Simple Transmission Control Protocol

Kushal Lakhotia
Department of Computer Science, Purdue University

This document contains a short description of the SCTP protocol explaining the steps involved in the implementation as the design level.

State Variables Maintained in the Implementation

The following STCP state variables are maintained which are updated whenever an event occurs and relevant actions are taken. The comments in “/* */” describes the function of a variable. In the steps described below there is frequent reference to these state variables.

```
bool_t done;                          /* TRUE once connection is closed */
int connection_state;                 /* state of the connection (established, etc.) */
mysocket_t sd;                        /* socket descriptor */
tcp_seq initial_sequence_num;         /* initial sequence number of this host */
tcp_seq receiver_initial_seq_num;  /* initial sequence number of the receiver or the other host */

tcp_seq receiver_window;      /* receiver window size last advertised by the receiver or the other host */

tcp_seq send_base;                 /* first unACKed sequence number */
tcp_seq send_base_ptr;             /* pointer to the send_window corresponding to the send window */
tcp_seq next_sequence_num;      /* next sequence number that is free */

tcp_seq recv_window_size;                /* current receive window of the host */
tcp_seq expected_sequence_num_ptr;  /* pointer to the recv_window corresponding to the receive window */
tcp_seq expected_sequence_num;        /* expected sequence number from the sender or other host */

char send_window[WINDOW_SIZE];           /* send buffer of the host */
char recv_window[WINDOW_SIZE];           /* receive buffer of the host */
int recv_window_lookup[WINDOW_SIZE];  /* lookup table for the receive buffer of the host */

bool_t timer_running;        /* to indicate whether the timer is running or not */
int retransmission_count;    /* to keep a count of how many retransmissions have been done */
bool_t close_initiator;        /* to indicate if the connection close has been initiated by this host or not */
```

Network Initiation

In the network initiation the active and passive hosts go through the following states which comprise of the 3-way handshake.

Active Host:

1. SEND_SYN: The active host starts at this state. It sends the SYN and moves to the next state and waits for an ACK from the passive host. The active host will try to send the SYN for a maximum of 6 times and then close the connection.
2. WAIT_FOR_ACK: The active host waits for SYN_ACK and then sends an ACK to the other host. If it does not get SYN_ACK within a timeout period, it moves to SEND_SYN to resend SYN. If it gets SYN_ACK within the timeout, it initializes the state variables, moves to the ETABLISHED state which implies connection is established and then unblocks the application.

Passive Host:

1. WAIT_FOR_SYN: The passive host waits for a SYN to arrive. It it gets a SYN, it sends a SYN_ACK and moves to the next state to wait for an ACK. The passive host tries to send SYN_ACK for a maximum of 6 times and then closes the connection.
2. WAIT_FOR_ACK: The passive host waits for an ACK to arrive in this state for an interval of time and then times out. If timeout occurs then it moves to the previous state but does not wait for a SYN to arrive. Instead it just resends SYN_ACK. If it does not time out then it moves to the ETABLISHED state which implies connection is established and then unblocks the application. It checks for any application data that the segment may contain and then handles that appropriately as described below.

<u>Transfer of Data Between the Hosts</u>

A host keeps looping over a series of checks and corresponding actions till the connection is alive. The following steps summarize the same. The timeout mechanism is explained after these steps.

1. Wait for an event to occur for a particular timeout period.
2. If the event is that application wants to send data then perform the following steps
   (a) If the current send window size is greater than zero then perform steps (b) to (g) else go back to step 1.
   (b) Initialize the maximum application data STCP is willing to accept from the application depending on the state variables.
   (c) Accept data from the application in a temporary buffer.
   (d) Construct the STCP header to encapsulate the application data.
   (e) Construct the STCP segment to be sent using the header and application data.
   (f) Send the segment to the network layer and start the timer if it is not running.
   (g) Buffer the sent data in the send buffer and update the state variables.
3. If the event is that network wants to deliver a segment then perform the following steps
   (a) Accept the segment, extract the header and the application data from it.
   (b) If the header has ACK bit set then read the ACK. If the ACK is within the send window then move the send window forward. Stop the timer if it is running. If there are still unacknowledged bytes in the send window then start the timer again.
   (c) If the segment contains application data then handle the application data as described in steps (d) to (f).
   (d) If the sequence number in the header is the expected sequence number then update the state variable to reflect the bytes that have been received. Deliver all bytes of data that have been received in order and move the receive window forward. Update the other state variables and send an ACK to the other host.
   (e) If the sequence number is lesser than the expected sequence number and lies in the last receive window but still has some bytes in the present receive window then extract the bytes that lie in the present window. Deliver all bytes of data that have been received within the window and move the receive window forward. Update the state variables and send an ACK to the other host. If no data lies in the current receive window then send an ACK in response to the duplicate data received.
   (f) If the sequence number is greater than the expected sequence number but still lies in the present receive window then out of order data has been received. If there is space in the receive window then extract the data that lies within the receive window and buffer it.

Update the state variables and send an ACK to the other host.

(g) If the header has the FIN bit set then initiate the network termination as the passive host as described below.

4. If the event is that the application wants to close the connection then initiate network termination as the initiating host as described below.
5. Free up all temporary buffers created for storing application data, header or segment.

Timeout mechanism: The timeout mechanism is implemented using an alarm functionality as described below:

1. Start Timer - Set up the signal handler for SIGALRM and set the alarm to the timeout value.
2. Stop Timer - Stop the alarm and set the retransmission count to 0.
3. Resend – If the retransmission count is 6 then close the connection (the count starts from 0). Otherwise increment the retransmission count by 1. Start at the first unacknowledged byte and retransmit all the buffered bytes in chunks of MSS or leftover bytes, whichever is lesser. Start the timer.

Network Termination

In the network termination the initiating and passive hosts go through the following states which comprise of the 4-way tear-down.

Initiating Host:

1. SEND_FIN: If the application data has requested for connection termination then this host is the initiator and it sends a FIN segment and moves to he next state waiting for an ACK. The initiator tries to send FIN for 6 times and then closes the connection.
2. WAIT_FOR_ACK: The initiator waits for ACK to arrive from the other host. If it does not get ACK within a timeout period, it will move to SEND_FIN to resend the FIN. If it gets ACK within the timeout, it moves to the WAIT_FOR_FIN state.
3. WAIT_FOR_FIN: It waits for FIN and when received check if it has data in it. If it does then it handles the data as explained above. It then moves to FIN_RECVD state.
4. FIN_RECVD: It sends an ACK in response to the FIN and terminates the connection.

Passive Host:

1. FIN_RECVD: If the other host has requested for connection termination then this host is the passive host and it sends an ACK in response to the FIN. It then moves to SEND_FIN state to send the FIN from its side.
2. SEND_FIN: Send FIN and wait for ACK to be sent by the other host. It tries sending FIN for a maximum of 6 times after which it closes the connection.
3. WAIT_FOR_ACK: It waits for ACK from the other host to arrive. If it does not get ACK within a timeout period, it will move to SEND_FIN to resend the FIN. If it gets ACK within the timeout, it terminates the connection.