BITS Pilani - Hyderabad Campus
CS F303 (Computer Networks)
Second Semester 2023-24, Lab Sheet 7
TCP: Slowstart, Congestion Avoidance and Timeout

1. Overview

In this lab, we will emulate Slowstart, AIMD and Timeout, which are the key features of any TCP protocol. We will use socket programming to utilize sockets for the communication between the sender and the receiver. Additionally, we will also utilize Mininet, to utilize virtualized environment for simulating hosts, switches and routers.

2. Slowstart, Congestion Avoidance and Timeout

A complete Finite State Machine (FSM) description of TCP congestion control is given in Figure below. This FSM shows the description of TCP Reno where TCP Reno does not have Fast Recovery State. For TCP Tahoe, there is no fast recovery, and if there are three duplicate ACKs, the cwnd is reduced to 1.

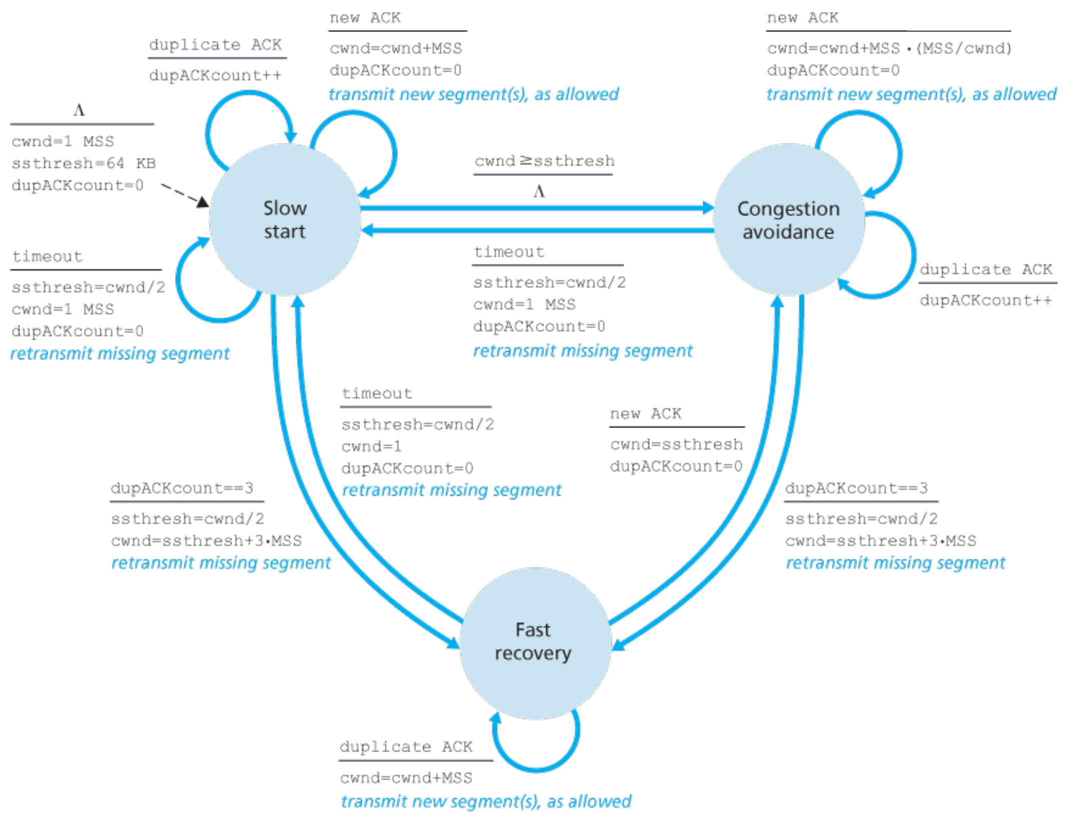

Figure 1 TCP: Finite State Machine

In this work, we will only emulate basic slow start and congestion avoidance.

3. Deployment of Slowstart, Congestion Avoidance and Timeout with Mininet

In this lab, we will be using Mininet to develop a network topology (one sender and one receiver) and implement sliding window protocols. Please follow these steps:

- To setup and use Mininet, follow the previous labsheets.
- Create a standard topology as shown in Figure 3.

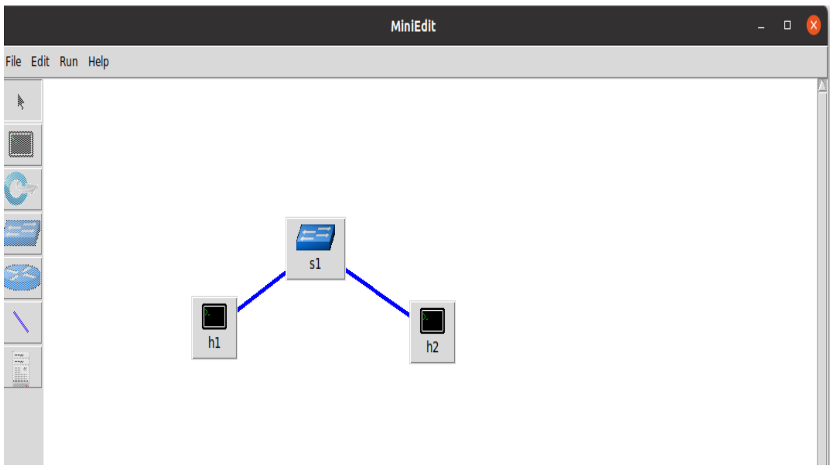

Figure 2 Topology (One sender and one receiver)

- Configure IP addresses of h1 and h2 nodes, and start running mininet.
- Develop C programs for both the sender and receiver functionalities. Place the **sender.c file** on host 1 (h1) and the **receiver.c** file on host 2 (h2). These codes are given in the next section.

Note: Prior to compilation, ensure to input the IP address of the receiver into the sender program. Then, compile both the sender and receiver programs on mininet terminals of both h1 and h2.

4. C programs

(sender.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8081
#define BUFFER_SIZE 1024
```

```
#define INITIAL_CWND 1
#define INITIAL_SSTHRESH 16
// TCP State
enum TCPState {
  SLOW_START,
  CONGESTION_AVOIDANCE,
  TIMEOUT
};
int sock = 0;
int ssthresh = INITIAL_SSTHRESH;
int cwnd = INITIAL_CWND;
enum TCPState state = SLOW_START;
int main() {
  struct sockaddr_in serv_addr;
  char buffer[BUFFER_SIZE] = {0};
  const char *message = "Hello from sender";
  // Create socket
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("Socket creation error");
    exit(EXIT_FAILURE);
  }
  // Initialize sockaddr structure
  memset(&serv_addr, '0', sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(PORT);
  // Convert IPv4 and IPv6 addresses from text to binary form
  if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
    perror("Invalid address/ Address not supported");
    exit(EXIT_FAILURE);
  }
  // Connect to server
  if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    perror("Connection failed");
    exit(EXIT_FAILURE);
  }
  // Send data to server
  while (1) {
    if (state == CONGESTION_AVOIDANCE) {
     cwnd++; //congestion avoidance
     printf("\n The current cwnd size is %d",cwnd);
     send(sock, message, strlen(message), 0);
     printf("Message sent to server: %s\n", message);
```

```
            }
          else {//slow start
          printf("\n The current cwnd size is %d",cwnd);
          for (int i1=0; i1<cwnd; i1++)
          {
                    send(sock, message, strlen(message), 0);
                    printf("Message sent to server: %s\n", message);
          }
          cwnd = 2*cwnd;
          if (cwnd > ssthresh) {
              state = CONGESTION_AVOIDANCE;
              cwnd=cwnd/2;
        }
      }
          sleep(1); // Simulate delay between packets
  }
  // Close the socket
  close(sock);
  return 0;
}
```

(receiver.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8081
#define BUFFER_SIZE 1024
int main() {
  int server_fd, new_socket;
  struct sockaddr_in address;
  int opt = 1;
  int addrlen = sizeof(address);
  char buffer[BUFFER_SIZE] = {0};
  // Create socket file descriptor
  if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
     perror("Socket creation failed");
     exit(EXIT_FAILURE);
  }
  // Forcefully attaching socket to the port
```

```
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
    perror("Setsockopt failed");
    exit(EXIT_FAILURE);
}
// Initialize sockaddr structure
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Bind the socket to the address
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

// Listen for connections
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}
// Accept incoming connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
    perror("Accept failed");
    exit(EXIT_FAILURE);
}
// Receive data from client
while (1) {
    ssize_t bytes_received = read(new_socket, buffer, BUFFER_SIZE);
    if (bytes_received < 0) {
        perror("Read failed");
        exit(EXIT_FAILURE);
    } else if (bytes_received == 0) {
        printf("Connection closed by client\n");
        break; // Exit the loop if the client closes the connection
    } else {
        printf("Message received from client: %s\n", buffer);
        memset(buffer, 0, BUFFER_SIZE); // Clear the buffer
    }
}
// Close sockets
close(new_socket);
close(server_fd);
```

```
    return 0;
}
```

5. Lab Exercise

Modify the program (sender.c) such that you will implement Timeout event (already a part of TCPState in sender.c). When the timeout event occurs, you set sshthresh = cwnd/2 and new cwnd = 1. You can assume in your code that a timeout event occurs when cwnd = 18.