18-03-2020

Chat Client Unreliable

Summary

A chat client has been developed using an unreliable protocol called UDP. Whilst an unreliable protocol is used, we do want reliable transport. This involves guaranteed delivery of messages, error protection and in-order message delivery.

Solutions

Below are our solutions.

Guaranteed Message delivery

For guaranteed message delivery we use a form of sliding window protocol. Both the sender and receiver keep a list about each other. This is done after the sender sends a message labeled with INITIALIZE to the receiver. The sender then already constructs the list, and the receiver will know that it has to do the same for the sender. However, only the sender buffers the messages that have yet to be sent. The receiver only saves what sequence number it expects and discards any packet that is not that expected sequence number. B keeps retransmitting the packets till it gets an acknowledgement for each packet. This way the message delivery is guaranteed.

Error Protection

CRC-32 has been used for error detection. When an error is detected using CRC the message is retransmitted as long there is no error. The CRC-32 polynomial is used for XOR division and when the result is zero there is no error and there is an error when it is not. It will then be retransmitted accordingly. The code basically performs the known CRC-check but not directly on the message itself but only the part that is received at the end user. That means that the sender would send SEND receiver_name message, but what we do is create a ‘mock’ message, since we know our protocol so we know how the receiver will receive our message and calculate the CRC-check on: DELIVERY sender_name message. We then send this along with the message and use it to check whether the message is correct or not. However, for some reason the check seems to be unstable sometimes. Due to time constraints, we have not been able to find out why. We believe it might be because we are processing a string.

In-order Message Delivery

To test our solution, we forced a delay from 5 to 10 seconds. A sliding window protocol has been used for this. We start with an initial sequence number of A0 which increments with every new message. This goes to a maximum number of 7, we did this to keep it in one byte. We then start from 0 again but in B0 instead of A because if it gets added in A the eighth message becomes the earliest message.

When there is a new sender the INITIALIZE state is sent within the message so that the server knows there is a new client. Imagine client A sends a message to client B. UDP is not connection-oriented so client B initially does not know that it is going to receive from client A. When the INITIALIZE state is sent to B it knows to start expecting specific sequence numbers, so it knows the order to receive it. Client A prepares a buffer for B and client B prepares a buffer for A. Any following messages will hold the NEW_PACKET state, meaning a buffer is already in place in both clients who then fix the in-order delivery. Any message that does not have the expected sequence number will be discarded until the expected message arrives.