CS 3251: Networking 1: Spring 2016
Programming Assignment 2: Reliable Transport Protocol
Brian Eason Nico de Leon Duc Tran

I. High level description of how RTP works:

1. Connection establishment and connection termination:
   3-way handshake is used for both connection establishment and termination.
   - First, one of the endpoint (first endpoint) will initiate by sending a SYN (for establishment) or FIN (termination) packet to the other (second endpoint).
   - The second endpoint if got the packet, will send a SYN+ACK (or FIN+ACK) packet back to first endpoint and open/close the connection on its side.
   - The first endpoint after getting back the SYN+ACK/FIN+ACK will also open/close the connection on its side (for connection termination, it will have to wait for a timeout before closing the connection), and send an ACK for the SYN+ACK/FIN+ACK..

2. Window-based flow control:
   - If the receiver window is the size of 1 packet, the sender will send 1 packet at a time. Once an ACK is received, another packet will be sent.
   - The receiver window will essentially restrict the number of un-ACKed packets that are in transmission. So if the receiver window is 5000 bytes and packet size is 1000 bytes, the maximum number of un-ACKed packets in transmission is 5.

3. Congestion control:
   - Congestion window begins at 2*MSS which is 2000 bytes (since MSS is 1000 bytes), unless this exceeds the receiver window in which case, the congestion window is the receiver window.
   - Maximum congestion window size is receive window
   - Whenever there is a packet loss or timeout, decrease congestion window by length of data.
   - Whenever there is a successful packet ACK, increase congestion window by length of the data.

4. Duplicate packets handling:
   - Packets are organized by their sequence number.
   - If the received packet has a sequence number that is lower than the current ACK number, it will be ignored, because all sequence numbers lower than the current

ACK number have already been ACK. Therefore, packets with a sequence number less than the current ACK number are duplicates.

5. Corrupted packets handling:
    - We check the integrity of data through a checksum field, which is calculated using Java library CRC32.
    - Before delivering the packet, the checksum field is set to 0, and CRC32 is used to calculate the checksum of all of the fields in the packet, then the checksum is set to that new checksum.
    - When receiving a packet, the recipient will first get the value of the checksum field in the packet, set the checksum field of the packet to 0 for recalculation, recalculate the checksum of that packet, and compare the two values to see if the packet is corrupted or not.
    - Corrupted packet is simply ignored and considered as lost.

6. Lost packets handling
    - Lost packets are handled by resending with each timeout
    - The sender keeps track of all the un-ACKed packet, and when a packet has not been ACKed for a particular period of time, it will be resent.

7. Re-ordered packets handling
    - Packets are organized by their sequence numbers
    - Only the packet with the correct sequence number will be sent to the upper layer. Packet with bigger sequence number will be put to a queue, so it can be processed and ACKed later when all of the previous packets are received.

8. How does RTP (de)-multiplex data to different RTP connections at the same host?
    - The host can easily extract the socket address of where the packet is coming from, and pull out the corresponding connection because it keeps a table of the connections with their socket addresses.
    - If there has not been a connection for that address, the host will create one and put on the table.

9. Bi-directional data transfers:
    - RTP supports bi-directional transfer by allowing data coming in and going out both ways through packets.
    - All of the abstraction of handling which data belongs to which process is handled using Threads in the Application layer.

10. Byte-stream semantics support:

- RTP packet has a packet to Bytes method, which convert all of the information the packet has into an array of bytes for delivery.
- At recipient end, the packet can be recreated by passing the array of bytes into an RTP constructor. The constructor will create a new RTP packet, extract all of the field information from the bytes and put it in the new packet.

11. Special values or parameters:
    - Maximum size in byte for a RTP packet
    - Timeout to wait for a packet

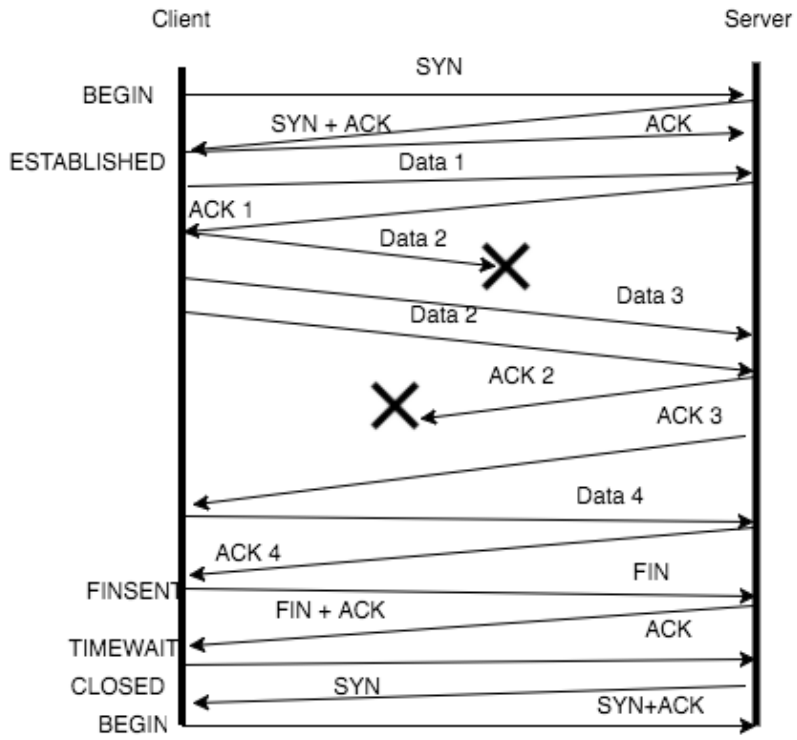
II. RTP header structure and its header fields

RTP Header

<table>
<tr><td colspan="4">Sequence number (32 bits)</td></tr>
<tr><td colspan="4">ACK number (32 bits)</td></tr>
<tr><td>ACK</td><td>SYN</td><td>FIN</td><td>Optional (29 bits): Reserved for future implementation</td></tr>
<tr><td colspan="4">RTP checksum (32 bits):</td></tr>
<tr><td colspan="4">data</td></tr>
</table>

Bit 0-31: sequence number of the packet (if the packet contains data)
Bit 32-63: acknowledged number of the packet (if the packet is a ACK packet)
Bit 64: ACK flag (set to 1 if the packet is a ACK packet)
Bit 65: SYN flag (set to 1 if the packet is a SYN packet for connection establishment)
Bit 66: FIN flag (set to 1 if the packet is a FIN packet for connection termination)
Bit 67-95: Optional bits, can be used for future implementation
Bit 96-127: Packet checksum for integrity check
Bit 128 - varied: packet data (if applicable).

III. FSM for two endpoints



IV. API formal description

A formal description of the protocol's programming interface (the functions it exports to the application layer, including return values and any error conditions).

1. RTPServerSocket.java
    - constructor(int port): create a new UDP socket for the RTP socket to use.
    - accept(): accepting a new connection from a client
        - Return: a new RTP socket for that connection
        - May get error if interrupted.
    - close(): Close the connection
    - setReceiveBufferSize(int size): set the size of the receive buffer for each client of the server
2. RTPSocket.java:
    - constructor(InetAddress address, int port):
        - Create I/O stream for/to application layer
        - Create a connection to the server
        - Create a thread to listen for data come from application layer
        - Create a thread to listen for packets sent from server.
        - Establish a three-way handshake with the server in order to be ready for data transfer.

- getInputStream(): return input stream from application layer
- getOutputStream(): return output stream to application layer
- setReceiveBufferSize(int size): set the size of its receive buffer
- close():
    - Perform a three-way handshake to terminate the connection with the server.
    - Close the I/O streams from/to the application layer