**RTP Protocol Design Documentation**

**Cameron Gallahue and Bill Feng**

1. High Level Analysis
   1. In order to send data via RTP, several pieces of information must be known. We must know where to send the data (IP address and port number), what data to send, and how much data can be sent at once (the window size in packets). The actual functionality of RTP runs as follows:
      1. The server is set up to listen for incoming packets on a particular port number.
      2. The client sends a connection request packet to the server, containing the client’s IP and port number, as well as the specific data requested.
      3. The server accepts the connection, and divides up the requested data into packets.
      4. All of these packets are added into the queue, and as many are sent at once as the maximum window size allows. The client acknowledges each packet as it is received, and the server shifts the sender window over to continue sending more packets in its queue. As each packet has its own destination IP and port number, the server can service more than one connection at a time by simply adding the packets for each new request to the end of the sending queue.
      5. The client then closes the connection, which sends a message to the server saying that it wants to close the connection. The server then sends an acknowledgement of the connection closing, and closes the connection. The client then closes the connection once this packet is received.
   2. RTP Header Structure
      1. ip\_src: the source IP address
      2. sPort: the virtual source port number
      3. ip\_dest: the destination IP address
      4. dPort: the virtual destination port number
      5. sPort\_udp: the source port number used by UDP
      6. dPort\_udp: the destination port number used by UDP
      7. seqn: the sequence number of the packet
      8. SYN: Boolean representing whether the packet is a SYN packet (used to initiate a connection)
      9. ACK: Boolean representing the acknowledgement of a packet
      10. BEG: Used when the payload is data to be stored by the recipient
      11. FIN: Boolean, used to indicate closing a connection
      12. GET: Boolean used when the payload is a request for data
      13. Timestamp: the current timestamp of the packet used to determine timeout
   3. Finite State Machine Diagrams
   4. RTP API
      1. \_\_init\_\_(ip\_addr, udp\_port, rtp\_port, server, receiveWindow):
         1. ip\_addr is the IP address to be used, either the IP of the server if used client-side, or the local address (127.0.0.1) if used server-side
         2. udp\_port is the UDP port number of the server. Should be whatever number the server listens on.
         3. rtp\_port is the RTP port number used to number the particular connection. This allows the connection to be multiplexed, sending to the same physical port, but separating the data at the source by its “virtual” port number, or its connection number.
         4. server is a Boolean that is set to true if this is created server-side, or false from the client side.
         5. receiveWindow is the size of the receiver window in packets. RTP will allow this number of in-transit packets at any given time.
      2. connect(ip\_client, uPort, dPort):
         1. This method is only called client-side.
         2. ip\_client is the IP address of the local machine (the client, not the server)
         3. uPort is the physical UDP port to send the data back to
         4. dPort is the RTP port to send the data back to
      3. close(ip\_dest, uPort, dPort):
         1. ip\_client is the IP address of the local machine (the client, not the server)
         2. uPort is the physical UDP port to send the data back to
         3. dPort is the RTP port to send the data back to
      4. listen():
         1. This is called in an infinite loop server-side, and automatically handles all traffic internally, accounting for all file sending and posting, as well as establishing connections with a client
      5. getFile(filename, ip\_client, uPort, dPort):
         1. This is used for requesting a file from a server
         2. Filename is the name of the file to be retrieved
         3. ip\_client is the IP address of the local machine (the client, not the server)
         4. uPort is the physical UDP port to send the data back to
         5. dPort is the RTP port to send the data back to
      6. sendFile filename, ip\_client, uPort, dPort):
         1. Filename is the name of the file to be sent
         2. ip\_client is the IP address of the local machine (the client, not the server)
         3. uPort is the physical UDP port to send the data back to
         4. dPort is the RTP port to send the data back to
      7. send(message, ip\_dest, uPort, dPort):
         1. message is the byte array of data to be sent to the recipient
         2. ip\_client is the IP address of the local machine (the client, not the server)
         3. uPort is the physical UDP port to send the data back to
         4. dPort is the RTP port to send the data back to
      8. recv(ip\_dest, uPort, dPort):
         1. ip\_client is the IP address of the local machine (the client, not the server)
         2. uPort is the physical UDP port to send the data back to
         3. dPort is the RTP port to send the data back to
      9. getPost(getName, postName, ip\_dest, uPort, dPort):
         1. getName is the filename for the server side file
         2. postName is the filename for the client side file
         3. ip\_dest is the IP address for the server
         4. uPort and dPort are the UDP and RTP port of the server, respectively
         5. This method is only called by the client
         6. This method makes the client send and receive packets synchronously
   5. Algorithms of Non-Trivial Methods
   6. Other Information
      1. How does RTP perform connection establishment and connection termination?
         1. Connections are established by sending a SYN packet to the server, which is responded to by a SYNACK. Once the client receives the SYNACK, it will be able to send its request to the server. To terminate the connection, the client sends a FIN packet to the server, which responds with a FINACK. Once the client receives the FINACK, the connection is closed.
      2. How does RTP perform window-based flow control?
         1. The RTP takes in a maximum window size on creation. The window size is maintained by allowing the first n packets in the queue to be sent given that n is the window size. Once the first packet in the queue is ACKed, it is removed from the queue, allowing the next packets in line to be sent.
      3. How does RTP detect and deal with duplicate packets?
         1. RTP also uses sequence numbers to determine packet ordering, and will ignore packets that are either outside of the receiver window or have already been ACKed (for example, those with sequence numbers less than the current packet waiting to be received)
      4. How does RTP (de)-multiplex data to different RTP connections at the same host?
         1. RTP uses both the physical port number (used for UDP messages) as well as a “virtual” port that’s used to determine what particular connection a particular packet is intended for. This way, a single UDP port can be used to service multiple information streams at once.
      5. How does RTP support bi-directional data transfers?
         1. Both the client and server handle outgoing data and incoming ACKs (or vice versa) already, so during bi-directional data transfer, the RTP header is checked to see what type of packet is being sent, and handles it whether it is a data packet or an ACK.
      6. How does RTP provide byte-stream semantics?
         1. Byte-streaming is supported as the message to be sent is supplied as a byte array, which is split into payloads to be sent one after the other.
      7. Are there any special values or parameters in your design (such as minimum packet size)?