8005 Asn2 - Report

There is never too much performance

Isaac Morneau; A009584050
Aing Ragunathan; A00765949

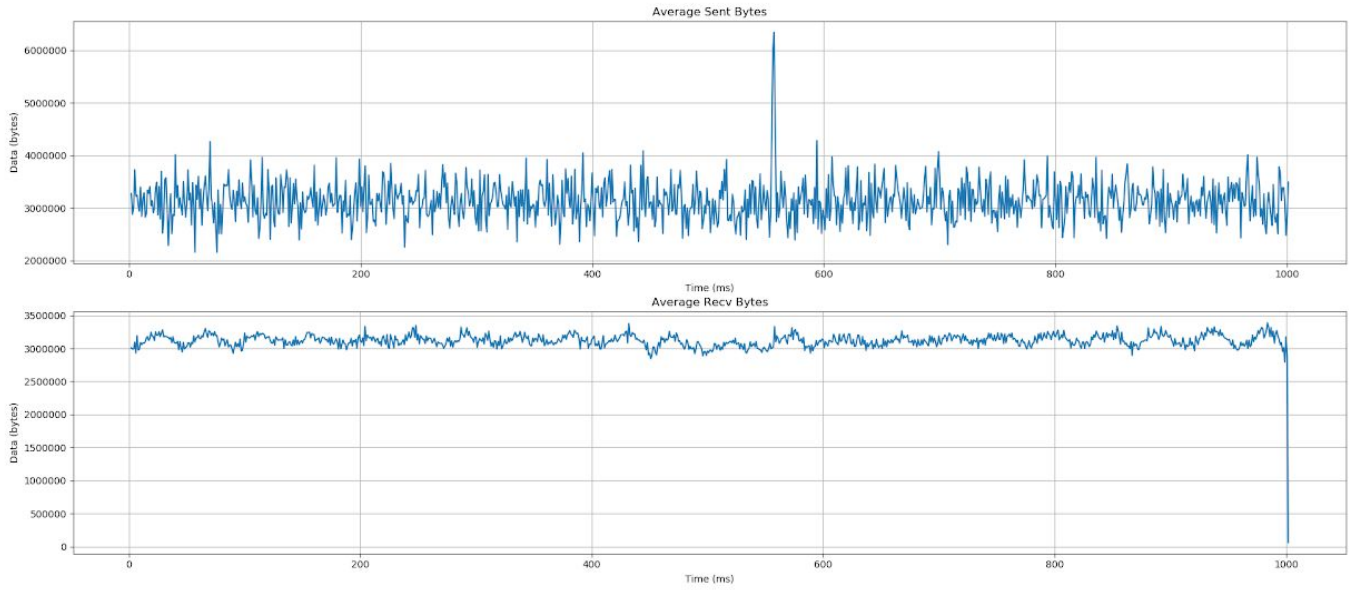
Server Type Tests

The following tests were of 30 seconds of sending after all connections were established with the exception of the 10,000 connection epoll test which was increased to 120 seconds to better show the performance.
All of the tests were performed against a high performance epoll client to ensure that differences seen were solely the result of the server itself instead of client variances.

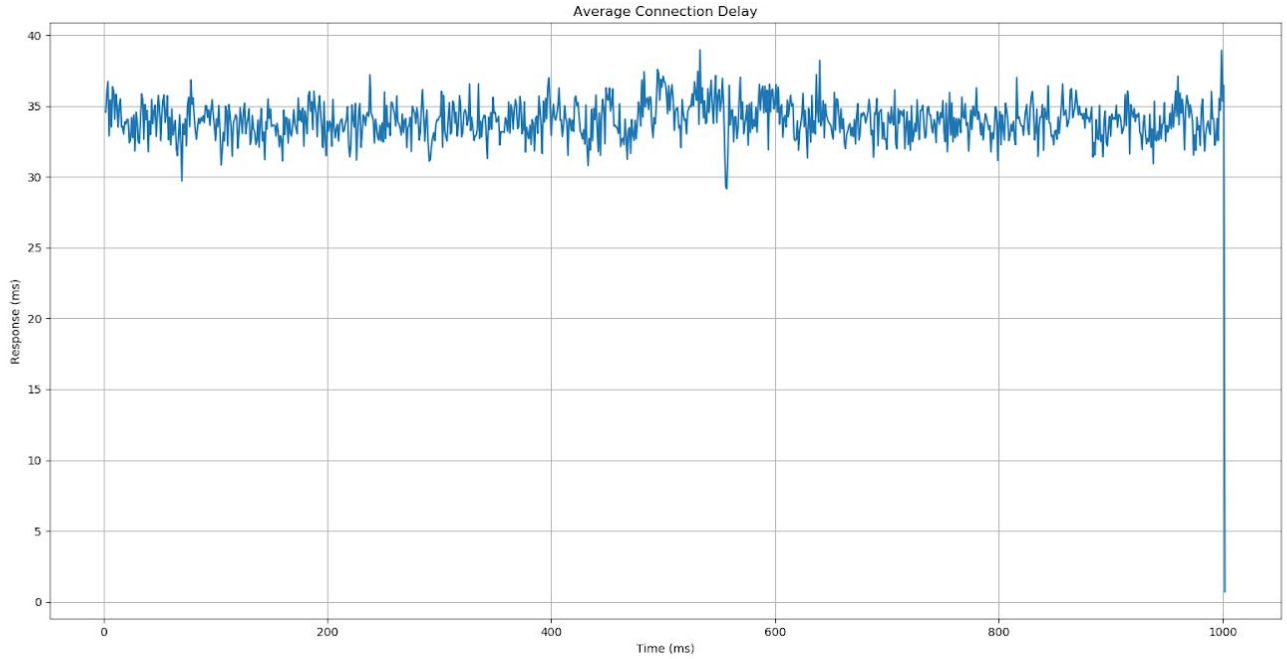
Traditional Multithreaded Server

The traditional multithreaded server was the simplest server out of the three. It was also outperformed by the other two servers in the number of clients it was able to accept while using 100% of the testing machines CPU resources. However it has proved to still be a viable for a use case such as a echo server when the number of clients is less than a few 1000.

100 Clients



The traditional multithreaded server performs quite well while it has a low number of connected clients since a modern CPU is able to manage a thread for each connection. The number of bytes sent in this test is very close to that of poll and epoll while the bytes received is the most consistent and keeps up with the alternative types of servers.

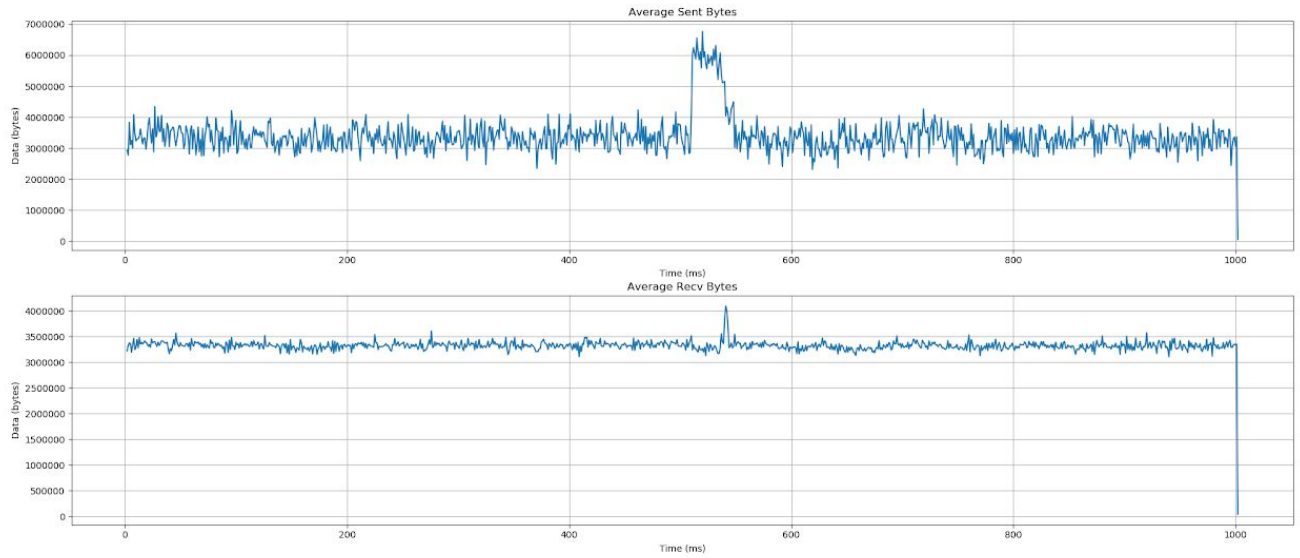


The traditional multithreaded server surprisingly performed the best out of the three while the number of clients was limited to 100. The results for this test show that it outperformed the poll and epoll server significantly in the delay time between echos. The lack of needed complexity in the traditional server resulted in the server functioning as a very minimal echo in a spin lock. This is what caused the delays to be so small between the echos over all. The other impacting design choice of note is that as all the other implementations used non blocking sockets traditional used entirely blocking. This means that it was able to function similarly to an event driven model with the kernel waking up threads that could read or send data.

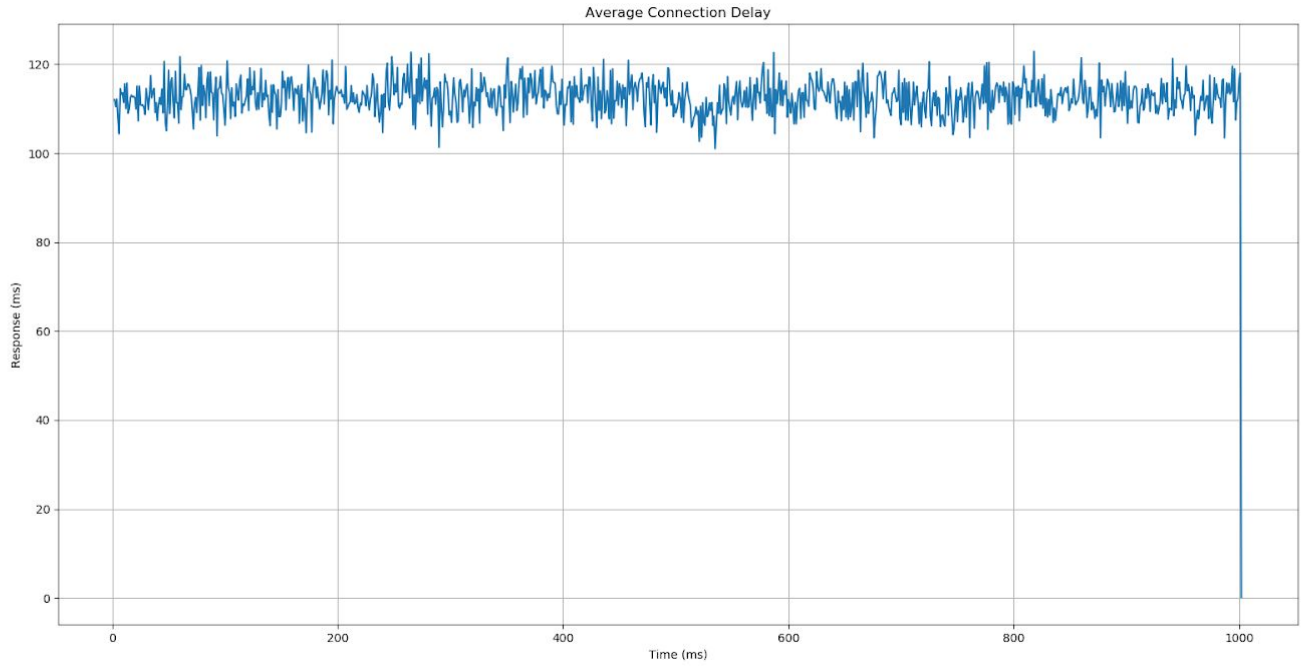
```
100% [2965377/2980277]
finished parsing
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 1
==>totals<==
connections: 100
sent(bytes): 3132256160
recv(bytes): 3120875372
```

```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 2
==>averages<==
delay(ms): 34
packets sent: 29803
data sent(bytes): 31322562
data recv(bytes): 31208754
```

1000 Clients



The traditional multithreaded server is also able to handle 1000 clients as well without any issues. It is a still able to send a about the same amount of data as the poll and epoll servers. Scheduling differences in the threads explain the brief increase in sending data it was able to maintain a consistent speed throughout the test.



There is a significant rise in delay between 100 and 1000 clients as the multithreaded server rises from 34ms to 295ms. This increase of about 10x in delay is expected since it correlates with the number of connected clients.

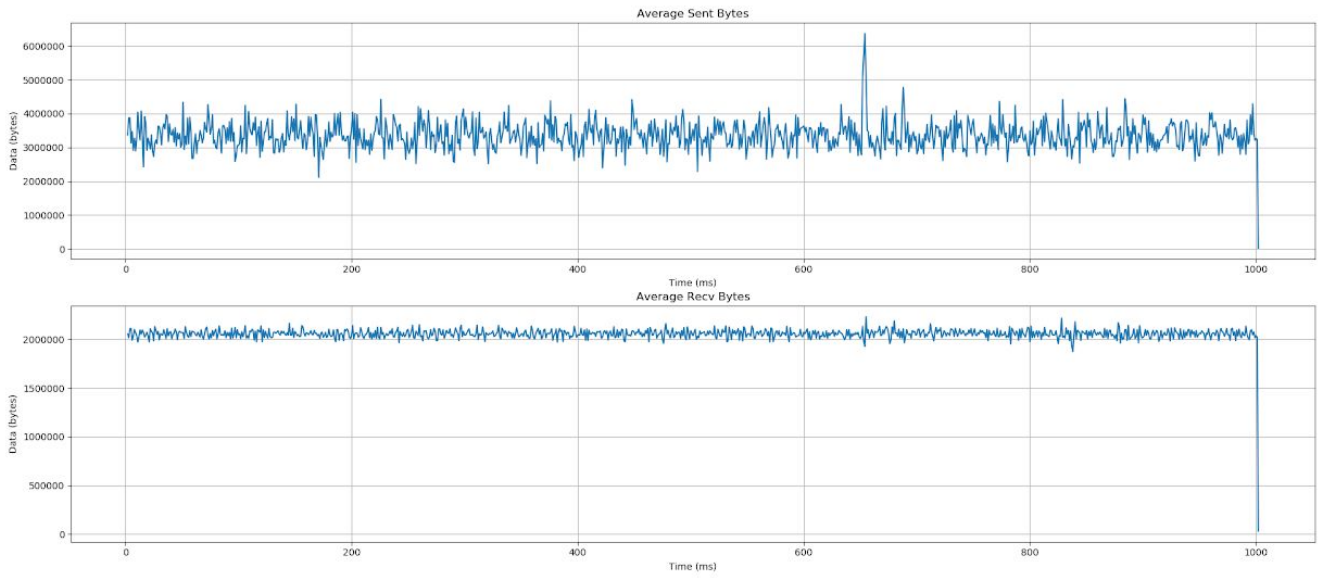
```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 1
==>totals<==
connections: 1000
sent(bytes): 3392406880
recv(bytes): 3322536924
```

```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 2
==>averages<==
delay(ms): 295
packets sent: 3410
data sent(bytes): 3392407
data recv(bytes): 3322537
```

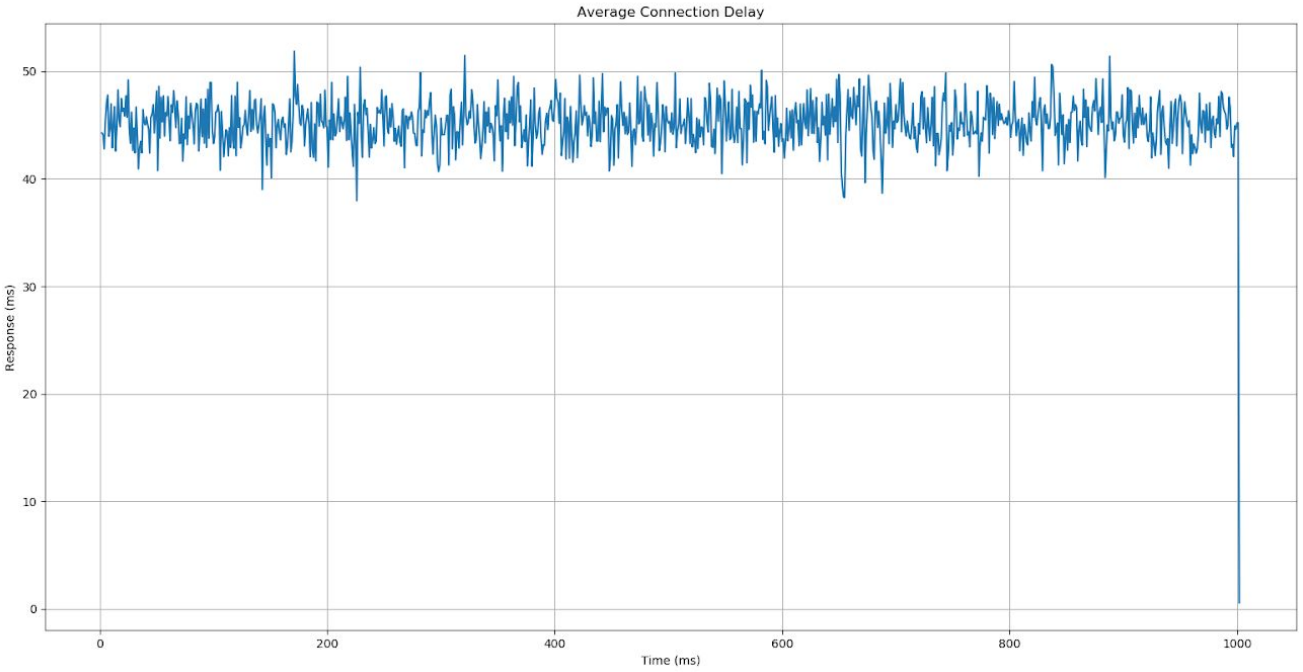
Poll Server

The poll Level Triggered server came in between the traditional multithreaded server and the epoll server in overall performance. This is because it has the added complexity of managing its clients which allows it handle more clients but with less focus on each.

100 Clients



Overall poll easily handles 100 clients without any issues and all variance is quite minimal. With the capped sending and receiving windows There are not enough clients to maximize the network usage so all the variance is due to the scheduling differences in the threads.

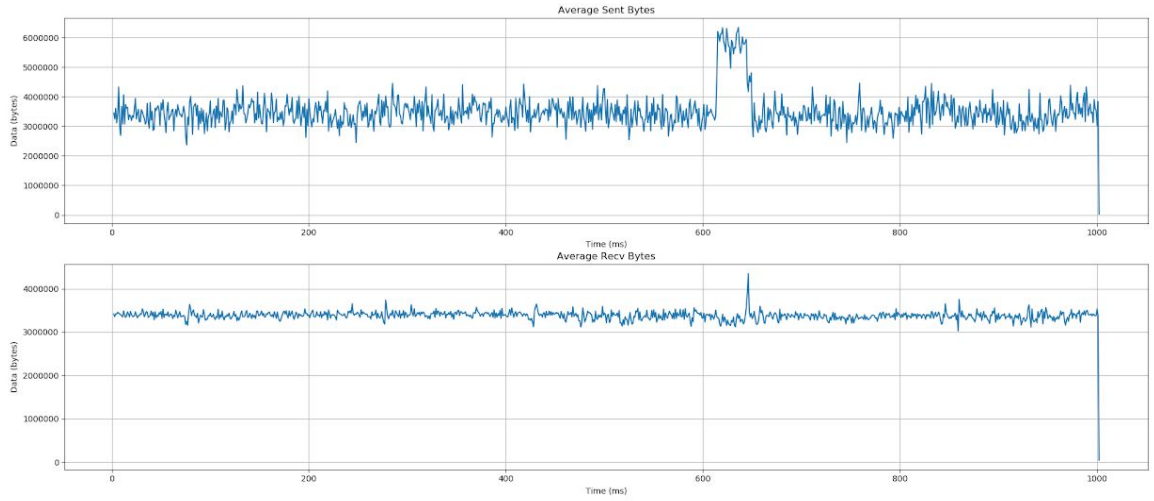


The delay hovers around 45ms never rising above 50ms and does not have any significant changes throughout the capture. This is expected as the overall system usage is very low for so few connections.

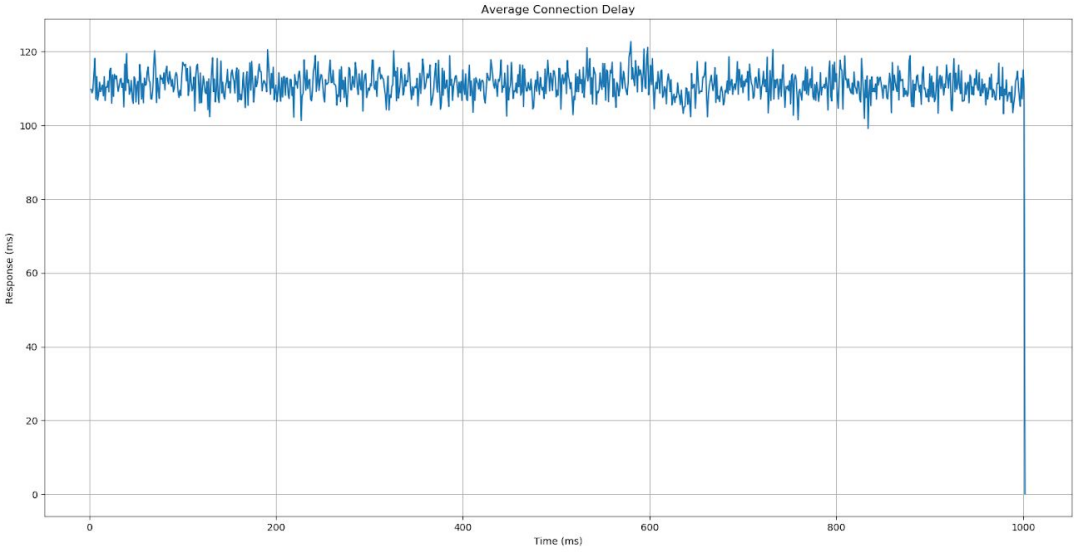
```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 1
==>totals<==
connections: 100
sent(bytes): 3363628240
recv(bytes): 2057544608
```

```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 2
==>averages<==
delay(ms): 47
packets sent: 22792
data sent(bytes): 33636282
data recv(bytes): 20575446
```

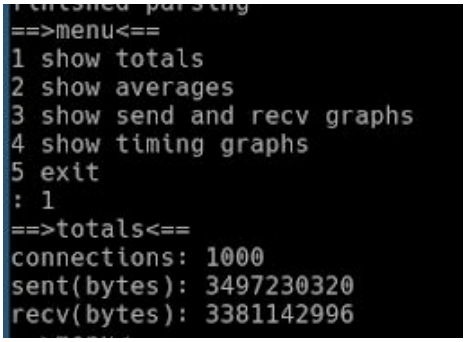
1000 Clients

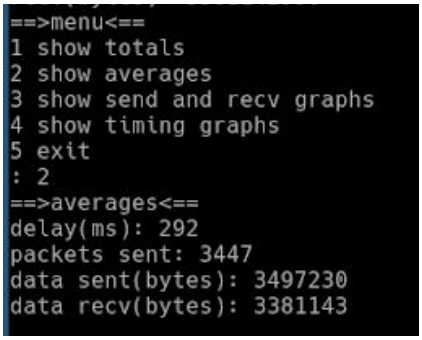


Overall poll easily handles 1000 clients as well without any issues and all variance is quite minimal. Scheduling differences in the threads explains the brief increase in sending data but overall both sending and receiving nearly have maximum network card usage.



The delay is the most noticable difference from the 100 clients as it has grown to about 100 - 120ms. The delay rise is less than linear which makes it on par with both the traditional multithreaded and epoll servers.

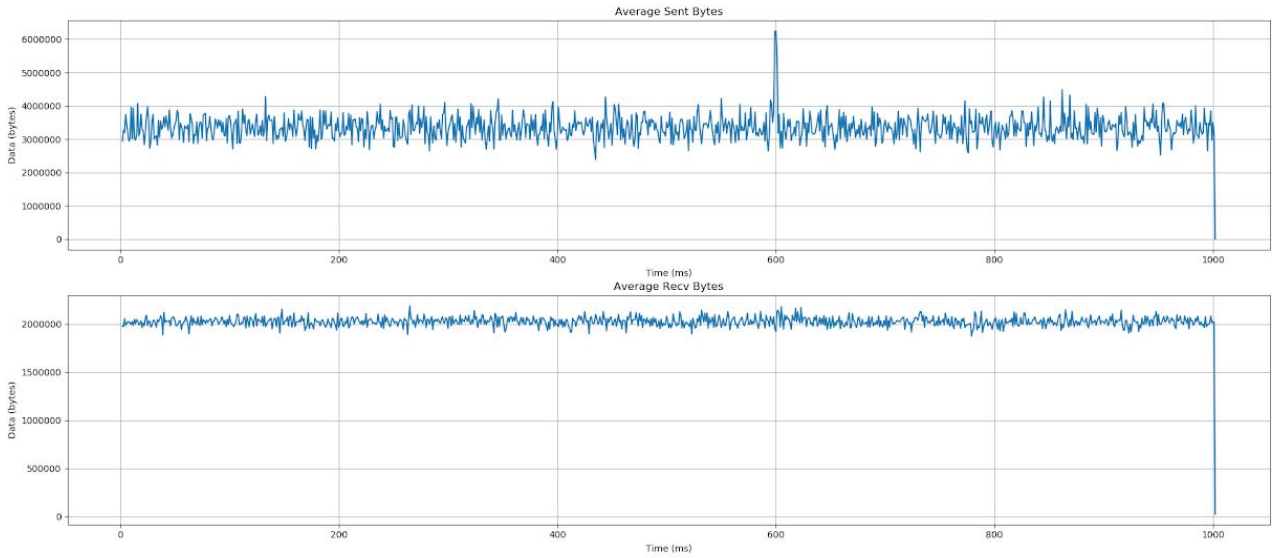

==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 1
==>totals<==
connections: 1000
sent(bytes): 3497230320
recv(bytes): 3381142996


==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 2
==>averages<==
delay(ms): 292
packets sent: 3447
data sent(bytes): 3497230
data recv(bytes): 3381143

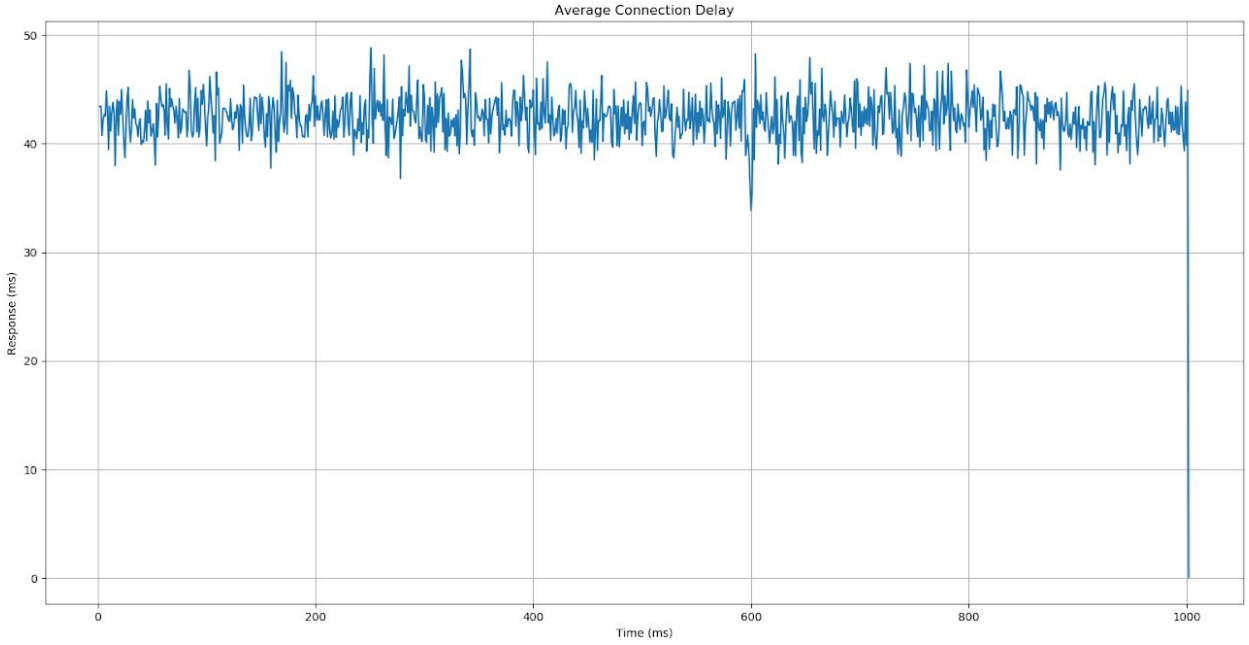
Epoll Server

The epoll Edge Triggered server was by far the fastest server out of the three. While it matched the poll server for raw throughput it was a limitation of the network card that allowed it to seem comparable. This was most noticeable when adding new connections as epoll could quickly add 10,000 clients in 30 seconds approximately, while after 15 minutes poll had only added 5,000. Should a faster network card be provided epoll would show its superiority far more quickly as well as its ability to maintain a higher throughput. For this reason epoll was also selected as the backing structure of the client application

100 Clients



Overall epoll easily handles 100 clients without any issues and all variance is quite minimal. With the capped sending and receiving windows There are not enough clients to maximize the network usage so all the variance is due to the scheduling differences in the threads.

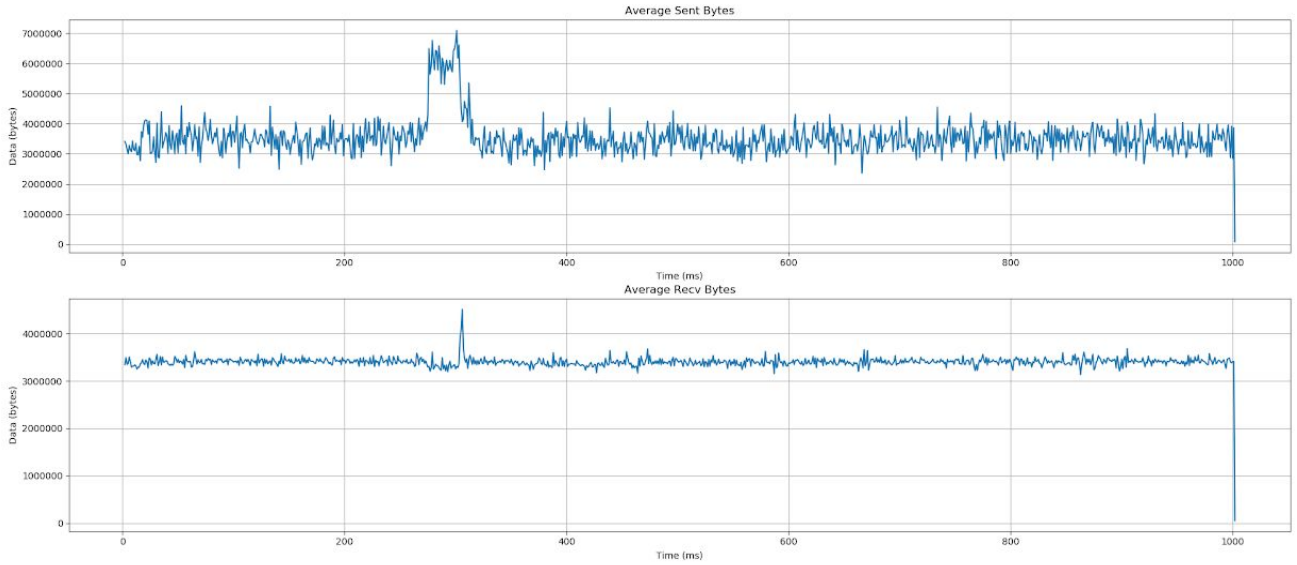


The delay hovers around 45ms never rising above 50ms and does not have any significant changes throughout the capture. This is expected as the overall system usage is very low for so few connections.

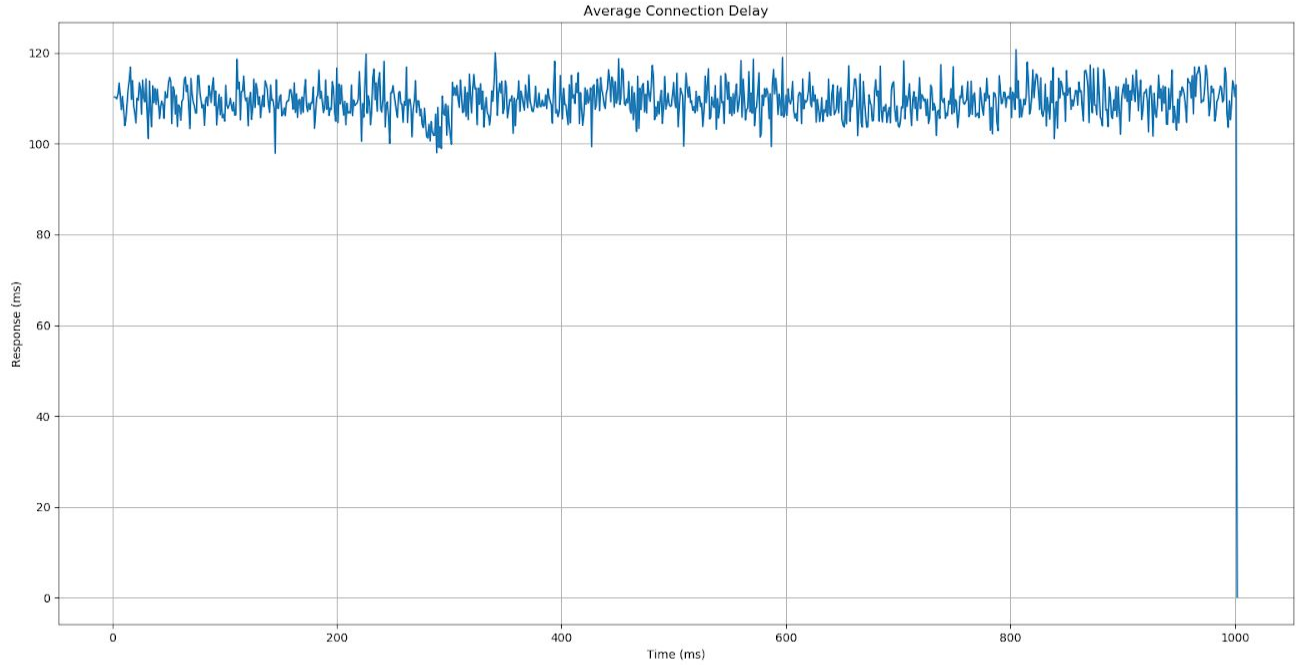
```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 1
==>totals<==
connections: 100
sent(bytes): 3353788640
recv(bytes): 2026934104
```

```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 2
==>averages<==
delay(ms): 143
packets sent: 22966
data sent(bytes): 33537886
data recv(bytes): 20269341
```

1000 Clients



Overall epoll easily handles 1000 clients as well without any issues and all variance is quite minimal. Scheduling differences in the threads explains the brief increase in sending data but overall both sending and receiving nearly have maximum network card usage.

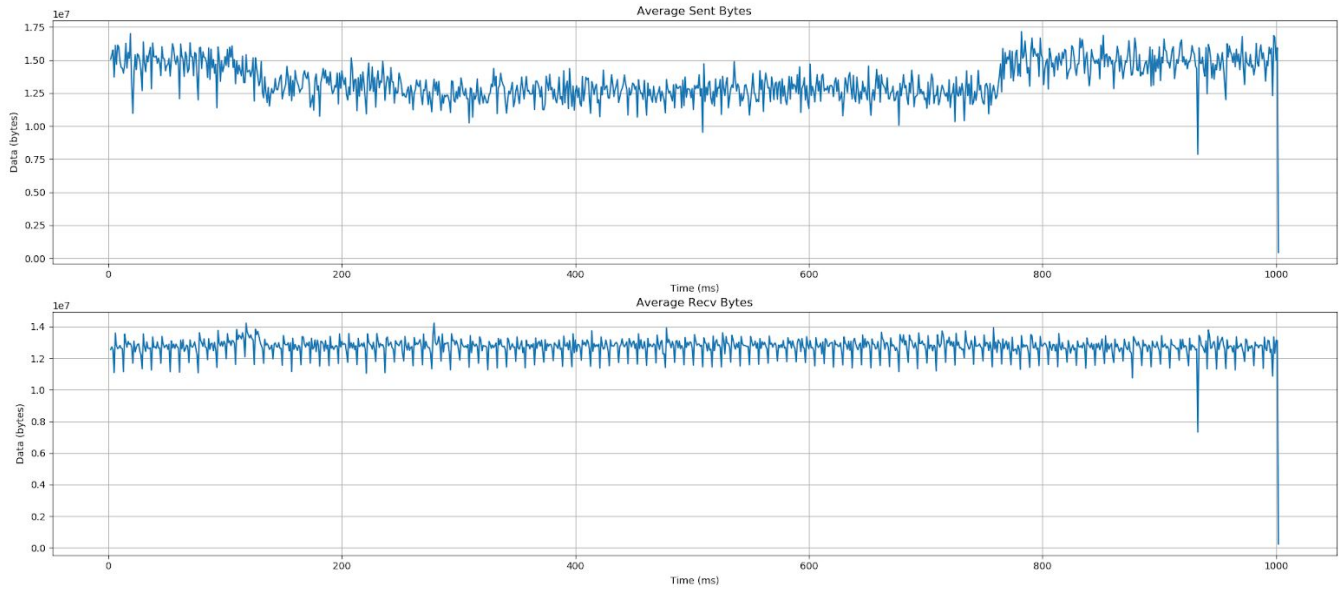


The delay is the most noticable difference from the 100 clients as it has risen to hover around 110ms. This is slightly more than double with ten times the clients. This is expected as the quantity of clients increased an order of magnitude from the previous test.

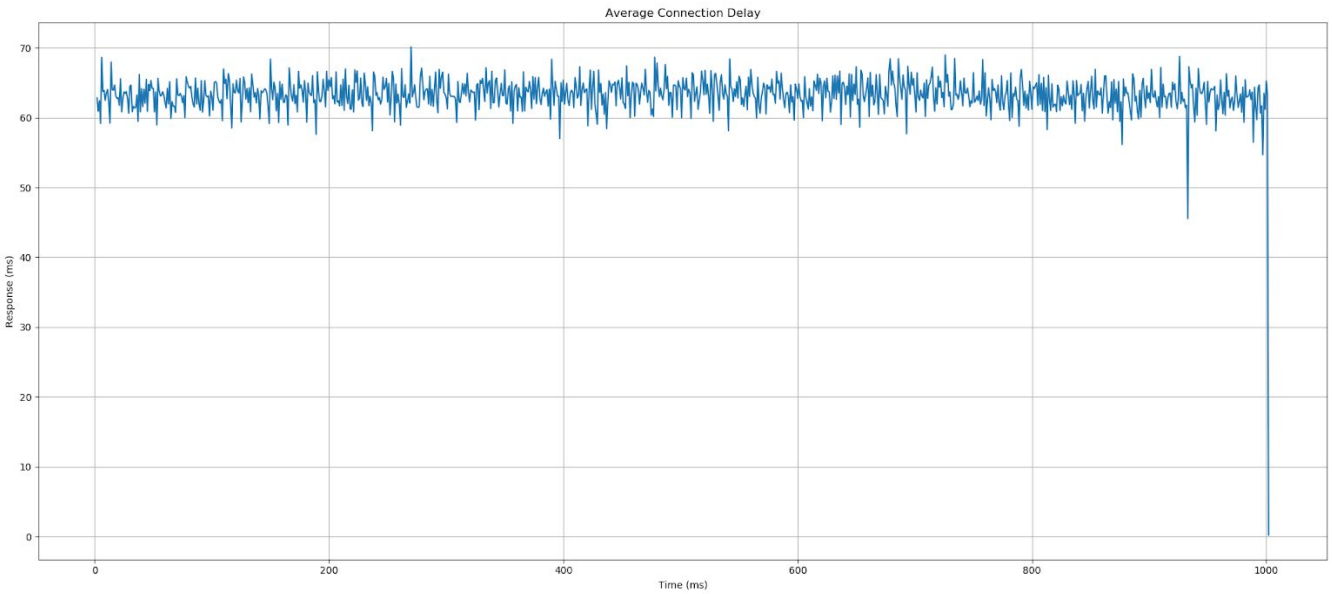
```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 1
==>totals<==
connections: 1000
sent(bytes): 3532053136
recv(bytes): 3400692772
```

```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 2
==>averages<==
delay(ms): 412
packets sent: 3446
data sent(bytes): 3532053
data recv(bytes): 3400693
```

10,000 Clients



This test differed from the others as 30 seconds is not enough time to accurately ascertain how well it performs so for this test only it was increased to 120 seconds. As you can see above the receiving was consistent throughout but the sending took a dip until the connections were saturated at 800ms and it then rises back up to the maximum throughput of the network card.



The delay for all of this hovers between 60ms and 70ms throughout. This is surprisingly low but its cause is the fact that not all connections are serviced at ever time slice. As the total graph stats show the average delay per client were instead 1027ms. The complexity that caused the initial performance hit in the lower number of connections ends up paying off as only epoll can handle this number of connections and still service them

```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 1
==>totals<==
connections: 10000
sent(bytes): 13485964376
recv(bytes): 12712885464
```

```
==>menu<==
1 show totals
2 show averages
3 show send and recv graphs
4 show timing graphs
5 exit
: 2
==>averages<==
delay(ms): 1027
packets sent: 1325
data sent(bytes): 1348596
data recv(bytes): 1271289
```

Conclusion

The most noticeable result from all of the testing performed is just how performant the networking programs are. The traditional server easily handled a hundred connections when our starting assumption was that it would not handle more than a few well. Poll was able to max out a gigabit network connection as was epoll though only epoll was expected to be able to.
The whole testing suite that was designed clearly illustrates the benefits of using event driven, edge triggered models for handling large numbers of connections. When the test for epoll is left on indefinitely with a thread to continually add new connections it can support up to 28,000 connections

while the next best poll can only support 6,000. In these extreme cases such efficient notification strategies give a clear advantage.