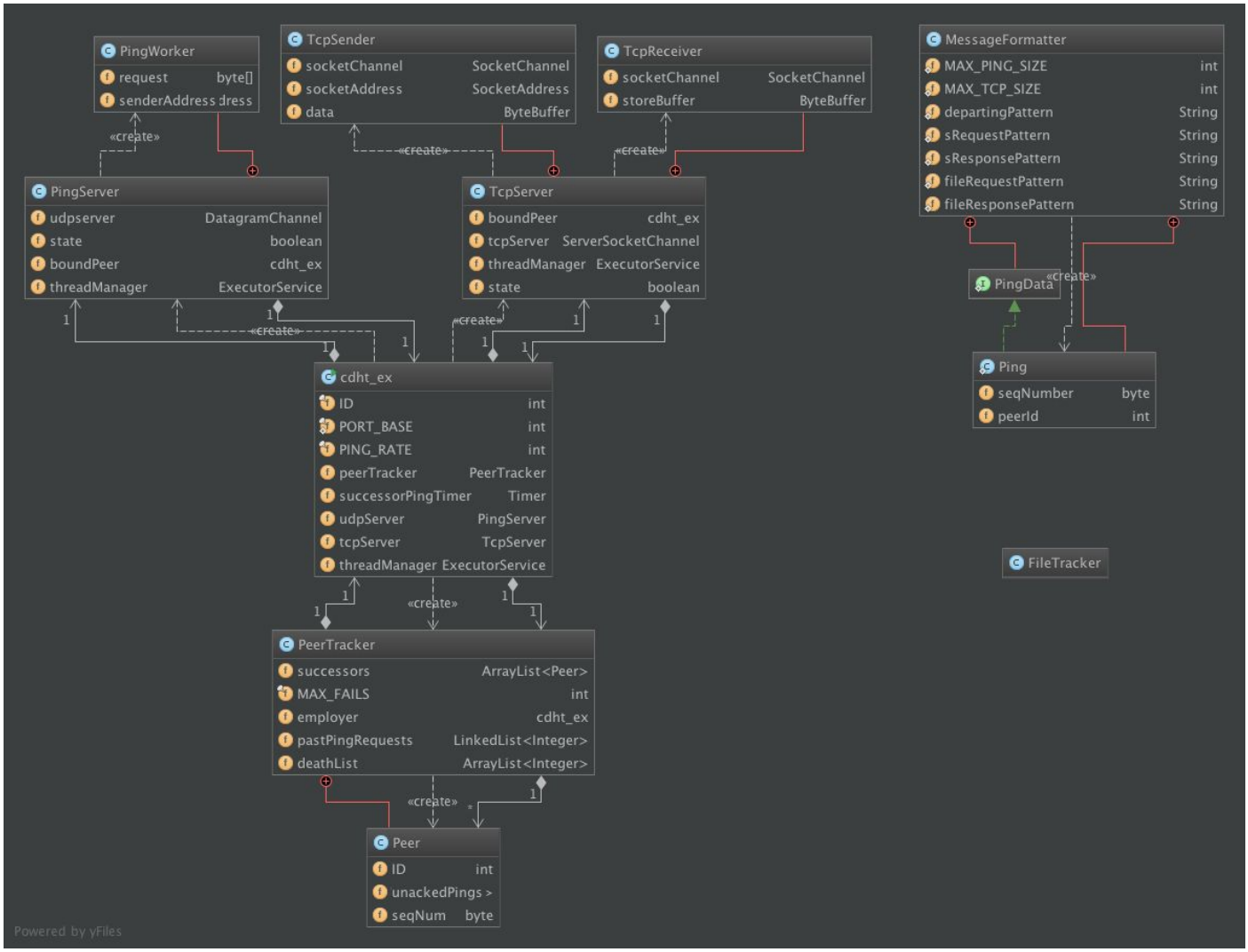
COMP3331
Circular DHT

By Adishwar Rishi - z5011984

Program Design



The program has 6 main classes; cdht_ex, PeerTracker, PingServer, TcpServer, MessageFormatter and FileTracker. Let me briefly describe each class

cdht_ex - This is the main class, and represents a peer in the cdht. it holds instances of the PingServer, TcpServer and PeerTracker. It also contains a threadpool which it uses to run both

server classes concurrently alongside itself. Once it starts all other code, this class reads input from the command line.

PeerTracker - The PeerTracker is a class that is bound to a certain cdht_ex class. It keeps track and manages the successors and predecessors of a cdht_ex class. It is responsible for sending out pings to make sure said successors are alive.

PingServer - This class holds the server code to receive and send Datagram packets. The class implements Runnable, which allows it to be run separately in a thread. The server itself is a multithreaded server, and has an inner class for handling incoming messages.

PingWorker - This is an inner class for the PingServer. The role of this class is to determine what the message type is, and then route it to the correct part of the program. E.g. when a ping request message is received, it tells the PeerTracker about the message, allowing the PeerTracker to update its list of past pings.

TcpServer - This class holds the server code that handles the sending and receiving of Tcp messages. As with the PingServer class, it implements Runnable, which allows it to run in a thread alongside the Tcp class. This class has two inner classes, TcpReceiver and TcpSender.

TcpReceiver - This is an inner class for the TcpServer. The role of this class is to take an incoming connection, read the data and then route it across the program based on the message type.

TcpSender - This is an inner class for the TcpServer. This class has the job of sending a new tcp message. It is put in its own thread so that the main server does not need to stop for the send to work.

How does the system work?

The working of the system can be explained in 3 main stages.

Initialisation - This is the startup phase for the system. This is managed by the cdht_ex class. During this phase, the cdht class starts the PingServer and the TcpServer classes. It passes them to a thread pool which then simulates simultaneous execution of the servers.
The cdht also sets the PeerTracker to be run every 1 second.
After setting up everything, the cdht class then starts listening to the command line for input.

Event Driven processing - This can be thought of as the running phase of the program. During this time all processing is done when one or another event is triggered. The main events are -

TCP or UDP message received, PeerTracker timer triggered (every 1 second) or command line input received.

Shutdown - This is the final stage of the program. It is triggered when a ‘quit’ is typed on the command line. This stage will trigger a tcp message that is sent to the peers predecessors. Then the servers and other threads are shutdown and the program will stop.

Extra : Death Detection - The extension part, detecting successor death is implemented in the following way. The PeerTracker analyses if a successor has failed a ping. The rule is - if a ping a ping message has not been acked by the time the next message is ready to send, the ping is declared to be unsuccessful. If 4 such pings are detected, the successor is said to be dead. Now the system sends pings every 1 second, hence it will take 4 seconds for a peer to realise if its successor is dead. If such a death is detected, the appropriate action is taken.

Message Design

The message design for UDP and TCP is drastically different. The UDP design is small and efficient, and the Tcp design is made for ease of development.

UDP Design -

- Ping request - R[0-128)
    - A ping request is a capital ‘R’ followed by a number in the rage 0 to 127
- Ping Response - r[0-128)
    - A ping response is a lowercase ‘r’ followed by a number inthe range 0 to 127.

Both udp messages are 2 bytes long. The number following the character is the sequence number. It wraps at 128. This is because java only supports signed bytes and hence a byte value ranges from -128 to 127.

TCP Design -

- File request message - “FR:[0-255][0-9999]”
- File response message - “Fr:[0-255],[0-9999],[0|1],[0-255]
- Graceful quit message - “D:[0-255],[0-255],[0-255]

TCP messages are encoded as strings, which means the longest message (file response) is 14 bytes long.

Note - I know that the file response message has a lot of useless information. I made this design with extensibility in mind.

Failure Cases

The program should work in all cases defined in the spec. However I will mention here the different things this program does not handle.

1. If two peers are forced quit one after another (before the cdht is able to mend itself), then the system will break.
2. If two peers are left and one is made to force quit. Then the last remaining peer will throw an exception

Design Tradeoffs

One main design tradeoff made was increasing the length of the tcp message. As mentioned, the TCP messages are all encoded as strings, greatly increasing their length. This tradeoff was made for ease of development, since strings are much easier to work with then bitstrings.

Another design tradeoff made was to make ping messages run every 1 second. This makes the command line interface practically unusable, but I felt that this was necessary for death detection to occur at a reasonable rate (4 seconds).

Possible improvements and extensions

There are two main ways to improve the current code. Both are performance improvements.

One is to combine both tcp and udp servers into one thread. This can be done quite easily, since both servers are already configured to be non-blocking. One just needs to move the code into one loop. This would decrease the burden on the process, since it will need to handle one less loop.

The second is to compress the size of a tcp message. Currently it is encoded in strings, but a future version could use bitstring encoding. This would greatly decrease the size of the message, and in turn improve performance.

Extensions to this program will be actually holding files, and passing them around the cdht, this can be done through tcp itself. If the file requested exists, then you can send the file through tcp. The receiving peer can reconstruct the file.