COMP3331 Assignment Report

In this assignment an implementation of a peer to peer (p2p) circular DHT protocol was created. Each peer in the network was required to be connected to the network at all times and each peer was required to perform the following:

- Ping its two successors at the end of every time interval
- Handle file queries and responses
- Transfer any required files that the peer holds to the requesting peer
- Safely handle peer churn in the network
- Detect and resolve abrupt peer departures from the network

Every message sent through the network was formatted into a string, this format was chosen as it was a quick and simple way of transferring information between peers. Due to the nature of the information being transferred and speed at which these messages were being transferred the main concern was efficiency. By using strings these messages could be structured quickly and easily by the sending peer and also translated easily by the receiving peer.

For each peer in the network, the following threads are initiated on startup to perform various actions:

- TCPServer Thread – this thread performs similarly to how a TCP server functions. This thread listens to the peer's port number for any new TCP connections at which point it forks a new thread (known as a serverHandler thread) specifically for the new connection.
- ServerHandler Thread – this thread performs the server actions of a peer connected by a TCP connection. This thread listens to the TCP messages received from its sender and dissects the message and perform the required actions.
- TCPClient Thread – this thread performs all client side actions of a TCP connection. Its main functionality is required to listen to the user input. This thread initiates any file request or peer departures.
- Client Thread – this thread sends the ping requests to its successors using UDP. It also determine if the corresponding ping responses were received. If a certain number of successive ping responses were not received it will initiate a new thread (KillPeer) that updates its successors.
- Server Thread – this thread listens to the UDP socket of its port number. Upon receiving any UDP messages the thread will dissect the message and determine the appropriate action

Other threads are established by the peer to perform certain functionalities.

- KillPeer thread – this thread is initiated by the peer when a peer has been detected to have ungracefully exited the network. Its main functionality is to update the peer's successor information.
- FileTransfer thread – this thread transfers the file packets and initiates the log file of the file holding peer. This thread will send datagram packets of a predefined size to the requesting peer until all packets are sent.

Along with these threads a peer object is passed between all the threads, this peer object holds all information about the peer, including its ID, its successors and predecessors, its max segment size and drop rate. This peer object is passed between all threads to ensure all the peer's information can be updated and obtained. This class also ensures that all information about the peer can be reached from one location. One concern that may arise through this implementation is the impact of concurrency. With multiple threads accessing the same class, it is appropriate to use synchronized methods and blocks to eliminate race conditions. This however is more of an issue for networks that are more active and require multiple packets to be sent at the same time.

Pinging Successors

Each peer in the network is required to send ping requests to its allocated successors in the circular network. The ping requests were sent using the UDP protocol where each request message is encapsulated in a datagram. The message was formatted as a string in the following structure:

Ping message = "Ping " + Peer ID

The structure of the following message was decided due to the UDP protocol used. Since UDP connections receive multiple datagrams from multiple sources, the initial word "Ping" was used to indicate to the receiving peer the purpose of the message. The peer ID was also included to notify the receiving peer of sender of the ping request but also more importantly determine the destination of its ping acknowledgements.

The time intervals between ping requests was set to 30 seconds, this was set to ensure that the time interval before timeout was wide enough to receive acknowledgements from its ping requests and not inadvertently timeout. More importantly this was to ensure that each peer sufficiently recognized its successors without the network being overloaded with ping requests and ping responses that may disrupt and delay other actions of the peer such as file requests and file transfers.

Ping requests also serve as an alternative use in the network in the form of determining a peer's predecessors. Each peer keeps a knowledge of its two predecessors through these ping requests. Predecessor information is updated every time a ping request is received to ensure the peer's predecessors are correct even in situations where peer churn or abrupt peer departures occur.

File Request

In this assignment each peer is able to request a file from other peers in the network. A peer sends a file request by sending a file request message to its successors. File requests are received initially through user input and is then sent through the network using TCP connections. The following message structure is used:

File Request Message = "request " + filename + (requesting Peer ID);

The initial "request" is to distinguish to the peers in the network the nature of the message received. The filename is assumed in this assignment to be a four digit number. Each peer translates the following filename into its corresponding hash value. A peer is determined to hold a file if its peer ID is the next closest ID after the file's hash value. This request message is passed to its successors until a peer is determined to hold the file. The requesting peer ID is added to the file request message so a direct TCP connection can be setup between the file holder and the requesting peer.

File Transfer

A new thread is initialized by the file holding peer to transfer the file to its requesting peer. This was done to ensure that the peer could continue listening to its TCP connection without any delay that may occur when transferring a file. File transfers was done through a UDP connection with the file being broken down into predefined file packets and sent using datagrams.

Each datagram packet contains file data up to the allocated maximum segment size as well as a header containing the file holder's ID, the current sequence number, the acknowledgement number and the size of the file data. The header of each packet was allocated 100 bytes, this may have been excessive but the excess bytes were used to account for the fluctuations in the header size but it also made it much more easier to separate the header and the file data.

To replicate a stop and wait protocol for reliable file transfer, a blocking queue was used to check if acknowledgements was received from the requesting peer. The following actions of listening to file packet acknowledgements and sending file packets were done on two separate threads, so a blocking queue was used to combine these two functionalities. When an file acknowledgement was received the peer would push it onto a blocking queue to be read by the other thread. The thread sending the file packets would poll the blocking queue until a file acknowledgement was pushed onto the blocking queue. At this point the thread would send the next packet to the requesting peer. The thread polling the blocking queue also has the possibility of timing out when no acknowledgement is pushed onto the blocking queue within one second. The timeout forces the thread to resend the previous data packet.

To replicate the dropping of packets, a drop rate is included in the implementation which is compared against a randomly generated value. When the randomly generated value is greater than the drop rate, the packet will not be sent and a timeout will occur.

To indicate a file has finished being sent, the file holder will send a finishing message containing "EOF" to the requesting peer to notify the peer that the file transfer has finished.

Peer Departure

A peer is able to depart from a network if it inputs into its terminal the string "quit". This will notify its two predecessors through a TCP connection that the peer is leaving the network. The structure of the message is as below:

Depart Peer message = "depart " + peer ID + new Successor

"depart" is used to indicate to the predecessor peers the intentions of the message. Also included in the message is the departing peer's ID as well as the ID of one of its successors. The ID of its successor depends on which predecessor the departing peer is sending the message to. The closest predecessor will receive information about the departing peer's second successor and the furthest predecessor of the two will receiving information about the departing peer's first successor.

On receiving the departing messages, the predecessors will update its successors will the new information received from the departing peer. The first predecessor will move its second successor to now be its first successor and its second successor is now the departing peer's second successor. The second predecessor will now allocate the departing peer's first successor as its second successor. Immediately after the changes to its successors are made, the peers will ping its new successors. This serves two purposes, first is to ensure that the peers pings the correct successors, second is to ensure that the new successors of each peer also updates its predecessor information to account for the departing peer.

Kill a Peer

A peer in the network can leave a network without notifying any peers in the network, this is known as leaving the network ungracefully. Peers in the network will be notified of a peer leaving the network ungracefully by the lack of ping responses its predecessors will receive from the departed peer. To distinguish if a peer has departed the network, number of ping requests not responded to has been set to 4. This value was chosen to ensure that the occasional missed ping request is not indicative of a departed peer. The value of 4 was also chosen as it was determined that any extra missed pings would not be of benefit in determining the status of a successor peer. The number 4 was based on the methods of the triple duplicate acknowledgements. Both scenarios suggest that the lack of results suggest that information has to be updated or changed. Given that ping requests are sent in 30 second intervals, peers will change successors due to an ungraceful exit at a minimum of 2 minutes after the peer has supposedly exited the network. For highly active and congested networks this may be inappropriate but given the functionality and simplicity of this network, the delay in updating the network may be sufficient due to the functionality of the network.

The third concern when developing an implementation of a peer's ungraceful exit is the time interval before timeout of the ping request occurs. In this assignment this is less of a concern since a ping request is never dropped in the network and is always received. Therefore ping responses are only dealt at the end of each ping request interval of 30 seconds. After the end of each interval, the responses are checked from a blocking queue to see if any ping response was sent. This is most likely sufficient for the purposes of this assignment as the ping responses are used only as a indication of a peer exiting ungracefully. In reality, timeouts on ping response are much shorter for much more active networks since peers can make numerous changes to the network in a very short amount of time and as a result require each peer to be continually updated with the changes.

To determine successive missed responses a count is implemented to count the number of successive ping responses. If the number of successive missed ping responses reaches 4 it will initiate a new TCP connection that will update its new successors. Once the peer updates its successors it will send ping requests to the two successors so that they can also update their predecessor information.

Assignment Notes

- Inside the folder is a run script, run the shell script to start the program.
- Enclosed in the folder is the resulting duplicate file and the two file logs. Delete these files to generate new logs and a new duplicate file.
- Testing was done in Vlab using port numbers other than 50000 because the following ports were always in use. If the same errors occur when marking this assignment please use port 5000 as these ports were used for testing.
- The following demonstration video was run using ports 5000.

YouTube Link to Demo – A demonstration of the project

https://youtu.be/BYO_4TSugmc