Peer-to-Peer Assignment, Report

by Stratton Sloane

Implemented in: Python 3

Peer ping

My implementation first began with testing of basic ping messages between two peers. These were called ‘check-alive’ and ‘is-alive’ and were sent as text messages with the peer name appended to the end. I then begun work on a class named ‘PeerManager’, which would automate the ping process, along with other peer-related functions in the future.

It was during the creation of PeerManager that I realised that handling the ping requests and responses sequentially would be difficult, so I started work on a ‘Listener’ class. This class would run a listener function in its own thread, processing requests that came through and storing them in a queue, which the PeerManager class would consume. Around this time I created the ‘MessageSender’ class, in order to centralise and simplify the sending of messages to other peers.

The ‘ping_successors’ function in PeerManager started by sending a ‘check-alive’ message to each successor, then waiting in a loop for ‘is-alive’ responses until either the timeout (1 second) was reached or all successors has responded. If the timeout was reached before all successors had responded, it would restart and try again, up to a maximum of 3 attempts. Unresponsive peers after 3 attempts were considered ‘dead’.

File transfer

The ‘FileManager’ class handles file request and transfer. Here it became clear that the text-based message was inadequate for transferring segments, so I created a dedicated ‘Message’ class. This is packed into a byte stream to send to other peers, and unpacked into an object on the other end. Each message has a type (a MessageType enum), source peer, destination peer, and other headers, followed by file data of variable length. This format allowed easy creation of new message types and headers and simplified handling.

File request handling was the first thing to be implemented. The peer requesting the file creates a new message of the type ‘FILE_REQUEST’, storing the name of the requested file in the ‘requested_file’ header and their own identity in the ‘requested_peer’ header. This message is forwarded using TCP to the first successor, who checks whether they have the file (through the hash function) and forwards it if not.

When a peer identifies that they have the requested file, they send a ‘HAS_FILE’ message using TCP to the peer identified in the ‘requested_peer’ header. After this it immediately starts transferring segments, sending a ‘PACKET_TRN’ message with ‘seq_number’ and ‘segment_size’ headers, followed by the segment data stored under the ‘file_data’ field. It awaits for a ‘PACKET_RCV’ message from the peer, with a timeout of 1 second. If not response is received, the message is re-transmitted, else a further part of the file is read into a new message and the sequence number advanced.

When a packet is dropped ‘randomly’, no message is sent but the rest of the process proceeds as normal.

Peer departure

A departing peer calls the ‘depart’ function of PeerManager, which sends a ‘DEPART’ message to each of the peers predecessors. This message has a ‘successors’ header that contains the two successors of the departing peer.

On arrival of a ‘DEPART’ message, each peer replaces the departing peer with one of the two successors (depending on the position of the departing), and pings them.

Sending messages via TCP and UDP requires two listeners in the Listener class, however received messages are handled as normal so no changes were needed. TCP transmission can be toggled on and off with an argument to the ‘send_message’ function.

Peer death

The sudden departure of a peer is determined by the ‘ping_successor’ function of PeerManager, which runs every 10 seconds. After 3 attempts to contact a peer, they are considered ‘dead’ and the process to replace them begins. Using the ‘get_successors’ function, which sends a ‘SUCCESSORS_REQ’ message to a given peer to retrieve its successors, the successors of the immediate following peer are determined, and if required, also the successors of the following peers’ first successor.

Message format

m_type – Message type, one of:

- PING_REQ
- PING_RESP
- FILE_REQUEST
- HAS_FILE
- PACKET_TRN
- PACKET_RCV
- DEPART
- SUCCESSORS_REQ
- SUCCESSORS_RESP

source – Source peer, 8 bits
destination – Destination peer, 8 bits
successors – Successors of peer, 16 bits
requested_peer – Peer that requested file, 8 bits
requested_file – File name, 16 bits
seq_number – Sequence number, 32 bits
segment_size – Segment size, 32 bits
file_data – Segment data, variable size (corresponds to segment size)