Simple Reliable Transport Protocol (SRTP)

PROJECT REPORT

Varun Tutpet Keshava Murthy (000922205)

Shruthi Hebbur Prasanna Kumar (000917498)

Aim of the Project:

To implement a simple reliable transfer protocol which provides reliable transmission of data over the unreliable network layer UDP. The reliability in SRTP is provided on lines of the TCP functionality.

Implementation:

1. SRTP Packet structure:

We are using the following packet structure in our SRTP for communication between the peers.

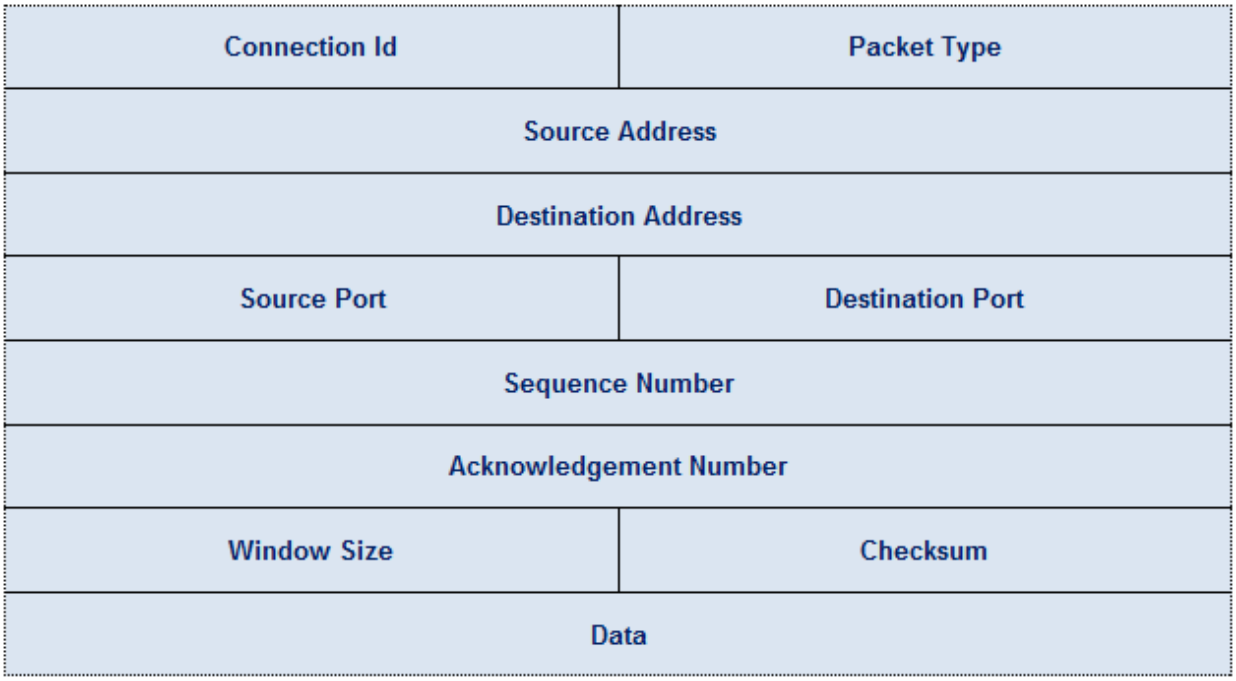


Figure 1 : Packet structure

Connection Id :Identifier to identify the connection.
Packet Type : SRTP_SYN , SRTP_SYN_ACK , SRTP_FIN, SRTP_RESET, SRTP_DATA: SYN , Acknowledge , Reset , Data
Source Address : IP address of the packet originating host.
Destination Address : IP address of the destination host.
Source port : Port number of the originating host.
Destination port : Port number of the destination host
Sequence number : Identification number to identify the packet (For flow control)
Acknowledgement number : Acknowledgement number for the received packet.
Window size : Maximum buffer size advertised by the sending host.
Checksum : For error checking of the header. (for reliable transfer)
Data : Data to be transferred.

2. **Connection Management:** This module handles connection establishment and termination.

Class Diagram for Connection Class:

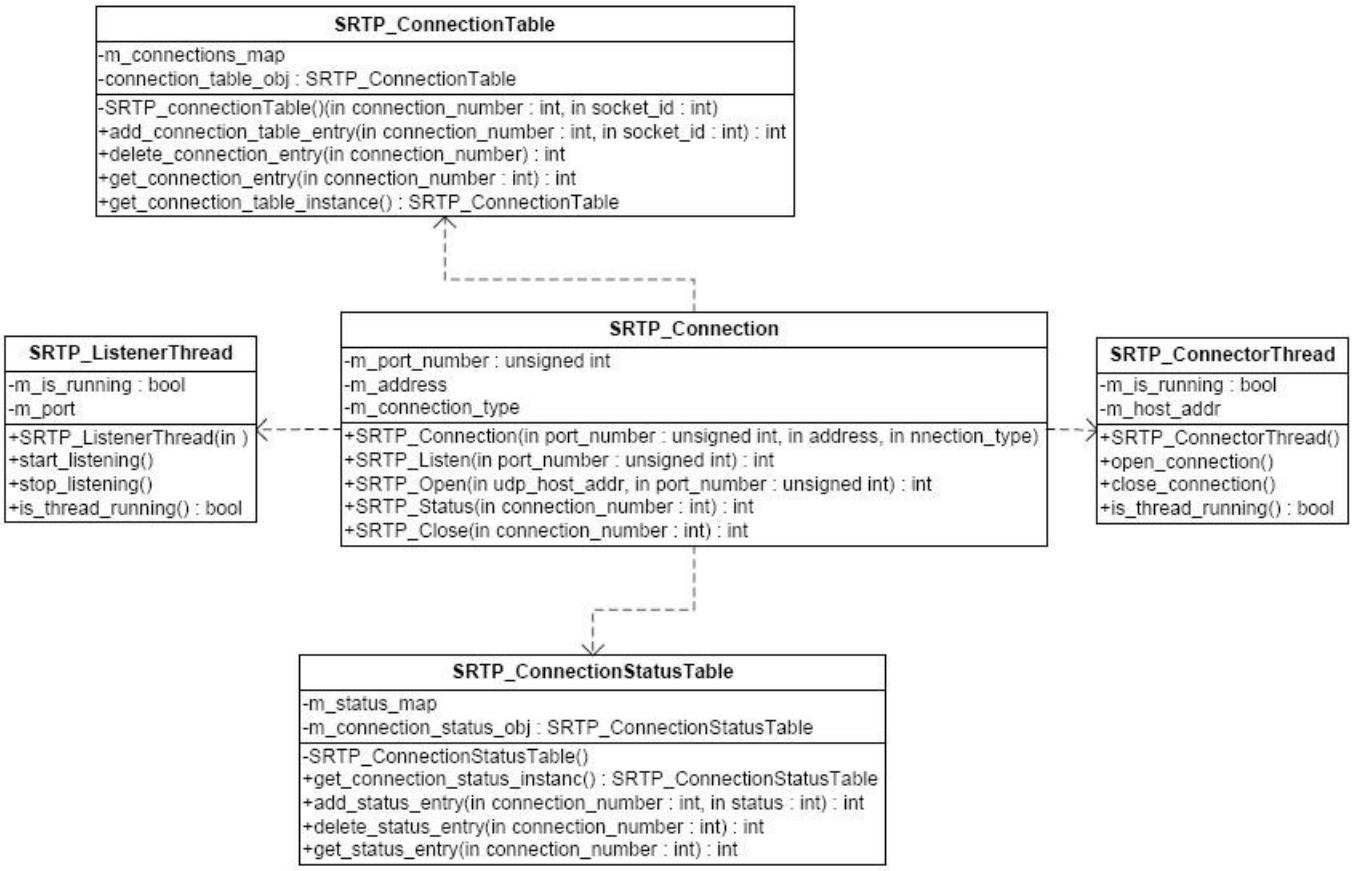


Figure 2: Connection Management

1. Class SRTP_Connection: This class is used to create/close the connection for the end applications.

2. Class SRTP_ConnectionInfo: This class is used to store the information for each connection (identified by connection number) like the socket id , local port, local address, remote port and remote address.

3. Class SRTP_ConnectionTable: This class is Singleton class and is used to store the mapping of established connections information present in objects of SRTP_ConnectionInfo class and their connection numbers.

4. Class SRTP_ConnectionStatusTable: This class is Singleton class and is used to store the mapping of connections status and their connection numbers.

5. Class PeerInfo : This class is used to store the information of the peer like the status of the peer, remote address , remote port, expected packet number from the peer, current sequence number and the window size of the peer.

6. Class PeerTable: This class is a Singleton class and is used to store the mapping between the connection number of the connection and the object of the Peer Info class.

Client side transition diagram: We follow the state transition similar to that of TCP.

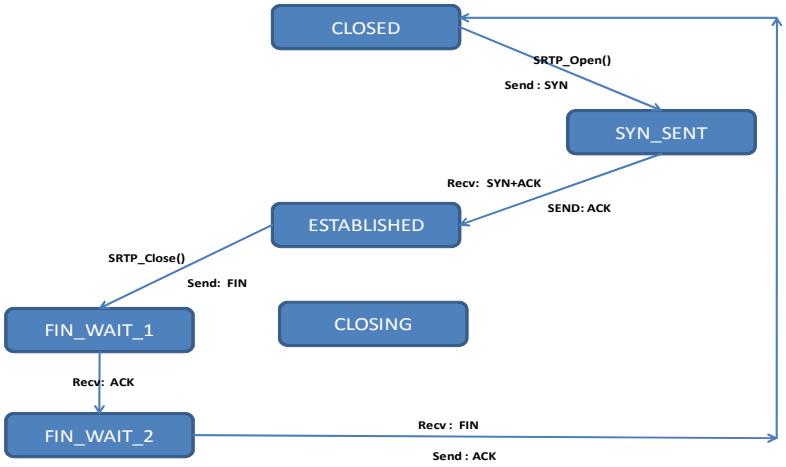


Figure 3: Client side transition diagram

Server side transition diagram:

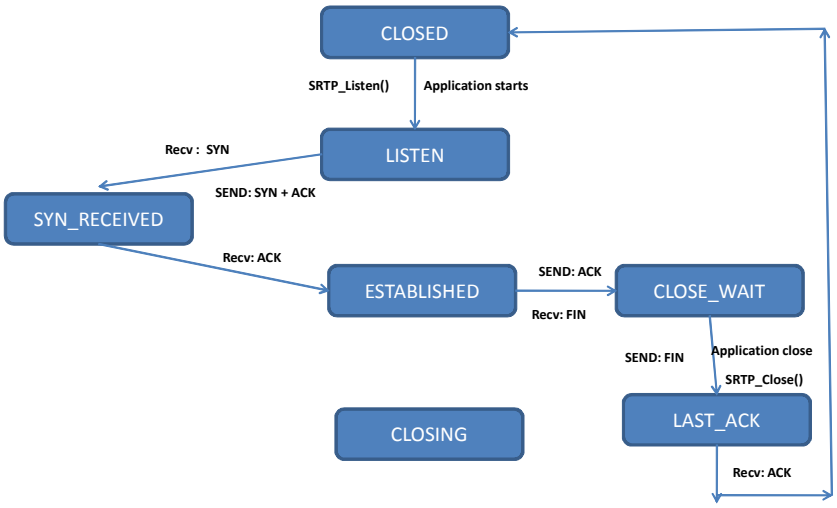


Figure 4: Server side transition diagram

The 4 interfaces used in SRTP_Connection class to achieve this are as follows:

1. SRTP_Listen(): Connection establishment at server side (passive open)
2. SRTP_Open(): Connection establishment at client side (active open)
3. SRTP_Close(connection_number): For graceful shutdown
4. SRTP_status(connection_number): To find the status of the connection

Connection establishment: TCP's 3 way handshake is used for Connection Establishment

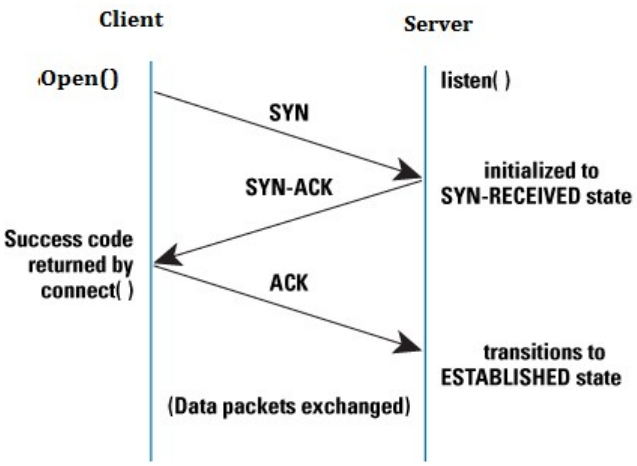


Figure 5: Connection Establishment

Connection termination: The connection termination uses the TCP's 4-way disconnection with half close.

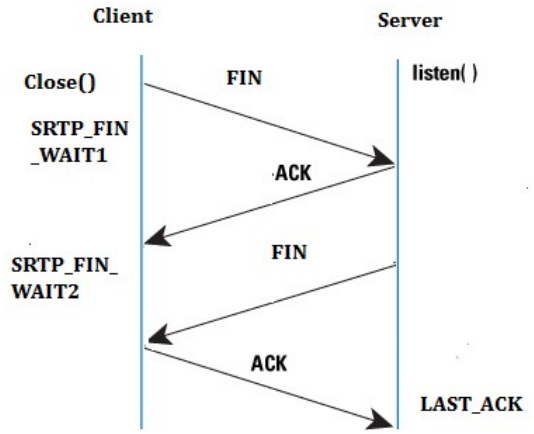


Figure 6: Graceful disconnection

3. Handling multiple Connections:

To handle multiple connections we use threads.

Thread Architecture:

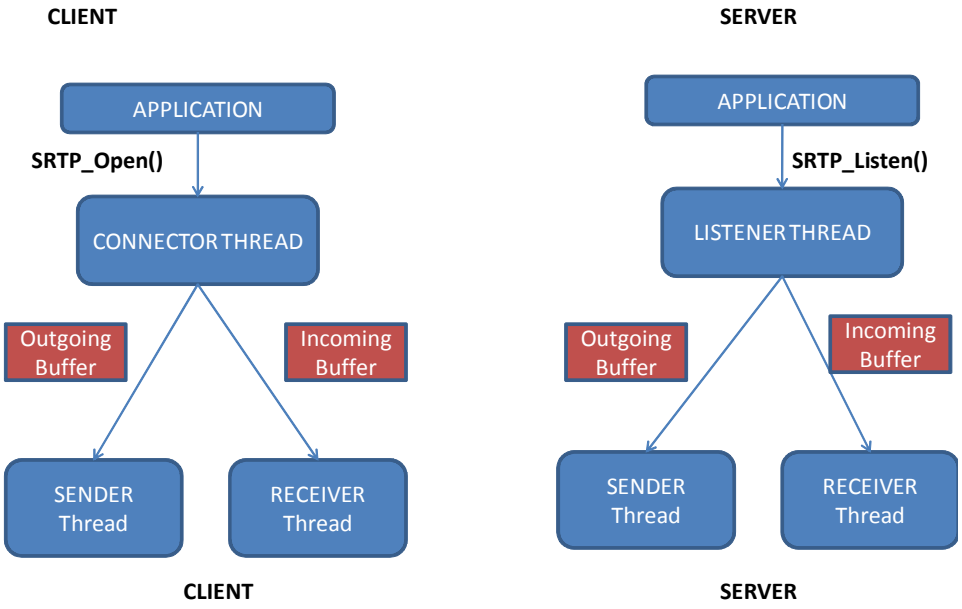


Figure 7: Thread Architechure

The application creates an object of class Connection which in turn spawns Connector thread if it is client and the listener thread if it is server. Each of the listener and the connector threads spawn sender and receiver threads to send and receive SRTP Packets.

1. **SRTP_ListenerThread**: This class is used to create a listening port to receive new connections from the peer. The start_listening() interface is used to create a new UDP socket, bind the local address and handle the incoming connections. The stop_listening() interface is used to close the socket associated with the listening port.

2. **SRTP_ConnectionThread:** This class is used to initiate/close a new connection to the peer. The SRTP_Open() interface is used to create a new UDP socket and bind the remote address and send a connection request.

3. **SRTP_ReceiverThread and SRTP_SenderThread:** These classes are used to receive and send the SRTP packets through the created sockets.

Thread Synchronization:

Since we are using multi-threading, thread synchronization is very important. We are using the pthread libraries built-in synchronization functionalities like Mutexes, Condition Variables & Joins. Mutexes are important whenever a global variable is being accessed. Condition variables are used in order to wait or notify a thread about a certain condition.

4. Buffer Management:

We are using the STL container deque of SRTP_packets as the buffer.

There is an Incoming Buffer and an Outgoing Buffer on either side (server and the client).

There is an IncomingHandler Class and an OutgoingMessageHandler class to manage the flow of packets between the application and the Incoming / Outgoing buffers.

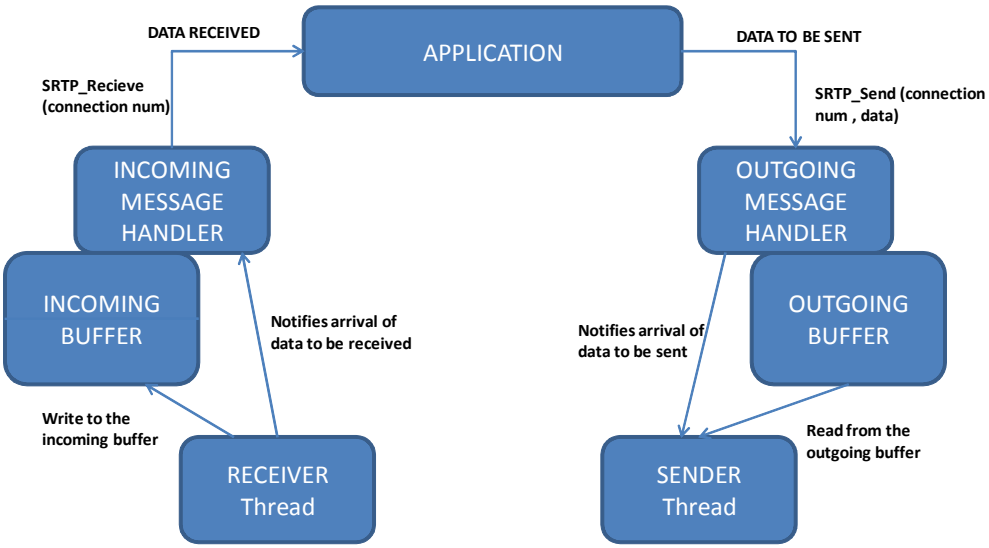


Figure 8: Buffer Management

The receiver thread receives the data packets and puts it in the Incoming Buffer. It notifies the Incoming Message Handler about the arrival of new data. The application uses SRTP_receive (Connection Number) interface to read the arriving data.

The application puts the data to be sent using SRTP_Send(connection number) in outgoing buffer through OutgoingBuffer Handler which notifies the Sender thread to send the data packets.

1. **Class SRTP_IncomingBuffer:** This class is used to store the incoming packets. It contains a double ended queue of SRTP_Packets with head and tail pointers. There is also a field to indicate the buffer window size. The add_packet() interface is used to add new arriving packets into the buffer. The remove_packet() interface is used to remove the acknowledged packets from the buffer. The increase size() and decrease_size() interfaces are used to modify the buffer window sizes. The get_packet() interface is used to get a packet from the buffer.

2. **Class SRTP_OutgoingBuffer:** This class is used to store the outgoing packets. It contains a double ended queue of SRTP_Packets with head and tail pointers. There is also a field to indicate the buffer window size. The add_packet() interface is used to add new arriving packets into the buffer. The remove_packet() interface is used to remove the acknowledged packets from the buffer. The increase size() and decrease_size() interfaces are used to modify the buffer window sizes. The get_packet() interface is used to get a packet from the buffer.

5. Reliable Transport:

We use the sequence numbers and checksum in our SRTP packet to provide reliable transport. We also use retransmission timers to handle the lost, corrupted , duplicated and delayed messages.

Sequence numbers: Each outgoing SRTP packet has a sequence number to be identified by the receiver to check if the packets are in order and no packet is lost. The receiver compares the incoming packet sequence number with the sequence number it expects. If it doesn't match , it drops the packet.

Checksum: For each SRTP packet sent from the sender thread, we calculate the checksum of the header using the checksum function and put it in the checksum field of the SRTP packet. At the receiver the checksum of the received packet is recalculated to validate the packet. If it is not validated as right, the packet is dropped.

Retransmission strategy: Whenever a packet is sent from the sender thread, a timer is started using alarm() function The timer is set to SRTP_Retransmission time. Once the acknowledgement for the packet is received, the timer is reset. If the ACK for the expected packet acknowledgement doesn't match the acknowledgement for the sequence number of the incoming packet, the packet is dropped. Once the set retransmission timer expires, there will be SIGALRM to notify this which will trigger the retransmission of the packet.

6. Flow control mechanism:

Sliding window: The Go-Back-N protocol is chosen.

We use alarm() function to initialize a timer when a packet is sent from the sender thread.

Each packet received has a sequence number . When the status of a connection is SRTP_Established and the received packet is of the type DATA , then the sequence number of the received packet is checked if it matches the expected packet sequence number for that connection. If not, the packet is discarded and no ACK for it is sent. Hence all the subsequent packets also will be dropped. After the expiration of the timer, on the sender side, the outgoing buffer head pointer will be readjusted to point to the beginning of the deque of the outgoing buffer and the packets will be retransmitted.

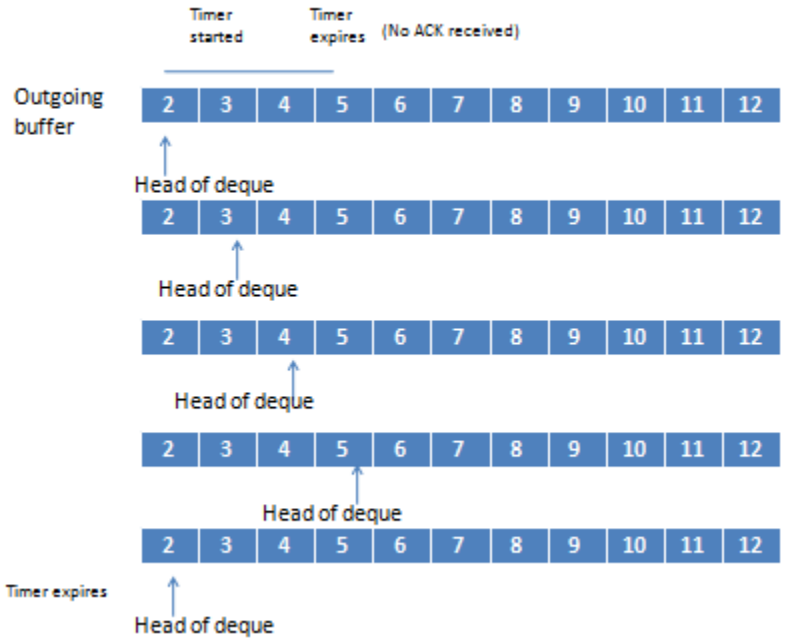


Figure 9: Sliding Window

7. The congestion-control technique:

Slow-start, congestion-avoidance is implemented by varying the buffer window size.

It is implemented as explained below:

Window size of the sender (the number of packets sent from outgoing buffer) will be minimum of receiver window advertisement in the SRTP packet in the first packet which is the SYN packet and cwnd (cwnd is the window size maintained at the sender) at a given point of time

window size = Min (cwnd , window size adv received)
Once connecton is established "cwnd" = 1;
Slow start threshold "ssthresh"= (window size in SYN packet) / 2
Each time ACK, received

```
if (PacketType == ACK)
{
        if ( ssthresh > = cwnd)
        {
                cwnd = cwnd + 1;

                windowsize = min (cwnd , window size ad recieved);

        }
        else
        {
                cwnd = (cwnd  + 1) /cwnd;

                windowsize = min (cwnd , window size ad recieved);

        }
}
else
{
        if (SIGALRM )
        {
                cwnd = 1;
                ssthresh = Max ( 2 , windowsize / 2 );

                windowsize = min (cwnd , window size ad recieved);

        }

}
```

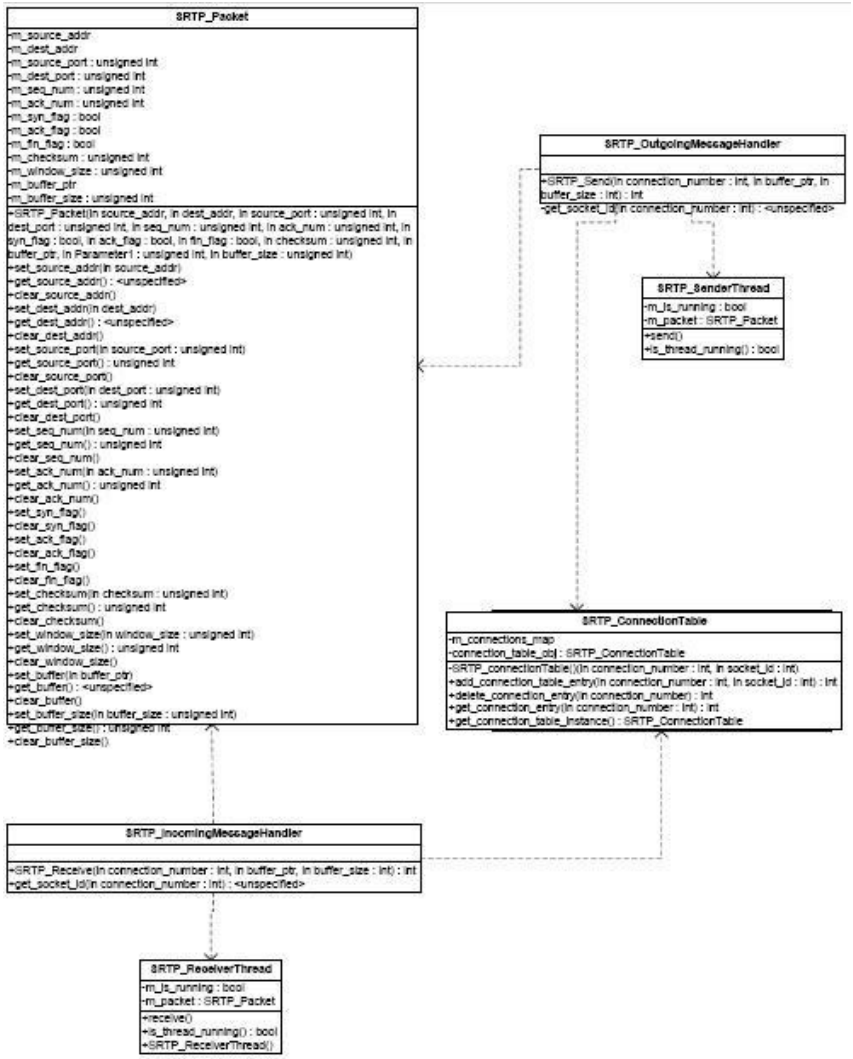
Fast retransmit and Fast recovery has been implemented almost similar to that existing in TCP.

8. Packet Transmission:

1. **Class SRTP_Packet:** This class is used to hold all the information regarding the SRTP packets. The packet structure is shown in the first section of the report. The header fields of the packet are implemented as separate fields in the class. The data portion of the SRTP packet has to be stored in a buffer (char*) and the buffer pointer and size has to be passed.

2. **Class SRTP_IncomingMessageHandler:** This class provides an interface for the applications to receive the packets from the peer. This class creates a SRTP_Receiver thread which is used to receive the data from the sockets.

3. **Class SRTP_OutgoingMessageHandler:** This class provides an interface for the applications to send the packets from the peer. This class creates a SRTP_Sender thread which is used to send the data to the sockets.


SRTP_Packet
-m_source_addr
-m_dest_addr
-m_source_port : unsigned int
-m_dest_port : unsigned int
-m_seq_num : unsigned int
-m_ack_num : unsigned int
-m_syn_flag : bool
-m_ack_flag : bool
-m_fin_flag : bool
-m_checksum : unsigned int
-m_window_size : unsigned int
-m_buffer_ptr
-m_buffer_size : unsigned int
+SRTP_Packet(in source_addr, in dest_addr, in source_port : unsigned int, in dest_port : unsigned int, in seq_num : unsigned int, in ack_num : unsigned int, in syn_flag : bool, in ack_flag : bool, in fin_flag : bool, in checksum : unsigned int, in buffer_ptr, in Parameter1 : unsigned int, in buffer_size : unsigned int)
+set_source_addr(in source_addr)
+get_source_addr() : <unspecified>
+clear_source_addr()
+set_dest_addr(in dest_addr)
+get_dest_addr() : <unspecified>
+clear_dest_addr()
+set_source_port(in source_port : unsigned int)
+get_source_port() : unsigned int
+clear_source_port()
+set_dest_port(in dest_port : unsigned int)
+get_dest_port() : unsigned int
+clear_dest_port()
+set_seq_num(in seq_num : unsigned int)
+get_seq_num() : unsigned int
+clear_seq_num()
+set_ack_num(in ack_num : unsigned int)
+get_ack_num() : unsigned int
+clear_ack_num()
+set_syn_flag()
+clear_syn_flag()
+set_ack_flag()
+clear_ack_flag()
+set_fin_flag()
+clear_fin_flag()
+set_checksum(in checksum : unsigned int)
+get_checksum() : unsigned int
+clear_checksum()
+set_window_size(in window_size : unsigned int)
+get_window_size() : unsigned int
+clear_window_size()
+set_buffer(in buffer_ptr)
+get_buffer() : <unspecified>
+clear_buffer()
+set_buffer_size(in buffer_size : unsigned int)
+get_buffer_size() : unsigned int
+clear_buffer_size()
SRTP_OutgoingMessageHandler
+SRTP_Send(in connection_number : Int, in buffer_ptr, in buffer_size : Int) : Int
-get_socket_id(in connection_number : Int) : <unspecified>
SRTP_SenderThread
-m_is_running : bool
-m_packet : SRTP_Packet
+send()
+is_thread_running() : bool
SRTP_ConnectionTable
-m_connections_map
-connection_table_obj : SRTP_ConnectionTable
-SRTP_connectionTable()(in connection_number : Int, in socket_id : Int)
+add_connection_table_entry(in connection_number : Int, in socket_id : Int) : Int
+delete_connection_entry(in connection_number) : Int
+get_connection_entry(in connection_number : Int) : Int
+get_connection_table_instance() : SRTP_ConnectionTable
SRTP_IncomingMessageHandler
+SRTP_Receive(in connection_number : Int, in buffer_ptr, in buffer_size : Int) : Int
+get_socket_id(in connection_number : Int) : <unspecified>
SRTP_ReceiverThread
-m_is_running : bool
-m_packet : SRTP_Packet
+receive()
+is_thread_running() : bool
+SRTP_ReceiverThread()

9. Timer Management:

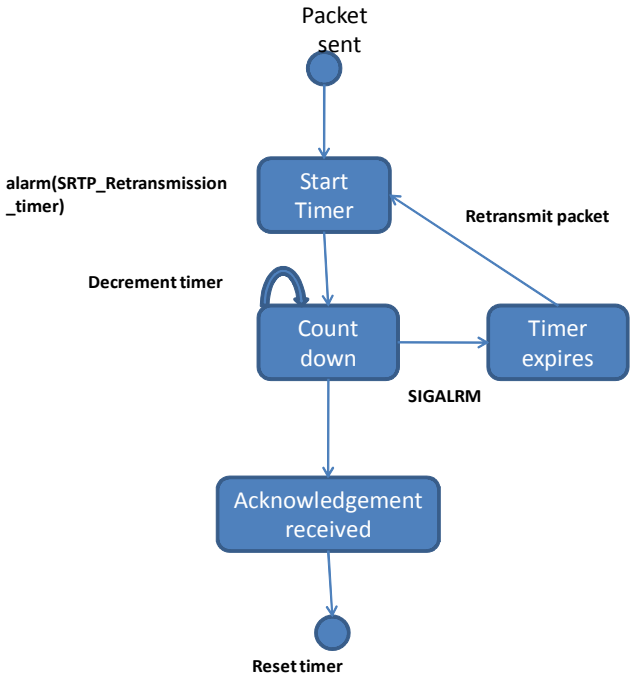


Figure 10: Retransmission timer

For managing the timer for retransmission, we are using alarm() function to set the retransmission timer each time a packet is sent. Once the ACK for the packet is received, the timer is reset. If the timer expires, a signal SIGALRM notifies the event and the packet is retransmitted.

To calculate the delay we use a timeval struct.

10. **Network Management (counters):** The counters are used to store the various network management related statistics like number of packets transmitted, received , dropped , retransmitted , checksum errors etc.

Various statistics like the number of packets transmitted successfully, number of packets retransmitted, number of acknowledgements received, and the average delay are maintained for each connection.

For example: Statistics Output of one connection for data transfer using SRTP.

```
*****************Upload***************

--------------------------------------
Run Time Statistics
--------------------------------------
Number of packets sent: 6
Number of packets received: 0
Number of acknowledgements sent: 0
Number of acknowledgements received: 6
Number of packets retransmitted: 0
Number of checksum errors: 0
Average delay of packets: 2432.83 microseconds
Data rate = 1644.17Kbps
--------------------------------------

***************Download*****************
--------------------------------------
Run Time Statistics
--------------------------------------
Number of packets sent: 1
Number of packets received: 5
Number of acknowledgements sent: 5
Number of acknowledgements received: 1
Number of packets retransmitted: 0
Number of checksum errors: 0
Average delay of packets: 1595 microseconds
Data rate = 2507.84Kbps
--------------------------------------
```

11. Applications:

We have developed two applications – FTP and simple talk using the SRTP interfaces.

FTP is used to download and upload files.

Talk application is used like a simple chat application.

12. Extra features:

We have implemented encryption and decryption features for our data . The outgoing and incoming data are encrypted and decrypted on sender and receiver side respectively.

It is similar to secret key encryption. Client and server share a secret key. Data transfer between the server and client are encrypted from the sender end and decrypted at the receiver.