

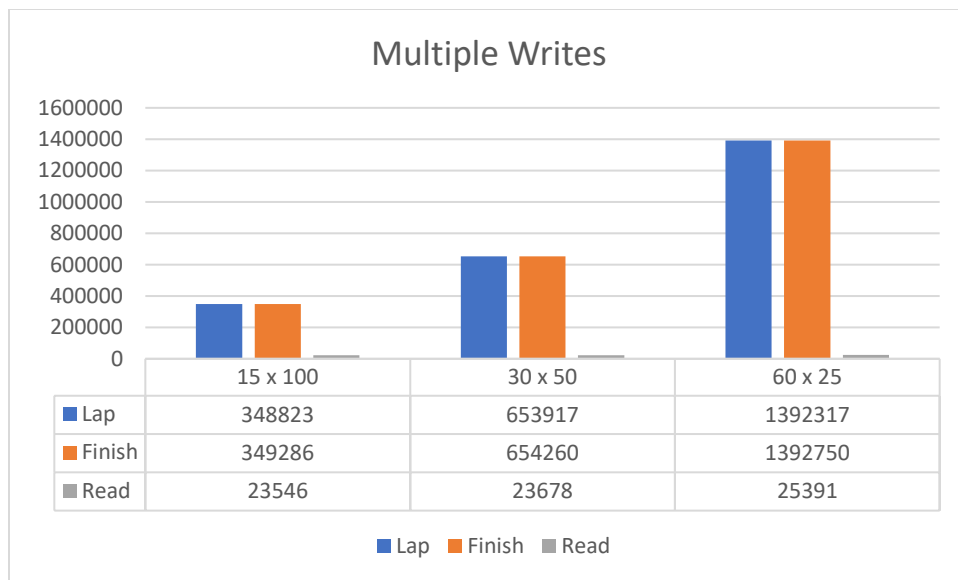
CSS432 – HW1 – Jun Zhen

Performance Evaluation

Type	Lap	Finish	Read	N-Buffer	Buffer Size
1	348823	349286	23546	15	100
1	653917	654260	23678	30	50
1	1392317	1392750	25391	60	25
2	134398	134669	20856	15	100
2	133904	134477	20766	30	50
2	201823	202237	21091	60	25
3	66175	67778	20078	15	100
3	62859	67167	20088	30	50
3	54090	56797	20092	60	25

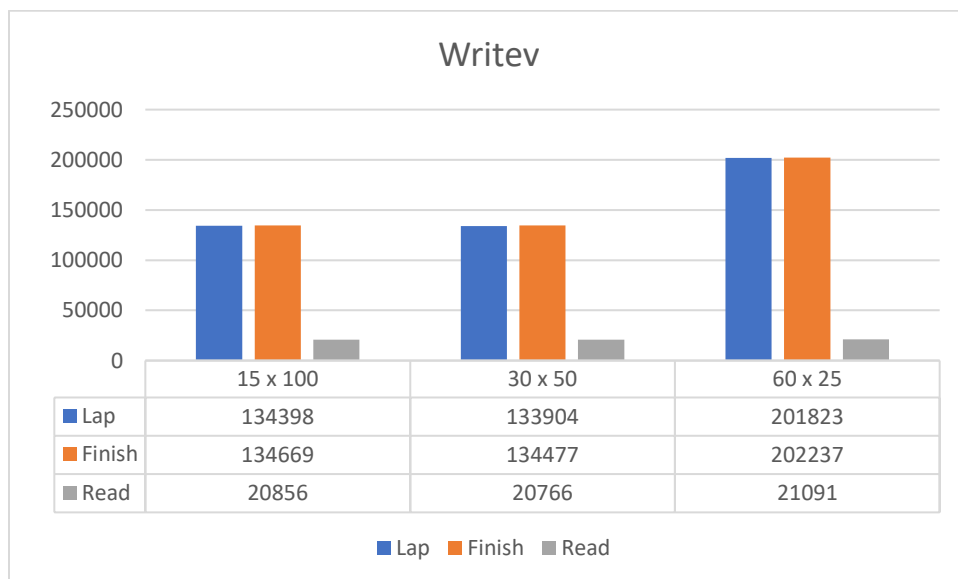
The fastest method to write from client to server is the type 3 (single write). With a packet sending by interval, there won't be a bunch of data holding up the queue, thus decreasing the travel time. Unlike type 3, type 1 (multiple writes) doesn't wait and send multiple of bytes all at once. This will likely cause the queue to clog up. Which is the main reason why type 3 is the slowest method compared to the other methods. Type 2 is interesting because it packaged up all it's bytes into one and send a large amount data at once. This also not the quickest but it is not the slowest either. The advantage is that there is less problem from queue delay, but at the same time the bytes become too large to fit in one packet, thus it must be divided into parts. In conclusion, even though each type sends the same about of bytes, the fact that type 3 is not congesting the queue, allow its bytes to go one by one result in the best performance. While type 2 tries to alleviate the stress on the queue by sending one large packet, but the size of packet will cause a slowdown either way. The least affective type being type 1. The main reason is due to the method of it sending its bytes. By clogging up the queue with multiple writes into the socket.

Type 1



The speed of type 1 drastically become slower due to the fact it has to transmit smaller bytes but more frequently.

Type 2



The speed stated the same for the first two, but it got a bit slower towards the end with 60 x 25. Might due to same reasoning as type 1 with it having transmitted more frequently.

Type 3



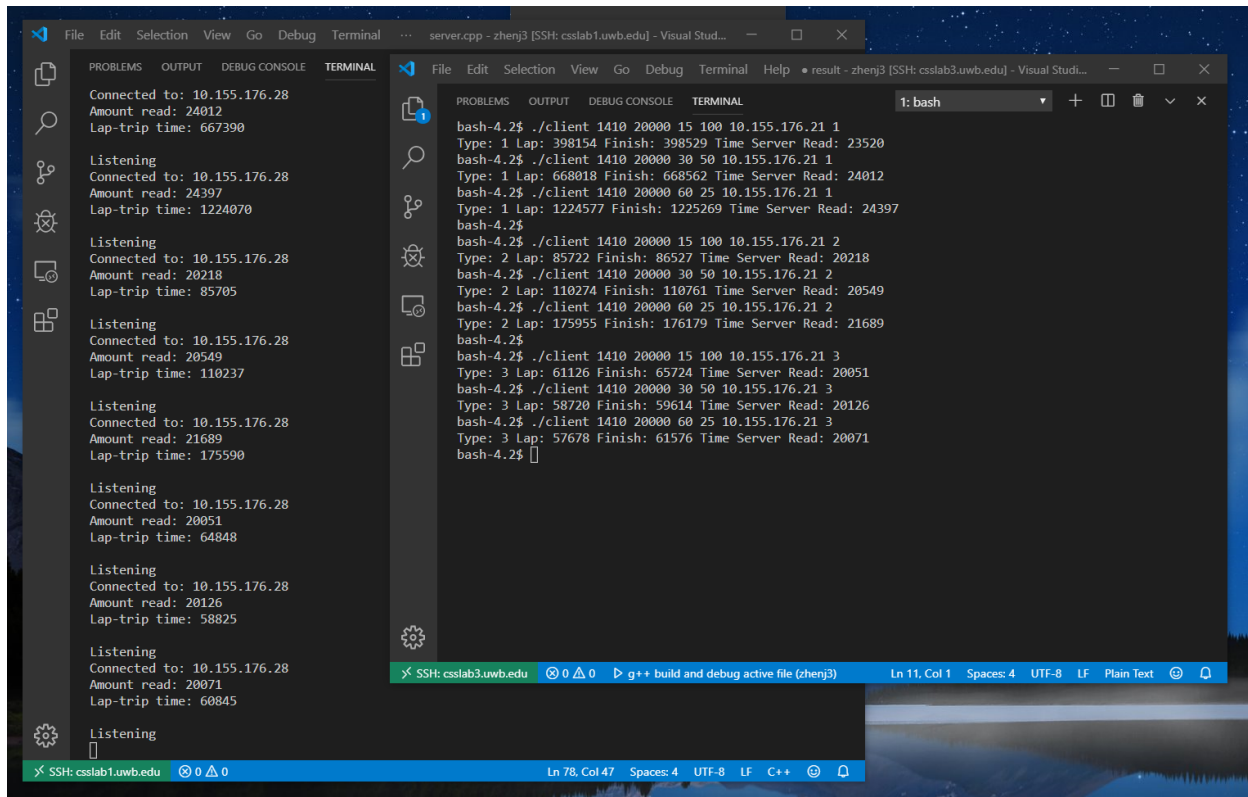
The speed of type 3 actually became fast, this due to the size of the packet being smaller in bytes.

Conclusion

The best method to transmit data with our program is to use type 3. Type 3 showed that transmitting between interval allow the data to freely traverse the network without causing congestion. This even better for network that are slow. Slow network will have longer nodal delay, having packets not transmitted all together will give the network time to process all its data before more data arrives. It gives the network a bit of breathing room. In a case where type 2 or type 1 is better if the bandwidth of the network is large enough to support big data, this will actually allow type 2 send one big packet without hindering its speed.

For the server side, type 3 is still the best method to transmit data. The reason is because it allows the server to finish all of its processes before more connections arrives. Another way for server to get around this is to have multi-threading. Having separate threads handling each connection will vastly increase the speed of the server due to thread is able to run concurrently with each other. If main was the only process that will become the same issue as the nodal queue. Main will require to process each connection one by one, if hundreds of connections connects at one time, this will create a bottleneck like a network queue.

Output server (left), client(right)



The image shows two side-by-side Visual Studio Code terminal windows. The left window, titled 'server.cpp - zhenj3 [SSH: csslab1.uwb.edu] - Visual Stud...', displays the output of a server program. It shows a series of 'Listening' messages followed by 'Connected to: 10.155.176.28', 'Amount read: [value]', and 'Lap-trip time: [value]'. The right window, titled 'result - zhenj3 [SSH: csslab3.uwb.edu] - Visual Studi...', shows the output of a client program. It displays a series of commands and responses, including 'bash-4.2\$./client 1410 20000 15 100 10.155.176.21 1' and 'Type: 1 Lap: 398154 Finish: 398529 Time Server Read: 23520'. The status bars at the bottom of each window indicate the current file, line, column, and encoding.

```
Connected to: 10.155.176.28
Amount read: 24012
Lap-trip time: 667390

Listening
Connected to: 10.155.176.28
Amount read: 24397
Lap-trip time: 1224070

Listening
Connected to: 10.155.176.28
Amount read: 20218
Lap-trip time: 85705

Listening
Connected to: 10.155.176.28
Amount read: 20549
Lap-trip time: 110237

Listening
Connected to: 10.155.176.28
Amount read: 21689
Lap-trip time: 175590

Listening
Connected to: 10.155.176.28
Amount read: 20051
Lap-trip time: 64848

Listening
Connected to: 10.155.176.28
Amount read: 20126
Lap-trip time: 58825

Listening
Connected to: 10.155.176.28
Amount read: 20071
Lap-trip time: 60845

Listening
[]
```

```
bash-4.2$ ./client 1410 20000 15 100 10.155.176.21 1
Type: 1 Lap: 398154 Finish: 398529 Time Server Read: 23520
bash-4.2$ ./client 1410 20000 30 50 10.155.176.21 1
Type: 1 Lap: 668018 Finish: 668562 Time Server Read: 24012
bash-4.2$ ./client 1410 20000 60 25 10.155.176.21 1
Type: 1 Lap: 1224577 Finish: 1225269 Time Server Read: 24397
bash-4.2$
bash-4.2$ ./client 1410 20000 15 100 10.155.176.21 2
Type: 2 Lap: 85722 Finish: 86527 Time Server Read: 20218
bash-4.2$ ./client 1410 20000 30 50 10.155.176.21 2
Type: 2 Lap: 110274 Finish: 110761 Time Server Read: 20549
bash-4.2$ ./client 1410 20000 60 25 10.155.176.21 2
Type: 2 Lap: 175955 Finish: 176179 Time Server Read: 21689
bash-4.2$
bash-4.2$ ./client 1410 20000 15 100 10.155.176.21 3
Type: 3 Lap: 61126 Finish: 65724 Time Server Read: 20051
bash-4.2$ ./client 1410 20000 30 50 10.155.176.21 3
Type: 3 Lap: 58720 Finish: 59614 Time Server Read: 20126
bash-4.2$ ./client 1410 20000 60 25 10.155.176.21 3
Type: 3 Lap: 57678 Finish: 61576 Time Server Read: 20071
bash-4.2$ []
```