Networks Assignment 2, Phase 1

Group 1: Rudrajit Kargupta - 2017B3A70452H, Raunak Mantri - 2017B5A71340H, Jalaj Bansal - 2017B3A71610H, Manisha Katariya - 2017B3A70354H, Aditya Agarwal - 2017B1A71075H

Group 2: Prateek Rai Verma - 2017A7PS0213H, Sarwadnya Mutkule - 2017B3A70450H, Pranav Sai Itapu - 2017B3A70373H, Godhala Meganaa - 2017B3A70973H, Malladi Spoorthi Siri - 2017B4A70580H, R Pratik Kamdar - 2017B3A70976H

In this document we describe a reliable Application layer middle-ware protocol which works using UDP as the underlying transport layer protocol and adds sufficient/desired reliability features on top of UDP in the Application Layer. As we describe this protocol, we set out a few specifications as well, which are described in the following section.

Specifications

- The client and server can maintain a single connection at a given point in time, any new requests while a connection is already in place are discarded.
- Connections are persistent, the same connection can be used to send one multiple files.
- The window size is set to be 5. That is, at a time the send or receive buffer contains 5 packets. This window is crucial in implementing the Selective Repeat Paradigm.
- Sequence numbers range from 0-9 to avoid packets being recognized incorrectly in cases of retransmission.
- Protocol will handle lossy networks and corrupt packets by buffering, ERROR packets (defined later), retransmission, delivering only in-order packets to application.
- The application layer adds the required headers to equi-sized data chunks and makes a packet that is sent using UDP, thus, UDP is just involved in the transfer of packets while the application layer deals with reliability.

Application Layer

When the application program requests file transfer, the application layer breaks the data to be sent(client)/received(server) into chunks and then encapsulates them into packets along with certain headers and hands them over to the transport layer. These headers are required for defining the type of the packet, routing the packets, ordering the packets. This layer also checks the correctness of the data transfer by the means of a checksum (1's complement sum based).

General Packet Structure

<table>
<tr><td colspan="2">Datagram Header</td><td>Opcode</td></tr>
<tr><td>Transfer ID (TID)</td><td>Sequence Number</td><td>Filename</td></tr>
<tr><td colspan="3">Data Body</td></tr>
</table>

- **Datagram Header:** This header consists of 4 components: the Source Port, Destination Port, Length and Checksum. The source and the destination port signifies the port numbers of the sender and the receiver respectively. Checksum is used to validate the correctness of the packet after it is transferred. The Length field specifies the size of the

UDP data and UDP header and Each of the 4 components of the Datagram Header are 2 bytes in size, a total of 8 bytes.

Source Port	Destination Port
Length	Checksum

- **Opcode:** The opcode signifies the type of packet being transferred. It is 1 byte in size. The opcode is discussed at length in the following section.
- **Transfer ID:** The Transfer ID is passed to the datagram layer which has to be used as the port numbers. A connection is uniquely identified by a pair of TIDs, that of the sender's port and the receiver's port. The TIDs are 1 byte in size.
- **Sequence Number:** Each of the packets in the buffer are assigned a sequence number according to the order in which packets are sent and the packet's position in the buffer. Sequence Numbers are 2 bytes in size.
- **Filename:** This field contains the filename being uploaded (or downloaded), it is 10 bytes in size.
- **Data Body:** The Data body contains the data chunk (which can also contain error/exception details) being sent in the packet. It is 256 bytes in size.

Total packet size: 8 + 1 + 1 + 2 + 10 + 256 = 278 bytes

Type of Packets
There are seven different types of packets that may be transferred over this protocol. These packet types are identified by the opcode -

Opcode	Packet Type
0	Error Packet
1	File Upload Request (Send)
2	File Download Request (Receive)
3	Acknowledgement (ACK)
4	Regular Data Transfer Packet
5	Connections Establishment Request
6	Connection Termination Request

0. Error Packet: The events which may trigger an error packet include: receiving a packet which is ill-formed (corrupt), the request cannot be fulfilled by the server etc. The details of the exceptions (type of error, negative acknowledgement for which sequence number etc.) are specified in the data part of the packet.
1. File Upload Request: Before the client initiates the process of sending data packets to the server for uploading, it sends this packet to the server as a request packet.
2. File Download Request: Sent from client to server to download a particular file that may exist on the server, so that the server knows which file to send (if it exists) and initiate the process of sending a copy of the same to the client.

3. Acknowledgement Packet (ACK): An ACK packet is an acknowledgement from the server/client for a previous request/packet. The sequence number in an ACK echoes the sequence number of the data/request packet being acknowledged.
4. Regular Data Transfer Packet: It contains a data chunk of the file being sent over the network and this packet must be placed in the buffer and arranged properly before it is appended to the download destination of the file.
5. Connection Establishment Request: This packet is used precisely for the purpose of establishing a connection. Once this packet is sent, it results in a 3-way handshake (so as to facilitate the full duplex connection between the client and the server).
6. Connection Termination Request: This packet can be sent by either of the two parties in a connection, sending this packet triggers a connection termination process which severs the connection between the two parties (client and server).

General scenario for file transfer
When the client wants to send a file (upload) to the server the following sequence of events take place (a similar process is followed when the client makes a download request to the server).

Step 1: The client sends a connection establishment request to the server at port 420. This packet also contains the TID of the port from which the client will be sending its data packets.
Step 2: The server acknowledges this request with an ACK packet and returns the TID on which it'll be listening to. Therefore, after this step, the server and client know each other's TIDs.
Step 3: The client acknowledges the receipt of the ACK in the previous step by sending a File upload request packet, informing the server to expect a data packet after this.

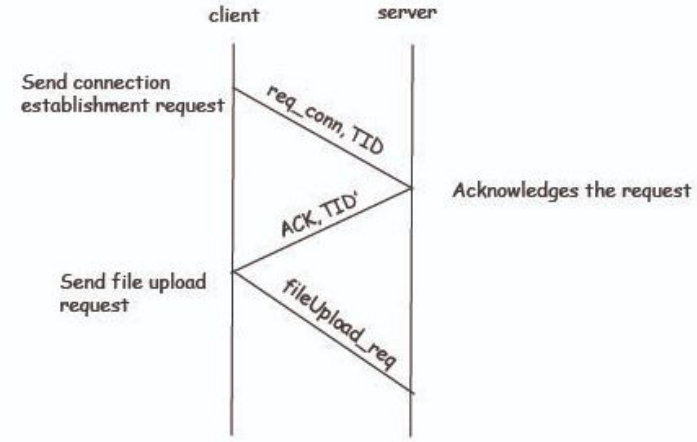

Figure 1: Connection Establishment Mechanism

Step 4: The application layer breaks the file into data chunks. These data chunks are then combined with the headers to form a packet. Packets are placed in a buffer of size 5 packets. Each packet in the buffer is assigned a sequence number depending on position in the buffer. Packet with the lowest sequence number is sent first. As soon as a packet is sent, a timer for that packet is started (to account for the timeout and retransmission). A timer is run for each packet. When the lowest sequence number packet is ACKed, it is removed from the buffer and the next chunk is appended to the end of the buffer, thus moving the "window" forward.
Step 5: The server acknowledges the receipt of the packet with an ACK packet with appropriate sequence number, provided the packet received is not corrupt or ill-formed or out of order (the sequence number of the packet must lie between the range of the sequence numbers at the server's end). These irregularities are handled by sending an ERROR packet. A buffer is maintained at the server's end and it has a range associated with it, which is the range of the

sequence number of packets allowed in the buffer at a time (in a way similar to receiver's end). If the packet is in-order then it is written to the file and the receiver window moves forward otherwise data remains in the buffer. The situation is depicted in figures 2 and 3 below.

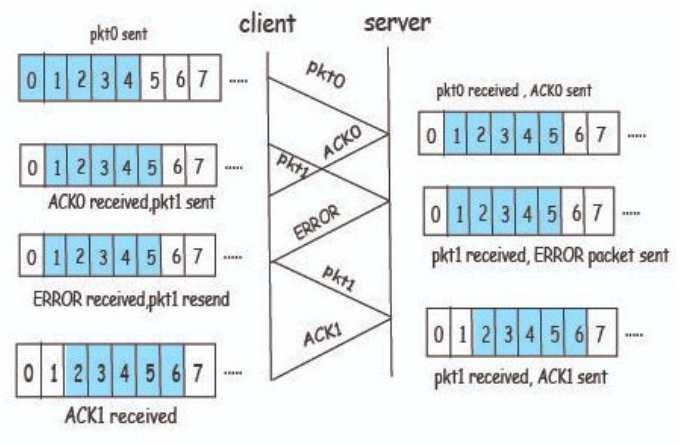

Figure 2: In case the server receives a corrupt packet requests for retransmission by using an error packet.

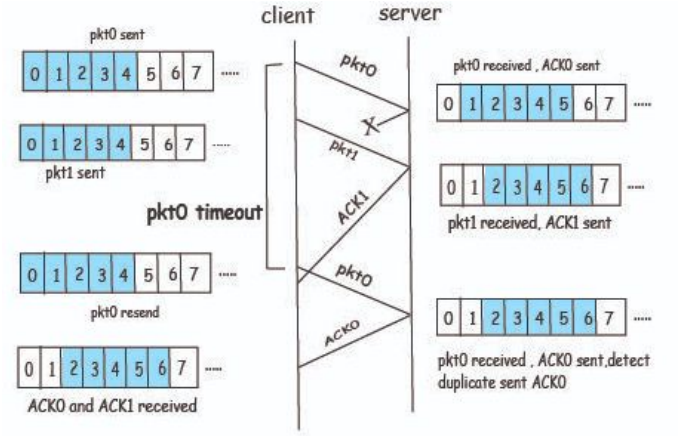

Figure 3: When the server receives duplicate packets it retransmits the previously sent ACK.

The last couple of steps describe the **selective repeat protocol** in action. We keep the windows size at both sides equal to 5 but the sequence numbers are from 0-9 to avoid packets being recognized incorrectly during retransmission. The process is depicted in figures 4 and 5.

Retransmissions

Step 6: In case the client receives an ERROR packet or the packet timer runs out (ACK lost), the corresponding packet is retransmitted. In case the client receives triple duplicate ACKS, then it does not wait for timeout and retransmits the next corresponding packet (according to sequence number received in the duplicate ACK packets). On the server side, if no packets are received for a (very) long time due to packets being lost, the server can terminate the connection but this is a very rare scenario where the network is down.

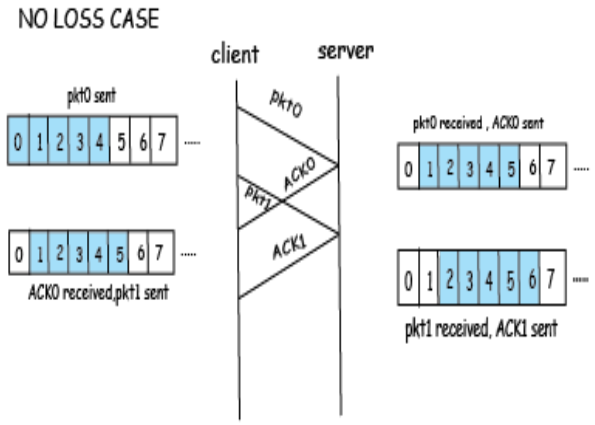

Figure 4: Selective repeat mechanism.

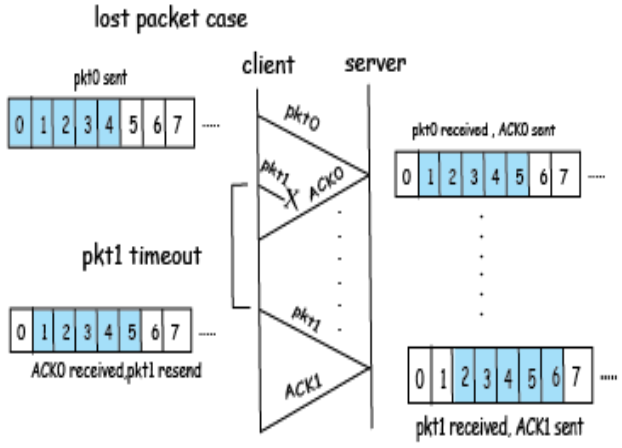

Figure 5:. Retransmission scenario in case of lost packet i.e client does not receive an ACK and therefore retransmits after timeout.

Connection Termination

Once a packet with data length less than 256 bytes is received by the server it signifies the end of the file transfer from client's end, which means that the server can empty its buffer, append it and compile the file (this is followed by a correctness check using a checksum and the server can reply with an ERROR packet for re-upload). However, even after the successful completion of the transfer, the server does not terminate the connection from its end and waits for receiving a new file upload/download request (persistent).

Step 7: The client may choose to terminate the connection by sending a connection termination request. Once this packet is received at the server, the server frees up all its resources, sends an ACK to this packet.
Step 8: As soon as the client receives the ACK from the server, it frees up all its resources and sends another ACK to the server, which then terminates the connection upon receiving this ACK. Please note, in case the last pair of ACKs get lost in the network and are not delivered, both the client and servers timeout and close their connection and free their resources anyway.

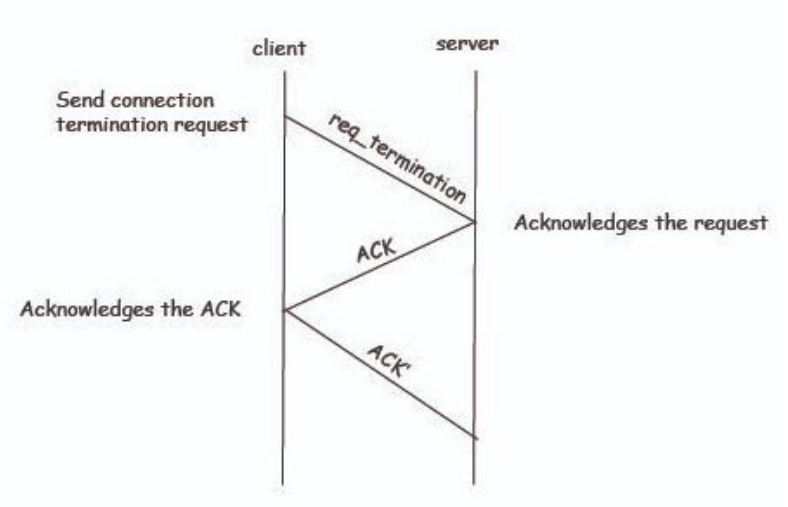

Figure 6: Connection Termination Mechanism