# Set Up

In converting the implementations to use ExecutorService, I created 4 different implementations:

- Single Thread (TCPEchoServerSingleExecutor)
- Thread Per Connection (TCPEchoServerThread)
- Cached Thread Pool (TCPEchoServerCachedPoolExecutor)
- Fixed Thread Pool (TCPEchoServerFixedPoolExecutor)

Upon comiling (mvn clean compile), I ran each and assigned them to their own port, so that I could run my Jmeter tests en masse. The following commands will run each of the implementations:

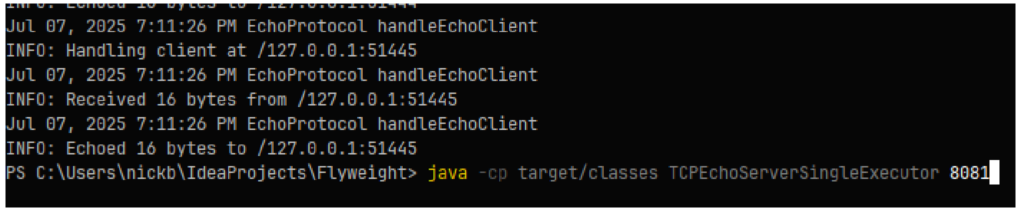
-java -cp target/classes TCPEchoServerSingleExecutor 8081

-java -cp target/classes TCPEchoServerThread 8082

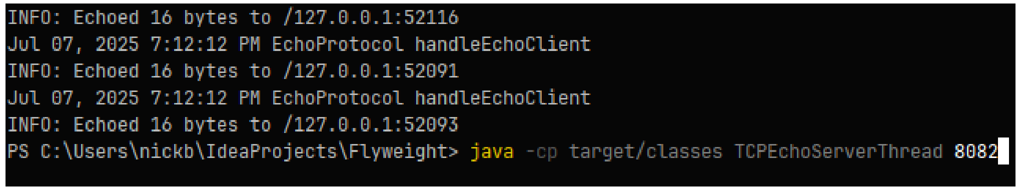
-java -cp target/classes TCPEchoServerCachedPoolExecutor 8083

-java -cp target/classes TCPEchoServerFixedPoolExecutor 8084 50 (or 10, this argument is the thread pool size)

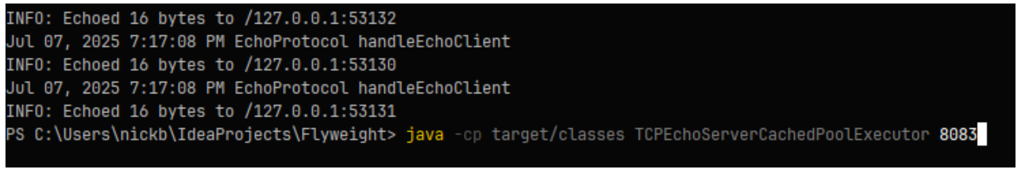
I also created an EchoProtocol class to handle receipt and echo of the messages, with delay when message size >= 4



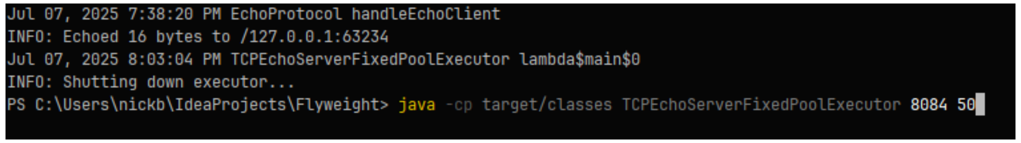
```
Jul 07, 2025 7:11:26 PM EchoProtocol handleEchoClient
INFO: Handling client at /127.0.0.1:51445
Jul 07, 2025 7:11:26 PM EchoProtocol handleEchoClient
INFO: Received 16 bytes from /127.0.0.1:51445
Jul 07, 2025 7:11:26 PM EchoProtocol handleEchoClient
INFO: Echoed 16 bytes to /127.0.0.1:51445
PS C:\Users\nickb\IdeaProjects\Flyweight> java -cp target/classes TCPEchoServerSingleExecutor 8081
```



```
INFO: Echoed 16 bytes to /127.0.0.1:52116
Jul 07, 2025 7:12:12 PM EchoProtocol handleEchoClient
INFO: Echoed 16 bytes to /127.0.0.1:52091
Jul 07, 2025 7:12:12 PM EchoProtocol handleEchoClient
INFO: Echoed 16 bytes to /127.0.0.1:52093
PS C:\Users\nickb\IdeaProjects\Flyweight> java -cp target/classes TCPEchoServerThread 8082
```



```
INFO: Echoed 16 bytes to /127.0.0.1:53132
Jul 07, 2025 7:17:08 PM EchoProtocol handleEchoClient
INFO: Echoed 16 bytes to /127.0.0.1:53130
Jul 07, 2025 7:17:08 PM EchoProtocol handleEchoClient
INFO: Echoed 16 bytes to /127.0.0.1:53131
PS C:\Users\nickb\IdeaProjects\Flyweight> java -cp target/classes TCPEchoServerCachedPoolExecutor 8083
```



```
Jul 07, 2025 7:38:20 PM EchoProtocol handleEchoClient
INFO: Echoed 16 bytes to /127.0.0.1:63234
Jul 07, 2025 8:03:04 PM TCPEchoServerFixedPoolExecutor lambda$main$0
INFO: Shutting down executor...
PS C:\Users\nickb\IdeaProjects\Flyweight> java -cp target/classes TCPEchoServerFixedPoolExecutor 8084 50
```

## Cached vs Fixed Thread Pool vs Thread Per Connection

The cached thread pool from ExecutorService allows for creation of new threads as new client requests are made (so lazy initialization). However, where it stands out from the other implementations is that the threads are cached and thus can be re-used when available. This is what makes it a great example of the Flyweight design pattern, since Flyweight uses a factory to create new objects when needed but reuse existing if it already exists. Fixed Thread Pool, on the other hand sets a predetermined pool size of threads and appropriates those as requests come in. Once all threads are used, no further all requests must wait until a thread is available. As I show a little later, Cached Thread Pool yields better performance.

Thread Per Connection has similarities to cached thread pool in that new threads are created as new connections come in. However, thread per connection does not reuse threads from a cache like cached thread pool does. Since there is not a direct similarity in ExecutorService for thread per connection, I used the original class to show how it compares performance-wise to the other implementations.

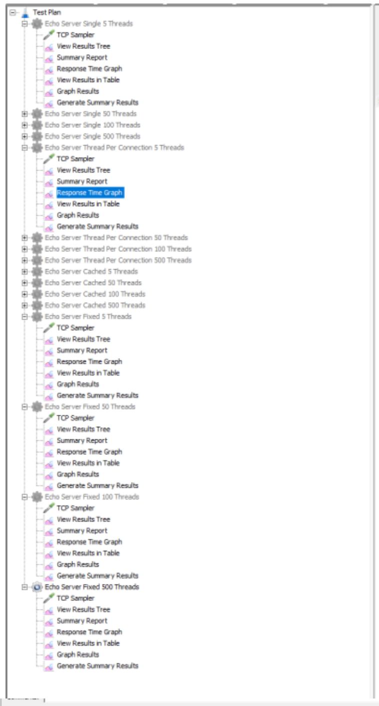
I also have a single thread implementation. As you will see, this is the worst of the group in terms of performance by a large margin.

# Jmeter & Power BI

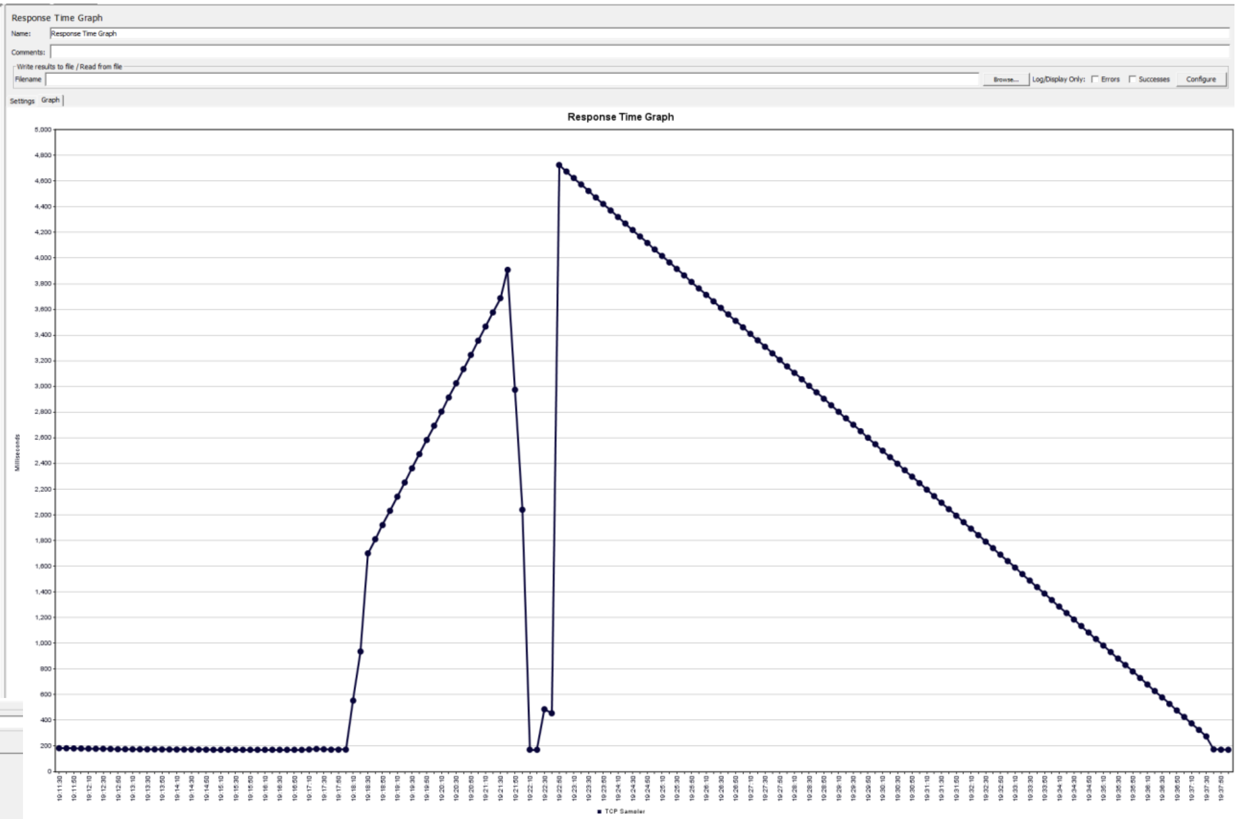
After starting each of my own implantations on their own port, I used Jmeter to simulate 5, 50, 100, and 500 requests to each of them. I ran the simulation for fixed thread pool twice, once with a pool size of 10 and once with a pool size of 50.

Jmeter has its own suite of visuals (as you can see in the examples to the right). However, I did not think they were good at telling the visual story about which implementation performed the best. Thus, I exported the results of the tests to create my own charts in Power BI to evaluate performance.

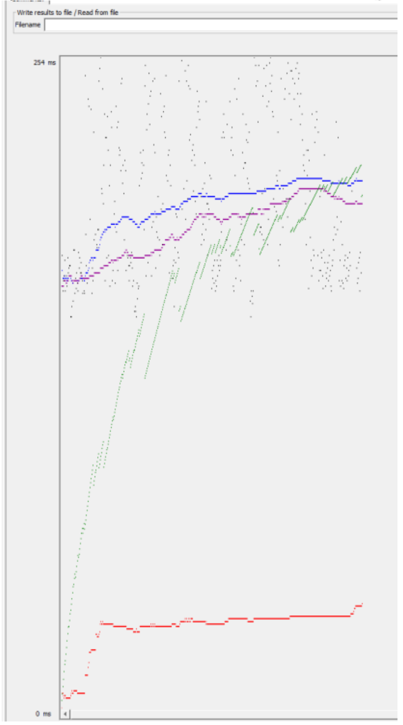
The following slides show my results for each connection type with 5, 50, 100, and 500 threads



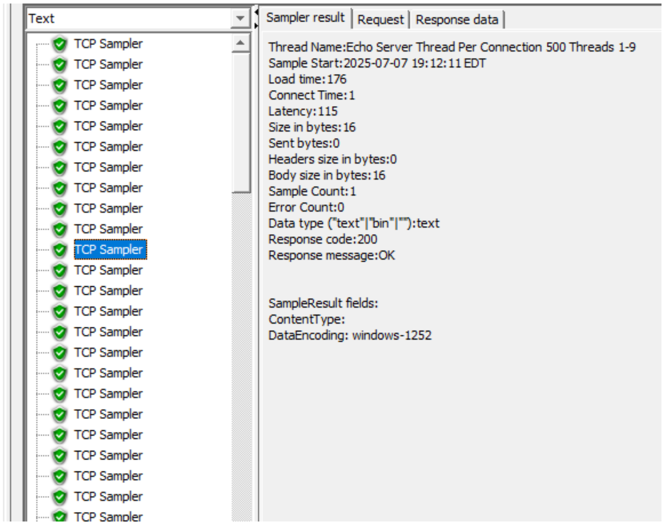
```
Test Plan
  Echo Server Single 5 Threads
    TCP Sampler
    View Results Tree
    Summary Report
    Response Time Graph
    View Results in Table
    Graph Results
    Generate Summary Results
  Echo Server Single 50 Threads
  Echo Server Single 100 Threads
  Echo Server Single 500 Threads
  Echo Server Thread Per Connection 5 Threads
    TCP Sampler
    View Results Tree
    Summary Report
    Response Time Graph
    View Results in Table
    Graph Results
    Generate Summary Results
  Echo Server Thread Per Connection 50 Threads
  Echo Server Thread Per Connection 100 Threads
  Echo Server Thread Per Connection 500 Threads
  Echo Server Cached 5 Threads
  Echo Server Cached 50 Threads
  Echo Server Cached 100 Threads
  Echo Server Cached 500 Threads
  Echo Server Fixed 5 Threads
    TCP Sampler
    View Results Tree
    Summary Report
    Response Time Graph
    View Results in Table
    Graph Results
    Generate Summary Results
  Echo Server Fixed 50 Threads
    TCP Sampler
    View Results Tree
    Summary Report
    Response Time Graph
    View Results in Table
    Graph Results
    Generate Summary Results
  Echo Server Fixed 100 Threads
    TCP Sampler
    View Results Tree
    Summary Report
    Response Time Graph
    View Results in Table
    Graph Results
    Generate Summary Results
  Echo Server Fixed 500 Threads
    TCP Sampler
    View Results Tree
    Summary Report
    Response Time Graph
    View Results in Table
    Graph Results
    Generate Summary Results
```



Response Time Graph
Name: Response Time Graph
Comments:
Write results to file / Read from file
Filename



Write results to file / Read from file
Filename



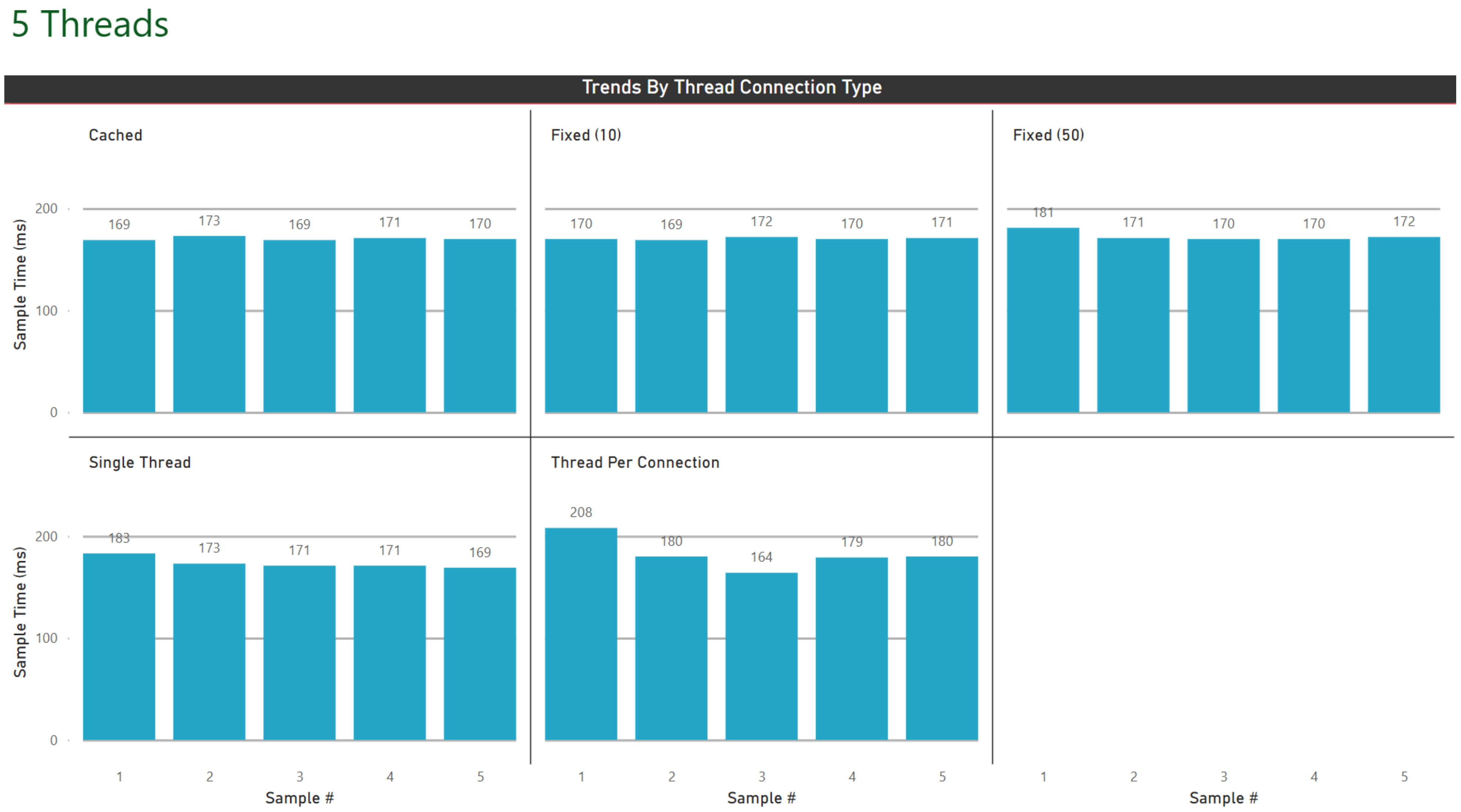
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler
TCP Sampler

```
Thread Name:Echo Server Thread Per Connection 500 Threads 1-9
Sample Start:2025-07-07 19:12:11 EDT
Load time:176
Connect Time:1
Latency:115
Size in bytes:16
Sent bytes:0
Headers size in bytes:0
Body size in bytes:16
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK


SampleResult fields:
ContentType:
DataEncoding: windows-1252
```

# 5 Threads

With only 5 threads, no implementation stood out with each performing relatively similarly

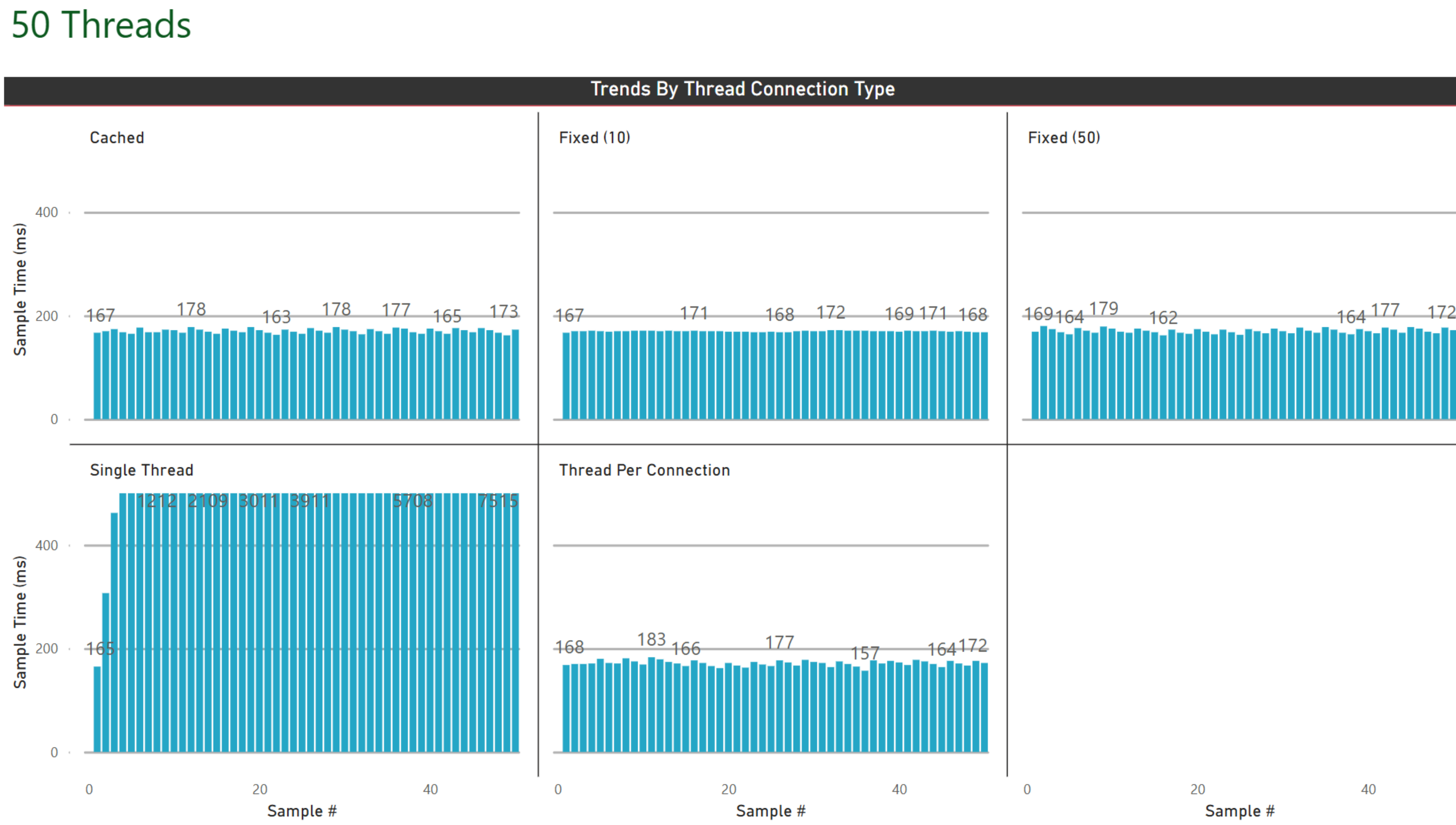


# 50 Threads

When the load is increased to 50 threads, the Sample Time of the Single Thread implementation skyrockets.

This makes sense as there is only one thread that must be waited on by each request.

The other 4 implementations stay relatively consistent throughout the run.



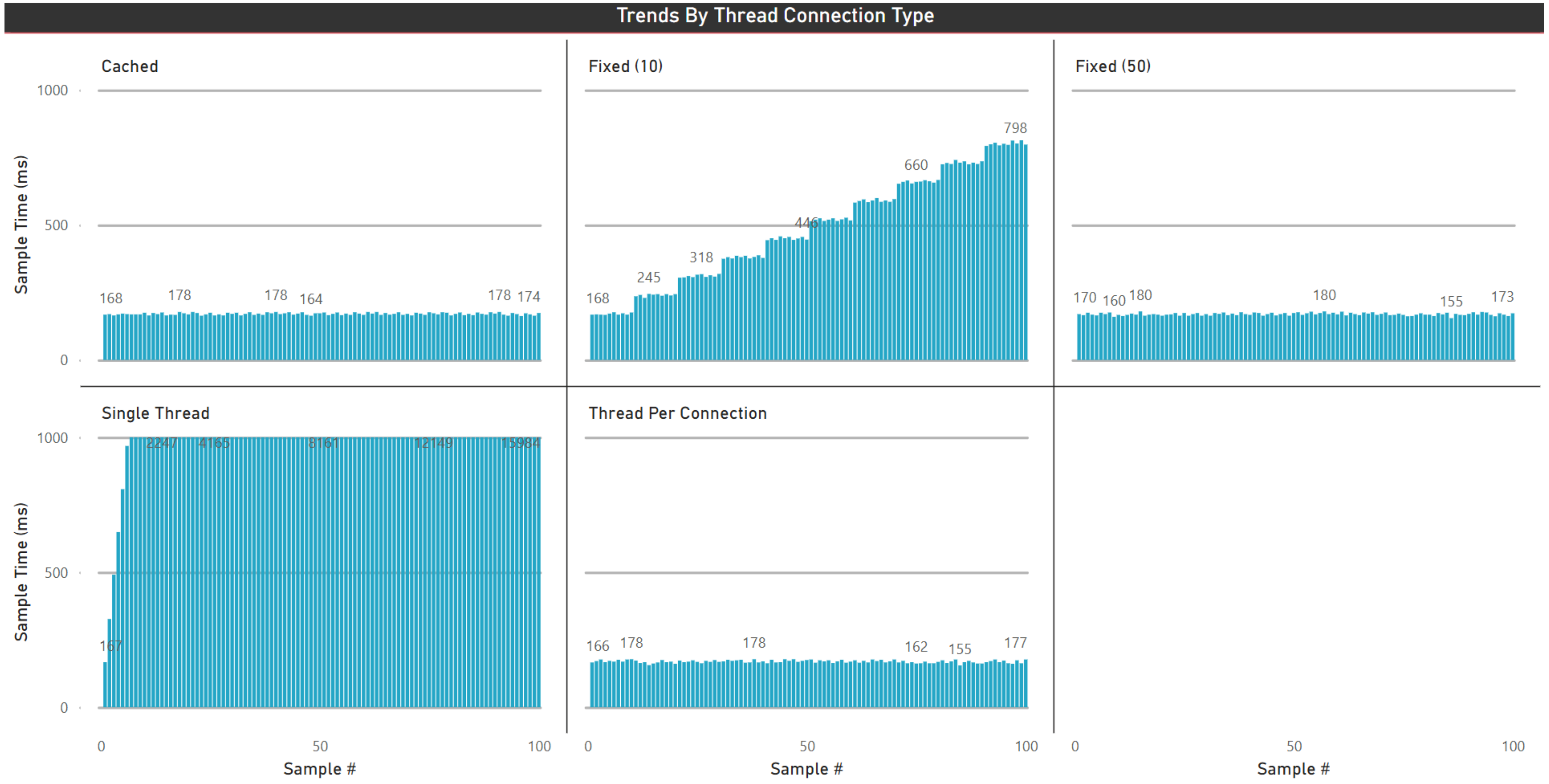
# 100 Threads

As the load increases, weak points in certain implementations begin to show themselves. Single Thread continues to balloon out of control while the Fixed Thread Pool with 10 pool size begins to suffer.

This makes sense, since there is only a pool of 10 threads to go around, a load of 100 threads is going to have many requests waiting.

Cached, Fixed (50 pool size), and Thread Per Connection continue to remain consistent.

## 100 Threads



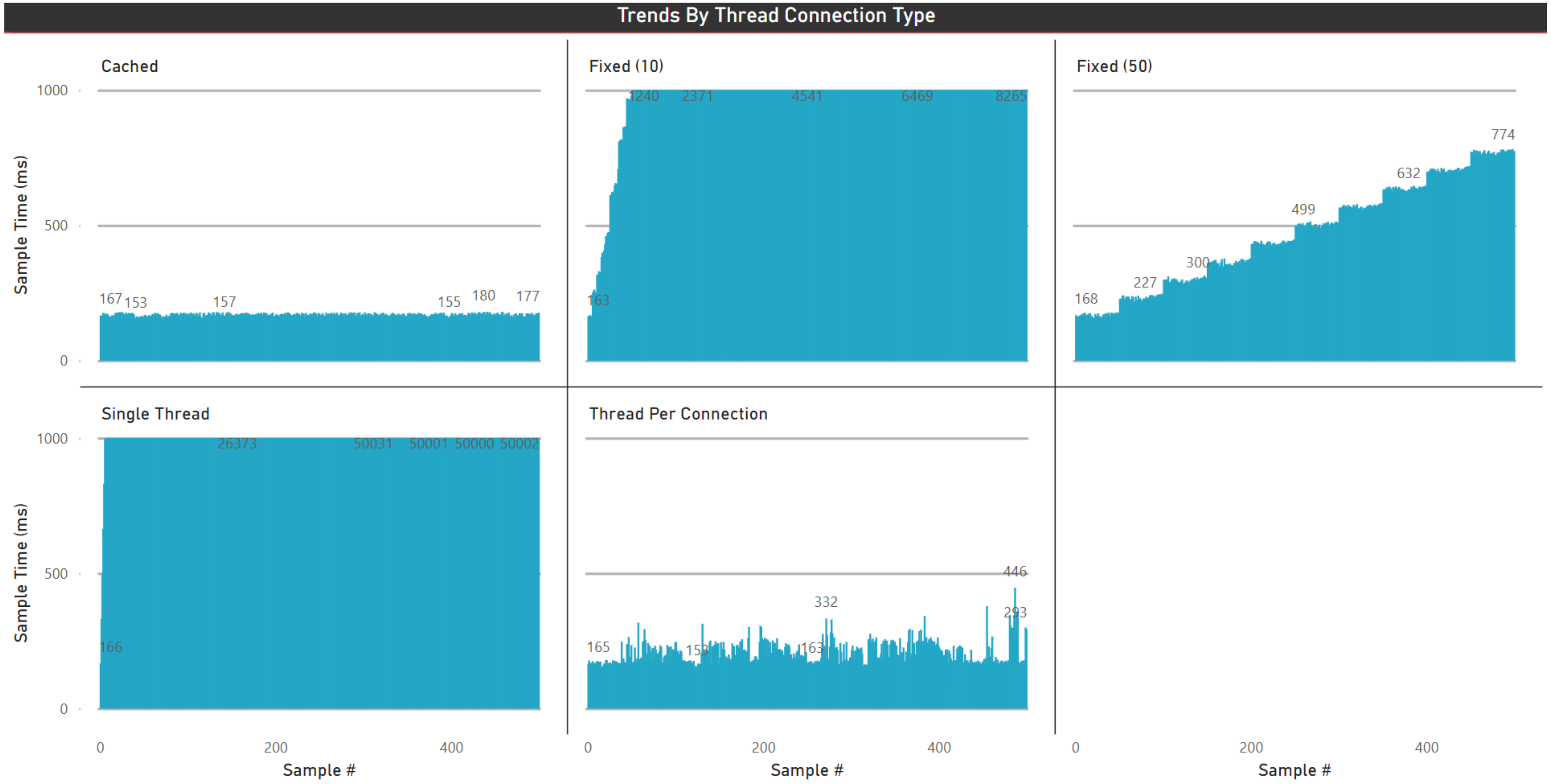
# 500 Threads

The 500 thread load showed the cached thread pool is the best implementation for performance. Single Thread hit the response time cap and began to fail. The Fixed Thread Pool (10) continued to degrade while the 50 pool size began its degradation (as expected).

The Thread Per Connection also began to suffer, albeit no where near as bad as the previous 3. It can be seen that there are much higher times at the end of the run than at the beginning whereas the cached implementation stays consistent throughout. This is likely due to the overhead Thread Per Connection has from creating a new thread for every connection where as cached can reuse threads from its cache.

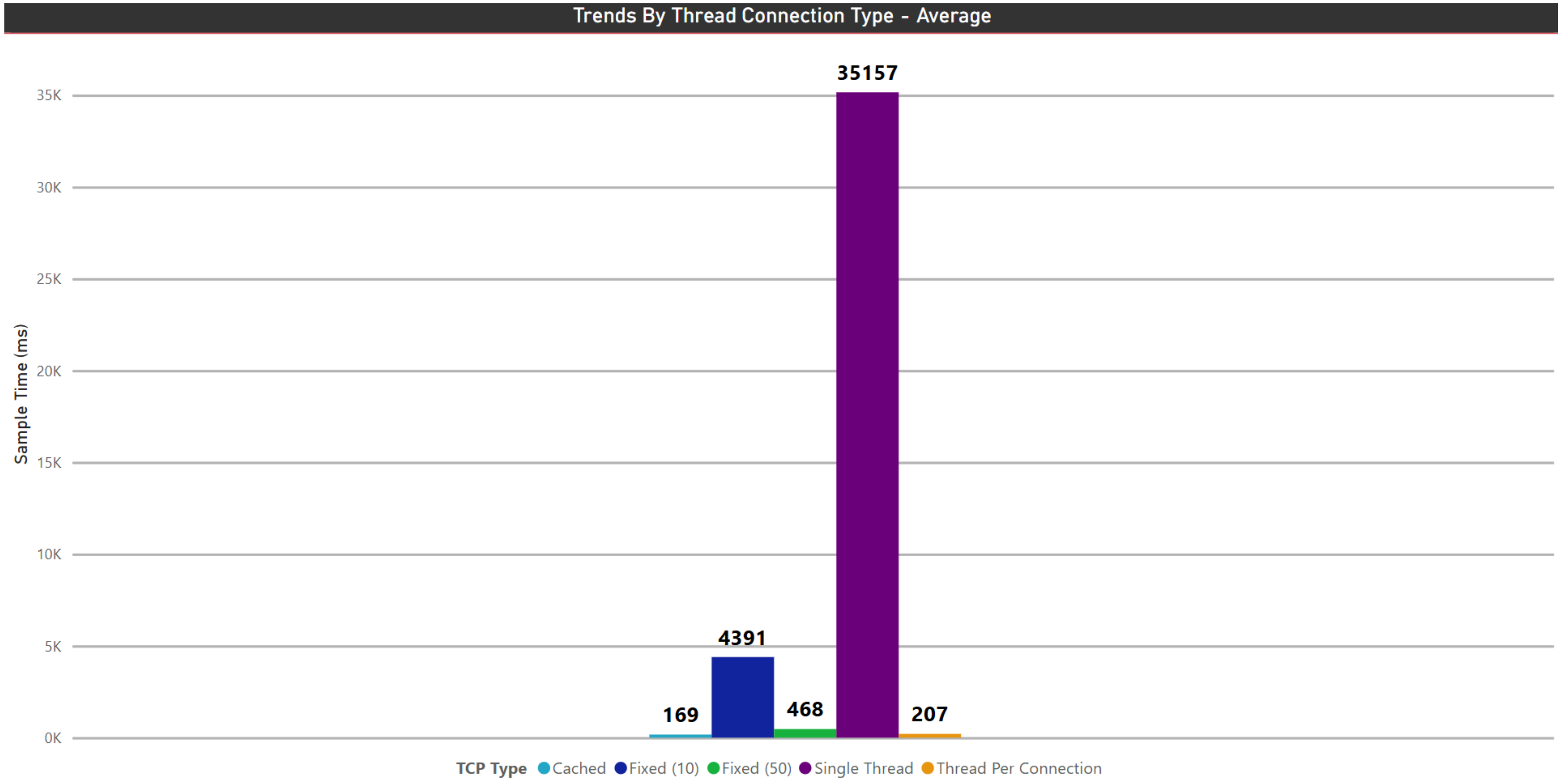
The following slide shows an average (rather than time series) of the 500 thread response times to show this point even better.

## 500 Threads



# 500 Threads (Average Time)

## 500 Threads


Trends By Thread Connection Type - Average
Sample Time (ms)
35K
30K
25K
20K
15K
10K
5K
0K
TCP Type
● Cached
● Fixed (10)
● Fixed (50)
● Single Thread
● Thread Per Connection