Reliable Transfer Protocol

CS 3251

HW 4

Dan Fincher
Drew Ritter

*Changes to original design are highlighted

~~Anything taken out has strikethrough~~

Description of RTP

The internet network layer provides a best effort service with no guarantee that packets arrive at their destination. Also, since each packet is routed individually, it is possible that packets are received out of order. It is necessary to have a reliable data transfer protocol to ensure delivery of all packets and their order to the application layer. A Reliable Transfer Protocol (RTP) guarantees the delivery of packets to a destination as well as the correct packet order by handling delayed, lost, corrupted, duplicate, and out of order packets.

Our Implementation

Our RTP is connection oriented. It utilizes a 3-way handshake to ensure that both client and server are connected, like a TCP connection.

Our RTP will utilize pipelining behavior, specifically implementing Go-Back-N. With pipelining behavior, the RTP can send multiple packets at once. This increases efficiency and only when a packet is lost or corrupted that all packets within a window must be resent. With window-based flow control that Go-Back-N provides, the transmission of data can be managed so that the receiver does not get overwhelmed with data.

A lost packet will be handled as a lack of acknowledgement from the receiver. We will have our protocol wait for a 0.5 seconds to receive multiple ACK signals from a window, and after that time it will resend all packets within a window.

To detect corrupted packets, a checksum will be implemented. After a checksum is calculated, and if it does not equal the checksum from the packet's header, the receiver will not send an ACK which will cause a timeout. ~~then a NACK packet will be sent to the sender so that all packets within a window are resent.~~ (Took out NACK flag)

For duplicated packets that are received, they will simply be ignored.

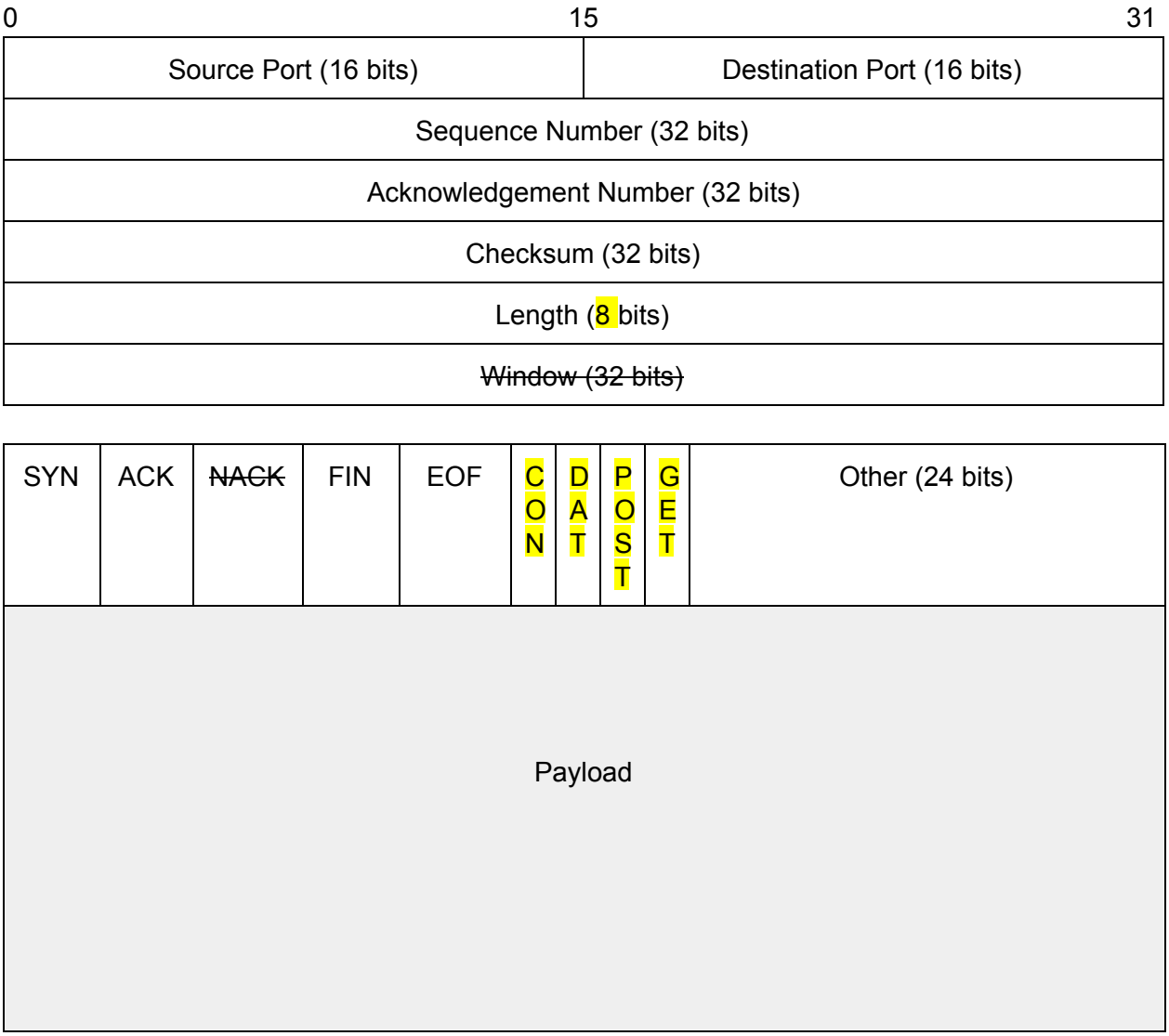
Our protocol will be able to handle bidirectional data transfers between the two connected hosts. The packets (including the acknowledgements) sent from Host A to Host B will be no different than those packets sent from Host B to Host A. Both A and B will be sending and receiving data at the same time.

We implement CRC32 for the checksum.

Packet Header Structure

Source Port - this is the 16-bit port number for the sender of the data
Destination Port - this is the 16-bit port number for the receiver of the data
Sequence Number - 32-bit sequence number to indicate order packet was sent
Acknowledgement Number - 32-bit sequence number to indicate packet was received with number corresponding to sequence number
Checksum - 32-bit checksum to verify that packet is not corrupted
Length - 8-bit field for length of the message in bytes
~~**Window - Indicates the size of the window that denotes how much data the receiver can get from the sender**~~
SYN Flag - If this bit is set, then the packet is a SYN packet. This tells the receiver that a sender wants to establish a connection
ACK Flag - If this bit is set, then the packet is an ACK packet. This tells the sender it successfully received a corresponding packet
~~**NACK Flag - If this bit is set, then the packet is an NACK packet. This tells the sender that a**~~ packet is corrupted
FIN Flag - If this bit is set, then one of the endpoints wants to end the connection
EOF Flag - If this bit is set, then the receiver knows that the packet is the final packet in the message and will return the data to the application.
CON Flag - If this bit is set, then the packet is for establishing a connection
DAT Flag - If this bit is set, then the packet is a data packet
POST Flag - If this bit is set, then the client is going to send a file to the server
GET Flag - If this bit is set, then the server is going to send a file to the client
Other - Left there in case others want to implement different features

Packet Header



Description of API

~~**RTPSocket createRTPSocket()**~~

~~Creates an RTP socket and returns it so that the program can send and receive messages.~~

~~**void listenRTP()**~~

void listenForPackets()

Invokes the socket to allow connections from clients to be made.

~~**void bindRTPSocket(InetAddress ip, int portNum)**~~

~~Hooks the RTPSocket up with the proper IP address and port number.~~

~~**booleon connectRTP()**~~

void startConnection()

Method called by the receiving host that will perform the handshake procedure to form a connection. It then returns true if the RTP socket can now effectively communicate.

~~**void closeRTPSocket()**~~

void endConnection()

Ends the connection made between the destination and source hosts by closing and destroying the socket.

~~**String receive(int numBytes)**~~

void receivedDataHeader(RTPPacketHeader receivedHeader)

This function will be responsible for returning the data in the correct order to the application. This function will guarantee the data received from the socket is not corrupt through the use of the checksum field within the RTP header.

void receivedConnectionHeader(RTPPacketHeader receivedHeader)

This function handles connection packets.

void receivedGetHeader(RTPPacketHeader receivedHeader)

This function handles packets with get flag raised

void receivedPostHeader(RTPPacketHeader receivedHeader)

This function handles packets with post flag raised

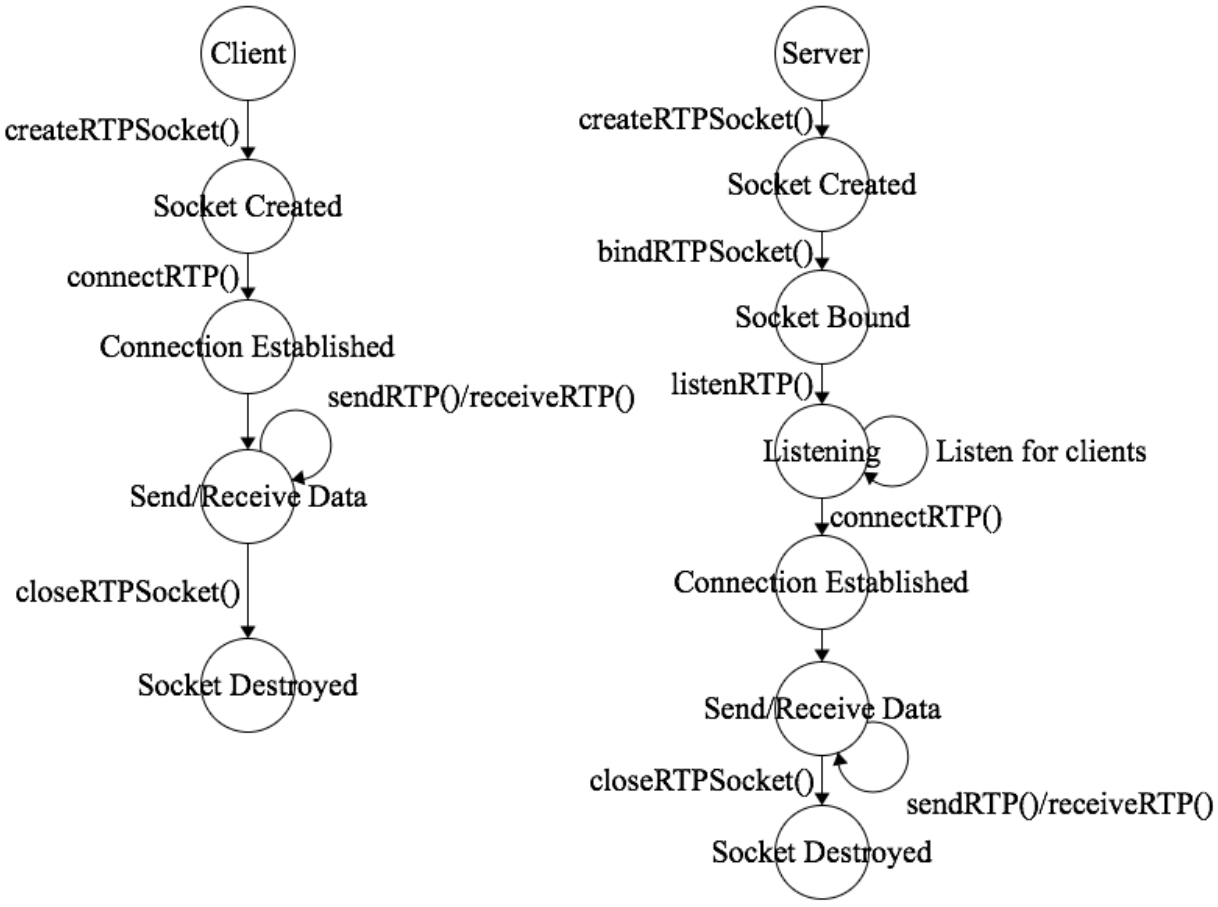
~~**void sendRTP(bytebuffer message)**~~

void sendData(String fileName)

This function will send the message in the parameter through the socket to the destination host with which the source host has formed a connection. For this to work properly, the source and destination hosts have to be connected and have a common open socket. The

sendRTP function guarantees that the message will be sent to the destination. It is also responsible for forming the RTP header in which the message will be encapsulated.

Finite State Machine Diagrams



Further Descriptions

Corrupt Packets

We will perform a checksum on the packet. If the packet does not pass the checksum, then an ACK is not sent back and the sender will resend the packet after timeout.

Lost Packets

We will use a timeout period of 0.5 seconds. If an acknowledgement is not received in that period, then we will resend all packets within the window.

Out of Order Packets

If we receive a packet with a sequence number ahead of the receiver's window size, we will ignore it. The sender will not receive an acknowledgement and will resend later.

Duplicate Packets

Duplicate packets will be ignored by the receiver and will not be saved. An acknowledgement will be sent back to the sender so that the sender's window can adjust its position.

Bi-Directional Transfer

The packet structure will be identical for the two directions of data transfer. A packet going from host A can be sent to host B alongside an acknowledgement for a previous packet from host B to host A.