1. Purpose

This assignment is intended for two purposes: (1) to exercise use of various socket-related system calls and (2) to evaluate dominant overheads of point-to-point communication over 1Gbps networks.

1. Algorithm and Explanation

Client.cpp and Server.cpp that establish a TCP connection between a pair of client and server, repeat sending a set of data buffers, (each with the same size) in three different scenarios from the client to the server, and send back an acknowledgment from the server to the client.

The client program is responsible for connecting to the server and depending on the input from the command line; send to the server data and receive back data and time them.

The server is responsible for creating a socket with a valid port number so that the client can connect to it. It is responsible for reading the data that the client send and return back how many read operations it performed.

1. Compile and run commands

Run Server first

$ g++ -pthread -std=c++11 Server.cpp

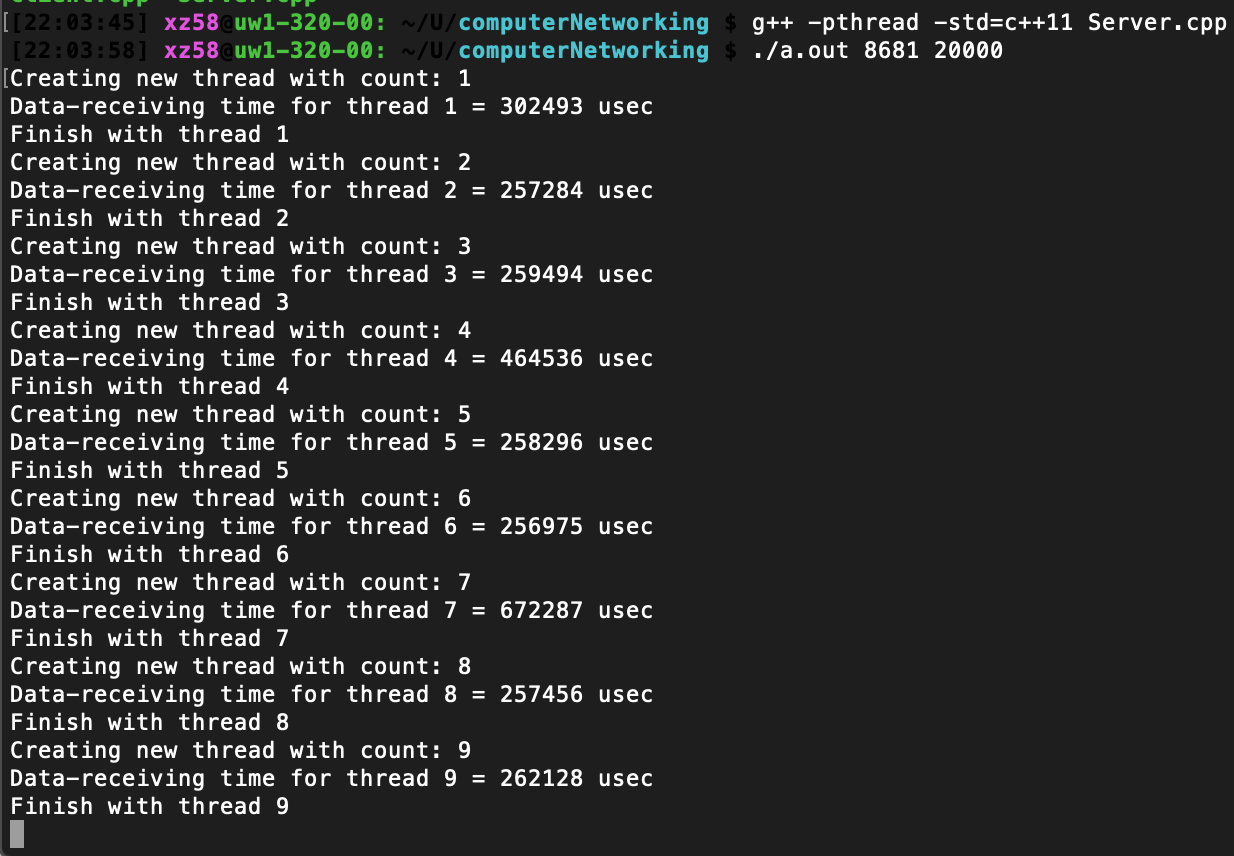
$ ./a.out [port number] [repetition]

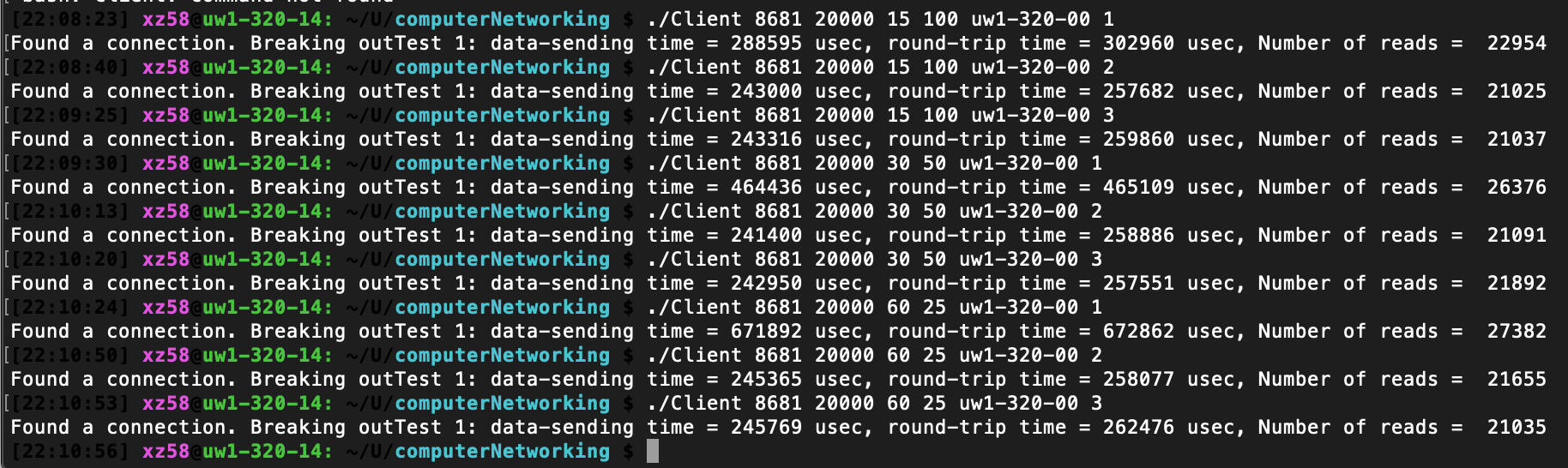
Then run Client

$ g++ -std=c++11 Client.cpp

$ ./a.out [port number] [repetition] [nbufs] [buffsize] [serverIp] [type]

1. Sample output





1. Performance Evaluation

For transfer type using different buffer and type, there are no significant different. This may be due to the ability to process the information of the machine.

For the number of read, there are differences based on how the data is write and how it is stored based on the explanation below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. of Buffer | Buffer Size | Type | Avg. Sending Time | Avg. Round-trip Time | Avg. no. of Reads |
| 15 | 100 | 1 | 288595 | 302960 | 22954 |
| 15 | 100 | 2 | 243000 | 257682 | 21025 |
| 15 | 100 | 3 | 243316 | 259860 | 21037 |
| 30 | 50 | 1 | 464436 | 465109 | 26376 |
| 30 | 50 | 2 | 241400 | 258886 | 21091 |
| 30 | 50 | 3 | 242950 | 257551 | 21892 |
| 60 | 25 | 1 | 671892 | 672862 | 27382 |
| 60 | 25 | 2 | 245365 | 258077 | 21655 |
| 60 | 25 | 3 | 245769 | 262476 | 21035 |

1. Discussions
   1. 1Gbps actual throughputs

Default Gigabit Ethernet has an impressive number of frames (about 81000 per second) possible and a high theoretical throughput for actual data (about 118 MB/s).

* 1. Multi-writes

Time: 15\*100 is the fastest, then 30\*50, 60\*25 is the slowest

Reads: 15\*100<30\*50<60\*25. However, the reads are really close.

* 1. Writev

Time: 15\*100, 30\*50, and 60\*25 all have similar sending and round-trip time for writev.

Reads: 15\*100, 30\*50, and 60\*25 tied again on the number of reads

* 1. single-write

Time: No significant difference between 15\*100, 30\*50, and 60\*25

Reads: No significant difference between 15\*100, 30\*50, and 60\*25

* 1. asynchrnoous reads vs blocking reads

The client sends a request to the server and waits for its response. Then it sends the next request and again waits for its response, and so on. Having to wait for the response before sending the next request is often a pretty huge bottleneck.

Then asynchronous I/O would mean "not synchronous I/O". That is, you issue an I/O command and it will be completed at some later time. It follows that to do asynchronous I/O you need an interface that allows you to issue I/O commands without blocking and to notify you when the I/O operation is completed.

The reason why we prefer asynchronous I/O over blocking reads is that asynchronous I/O boosts up the efficiency, throughputs, and lowers the average round-trip time.