Homework 1: Writing Component
Using Thread Pools to Manage Active Network Connections

Q1. What was the biggest challenge that you encountered in this assignment?

The biggest challenge that I encountered in this assignment was to set up the NIO connections properly and managing the keys accordingly. This was the first time I was working with NIO , so initially understanding the concept of NIO , socket channels,byte buffers , selectors and selection keys generated by the select function, handling those keys accordingly to accept the connection and to read the data from the socket channel took some time to understand. Initially after the writing the code for connections and sending and receiving messages , my code seemed to work correctly if I executed all the clients and the server in the same machine but it was not working properly when clients were being executed in different machines. The problem initially was I was just trying to read the data from the socket channel only once into the byte buffer , but when the clients are executed in different machines the time taken by the entire bytes of the message seem to bit longer as a result of which only partial messages were being read and wrong hash was sent back which was not present in the linked list of the client. More than one tasks instead of one task were created as a result of partial messages. But later I changed the code in a way such that we keep on reading from the socket channel until the byte buffer has no remaining bytes to fill. In this way I was able to get the entire message and only one task was created for this message. The part with managing the Thread pool , adding tasks to the task queue maintained in the thread pool, worker threads dequeuing tasks from the queue while managing to be thread safe by using synchronized dequeue function was also a little bit complicated.

Q2. If you had an opportunity to redesign your implementation, how would you go about doing this and why?

If I had an opportunity to redesign my implementation , I would do it in such a way that all the network operations like accepting the incoming connections , reading the messages from the socket channels and sending back the hash code of the received messages back to the client in the Thread pool itself. As of now my implementation is in such a way that the part where we accept the connections and read the messages from the clients are done in the main server thread itself, but the part where we calculate hash code of the received message and send it back the client is done by the worker threads in the thread pool. I did it in this way because several duplicate keys were being selected by the select function as the tasks created would take time to execute and if in the mean while select function is called the same selection key would be returned again resulting in creation of many duplicate tasks which would increase the work load on the worker threads. But I would redesign my implementation in such a way that all the tasks including accepting, reading and writing will be executed by the worker threads so that the entire computation is moved to the workers and the server will not become a bottleneck. The duplicate tasks created will be dropped or very little code will be executed for the duplicate tasks. For accept task , when a client connection is accepted the first time it returns the socket channel, however for the duplicate tasks accept function returns null , so I would check for null to know if the rest of the code should be executed or not. Similarly for read task , when we read the data into a byte buffer from the socket channel for the duplicate tasks it returns null which will be used to determine whether the subsequent code should be executed or not.

Q3. How well did your program cope with increases in the number of clients? Did the throughput increase, decrease, or stay steady? What do you think is the primary reason for this?

My program seemed to cope pretty well with the increase in the number of clients. The throughput through out the process of increasing the number of clients from one to hundred seemed to stay steady. Actually I tried to connect till around 150 clients , but the lab computers won't allow me to open more than 150 terminals , so I had to stop there. The throughput for 150 terminals seemed pretty steady. I think the primary reason throughput stayed steady was because I was using NIO channels to communicate, select function and selection keys based on the registered calls to create tasks which were being executed by the worker threads in the thread pool. The worker threads completed executing their tasks pretty quickly as the tasks were quite simple like sending a message back to the client using the socket channel. So almost all the tasks were executed as soon as they were added to the task queue in the thread pool with minimal delay. As the program description mentioned that we had to make our client thread sleep for some amount of time between sending messages to the server, I implemented that feature , which I think is also another factor for the throughput being steady because the amount of time for which I was making the client sender thread to sleep is considerably large the amount of time for the worker threads execute the tasks from the queue in the thread pool. However if we increase the number of clients to a much larger number , I think the throughput will gradually decrease as the sleeping time in the client would be a much less factor , and the fact that I used synchronized key work to dequeue the tasks from the thread pool might also result in decrease in throughput.

Q4. Consider the case where the server is required to send each client the number of messages it has received from that particular client so far. It sends this message at fixed intervals of 3 seconds. However, since each client has joined the system at different times, the times at which these messages are sent by the server would be different. For example, if client A joins the system at time T0 it will receive these messages at {T0 +3, T0 +6, T0 + 9, ...} and if client B joins the system at time T1 it will receive these messages at {T1 + 3, T1 +

6, T1 + 9, ...}
How will you change your design so that you can achieve this?

If I had to send messages to each client the telling the number of messages it has received from that particular client at fixed intervals of 3 seconds, also making sure that each client might join at a different time , so I would have to make sure that the messages indicating the count should be send accordingly. As the interval of time is 3 seconds, even if there are many clients , there will be only three set of clients for whom the count message has to be sent at a particular instance of time. For example consider if a client connects at second 1 , another at second 2 ,another at second 3, another at second 4 and so on, the count message for client connected at 1 and 4 should be send at the same time after client 4 is joined. Similarly this will be case for clients connected at seconds 2 and 3. In order to implement this , I will store the socket channel based on the hash value of the time the client has joined in a hash table. That hash table will have only 3 keys ie 0,1,2. I will be adding the information about the socket channel and count in the hash map and the key will be derived based on the time that particular client has joined ie key will be the remainder of time divided by 3. In this way I will store the socket channel and their count will be increased whenever a message is received from that client. For sending the count messages back to the client , I will have a separate thread which reads the values of socket channels and counts from the hash table and sends messages to the clients. It will have 3 sleep(each one second) statements. So count messages for the clients in the first key group will be sent first , then the thread will sleep for 1 second , then count messages will be sent for the second group , then sleep for 1 second , messages sent for the third group and sleep for 1 second. This process will continue in a while true loop.

Q5. Consider the overlay that you designed in the previous assignment. This overlay must support 10,000 clients and the requirement is also that the maximum number of hops (a link in the overlay corresponds to a hop) that a packet traverses is not more than 4. Assume that you are upgrading your overlay messaging nodes using the knowledge that you have accrued in the current programming assignment; however, you are still restricted to a maximum of 10 threads in your thread-pool and 100 concurrent connections. What this means is that your messaging nodes are now servers (with thread pools) to which clients can connect. Also, the messaging nodes will now route packets produced by the clients. Describe how you will configure your overlay to cope with the scenario of managing 10,000 clients. How many messaging nodes will you have? What is the topology that you will use to organize these nodes?

If the overlay has to support 10,000 clients , and if we are restricted to a maximum of 10 threads in the thread pool and only 100 concurrent connections then I will have 100 messaging nodes. In this way each messaging node manages 100 clients , so in total we will be managing 10,000 clients. The topology that I will be using to organize the nodes is half group based topology and half star based topology. The 100 messaging nodes will be divided into 25 groups and each group will have 4 nodes. A node within a group will be connected to all the other nodes present in that group. So each node in a group will have a minimum of 3 connections. Apart from that every group will have on end node(which will be chosen by some algorithm) which is connected to all the other 24 groups. So that particular node will have 28 connections. The messaging nodes while joining the overlay will advertise the details about the clients that are connected to that node to the entire group. The end nodes in a group will advertise the details of the entire group to all the other groups present in the overlay. In this way I will be able to route the packets from one client to another client using 3 hops within the overlay. Suppose a client sends request to send a message to another client to the messaging node it is connected to. Then that messaging node check the receiver's details and checks if any of the nodes within its group is connected to the receiver. If any node is connected then it will forward that message directly to the corresponding node. If any node within in that group is not connected to the receiver , the this node forwards the message to the end node within that group which then forward the message to the end node of the group which contains the receiver. After receiving the message the end node forwards it to the node which is connected to the receiver . So the number of nodes/hops this message has traversed is 3. In this way I will be configuring my overlay to cope with the scenario of managing 10,000 clients.