COMP9331 Assignment (Extended Version)

Session 1, 2015
Harold Jacob Hoare - z5021639

Program Design

The program is written in Java and consists of a main class cdht_ex with a main method and 3 static nested classes, each running a separate thread.
The main class defines several static fields (variables describing the state of the node) and fixed parameters.
The 4 simultaneous threads that the cdht_ex class runs are :

- main method - initialises fields, launches the other 3 threads, processes user input.
- PingTask - pings 2 successor peers, handles ungraceful peer departure.
- UDPServerThread - listens for and replies to ping messages, listens for ping replies.
- TCP ServerThread - listens for 4 message types :
  - a) a request for a file,
  - b) requested file received,
  - c) a request for successor peers,
  - d) graceful peer quitting.

Message Design

UDP Transport :

Message	Description
PING <sequence>	Send ping with sequence number <sequence>.
REPLY <sequence>	Reply to ping with incoming sequence number <sequence>.

TCP Transport:

Message	Description
REQUEST <wxyz> <sender> <requester>	Peer <sender> is forwarding a request message for file with name <wxyz> to be sent to peer <requester>.
FILE <wxyz> <id>	Inform recipient that peer <id> has file <wxyz>.
QUIT <first> <second> <id>	Peer <id> with successors or predecessors <first> and <second> is leaving the network.
QUIT_RECEIVED	Acknowledge a QUIT message.
LIST	Request recipient to reply with its list of successors.
PEERS <first> <second>	Sender has successors <first> and <second>.

How The System Works

Initialisation and Pinging

The main method sets the fields peerID, firstPeer and secondPeer according to the input arguments.
A new DatagramSocket is created with port number 50000 + peerID to send the pings over UDP.
UDPServerThread and TCPServerThread are both started, to listen for incoming messages.
A new Timer is created and scheduled to call the run method of class PingTask after 1 second and thereafter every 30 seconds.
The run method of class PingTask then continually sends DatagramPackets beginning with "PING" out of the UDP DatagramSocket to ping both firstPeer and secondPeer.

Ping Responses

UDPServerThread listens to the same port (50000 + peerID) as the outgoing ping requests. When a message is received it is parsed into words.

If the first word is "PING" a message is displayed that a ping has been received and the client's port number is translated to give the peerID that sent the ping. The incoming peerID is also used to update fields firstPredecessor and secondPredecessor that hold the IDs of the peers that sent the two most recent pings. A DatagramPacket with a ping response beginning with "REPLY" is then sent back to the sender via the DatagramSocket.
If the first word is of an incoming sentence is "REPLY" a message is displayed that a ping response has been received together with the ID of the responding peer.

File Requests

After initialisation the main method waits for user input. If the input is "request <wxyz>" (where <wxyz> is a 4 digit integer) a new TCP Socket is created with destination port of 50000 + firstPeer. The string "REQUEST <wxyz> <sender> <requester>" is sent where <sender> and <requester> are the same peerID in this first case (but are different when the request is forwarded). A message is displayed on screen that a file request message has been sent to the successor.
TCPServerThread creates a welcoming ServerSocket on port 50000 + peerID and when a client connects a connection Socket is created. As with UDP, incoming messages are parsed into words.
If the message is "REQUEST <wxyz> <sender> <requester>" the hash of the filename is calculated as <wxyz> mod 256. If (peerID - sender) mod 256 is greater than (peerID - hash) mod 256 then the file is stored on this node. A new TCP socket is created with destination port equal to 50000 + <requester> and a message beginning with "FILE" is sent to inform the requesting node that the file is here. A message is also displayed on screen that the file is here and that a reply has been sent to the requesting peer.
If the file is not stored here a file request message is forwarded to firstPeer via a new TCP Socket with destination port 50000 + firstPeer. If any node forwarding a file request cannot connect to its firstPeer then it forwards the request to its secondPeer as a fallback.
If TCPServerThread receives a message "FILE <wxyz> <peerID>" then a sentence is displayed on screen that the file <wxyz> has been received from <peerID>.

Graceful Exit

If the string input by the user to the main method is "quit" then 2 new TCP Sockets are created with destination ports of 50000 + firstPredecessor and 50000 + secondPredecessor. The string "QUIT <firstPeer> <secondPeer> <peerID>" is sent to both predecessors. Both successors are also informed of the departing node's predecessors with the message "QUIT <firstPredecessor> <secondPredecessor> <peerID>" sent over TCP. When all predecessors and successors have replied to acknowledge the quit request then a message is displayed on screen that this node is leaving and the program terminates.
If a TCP connection Socket receives a string beginning with "QUIT" then it displays a message that the sending peer is departing and replies with "QUIT_RECEIVED". The peer that received the quit message now checks whether the departing peer is its firstPeer or secondPeer. If it is its firstPeer then the current secondPeer is promoted to firstPeer and the new secondPeer becomes the departing peer's secondPeer. If the departing peer is the secondPeer then the new secondPeer is the departing peer's firstPeer. A message is displayed on screen showing the new first and second successors. The regular pings are naturally now directed to the updated firstPeer and secondPeer.
The peer that received the quit message then checks whether the departing peer is its firstPredecessor or secondPredecessor in a similar manner to the successor checking, and updates itself accordingly.

Peer Quit Without Warning

Sequence numbers are introduced to the ping messages in order to identify when a peer has left without warning. The initial sequence number is chosen randomly and thereafter incremented by 1 (mod 65536) each ping. The incoming sequence number is also appended to ping reply messages. The most recent sequence number that the first and second successor peers have acknowledged is recorded as firstAck and secondAck.
After sending a ping message a node waits 1000ms then checks whether the sequence number sent is the same as the most recent sequence number acknowledged by both successors. If a reply has not been received from either successor, its count of consecutive missing acknowledgements (firstLostPings or secondLostPings) is incremented. If the consecutive missing acknowledgement count for either successor

reaches 4 then that node is deemed to have left the network. A message is displayed that a successor is no longer alive and the successors of the remaining live peer are requested via the message "LIST" sent through a new TCP Socket.
If a TCP connection Socket receives a string beginning with "LIST" then it replies with "PEERS <firstPeer> <secondPeer>". The peer that requested the list then updates its first and second successor in the same manner as described above for graceful exit with one provision - if a node has just lost its second successor then it must check the response from its first successor to see if that first successor has already updated its successors or not, and act accordingly. A message is displayed with the new successors, who now receive the regular pings which are used to update their predecessors.

Design Tradeoffs and Considerations

An initial ping is sent after 1 second so that each peer can quickly determine its predecessors. Thereafter the ping frequency is every 30 seconds. This is a compromise between the waste of resources from pinging unnecessarily frequently and the need to regularly determine that successors are still alive.
A node that is gracefully quitting informs its successors as well as its predecessors. Although only predecessors are required to be informed by the spec, that causes nodes to take until the next received pings to update their predecessors. By informing successors directly they can update their predecessors faster and so the network is more robust.
The number of missing ping replies before a successor is deemed to have left without warning is set to 4. Setting this too low would remove peers that are in fact still alive despite a few pings have been lost in the network. Setting it too high would imply that a peer could be dead for too long before action is taken to rebuild the circular DHT.
Timeout before a ping is lost is set to 1000ms for illustrative purposes. In reality this could be set similar to the TCP timeout interval which is EstimatedRTT + 4 * DevRTT where EstimatedRTT and DevRTT are exponential weighted moving averages. This network models peers by using different ports on the same node and hence the RTT is very small compared to a real set of peers separated geographically.
The range of sequence numbers is 2^16. Since only 4 missing sequence numbers are required for a peer to be deemed dead this could be a smaller range but the number chosen is closer to actual networks.

Improvements and Extensions

1) Peer join. If a new peer wanted to join the network it could contact any node and a message would be passed around the circular DHT to find the peer that held the files to be transferred to the new peer. The neighbouring peers would update their successors and predecessors.
A more sophisticated approach to peer join would involve allocating a new peer id according to the current nodes that hold most files and minimisation of the gaps in the network.
2) Further ungraceful departure scenarios. When a peer is killed without warning the network requires 4 missed pings to identify the dead peer and then a further ping cycle to identify new predecessors. The program already includes a fallback by forwarding file requests to the second successor if the first does not respond. This could be further extended to handle graceful peer quit or multiple dead peers during the time that a peer has been killed but the network has not yet identified and corrected for that. This would involve sending messages to the other successor or predecessor when one of them does not respond.
3) Circle shortcuts. A node could track a certain number of peers in other parts of the network, enabling file requests to reach their destinations faster and performance approaching O(logN).
4) File request forwarding to second successor. If a node can determine that a file is not stored at its first successor then it could forward a file request message directly to its second successor, roughly cutting the number of file request forwards in half.
5) Bi-directional traffic. A peer could choose to send messages to either its successors or predecessors, meaning file request messages need not go around almost the entire circle to reach nearby predecessors.
6) Multiple peers holding same file. In a real network to protect against nodes leaving ungracefully and the files held there being lost there could be backup copies of files held at successor peers.