Reliable Data Transfer Protocol

Anushree Sitaram Das

To implement a reliable data transfer protocol between a client and a server, in which the packets are tunneled (from the network layer up) within UDP packets for the simulation, the following rules can be used:

- After a client sends a packet, the server responds with an acknowledgment(ACK).
- If the packet gets lost, the client must have a timeout before which it waits for an acknowledgment from the server and then retransmit that packet.
- If the packet gets corrupted, the server sends back a negative acknowledgment. The client then retransmits the packet.

To implement the above rules, in this project the client wait in an infinite loop till it receives an ACK. Since a UDP packet doesn't have an acknowledgment field, the server returns the checksum itself as data. The client checks if the received checksum matches the checksum it had sent for the corresponding packet. If the checksum received is different then the response from the server is considered as a negative acknowledgment(NACK). The algorithm used for error detection is checksum. To calculate checksum, all the data bits are summed up and the sum's 1's compliment is calculated, which is the final checksum.

This protocol is more efficient than TCP for the current problem because of the following reasons:

- This protocol doesn't need to follow a three-way handshake to setup a connection like it is required for a TCP connection. As a result, the number of packets needed is reduced.
- Since, UDP packets are used to exchange data, the size of the packet is reduced drastically.
- A congestion control mechanism is not necessary as one packet is sent at time. The next packet is sent only after the client receives a positive acknowledgement from the server.
- A sequence number that can be used to reorder arrived packets at the receiver in TCP is required in this protocol as the next packet is sent

only after the client receives a positive acknowledgement from the server.

In conclusion, the resulting protocol uses in smaller packets and fewer packets to transfer data from client to server, than a corresponding TCP/IP implementation must have used.