Part 2: Design of Algorithm to handle Transfers.

Assumptions:

1. There is always one server that is alive in a bank chain. This implies that there is always a head and tail for bank.

2. If an account is not present, a new one is created and instantiated with 0. This is a simplification used for transfers. So in the case of transfers, the destination account will hold the transfer amount after the transfer update is processed.

Client will build a update request with reqID, bankName, accountNum, amount, destBank, and destAccout. If the client doesn't receive a reply, it will will automatically retransmit the update to the head server. This will deal with failure of the source head server.

The servers of the banks will store all the heads and tails of the other banks. Initial head and tail servers are loaded with this information from the configuration file.

When a failure of the head or tail occurs, a notification is broad-casted to all servers of the various bank servers.

**Background about ACKing and SentUpdates:**

SentUpdates is a list that is used to store updates that have been processed by the server but have yet to be ACKed from the tail server in the chain. The tail server will send a ACK to its predecessor onces the update has been processed and will also send the client a response. The predecessor will then remove the update that the tail has processed from the SentUpdates list and send the ACK to it's predecessor so it can do the same. This will be repeated until the head receives the ACK and the update has been removed from SentUpdates. We are in essence sending a sequence number with all the updates so the ACK is just the same sequence number to show that we have replicated that data in all the severs.

**Failure Detection when transferring among Different Banks:**

When a transfer is received from a client at the source bank's head it will send the transfer update down the chain as seen in our pseudo code from part one. We will put the update entry in the sentUpdates list in each server. This will persist in the servers until the destination ACK is received at each server. Any failure in the source chain will fix itself from the failure detection algorithm in part one. When the transfer trickles down to the tail and the destination bank is different, the transfer update will be forwarded to the the head of the destination bank. In this case, we need to detect a failure of the destination head before sending the update down the chain. The source tail server will receive a message with the new head from Master. If the new head is in the Bank that we just sent a transfer to, we will re-send the transfer update again to the new head of the Bank. We can check if we just sent the transfer to the bank that just failed from the sentUpdates list that is maintained until the update has been processed. From the assumption above, we know that there is at least one head and one tail present in all bank chains at all times. This implies that we will eventually get a response for the destination tail at some point. If there is a failure in the destination chain, the failure detection from part one will detect the problem and relink the chain. The destination tail will send a ACK to its predecessor as well as the source tail. When the source tail receives the ACK, the update entry from the sentUpdates can be removed. This ACK will travel backwards all the way up to the source head. The sentUpdates is used to detect failures in our case and if we make it all the way back to the source head without failure then we remove it from the sentUpdate list. After sending both ACKs, the destination tail sends a response to the client.

