RTP: Reliable Transfer Protocol

Sagar Laud and Samarth Agarwal

Protocol Specifications:

1. The RTP protocol outlined in this design is a pipelined protocol. It will send as many packets as specified in the window size.
2. The protocol will handle lost packets. If packets are lost from the sender or receiver, there is a timeout feature which will result in a retransmission from the sender.
3. The protocol will handle corrupted packets. Each RTP header will contain a checksum computed by the algorithm. The sender will compute the checksum and send the packet. Upon receiving it, the receiver will also compute the checksum and verify that the one included in the packet matches the one that was computed. If it matches, an ACK is sent to the sender. If not, it is treated as not being received.
4. The protocol can indeed handle duplicate packets. This is handled by keeping track of sequence numbers from the sender. Even with two identical packets arriving, only one will be processed at a time and as a result only one will be in the proper order.
5. The protocol will handle out-of-order packets. As with duplicates, this will be handled by using sequence numbers. The protocol will keep track of the expected sequence number and will only add the data to the message if it arrives in the proper order. In this manner, the protocol uses a Go-Back-N approach.
6. The protocol will support a bi-directional data transfer. This is achieved by allowing the sender and receiver to be interchangeable for the two entities during the connection. By combining this ability with the feature to piggyback data back with an ACK, the protocol allows data to be sent in both directions with both directions receiving ACKS.
7. The checksum algorithm implemented by this protocol is more complicated than the basic IP checksum algorithm. The algorithm implemented by the protocol is called Fletcher’s checksum algorithm which aims to provide error detection similar to cyclic redundancy check, but with lower computational effort. This algorithm (implementation details provided at the end) allows for easy detection of corruption.
8. The protocol that we have implemented can handle corruption (and also does so using Fletcher’s Checksum), duplicates, out of order packets, and dropped packets. It does all of these simultaneously. For these we feel that we deserve some extra credit as while these were all on the rubric, they are all handled simultaneously and seamlessly. Fletcher’s Checksum was particularly interesting and is far better than the IP checksum as well.

Introduction:

The primary purpose of our RTP connection is to provide a reliable connection service between two endpoints. In order to provide this service, the protocol provides functionality in the following areas:

1. Basic Data Transfer
2. Reliability
3. Flow Control
4. Connections
5. Multiplexing

1. Basic Data Transfer

   RTP will transfer a continuous stream of octets in either direction between end points by packaging a fixed number of octets into segments, called packets, for transmission through the network. The sending user will specify a receiving user to assure that the data is transmitted. The RTPs will forward and deliver the data that is ready up to a point to the receiver through the send function defined in the programming interface.

2. Reliability

   In order to remain a reliable service, the protocol must be able to recover from data that is corrupted, delivered out of order, lost, or delivered multiple times. Each of these issues will be addressed individually. Each window of packets which is sent contains a sequence number (which determines the order it has in the overall assembled message) for each packet and waits for a positive acknowledgement (ACK) with the acknowledgement number being the next sequence number that the receiver expects. If an ACK is received with the acknowledgment number being the last packet in the window + 1, the sender can send an entirely new window of packets to the receiver. If not, the sender will go back to the last packet that the receiver received correctly + 1 (in order and uncorrupted) and send a window of packets starting at that packet. In the situation where an ACK is lost on the way back from the receiver, or the packets from sender are lost on the way to the receiver, a retransmit timer on the sender's end waits a specified amount of time for a response. If this response does not arrive in the given time, the packets are sent again. In the event of a duplicate packet (where the receiver has received the packet but the sender has not yet received an ACK), the packet does not need to be acknowledged as the receiver would have already acknowledged it the first time it arrived in the window.

3. Flow Control

   The protocol must provide a way for the receiver to manage the amount of data sent by the sender. The receiver will send a window size (specifically) a number of bytes that it can currently handle. The sender in turn will determine what is less: the window size, or the current number of packets that can be accepted, and will send the minimum of these two. The window size will be the acceptable number of octets that the sender may transmit.

4. Connections

   In order to maintain reliability and flow control, a connection must be established between the sender and receiver to maintain the status of the transmission. With the combination of sockets, window sizes, and sequence numbers, we define a connection. There can exist one and only one connection between a given pair of sockets at a given point in time. When the two sides of the transmission wish to communicate, they must establish a connection by exchanging the information that comprises a connection and initializing the necessary status variables. To do this, the protocol will use a three-way handshake. The handshake utilizes sequence numbers to avoid establishing an erroneous connection. Once this is done properly, the sender and receiver are free to communicate freely. Eventually, one of the entities involved in the connection will wish to terminate the connection. The connection is then terminated and the resources are released to be used by other processes.

5. Multiplexing:

   For the sake of allowing multiple RTP connections to exist at one given time on a host, RTP provides a set of ports within the host with each capable of being part of a connection. Each port combined with a network and host address represents an entity called a socket. Each pair of sockets in a connection serves as a unique identifier for that connection. A socket will bind to a port to serve its purposes. This protocol will take information from the network layer and multiplex it based upon the ports and addresses involved. This will map to a socket which in turn will send the information to the application layer.

High-Level Description

The protocol uses the functionality mentioned above in order to establish and manage the connection between two hosts in the network. After a connection is established, data can be sent and received. In order to send data to the receiver, data is broken up into packets containing a fixed number of octets. Data will be delivered reliably as the protocol ensures reliability through ACK packets. The protocol will be vary of overflowing the receivers packet buffer through the use of flow control. The connection is closed when it is no longer needed.

RTP Header Structure

Ports: The header will be comprised of several portions. It will contain a 16 bit source port number and a 16 bit destination port number. Both of these ports are self-explanatory and correspond to ports on both the source and destination ends.

Sequence Number and Acknowledgement Number:
The header will contain two 32 bit values, the sequence number and the acknowledgement field. The sequence number is used to determine the order of the packet. The earliest available sequence number will actually be 1 more than the generated number as the first number will be consumed in the handshake to setup the connection (using SYNs and ACKS). Similarly, tearing down the connection using FIN and ACK will also consume a sequence number. With the exception of these scenarios however, the sequence number will be sent with the packet and incremented for every next packet (not a retransmission) that is sent. The ACK field will simply echo the next sequence number that it expects.

Control Bits:
There will be five control bits present in the header as well: SYN, ACK, SYNC, FIN, LAST. These will designate the type that the packet is. However, if none of these control bits are set, then the packet is simply a normal data packet. It is worth noting that even with any of these bits set with the exception of SYN and FIN, the packet can still carry data with it.

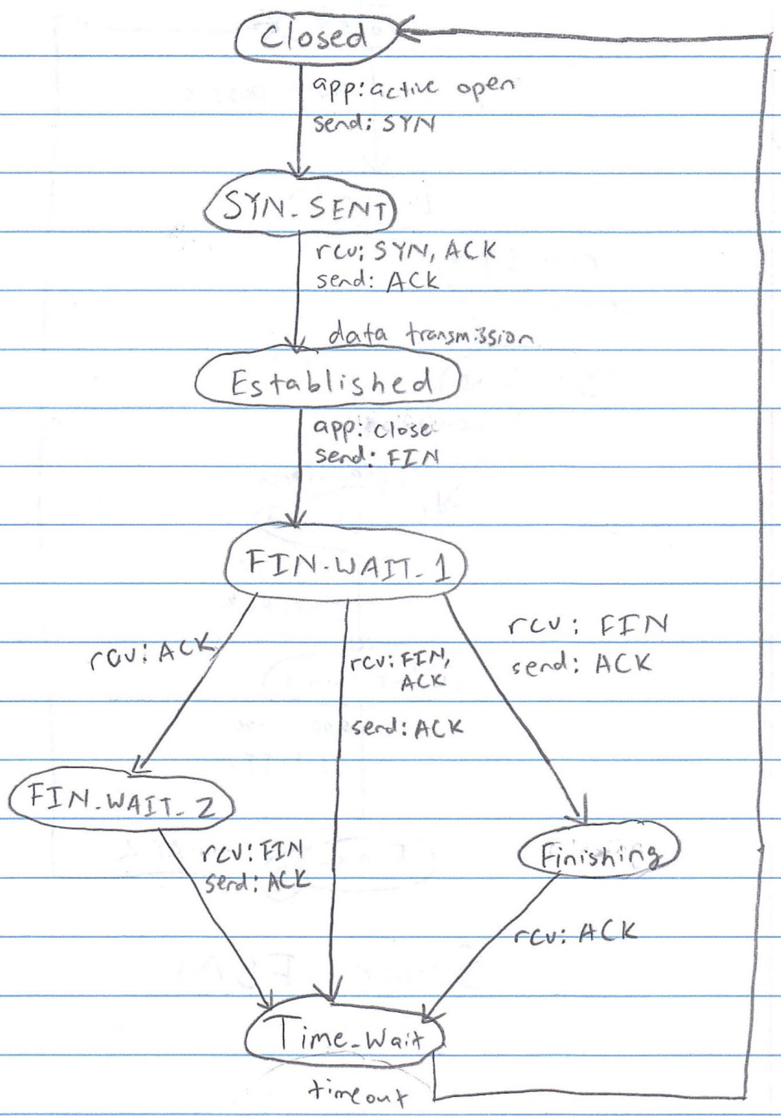
Checksum:
The header will contain a 16 bit checksum to assist with detecting corruption.

Window:
The header will allow for a 32 bit window to assist with flow control. The value of the window designates how much space is remaining in the buffer on the receiving side of the transmission.

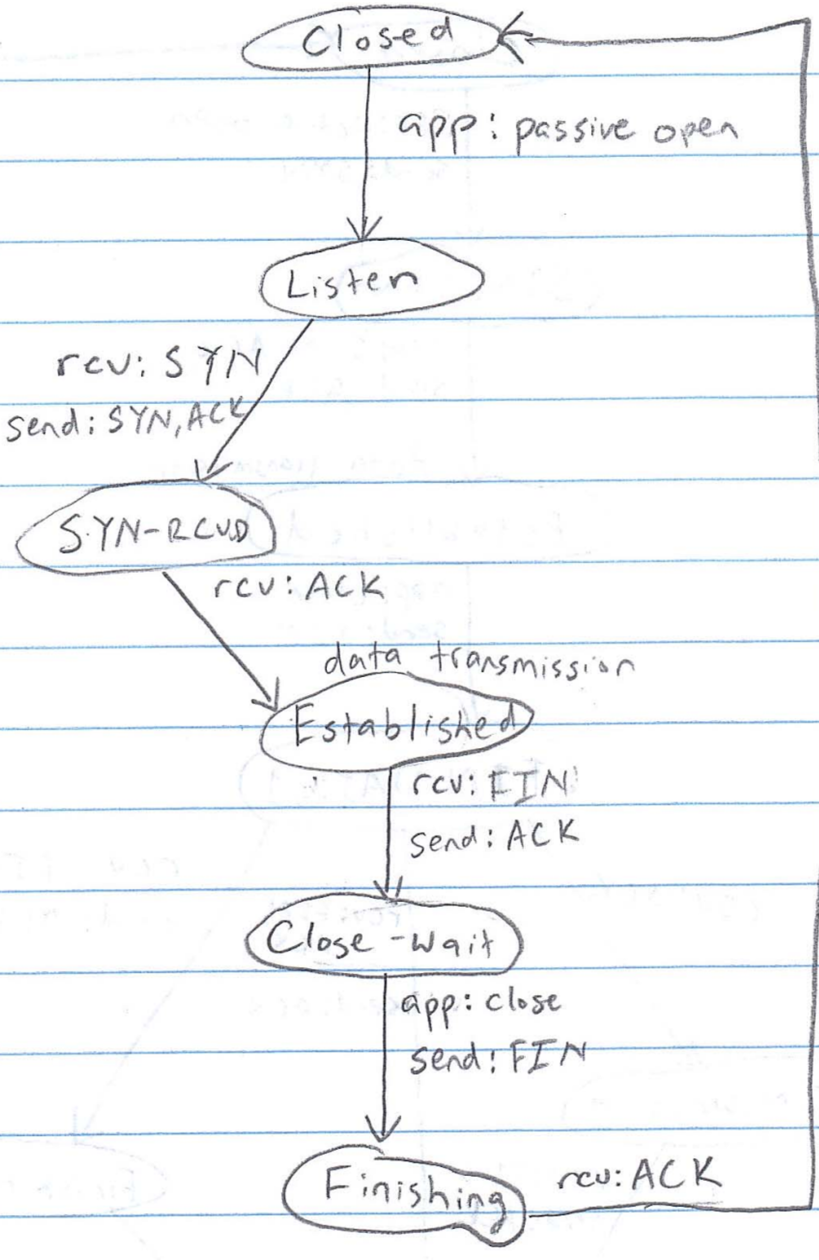
Diagram of header format:

```
31                                                                          0
|---------------------------------------------------------------------------|
|  Source Port : 16 bits                |       Destination Port : 16 bits   |
|---------------------------------------------------------------------------|
|                            Sequence Number: 32 bits                        |
|---------------------------------------------------------------------------|
|                        Acknowledgement Number: 32 bits                     |
|---------------------------------------------------------------------------|
| unused : 11 bits          |           Control bits : 5 bits        |                 Checksum : 16 bits          |
|-------------------------------------------------------------------------------------------------------------|
|                                          Flow Control Window: 32 bits                                       |
|-------------------------------------------------------------------------------------------------------------|
```

Finite State Machine



Client FSM


Closed
app: passive open
Listen
rcv: SYN
send: SYN, ACK
SYN-RCVD
rcv: ACK
data transmission
Established
rcv: FIN
send: ACK
Close-Wait
app: close
send: FIN
Finishing
rcv: ACK

Server FSM

Programming Interface

The user commands defined below specify the basic functions that the protocol provides to the user.

- connect()

    The connect function will be used to connect the two end hosts in the network. The local RTP should be aware of the identity of the process it serves. The destination host, as a foreign socket, will be specified by the sending host. The connect method will open a connection between the two hosts. The connection will have two different modes when in the open state. An active mode will be triggered by the subsequent execution of send(). A passive mode will cause the host to wait for a foreign socket (one that is specified by the local port that initiated the connection).

    This process will return a local connection name that can be used by the user to perform any required data transmission.

- post()

    This function is used to send data across the connection from the sender to the receiver. The address of the receiving host is specified at the time of connection establishment. The implementation of send will block the process in a sending state until transmission is complete and will return control after a successful/unsuccessful transmission.

- get()

    This function is used to receive data across the connection at the receiver's end. The address of the receiving host is specified at the time of connection establishment. The receiver will be blocked until the last packet is received successfully (specified by the LAST flag in the header).

- disconnect()

    This command causes the connection to be closed. The connection will stop sending and receiving any data and will shutdown.

Relevant Algorithms:

Checksum Algorithm:

The algorithm implemented by this protocol is called Fletcher's algorithm. The algorithm uses two summations, both initialized to 0 at the start. The first summation is calculated by computing the modular sum of the data, resulting in an 8 bit sequence. At every step of the first summation, the summed value is added to the second summation to compute another modular summation. After these two summations are computed for the entire packet (header and data), the final checksum value equals a 16 bit value with the most significant 8 bits being the second summation and the least significant 8 bits being the first summation.

This algorithm has two significant advantage over the simple checksum. The first advantage is that the algorithm is sensitive to the order of bytes in the data, through the use of the second summation. The second advantage is that the total possible checksum values is now the square of the possible values of the simple checksum, thus providing more guarantee about the integrity of the data.

Flow Control Algorithm:

Every time the receiver sends an ACK, within the packet it will specify the amount of room that is left in the buffer on that side. The sender will simply look at this size and only send a number of packets that will fit in the buffer. It is possible that the buffer will be full at some point in time. To handle this issue, the sender will simply keep repeatedly sending the same packet till it is accepted. Once the receiver does have room, it will send back an ACK along with the amount of space that is in the buffer now. The sender will then proceed to send more packets. This will continue till the entire message has been sent.