High throughput TCP server
Design Considerations

- **Maximum throughput** This comes down to a few aspects of design and testing form experience.
    - Optimal threading design to avoid threads spent time in blocked state on I/O, monitors etc
    - Usage Appropriate Concurrency datastructures and locking levels for Java code
    - JVM parameters tuning (Optimal JVM collector algo etc)
    - Resources (Memory /CPU)

- **Server Socket acceptor** thread logic is under TCPSocketServer. This thread's main responsibility is to listen to client connection requests and hand over managing I/O for each client connection to a new thread. Prevents any long running client read preventing other connection requests from being blocked. This thread listens infinitely until a terminate command is received. This maintains a state to understand the total number of client connections added to implement req 1 (at most 5 connections)
- **Client Socket I/O thread**: The design uses a one thread per established client socket connection model. Reading from an established client connection is handled by IncomingMessageHandler.
- **Log writer** : Since logging in another I/O work (other than reading client socket) it is done in another thread asynchronously (LogWriter) and order is not important in what order it goes to log based on requirement. A runnable thread that does the following:
    - Shares a LinkedBlockingQueue between the IncomingMessageHandler and the LogFileWriter in the producer consumer pattern.
    - This is done to make the writing to log file an asynchronous task to avoid blocking the incoming message listening thread from the client
- **Periodic StatisticsService**
    - This service's primary responsibilities are twofold
        1. Keep up to date the 3 metrics requested.
            - Each of the client reading processing will update this service when a valid 9 digit message is read.
            - **Concurrency choices:** Maintains a ConcurrentHashMap(Key is the integers seen, value is just a placeholder empty sting)) to check for dupes through the entire lifetime of the app. (The assumption here is that this is not a production so it is fine since this map could contain large amount of unique integers seen in history). Needs more calculation here. Java util concurrency packages are used to maintain the lowest level of atomicity of code e.g. AtomicInteger etc. At max 5 threads could be updating the state here. This service is ready for more thread concurrency.
        2. Periodic printing of report to the standard out is done by a scheduled harvest task. At the end of the harvest 10 sec cycle the period level stat values are set back to 0.

```
• private final Runnable periodicStatsTask = new Runnable() {
    @Override
    public void run() {
      try {
        logStatistics();
      } catch (Exception e) {
        log.error("Exception in the periodic reporting thread", e);
      }
    }

    private void logStatistics() {
      log.info("Received {} unique numbers, {} duplicates. Unique total: {}",uniqueIntegers.get(),
          newDuplicates.get(), totalIntegers.get());
      //now reset values
```

Other Considerations:
Java NIO was considered but since the maximum connections is restricted and the general understanding is that java nio helps scale and optimize for more client connections (100s or more avoid maxing threads handling client connection and read requests). In a typical production scenario for 100s/1000s of client requests NIO scales the best.

Requirement 1
This is taken care to maintain maximum of 5 connections as a count in the Acceptor loop. When this crosses the loop stops accepting more client connections.

```
05:30.431 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 1 unique numbers, 0 duplicates. Unique total: 1
05:40.041 [Thread-0] INFO  c.n.c.server.TCPSocketServer - Connection Accepted: Local Add /127.0.0.1 Remote Add /127.0.0.1:54765
05:40.042 [Thread-0] DEBUG c.n.c.server.TCPSocketServer - Current number of connections 2
05:40.042 [Thread-0] INFO  c.n.c.server.TCPSocketServer - Connection Accepted: Local Add /127.0.0.1 Remote Add /127.0.0.1:54766
05:40.042 [Thread-0] DEBUG c.n.c.server.TCPSocketServer - Current number of connections 3
05:40.042 [Thread-0] INFO  c.n.c.server.TCPSocketServer - Connection Accepted: Local Add /127.0.0.1 Remote Add /127.0.0.1:54767
05:40.043 [Thread-4] DEBUG c.n.c.handler.IncomingMessageHandler - Running message handler thread...
05:40.042 [Thread-5] DEBUG c.n.c.handler.IncomingMessageHandler - Running message handler thread...
05:40.043 [Thread-0] DEBUG c.n.c.server.TCPSocketServer - Current number of connections 4
05:40.043 [Thread-6] DEBUG c.n.c.handler.IncomingMessageHandler - Running message handler thread...
05:40.043 [Thread-0] INFO  c.n.c.server.TCPSocketServer - Connection Accepted: Local Add /127.0.0.1 Remote Add /127.0.0.1:54768
05:40.043 [Thread-0] DEBUG c.n.c.server.TCPSocketServer - Current number of connections 5
05:40.043 [Thread-7] DEBUG c.n.c.handler.IncomingMessageHandler - Running message handler thread...
05:40.043 [Thread-0] WARN  c.n.c.server.TCPSocketServer - Reached maximum connection limit 5; Stopping to accept more client connections
05:40.044 [Thread-0] WARN  c.n.c.server.TCPSocketServer - Reached maximum connection limit 5; Stopping to accept more client connections
05:40.044 [Thread-0] WARN  c.n.c.server.TCPSocketServer - Reached maximum connection limit 5; Stopping to accept more client connections
```

```
@Test
public void testMaxingConnections()
```

Could be improved to handle cases where a client connection goes away to decrement the count to allow back more connections. This is not implemented. But it is possible to do this with more time.

Requirement 2,3

The test generates a random integer with 9 digits. When it does break the requirement for instance 8 digits it gets dropped. The pattern check is done using java Matchers regex package.

c.n.c.handler.IncomingMessageHandler - Message 79339052 not matching expected pattern is being dropped

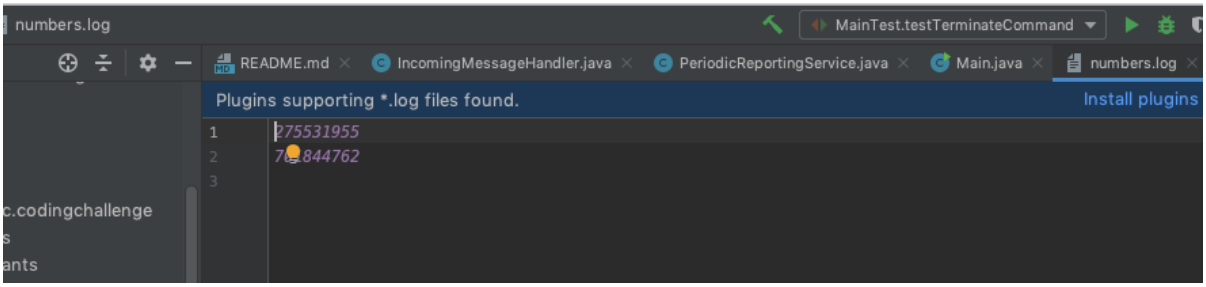
```
@Test
public void testSingleThreaded() throws InterruptedException, IOException {
```

Requirement 4 The log file, to be named "numbers.log", must be created anew and/or cleared when the Application starts.

The LogWriter when initialized using its constructor checks for a file named numbers.log.

- If there is, then clears it (Log below for that logic)
- If not, then creates a new file with that name in the root directory with the given name numbers.log

c.n.codingchallenge.utils.LogWriter - File numbers.log exists; Clearing it.



Requirement 7 Any data that does not conform to a valid line of input should be discarded and the client connection terminated immediately and without comment. Just to prove this is taken care added this log.

Processing message 80753266
08:58:09.665 [Thread-2] DEBUG c.n.c.handler.IncomingMessageHandler - Message 80753266 not matching expected pattern is being dropped
08:58:11.669 [Thread-2] DEBUG c.n.c.handler.IncomingMessageHandler - Processing message 268553232
08:58:11.670 [Thread-1] DEBUG c.n.codingchallenge.utils.LogWriter - Queue is not empty; Writing to file.

Requirement 8

PeriodicReportingService creates and updates the stats for printing below every 10 s. The printing job is managed by the period task executor provided by java.

Logging:
c.n.c.stats.PeriodicReportingService - Received 4055 unique numbers, 0 duplicates. Unique total: 53844

Requirement 9

Terminate condition is checked by each of the client socket handler to check for incoming message. This is in IncomingMessageHandler.

1. Once terminate is detected, an AtomicBoolean orderShutdown is set to true.

```
/**
 * Process the request and add to the log writer queue for async writing to file
  * @param msgString
 */
private void processMessageAndWriteToLog(String msgString) {
   log.debug("Processing message " + msgString);
   if(msgString.contains(Constants.TERMINATE_CMD)){
      log.info("Got a shutdown message :" + msgString);
      this.orderShutdown.set(true);
```

2. The Boolean flag is a shared state with the parent class TCPSocketServer which has ShutdownTask thread which continually scans for changes. Once it detects the shutdown value to be true, it starts a sequence of resources clearing tasks client connections as below:

```
3.    if (orderShutdown.get()) {
          log.info("Detected shutdown command ");
          terminateAllClientConnections();
          terminateWorkerThreads();
          log.info("Exiting shutdown task");
```

Test Case

```
@Test
public void testTerminateCommand() throws InterruptedException, IOException {
```

Logs

18:13:19.844 [Thread-2] INFO c.n.c.server.TCPSocketServer - Terminate command received; Closing sockets..
18:13:19.845 [Thread-2] INFO c.n.c.server.TCPSocketServer - Attempting closing client sockets..
18:13:19.889 [Thread-2] INFO c.n.c.server.TCPSocketServer - Attempting closing server listening socket
18:13:19.889 [Thread-2] INFO c.n.c.server.TCPSocketServer - Terminating worker threads..

Load Test Documentation

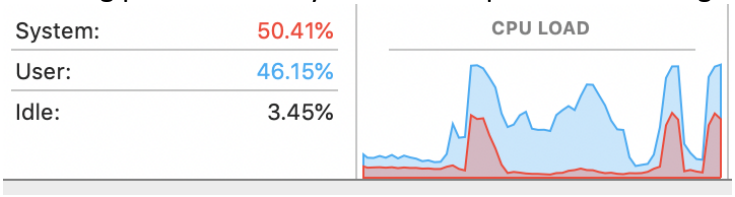
Reached almost to 180K in laptop setup

11:50:10.487 [pool-2-thread-1] INFO c.n.c.stats.PeriodicReportingService - Received 189058 unique numbers, 53 duplicates. Unique total: 397226
11:50:20.489 [pool-2-thread-1] INFO c.n.c.stats.PeriodicReportingService - Received 179314 unique numbers, 95 duplicates. Unique total: 576545
11:50:30.491 [pool-2-thread-1] INFO c.n.c.stats.PeriodicReportingService - Received 176526 unique numbers, 133 duplicates. Unique total: 753075
11:50:40.489 [pool-2-thread-1] INFO c.n.c.stats.PeriodicReportingService - Received 177078 unique numbers, 152 duplicates. Unique total: 930156
11:50:50.491 [pool-2-thread-1] INFO c.n.c.stats.PeriodicReportingService - Received 166406 unique numbers, 189 duplicates. Unique total: 1096566
11:51:00.490 [pool-2-thread-1] INFO c.n.c.stats.PeriodicReportingService - Received 165830 unique numbers, 204 duplicates. Unique total: 1262403

```
[YourKit Java Profiler 2019.8-b141] Log file: /Users/mramakrishnan/.yjp/log/Main-17335.log
10:28:43.046 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 152436 unique numbers, 207 duplicates. Unique total: 1256451
10:28:53.049 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 151550 unique numbers, 235 duplicates. Unique total: 1408004
10:29:03.046 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 164878 unique numbers, 278 duplicates. Unique total: 1572888
10:29:13.049 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 157980 unique numbers, 279 duplicates. Unique total: 1730868
10:29:23.049 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 151812 unique numbers, 354 duplicates. Unique total: 1882684
10:29:33.050 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 160188 unique numbers, 371 duplicates. Unique total: 2042877
10:29:43.050 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 168540 unique numbers, 386 duplicates. Unique total: 2211421
10:29:53.051 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 162363 unique numbers, 427 duplicates. Unique total: 2373786
10:30:03.051 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 166275 unique numbers, 503 duplicates. Unique total: 2540064
10:30:13.052 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 168458 unique numbers, 513 duplicates. Unique total: 2708527
10:30:23.052 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 165131 unique numbers, 563 duplicates. Unique total: 2873666
10:30:33.051 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 162190 unique numbers, 527 duplicates. Unique total: 3035859
10:30:43.053 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 170206 unique numbers, 629 duplicates. Unique total: 3206070
10:30:53.049 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 168680 unique numbers, 624 duplicates. Unique total: 3374753
10:31:03.050 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 150514 unique numbers, 580 duplicates. Unique total: 3525269
10:31:13.050 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 145793 unique numbers, 611 duplicates. Unique total: 3671068
10:31:23.053 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 152254 unique numbers, 667 duplicates. Unique total: 3823326
10:31:33.051 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 157883 unique numbers, 695 duplicates. Unique total: 3981214
10:31:43.054 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 153294 unique numbers, 686 duplicates. Unique total: 4134513
10:31:53.054 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 150659 unique numbers, 711 duplicates. Unique total: 4285175
10:32:03.055 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 151905 unique numbers, 732 duplicates. Unique total: 4437083
10:32:13.054 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 145226 unique numbers, 721 duplicates. Unique total: 4582316
10:32:24.515 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 125329 unique numbers, 663 duplicates. Unique total: 4707647
10:32:33.053 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 188481 unique numbers, 997 duplicates. Unique total: 4896136
10:32:43.053 [pool-2-thread-1] INFO  c.n.c.stats.PeriodicReportingService - Received 150468 unique numbers, 891 duplicates. Unique total: 5046607
```

CPU Bound ? Or More tuning on client load tests

I believe it could be potentially CPU bound potentially in my laptop since it was showing reaching peaks due to system + User process CPU usage.



Test setup

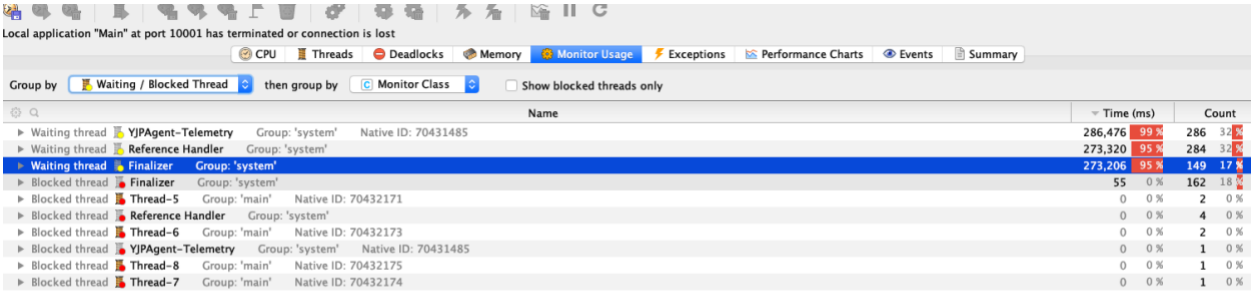
Client test

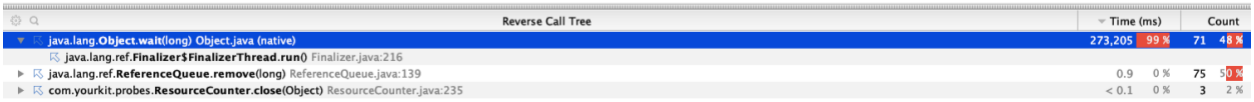
- One thread per socket writing uses a random int generation feature to keep generating continuous stream of integers for each of the 5 sockets

Analysis

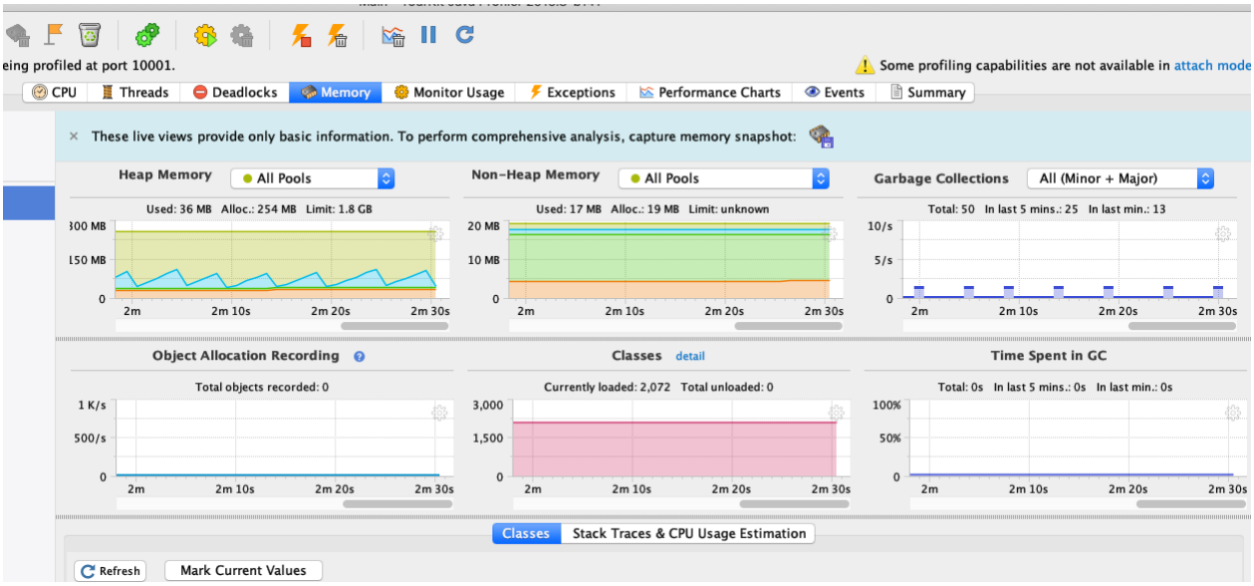
Concurrency Lock Analysis:

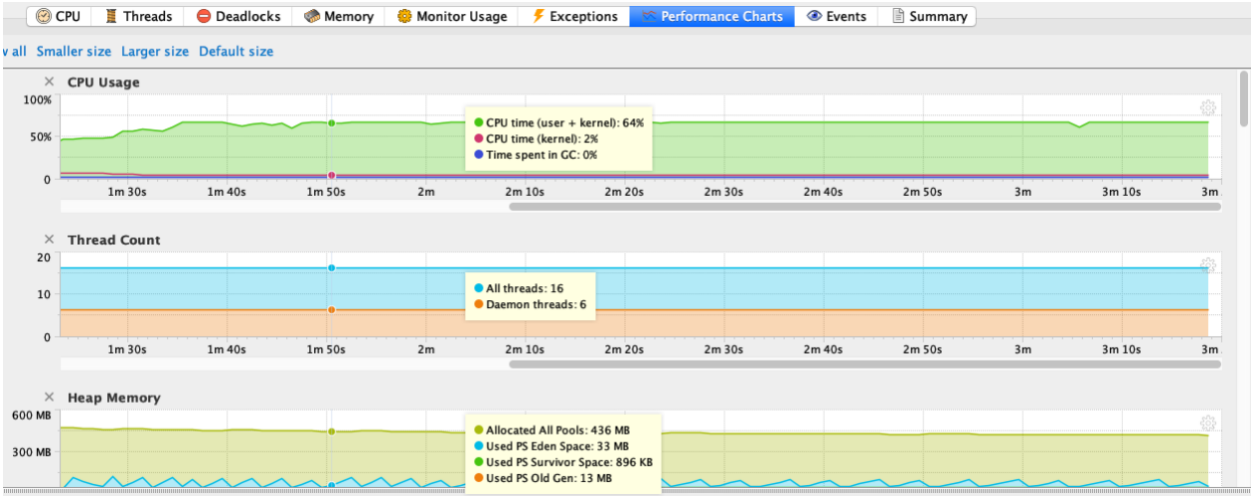
Used Yourkit's java monitor analysis tool to examine if any of the I/O running threads which means there is no more concurrency semantics that need optimization. The ones on waiting are the JVM GC threads



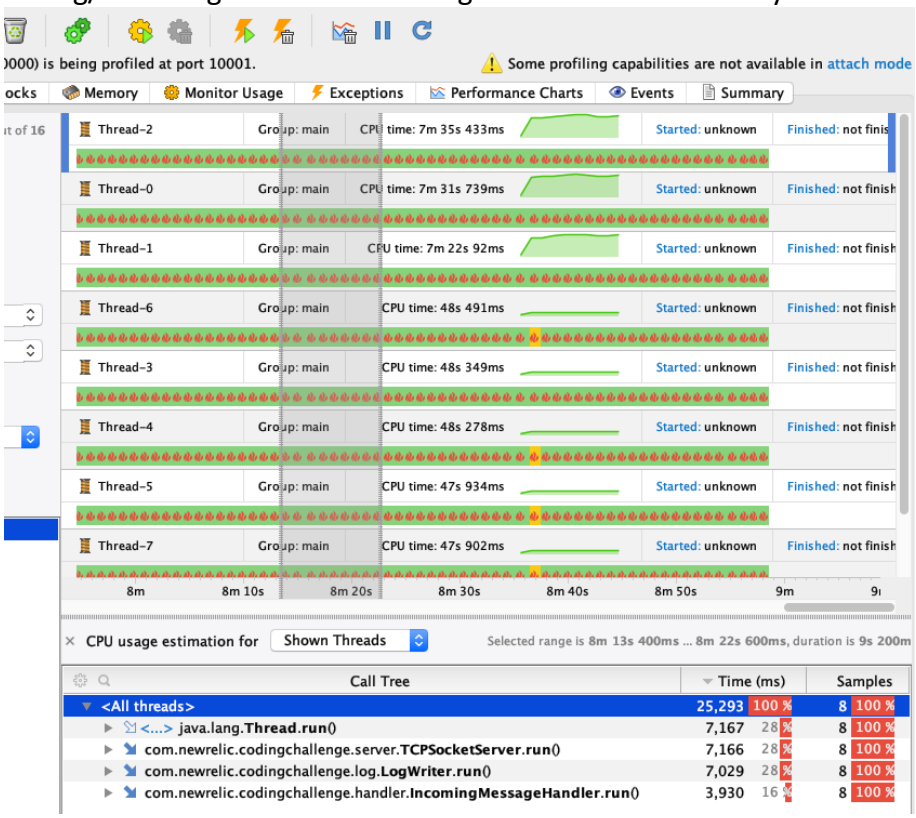


Heap Usage: Way less than allocated 1.8 Gb. No leak or memory issues. GC times were pretty minimal





I/O Threads are (Incoming message Handler) are pretty much in running state with no major locking/CPU usage issues as showing in the thread telemetry.




```
java -jar ./build/libs/coding-challenge-shadow.jar -Xms256m -Xmx2048m -XX:ParallelGCThreads=4 -XX:+UseParallelGC -XX:GCTimeRatio=9
```

Have specified some JVM GC options for the best possible GC optimization for throughput in my tests. It can be tuned more given the time.