Homework Assignment 3

Jim Miller
CPSC 433/533 - Computer Networks

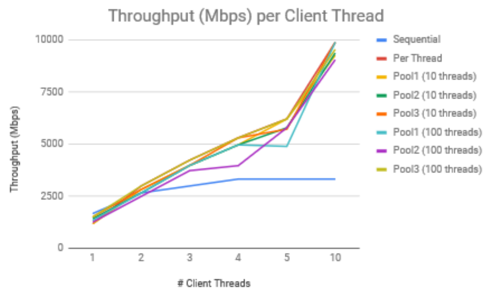
Exercise 1.

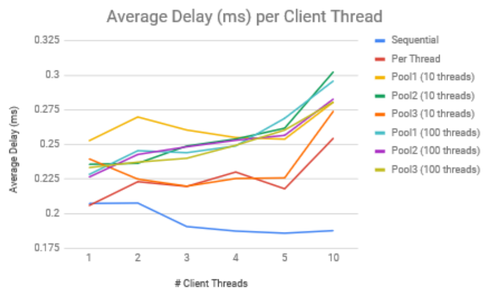
- See READ.ME

Exercise 2.

- a.) See READ.ME
- b.) Design questions:
  The timeout thread should not directly close the channel because we need the dispatcher to close the connection properly. The keep alive starting some sort of while loop. The server and client can periodically ping each other to ensure that the connection is still alive, and if a ping is not received after a given interval, the connection can be closed. HTTP 2 needs to implement mulitplexing, and so we need to be able to respond to multiple requests in parallel. We can implement multiple listener threads to respond to connections in parallel.
- c.) See READ.ME
- d1.)
  - The code for x-Socket simply adds another dispatcher in whenever every dispatcher has a full work load. There is not a set limit or maximum number of dispatchers. Therefore, the maximum number of dispatchers will be equal to the maximum value an Integer type can take in java. The dispatchers share the workload by using the nextDispatcher method, which uses a 'round-robin' approach.
  - The basic flow of the dispatcher thread is as follows. First, they call performRegisterHandleTasks, then performKeyUpdateTasks, then handleReadWriteKeys, then performDeregisteredHandlerTasks.
  - The onData method is called after data has been received from a connection. So, the server attempts to accept a connection, and when the connection is available, the server notifies the EchoHandler, which then calls onData.
  - The server sets a value for the maximum timeout. Then, under the hood of the 'connection' abstraction, a timeout event is registered if the maximum time has passed. The timeout event is then sent to and handled by the Idle Timeout Handler.

- d2.)
  - The boss group accepts an incoming connection, and worker group handles the traffic of the accepted connection. They achieve synchronization by grouping them b.group(boss, worker) and then creating a Future object from a b.bind(port).sync(). The sync method returns a future that allows for synchronization.
  - A future object is given when an event may not have completed; however, the future notifies when the event has occurred. To implement synchronization here, we could place locks around the future.
  - ByteBuf has to be explicitly released via the release() method.
  - The HTTP Hello World Server uses the LoggingHandler, an HTTP Request Encode/Decode handler (HTTPServerCodec), and a custom HTTPHelloWorldServerHandler for its handlers. The HTTP Snoop Server uses the LoggingHandler, the HTTPResponseEncoder handler, HTTPResponseDecoder handler, and the HTTPSnoopServerInitializer.
  - An inbound event to the ChannelPipeline is handled bottom-up by the inboundHandlers given to the Pipeline. For an outbound event, the opposite occurs. Outside of the pipeline, the inbound/outbound handlers communicate with I/O threads with nearby links/channels.
- e.) I was not able to connect to the apache server to compare benchmarks, but I compared the performance of my servers. I used the 100 doc.html files given in the tar file. I additionally used the request file included in the tar. Below are the comparisons of my servers Throughput and Average Delay. Each data point was received after running for 60 seconds.


Throughput (Mbps) per Client Thread
Throughput (Mbps)
# Client Threads
Sequential
Per Thread
Pool1 (10 threads)
Pool2 (10 threads)
Pool3 (10 threads)
Pool1 (100 threads)
Pool2 (100 threads)
Pool3 (100 threads)


Average Delay (ms) per Client Thread
Average Delay (ms)
# Client Threads
Sequential
Per Thread
Pool1 (10 threads)
Pool2 (10 threads)
Pool3 (10 threads)
Pool1 (100 threads)
Pool2 (100 threads)
Pool3 (100 threads)