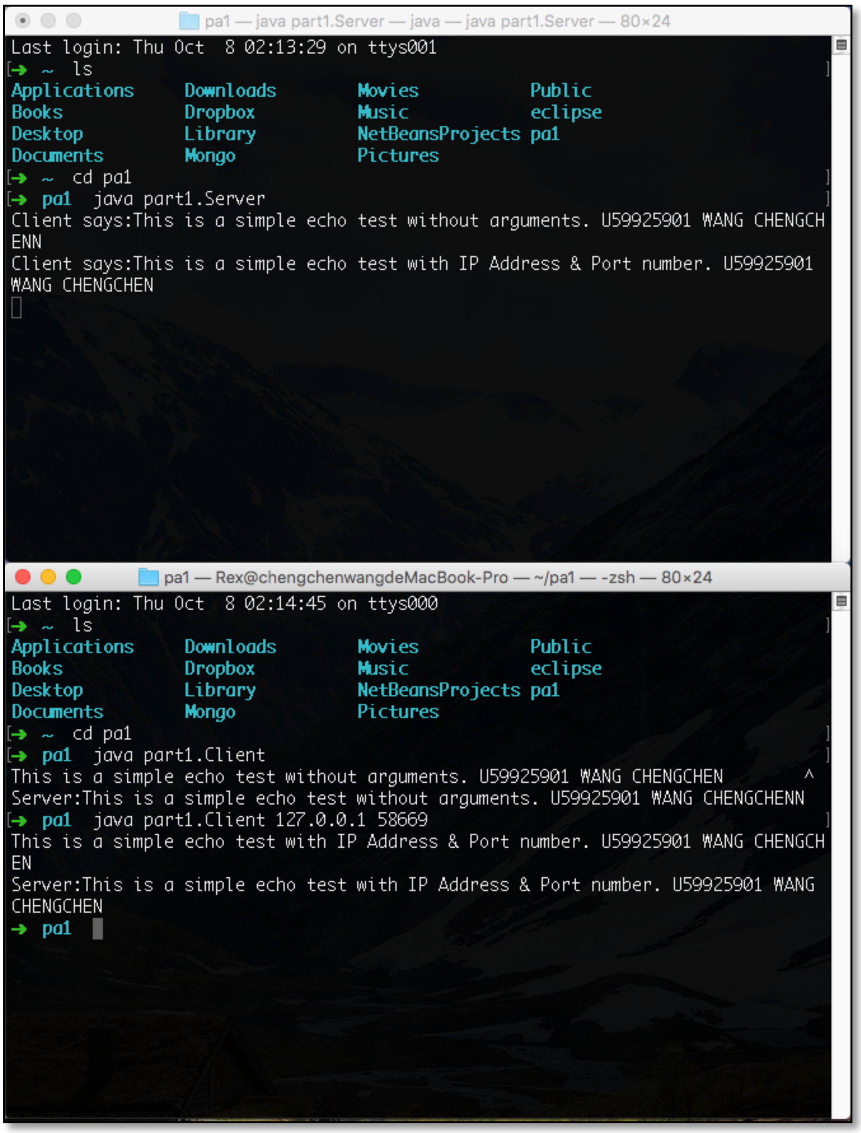

pa1 — java part1.Server — java — java part1.Server — 80×24

```
Last login: Thu Oct  8 02:13:29 on ttys001
→ ~ ls
Applications     Downloads        Movies           Public
Books            Dropbox          Music            eclipse
Desktop          Library          NetBeansProjects pa1
Documents        Mongo            Pictures
→ ~ cd pa1
→ pa1 java part1.Server
Client says:This is a simple echo test without arguments. U59925901 WANG CHENGCH
ENN
```

pa1 — Rex@chengchenwangdeMacBook-Pro — ~/pa1 — -zsh — 80×24

```
Last login: Thu Oct  8 02:14:45 on ttys000
→ ~ ls
Applications     Downloads        Movies           Public
Books            Dropbox          Music            eclipse
Desktop          Library          NetBeansProjects pa1
Documents        Mongo            Pictures
→ ~ cd pa1
→ pa1 java part1.Client
This is a simple echo test without arguments. U59925901 WANG CHENGCHEN        ^
Server:This is a simple echo test without arguments. U59925901 WANG CHENGCHENN
→ pa1
```

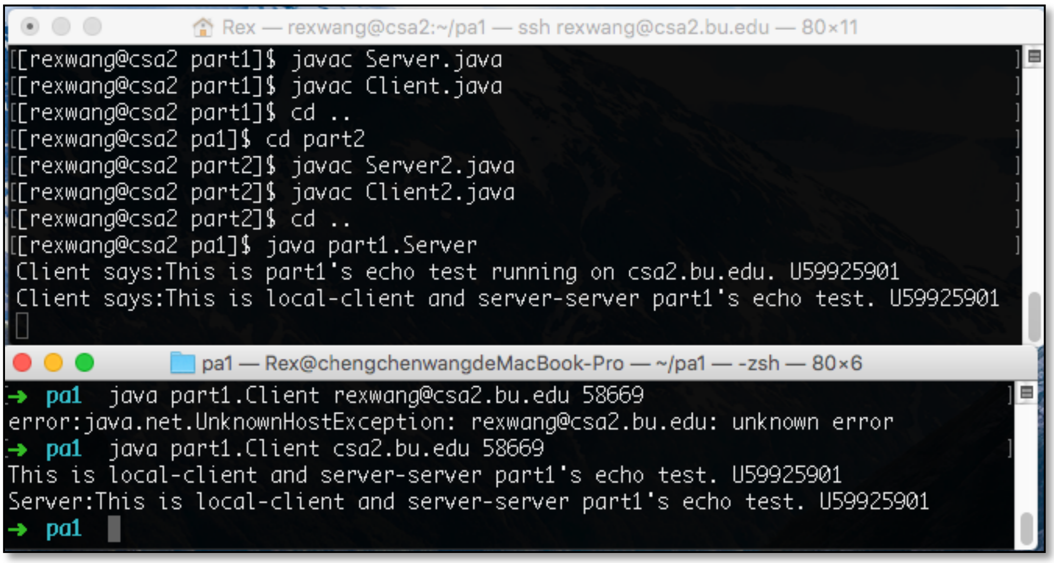
(2) Run client and server both locally with my specifying IP and Port (127.0.0.1 58669):



```
Last login: Thu Oct  8 02:13:29 on ttys001
➜ ~ ls
Applications     Downloads       Movies          Public
Books            Dropbox         Music           eclipse
Desktop          Library         NetBeansProjects pa1
Documents        Mongo           Pictures
➜ ~ cd pa1
➜ pa1  java part1.Server
Client says:This is a simple echo test without arguments. U59925901 WANG CHENGCH
ENN
Client says:This is a simple echo test with IP Address & Port number. U59925901
WANG CHENGCHEN
```

```
Last login: Thu Oct  8 02:14:45 on ttys000
➜ ~ ls
Applications     Downloads       Movies          Public
Books            Dropbox         Music           eclipse
Desktop          Library         NetBeansProjects pa1
Documents        Mongo           Pictures
➜ ~ cd pa1
➜ pa1  java part1.Client
This is a simple echo test without arguments. U59925901 WANG CHENGCHEN
Server:This is a simple echo test without arguments. U59925901 WANG CHENGCHENN
➜ pa1  java part1.Client 127.0.0.1 58669
This is a simple echo test with IP Address & Port number. U59925901 WANG CHENGCH
EN
Server:This is a simple echo test with IP Address & Port number. U59925901 WANG
CHENGCHEN
➜ pa1
```

(3) Run client locally and server on csa2@bu.edu with my specifying Port number 58669 (client runs on a different machine than server):


Rex — rexwang@csa2:~/pa1 — ssh rexwang@csa2.bu.edu — 80×11
pa1 — Rex@chengchenwangdeMacBook-Pro — ~/pa1 — -zsh — 80×6

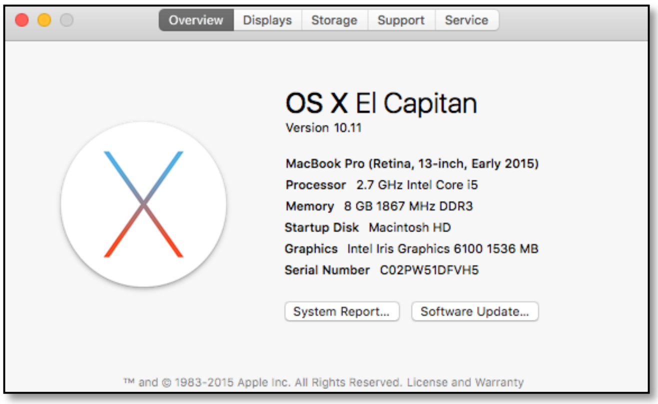
```
[rexwang@csa2 part1]$ javac Server.java
[rexwang@csa2 part1]$ javac Client.java
[rexwang@csa2 part1]$ cd ..
[rexwang@csa2 pa1]$ cd part2
[rexwang@csa2 part2]$ javac Server2.java
[rexwang@csa2 part2]$ javac Client2.java
[rexwang@csa2 part2]$ cd ..
[rexwang@csa2 pa1]$ java part1.Server
Client says:This is part1's echo test running on csa2.bu.edu. U59925901
Client says:This is local-client and server-server part1's echo test. U59925901
```

```
➜  pa1  java part1.Client rexwang@csa2.bu.edu 58669
error:java.net.UnknownHostException: rexwang@csa2.bu.edu: unknown error
➜  pa1  java part1.Client csa2.bu.edu 58669
This is local-client and server-server part1's echo test. U59925901
Server:This is local-client and server-server part1's echo test. U59925901
➜  pa1
```

Part2

Experiment Machine:

Type: MacBook Pro 13.3 retina with Mac OS 10.11



OS X El Capitan

Version 10.11

MacBook Pro (Retina, 13-inch, Early 2015)

Processor 2.7 GHz Intel Core i5

Memory 8 GB 1867 MHz DDR3

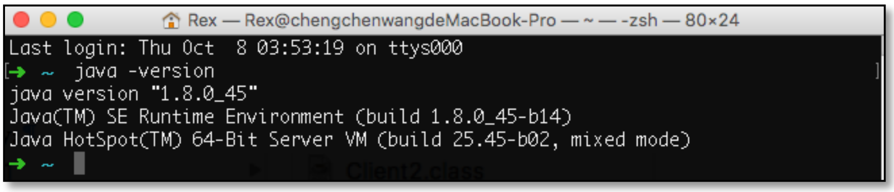
Startup Disk Macintosh HD

Graphics Intel Iris Graphics 6100 1536 MB

Serial Number C02PW51DFVH5

™ and © 1983-2015 Apple Inc. All Rights Reserved. License and Warranty

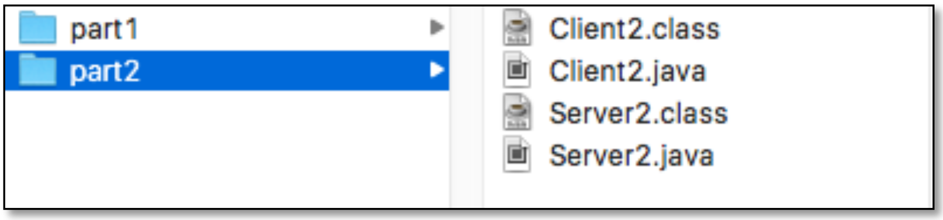
Java: 1.8



```
Last login: Thu Oct  8 03:53:19 on ttys000
➜  ~  java -version
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
➜  ~
```

Network: Comcast Service

Directory:



Compilation instruction:

Client2.java and Server2.java are included in a package called "part2", they can be compiled with command:

javac Client2.java
javac Server2.java

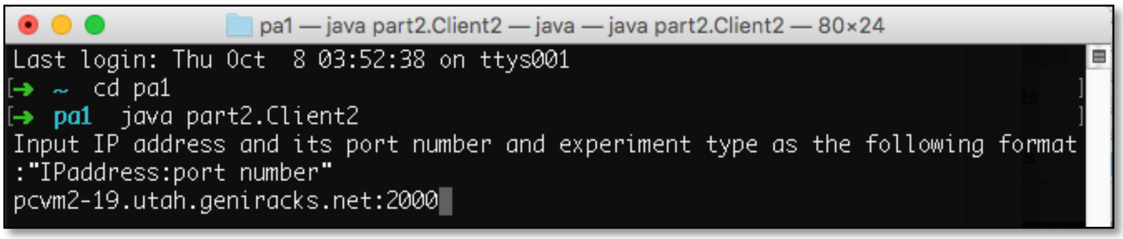
When we want to run the program, cd to the upper directory, and use:

java part2.Client2
java part2.Server2

Argument Specification:

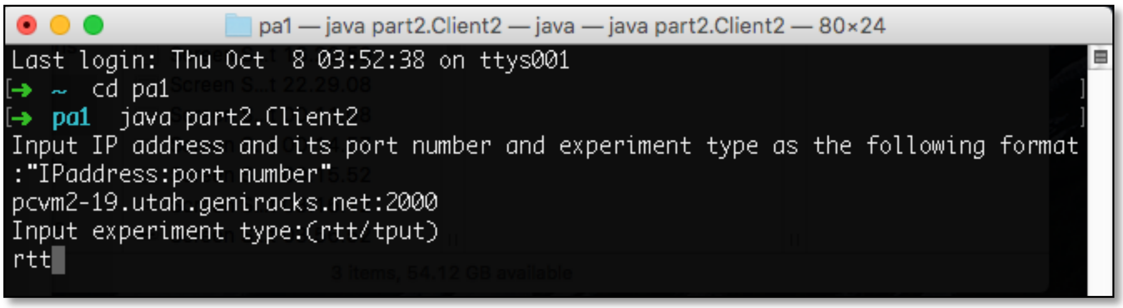
When using Client2, it required you to input several arguments to finish the experiment:

Input Server IP address: port number like: pcvm2-19.utah.geniracks.net:2000


pa1 — java part2.Client2 — java — java part2.Client2 — 80×24

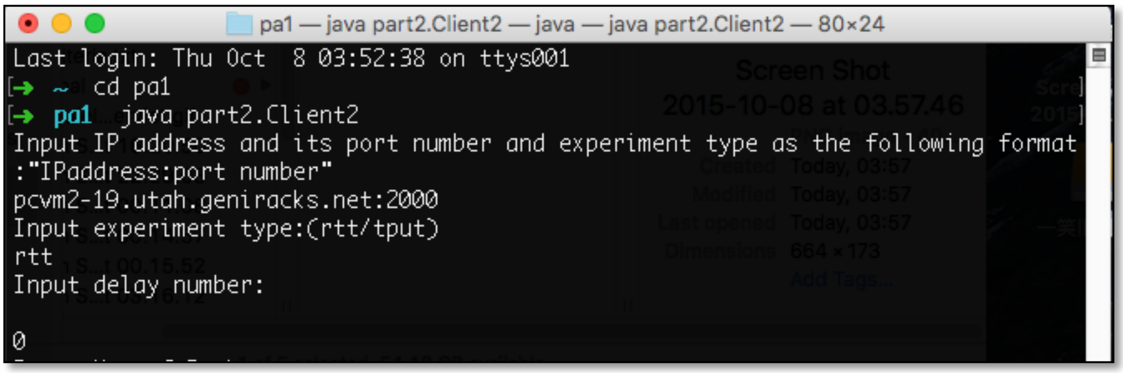
```
Last login: Thu Oct  8 03:52:38 on ttys001
[➜  ~ cd pa1
[➜  pa1 java part2.Client2
Input IP address and its port number and experiment type as the following format
:"IPaddress:port number"
pcvm2-19.utah.geniracks.net:2000
```

Input experiment type like: "rtt" or "tput" (without quotation mark and must be either rtt or tput)


pa1 — java part2.Client2 — java — java part2.Client2 — 80×24

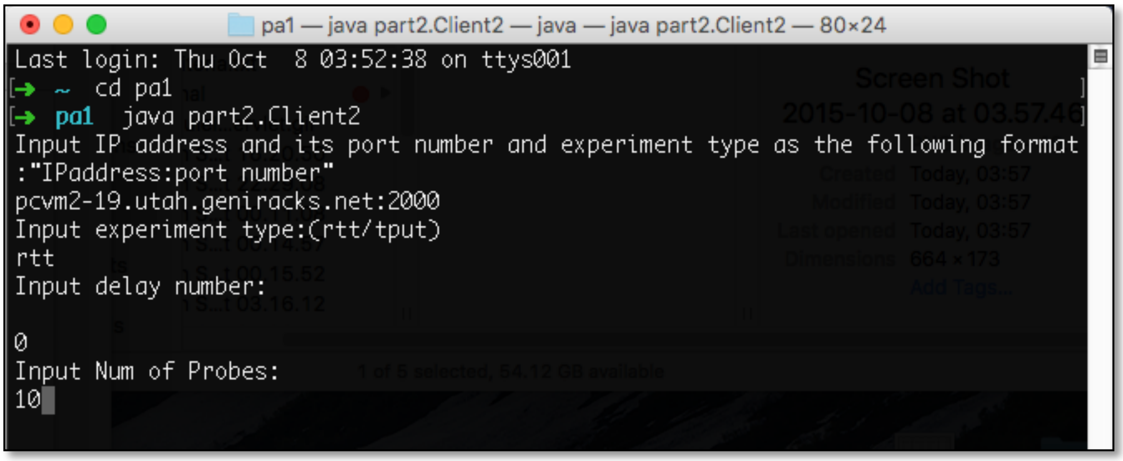
```
Last login: Thu Oct  8 03:52:38 on ttys001
[➜  ~ cd pa1
[➜  pa1 java part2.Client2
Input IP address and its port number and experiment type as the following format
:"IPaddress:port number"
pcvm2-19.utah.geniracks.net:2000
Input experiment type:(rtt/tput)
rtt
```

Input delay number like: 0


pa1 — java part2.Client2 — java — java part2.Client2 — 80×24

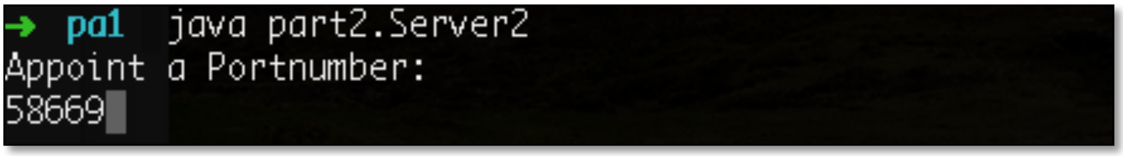
```
Last login: Thu Oct  8 03:52:38 on ttys001
[➜  ~ cd pa1
[➜  pa1  java part2.Client2
Input IP address and its port number and experiment type as the following format
:"IPaddress:port number"
pcvm2-19.utah.geniracks.net:2000
Input experiment type:(rtt/tput)
rtt
Input delay number:

0
```

Input number of probes (at least 10 times) like: 10


pa1 — java part2.Client2 — java — java part2.Client2 — 80×24

```
Last login: Thu Oct  8 03:52:38 on ttys001
[➜  ~ cd pa1
[➜  pa1  java part2.Client2
Input IP address and its port number and experiment type as the following format
:"IPaddress:port number"
pcvm2-19.utah.geniracks.net:2000
Input experiment type:(rtt/tput)
rtt
Input delay number:

0
Input Num of Probes:
10
```

In another hand, when using Server2, it required you to appoint a port number to run on, like:



```
➜  pa1  java part2.Server2
Appoint a Portnumber:
58669
```

Expected results of Client2:

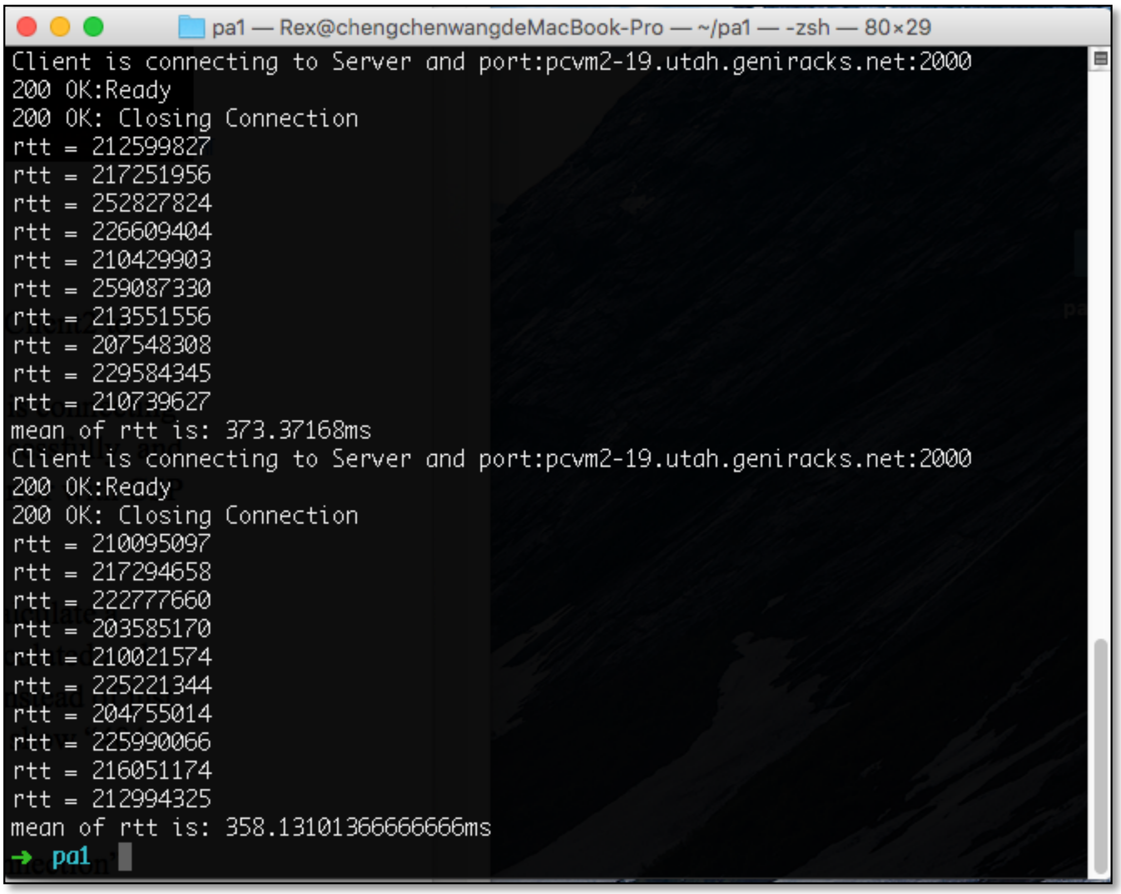
For each size of packet:

During the CSP-MP-CTP method experiment, I designed and expected Client2 to show information to tell the status of the connection:

In CSP, the Client2 is supposed to show: "200 OK: Ready" and “Client is connecting to Server and port:pcvm2-19.utah.geniracks.net:2000” if it connects successfully, and show “404 ERROR: Invalid Connection Setup Message” if there is an error with CSP message.

During the MP, the Client2 will show rtt values for every probes, and calculate a average of rtt/throughput value. (Note that this information will be calculated and showed after ctp, because we must calculate the whole round trip time instead of just mp's trip time) If there is something wrong with the mp message, it will show "404 ERROR: Invalid Measurement Message".

During the CTP, the Client2 is supposed to show "200 OK: Closing Connection" normally, and show "404 ERROR: Invalid Connection Termination Message" from server response, if there is something wrong with ctp message.



```
Client is connecting to Server and port:pcvm2-19.utah.geniracks.net:2000
200 OK:Ready
200 OK: Closing Connection
rtt = 212599827
rtt = 217251956
rtt = 252827824
rtt = 226609404
rtt = 210429903
rtt = 259087330
rtt = 213551556
rtt = 207548308
rtt = 229584345
rtt = 210739627
mean of rtt is: 373.37168ms
Client is connecting to Server and port:pcvm2-19.utah.geniracks.net:2000
200 OK:Ready
200 OK: Closing Connection
rtt = 210095097
rtt = 217294658
rtt = 222777660
rtt = 203585170
rtt = 210021574
rtt = 225221344
rtt = 204755014
rtt = 225990066
rtt = 216051174
rtt = 212994325
mean of rtt is: 358.13101366666666ms
→ pa1
```

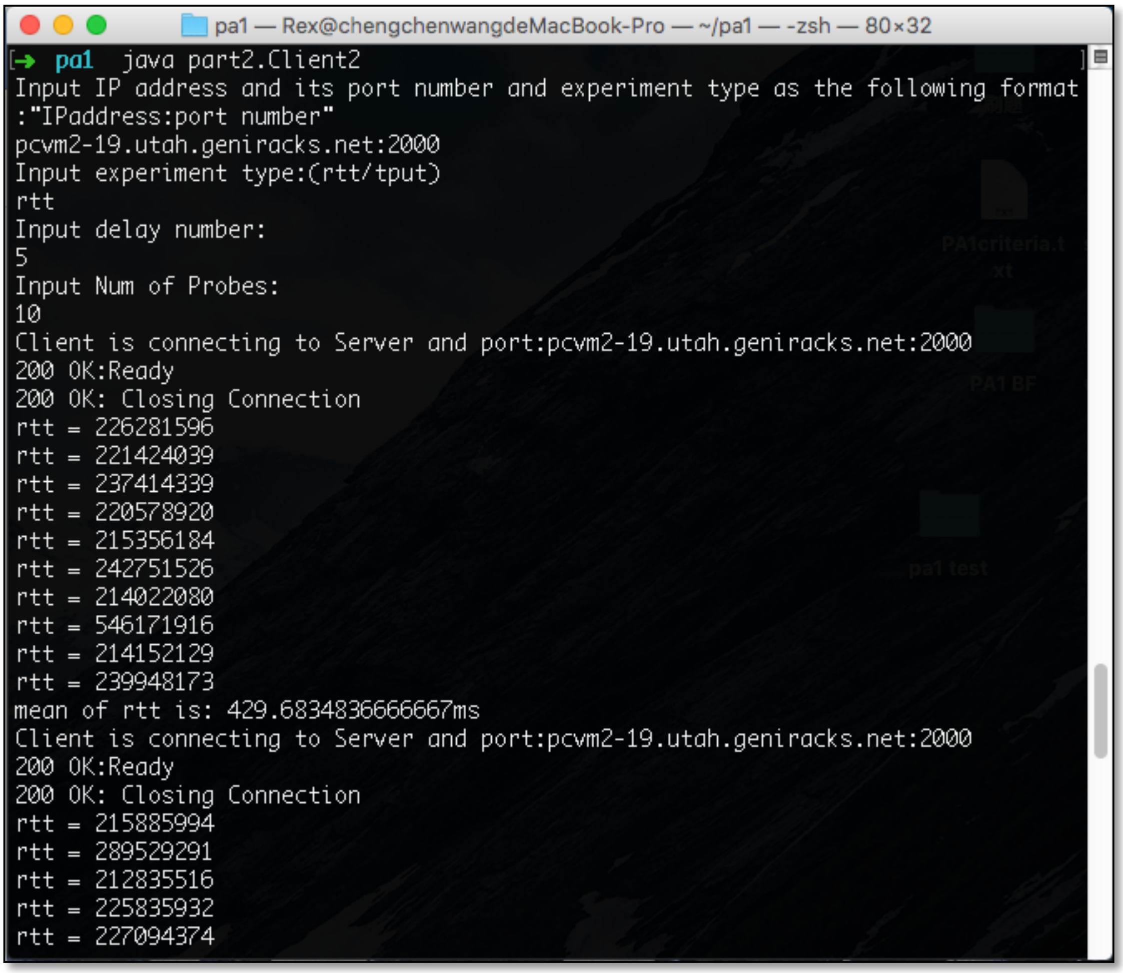
Design of program-Client2:
Client2 first define initializes some basic arguments, and then use method Scanner and Bufferedreader to scan the users' inputs, which provides an interactive interface to specify Sever IP address and port number, this experiment's type (rtt/ tput), server delay, number of probes (at least 10). Then Client2 uses these specified arguments to set up a socket connection with the server and start CSP-MP-CTP method process, using bufferedReader to read messages form server, and using printWriter with flush() to send messages to the server. During this process, whenever Client2 get a "404 ERROR" from server, it prints the message out and close the socket right away. During MP phase, it notes down the rtt values for each probe, after the CTP phase, Client2 calculate the mean rtt value or the mean throughput value, determined by what type of this experiment is.

Design of program-Server2:

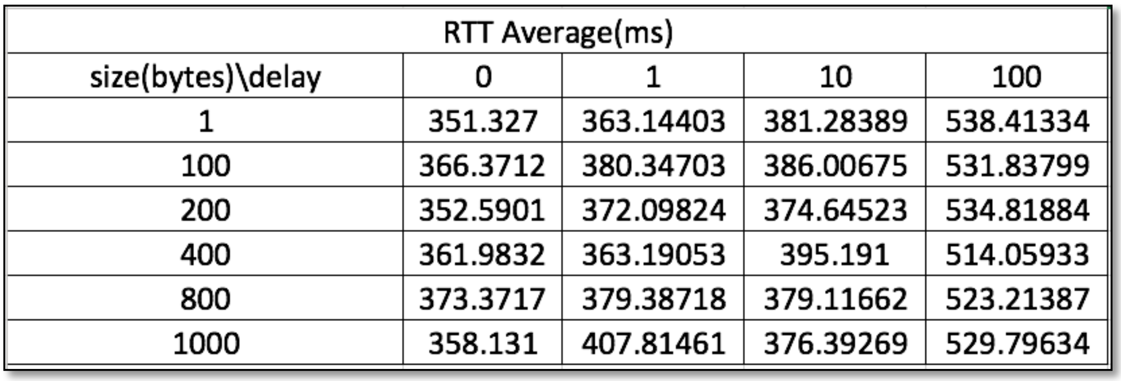
After users entering port number to run on, the Server2 establish a server socket by using method ServerSocket. And in a “while(true)” loop, which means Server2 continuously accepting connections from clients, Server2 examines CSP messages/ MP messages /CTP messages by splitting the messages by “whitespace”. It examines protocol type, number of probes and so on. When it comes to sequence number, Server2 uses a loop to self add the sequence number and examines it with the sequence number accepting from clients. During the process, each error type of message cause a socket close and Server2 prints the type of error out.

Description and results of Experiment:

For RTT: using **pcvm2-19.utah.geniracks.net:2000** as the Server, and use experiment type **rtt**, from **1byte to 100, 200, 400, 800, 1000 bytes**, for each size, the number of probes is **10**, so there will be 10 rtt values for each size. Calculate each size’s **average rtt value**, so we get 60 results and 6 average values. Varying delay number from **0 to 10 to 100**, and we can get different groups of average rtt values:

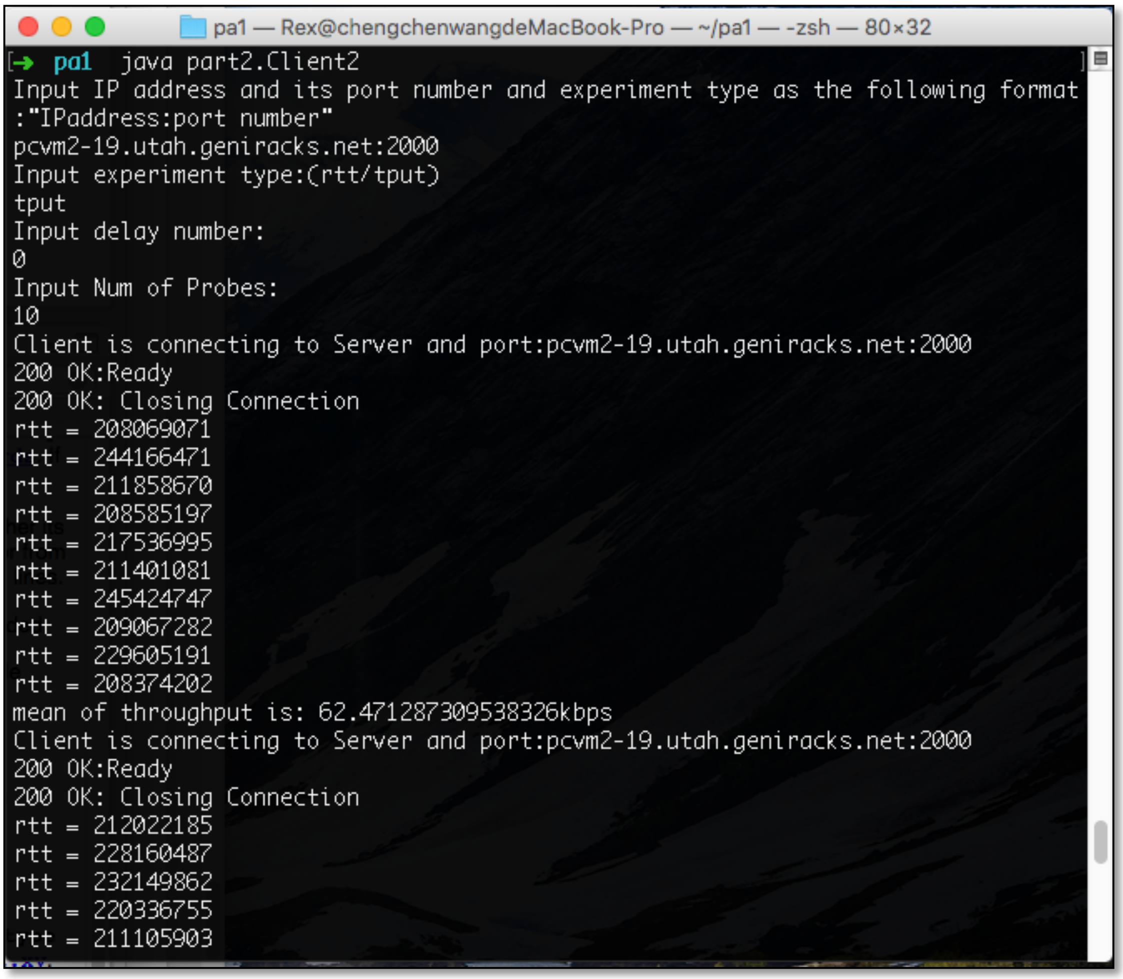


```
[➜  pa1  java part2.Client2
Input IP address and its port number and experiment type as the following format
:"IPaddress:port number"
pcvm2-19.utah.geniracks.net:2000
Input experiment type:(rtt/tput)
rtt
Input delay number:
5
Input Num of Probes:
10
Client is connecting to Server and port:pcvm2-19.utah.geniracks.net:2000
200 OK:Ready
200 OK: Closing Connection
rtt = 226281596
rtt = 221424039
rtt = 237414339
rtt = 220578920
rtt = 215356184
rtt = 242751526
rtt = 214022080
rtt = 546171916
rtt = 214152129
rtt = 239948173
mean of rtt is: 429.6834836666667ms
Client is connecting to Server and port:pcvm2-19.utah.geniracks.net:2000
200 OK:Ready
200 OK: Closing Connection
rtt = 215885994
rtt = 289529291
rtt = 212835516
rtt = 225835932
rtt = 227094374
```

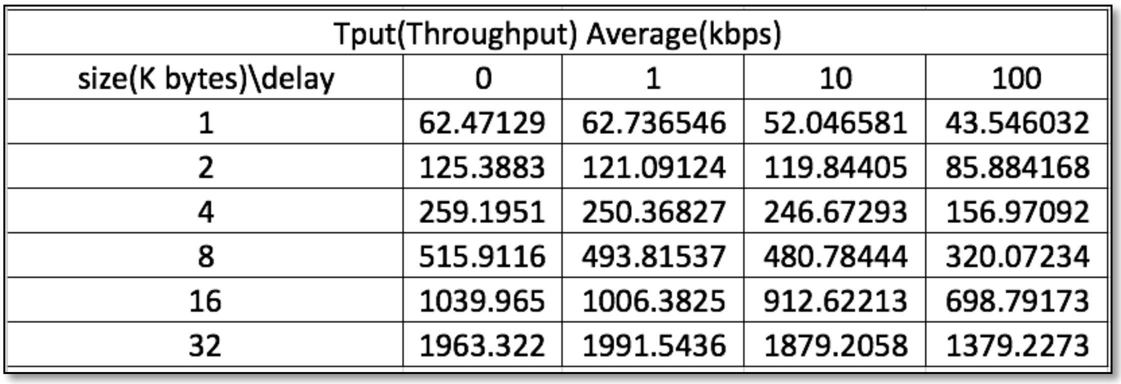


RTT Average(ms)				
size(bytes)\delay	0	1	10	100
1	351.327	363.14403	381.28389	538.41334
100	366.3712	380.34703	386.00675	531.83799
200	352.5901	372.09824	374.64523	534.81884
400	361.9832	363.19053	395.191	514.05933
800	373.3717	379.38718	379.11662	523.21387
1000	358.131	407.81461	376.39269	529.79634

For Tput: using **pcvm2-19.utah.geniracks.net:2000** as the Server, and use experiment type **tput**, from **1k byte to 2k, 4k, 8K, 16K 32K bytes**, for each size, the number of probes is **10**, so there will be 10 tput values for each size. Calculate each size's average tput value (using size/rtt, and then convert the unit of value to kbps), so we get 60 results and 6 average values. Varying delay number from 0 to 10 to 100, and we can get different groups of average tput values:



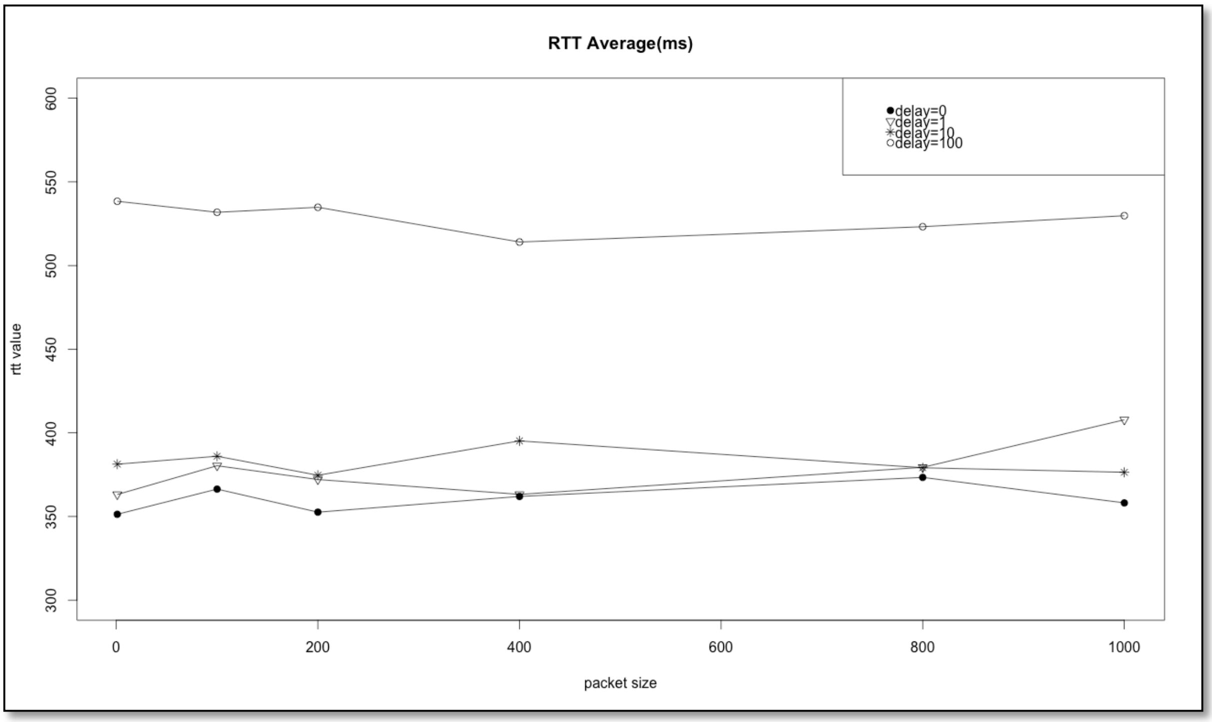
```
→ pa1  java part2.Client2
Input IP address and its port number and experiment type as the following format
:"IPaddress:port number"
pcvm2-19.utah.geniracks.net:2000
Input experiment type:(rtt/tput)
tput
Input delay number:
0
Input Num of Probes:
10
Client is connecting to Server and port:pcvm2-19.utah.geniracks.net:2000
200 OK:Ready
200 OK: Closing Connection
rtt = 208069071
rtt = 244166471
rtt = 211858670
rtt = 208585197
rtt = 217536995
rtt = 211401081
rtt = 245424747
rtt = 209067282
rtt = 229605191
rtt = 208374202
mean of throughput is: 62.471287309538326kbps
Client is connecting to Server and port:pcvm2-19.utah.geniracks.net:2000
200 OK:Ready
200 OK: Closing Connection
rtt = 212022185
rtt = 228160487
rtt = 232149862
rtt = 220336755
rtt = 211105903
```



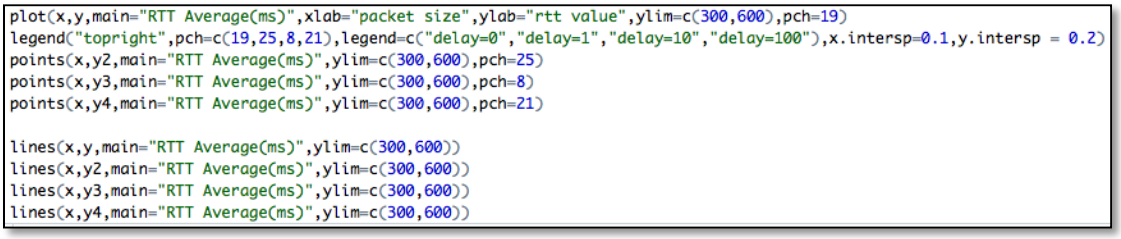
Tput(Throughput) Average(kbps)				
size(K bytes)\delay	0	1	10	100
1	62.47129	62.736546	52.046581	43.546032
2	125.3883	121.09124	119.84405	85.884168
4	259.1951	250.36827	246.67293	156.97092
8	515.9116	493.81537	480.78444	320.07234
16	1039.965	1006.3825	912.62213	698.79173
32	1963.322	1991.5436	1879.2058	1379.2273

Graphs and analysis:

RTT:



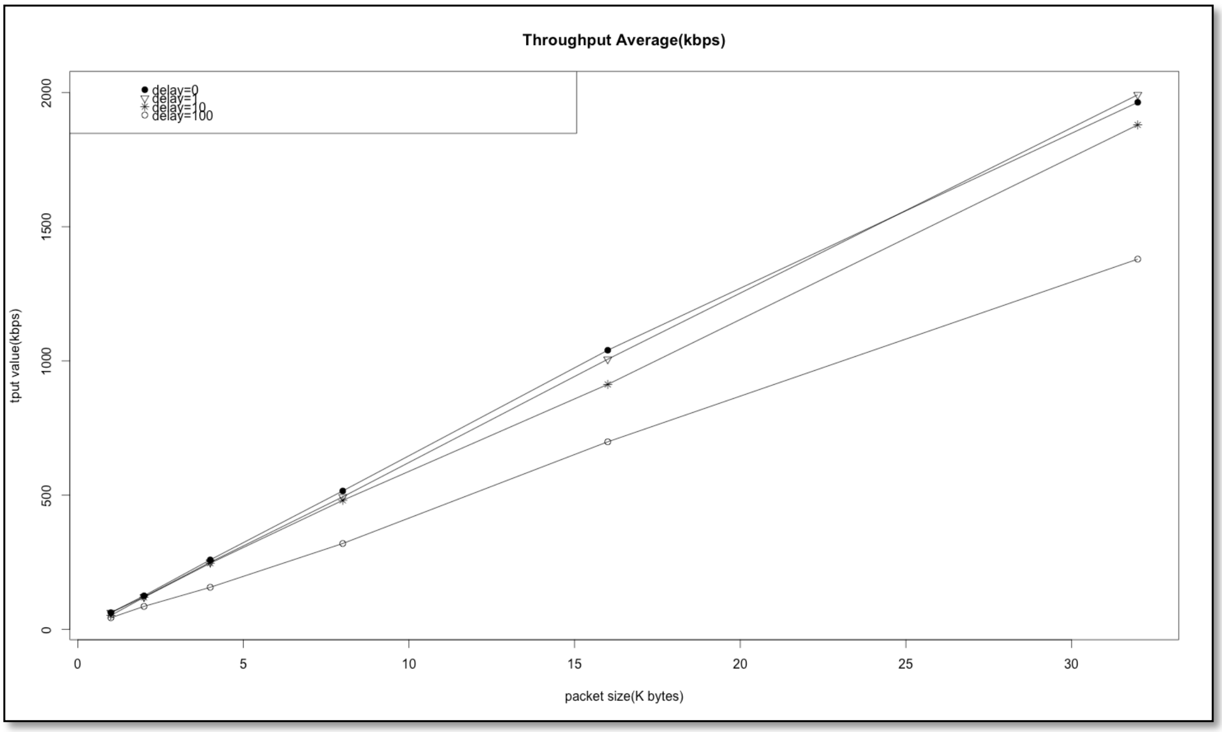
Plotted by R language:



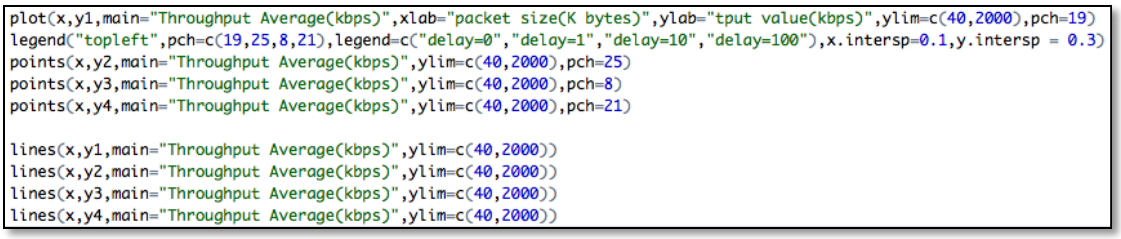
```
plot(x,y,main="RTT Average(ms)",xlab="packet size",ylab="rtt value",ylim=c(300,600),pch=19)
legend("topright",pch=c(19,25,8,21),legend=c("delay=0","delay=1","delay=10","delay=100"),x.intersp=0.1,y.intersp = 0.2)
points(x,y2,main="RTT Average(ms)",ylim=c(300,600),pch=25)
points(x,y3,main="RTT Average(ms)",ylim=c(300,600),pch=8)
points(x,y4,main="RTT Average(ms)",ylim=c(300,600),pch=21)

lines(x,y,main="RTT Average(ms)",ylim=c(300,600))
lines(x,y2,main="RTT Average(ms)",ylim=c(300,600))
lines(x,y3,main="RTT Average(ms)",ylim=c(300,600))
lines(x,y4,main="RTT Average(ms)",ylim=c(300,600))
```

According to the image above, when delay=0, we find that different packet sizes have nearly no effects on mean rtt values, when delay=1/10/100, we also find that different packet sizes have nearly no effects on mean rtt values. But as the delay increasing from 1 to 100, the mean rtt values are increasingly as well. So we know that even though increasing the server delay merely increases the processing time at the server, it causes the feedback delay, which observed by us, has an effect similar to increasing the path's propagation.

Tput:



Plotted by R language:



```
plot(x,y1,main="Throughput Average(kbps)",xlab="packet size(K bytes)",ylab="tput value(kbps)",ylim=c(40,2000),pch=19)
legend("topleft",pch=c(19,25,8,21),legend=c("delay=0","delay=1","delay=10","delay=100"),x.intersp=0.1,y.intersp = 0.3)
points(x,y2,main="Throughput Average(kbps)",ylim=c(40,2000),pch=25)
points(x,y3,main="Throughput Average(kbps)",ylim=c(40,2000),pch=8)
points(x,y4,main="Throughput Average(kbps)",ylim=c(40,2000),pch=21)

lines(x,y1,main="Throughput Average(kbps)",ylim=c(40,2000))
lines(x,y2,main="Throughput Average(kbps)",ylim=c(40,2000))
lines(x,y3,main="Throughput Average(kbps)",ylim=c(40,2000))
lines(x,y4,main="Throughput Average(kbps)",ylim=c(40,2000))
```

According to the image above, when delay-0, as the packet size is increasing, we find that the mean throughput values are also increasing. It's not a liner function, but the increasing speed is really high and nearly equals to liner. So we can know that when we increase a packet size, the path's throughput will be correspondingly increase. Besides, when delay increases from 0 to 100, the increase speed of throughput somewhat decreases, as the image shows, but the overall trend does not change.