SSN COLLEGE OF ENGINEERING
KALAVAKKAM-603110

Department of Computer Science and Engineering

UCS2501 – COMPUTER NETWORKS-CSE-B-23

III Year CSE - (V Semester)

Title: TCP Segment Communication Simulation

Academic Year 2023-24

Batch: 2021- 2025

Faculty Incharge: SV Jansirani

Project Students:

Y.V.Ojus	3122 21 5001 125
Rohith M	3122 21 5001 085
Rohit Ram	3122 21 5001 086
V.Sanjhay	3122 21 5001 093

INDEX

PROBLEM DEFINITION

The goal of this project is to design and implement a simplified Transmission Control Protocol (TCP) Segment Communication Simulation. TCP is a widely used transport layer protocol responsible for reliable data transmission over a network. In this simulation, we will focus on the allocation of the segment numbers based on the number of bytes transmitted and acknowledgment numbers.

Project Overview:

1. **Socket Programming:**

Implement a client-server model using socket programming in C. The server will be responsible for allocating segment numbers based on the client's data transmission requests.

2. **Segment Allocation Algorithm:**

Develop an algorithm that determines how segment numbers are allocated based on the number of bytes transmitted by the client. Consider factors such as sequence numbers and acknowledgment numbers in the TCP header.

3. **Data Transmission Simulation:**

Simulate data transmission between the client and server. The client should request the allocation of segment numbers based on the number of bytes it intends to transmit, and the server should allocate appropriate segment numbers. The server should also acknowledge received segments.

4. **User Interface:**

Create a simple user interface for both the client and server sides to input necessary parameters and display the allocated segment numbers and acknowledgment information.

5. **Documentation:**

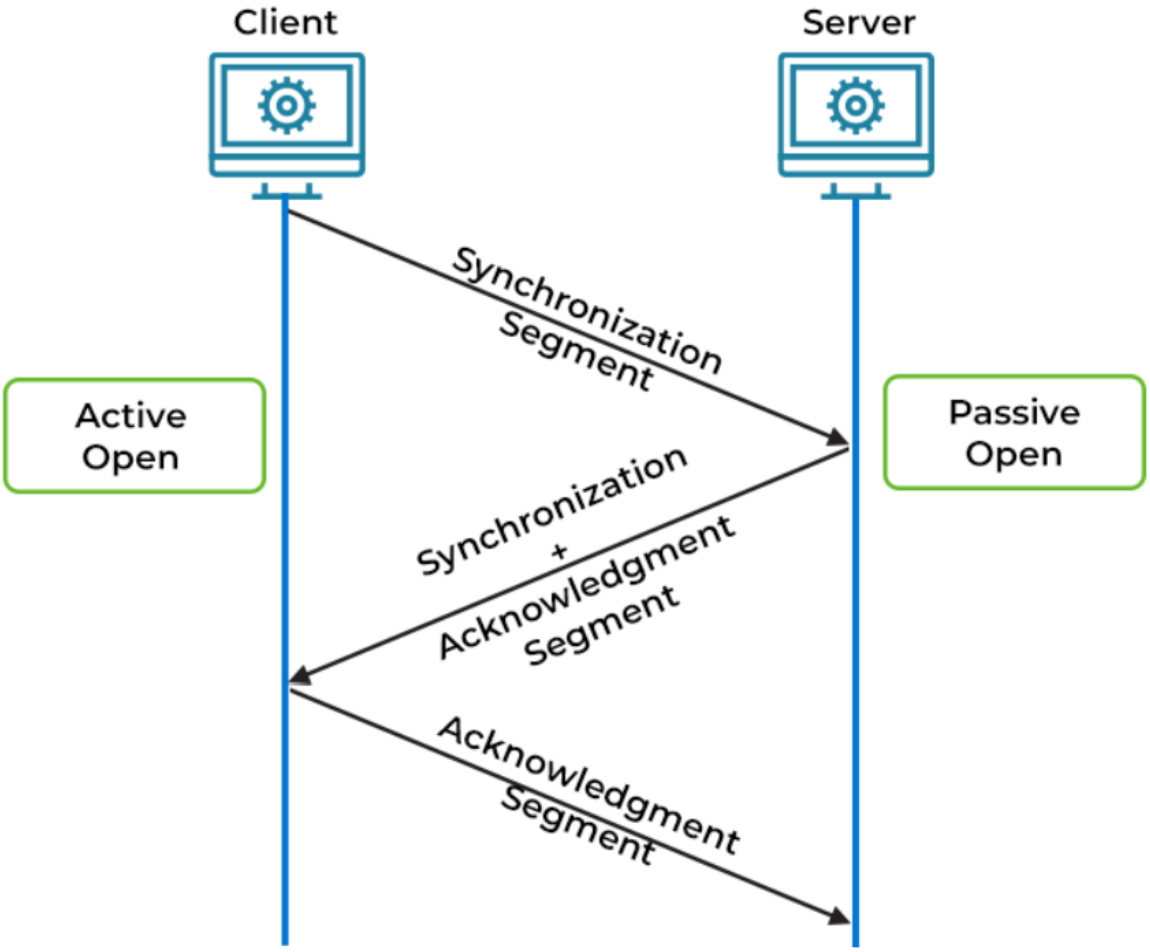
Provide comprehensive documentation that includes the design rationale, algorithms used, implementation details, and instructions for running the simulation.

Deliverables:

- Complete source code for the client and server programs.
- A detailed report documenting the design, implementation, and testing of the simulation.
- Instructions on how to compile and run the simulation.

PROTOCOL AND METHOD EXPLANATION

TCP Three-way Handshaking :



What is TCP Protocol?

Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol. TCP explicitly defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service. TCP uses a combination of GBN and SR protocols to provide reliability. To achieve this goal, TCP uses checksum, retransmission

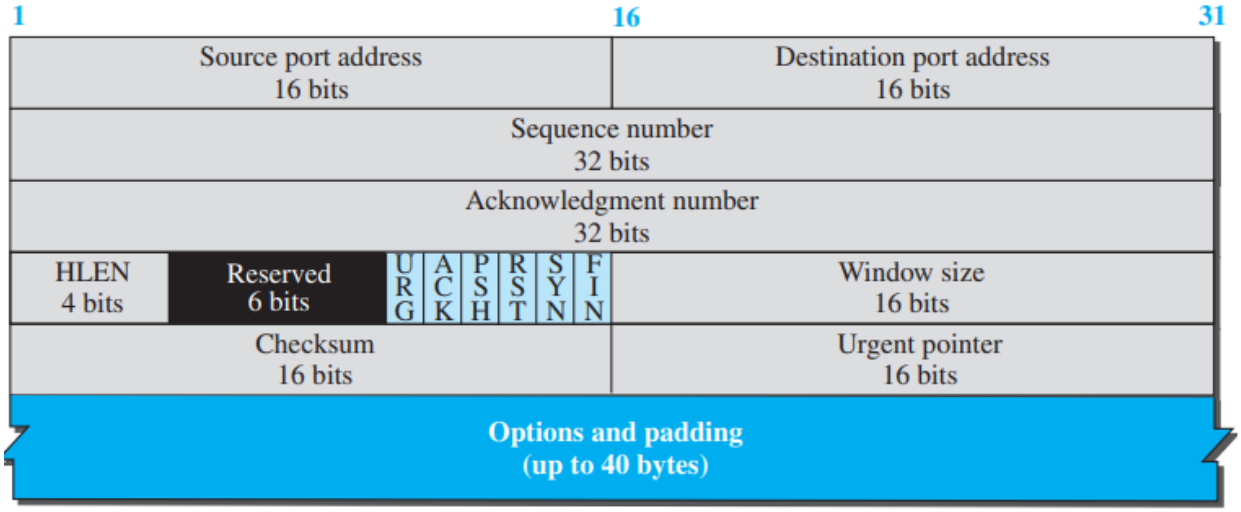
of lost or corrupted packets, cumulative and selective acknowledgments, and timer.TCP is a fundamental protocol in the Internet Protocol Suite (TCP/IP) and is widely used for reliable communication in various applications, such as web browsing, file transfer (FTP), email (SMTP), and more.

Features:

1. **Connection-Oriented**: Connection-oriented protocol, providing a reliable, stream-oriented connection between two devices. It establishes a virtual circuit before data transfer
2. **Reliability:** It uses acknowledgments and retransmission of lost packets to guarantee data integrity.
3. **Flow Control:** TCP incorporates flow control mechanisms to manage the rate of data exchange between sender and receiver.
4. **Full-Duplex Communication:** TCP supports full-duplex communication, allowing data to be transmitted in both directions simultaneously.
5. **Three-Way Handshake:** TCP uses a three-way handshake (SYN, SYN-ACK, ACK) to establish a connection between two devices.
6. **Header Information:** TCP headers include information such as source and destination ports, sequence numbers, acknowledgment numbers, window size, and checksum.
7. **Ordered Delivery:** TCP guarantees the ordered delivery of data packets, ensuring that data arrives at the destination in the same order it was sent.
8. **Acknowledgment:** The receiver acknowledges the receipt of data packets, and the sender retransmits any packets not acknowledged within a specified timeout.
9. **Port Numbers:** TCP uses port numbers to identify specific processes or services on a device, facilitating multiplexing of multiple applications on the same device.
10. **Sliding Window:** TCP uses a sliding window mechanism for flow control, dynamically adjusting the amount of data that can be in transit based on network conditions.

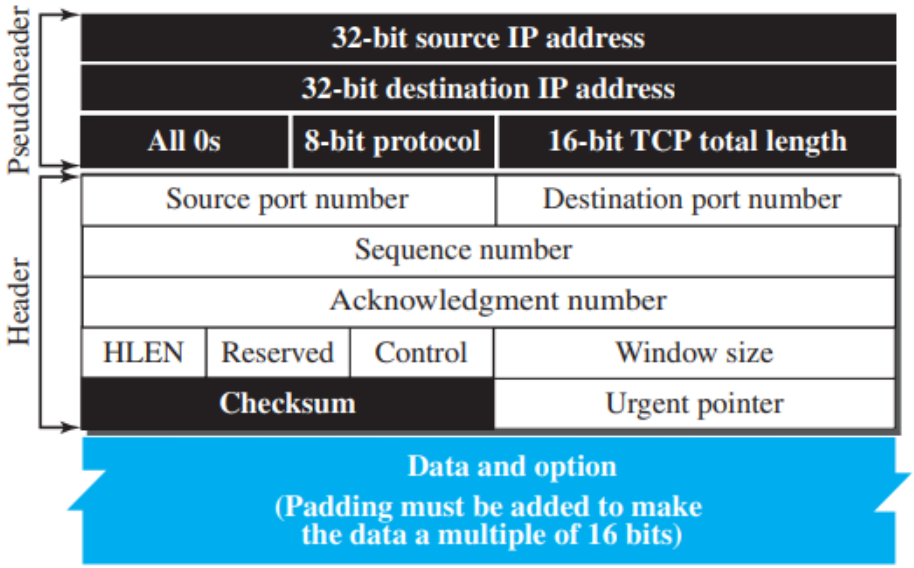
SEGMENT FORMAT:

a. Segment



b. Header

HEADER FORMAT:



METHOD EXPLANATION:

The method mainly involves 3 steps:

1. **SYN (Synchronize) - Client to Server:**
   The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for the synchronization of sequence numbers. The client in our example chooses a random number as the first sequence number and sends this number to the server. This sequence number is called the initial sequence number (ISN). There is no acknowledgment number initially set.

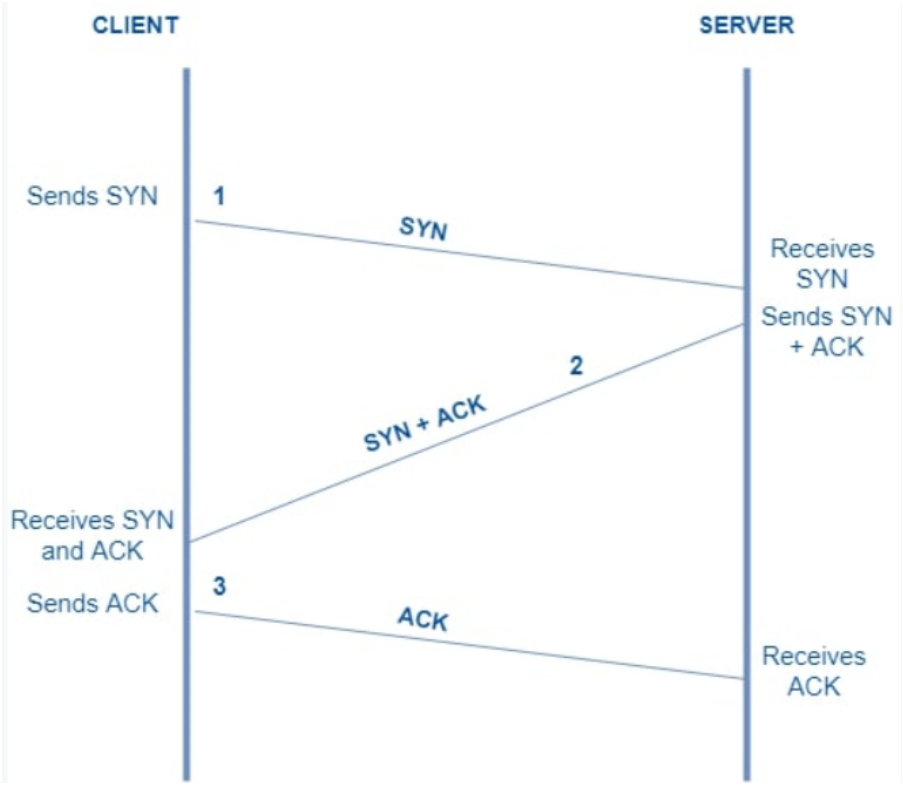
2. **SYN-ACK (Synchronize-Acknowledge) - Server to Client:**
   The server sends the second segment, a SYN + ACK segment with two flag bits set as SYN and ACK. This segment has a dual purpose. First, it is a SYN segment for communication in the other direction. The server uses this segment to initialize a sequence number for numbering the bytes sent from the server to the client. The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client. The segment contains an acknowledgment, it also needs to define the receive window size, rwnd.

3. **ACK (Acknowledge) - Client to Server:**
   The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. The segment consumes as many sequence numbers as the number of data bytes.

During Data Transmission

Sequential Data Transmission: As data is transmitted, each TCP segment's header contains a sequence number, which is the cumulative count of bytes sent. This number identifies the position of the first byte of data in this segment within the entire data stream.

Acknowledgment Process: The receiver, upon getting a segment, sends an ACK back to the sender. This ACK number is the next expected sequence number, indicating that the receiver has successfully received all bytes up to that number and is ready for the next byte in the sequence



Example Scenario

Imagine a client (A) communicating with a server (B):

Initial State: Let's say the window size is set to 1000 bytes.

Data Transmission: A sends 1000 bytes of data to B.

Window Update: After sending the data, A waits for an acknowledgment. During this period, A cannot send more data because its window is full (1000/1000 bytes sent).

Acknowledgment: Once B receives the data, it sends an acknowledgment back to A. This acknowledgment also informs A of B's available window size (e.g., B might still be able to receive another 1000 bytes).

Window Adjustment: Upon receiving the acknowledgment, A's window is updated, allowing it to send more data. If the acknowledgment says B has 1000 bytes of window size left, A can send another 1000 bytes.

CODE

Server:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

// Data structure representing the header of a network segment
struct header {
  char source_port_address[16];
  char destination_port_address[16];
  char sequence_number[32];
  char acknowledgment_number[32];
  char HLEN[4];
  char reserved[6];
  char type[7];
  char window_size[16];
  char checksum[16];
  char urgent_pointer[16];
};

// Data structure representing a network segment
struct segment {
  struct header head;
  char data[1024];
};

// Function to initialize the header with default values
```

```
void initializeHeader(struct header *hdr, char *random_str) {
  strcpy(hdr->source_port_address, "54321");
  strcpy(hdr->destination_port_address, "12345");
  strcpy(hdr->sequence_number, random_str);
  strcpy(hdr->acknowledgment_number, "0");
  strcpy(hdr->HLEN, "5");
  strcpy(hdr->reserved, "000000");
  strncpy(hdr->type, "000010", sizeof(hdr->type) - 1);
  hdr->type[sizeof(hdr->type) - 1] = '\0';
  strcpy(hdr->window_size, "8192");
  strcpy(hdr->checksum, "0000");
  strcpy(hdr->urgent_pointer, "0000");
}

int main(int argc, char *argv[]) {
  srand(time(0));
  char random_number_str[4]; // At least 3 digits for a random number
  snprintf(random_number_str, sizeof(random_number_str), "%d", rand() %
1000);

  // Check for correct number of command-line arguments
  if (argc != 2) {
    fprintf(stderr, "Usage: %s <port>\n", argv[0]);
    exit(1);
  }

  // Extract port number from command-line arguments
  int Port = atoi(argv[1]);

  int server_socket, client_socket;
  struct sockaddr_in server_addr, client_addr;
  socklen_t client_addr_len = sizeof(client_addr);

  // Create a socket for the server
  server_socket = socket(AF_INET, SOCK_STREAM, 0);
  if (server_socket == -1) {
    perror("Socket Creation Failed!");
```

```
    exit(1);
  }

  // Initialize server address structure
  memset(&server_addr, 0, sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(Port);
  server_addr.sin_addr.s_addr = INADDR_ANY;

  // Bind the server socket to the specified port
  if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) ==
-1) {
    perror("Error binding socket");
    exit(1);
  }

  // Listen for incoming connections
  if (listen(server_socket, 5) == -1) {
    perror("Error listening for connections");
    exit(1);
  }

  printf("Server listening on port %d...\n", Port);

  // Accept a connection from a client
  client_socket = accept(server_socket, (struct sockaddr *)&client_addr,
&client_addr_len);
  if (client_socket == -1) {
    perror("Error accepting connection");
    exit(1);
  }

  printf("Connection accepted from %s:%d\n", inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));

  struct segment receivedSegment;
```

```
  // Initialization for 3-way handshake
  ssize_t bytes_received = read(client_socket, &receivedSegment, sizeof(struct
segment));
  if (bytes_received <= 0) {
    printf("Client disconnected.\n");
  }
  printf("\nReceived:\nSource Port - %s\nDestination Port - %s\nSequence "
         "Number - %s\nControl Field - %s\n\n",
         receivedSegment.head.source_port_address,
         receivedSegment.head.destination_port_address,
         receivedSegment.head.sequence_number, receivedSegment.head.type);

  // Set control field for 3-way handshake
  strncpy(receivedSegment.head.type, "010010",
sizeof(receivedSegment.head.type) - 1);
  receivedSegment.head.type[sizeof(receivedSegment.head.type) - 1] = '\0';

  // Prepare acknowledgment and sequence numbers for 3-way handshake
  int temp = atoi(receivedSegment.head.sequence_number) + 1;
  char temp2[50];
  snprintf(temp2, sizeof(temp2), "%d", temp);
  strcpy(receivedSegment.head.acknowledgment_number, temp2);
  strcpy(receivedSegment.head.sequence_number, random_number_str);

  // Prompt for input and send it back to the client
  printf("Enter a window size to send back to the client: ");
  fgets(receivedSegment.head.window_size,
sizeof(receivedSegment.head.window_size), stdin);

receivedSegment.head.window_size[strcspn(receivedSegment.head.window_siz
e, "\n")] = '\0';

  // Send the entire structure (header + data)
  write(client_socket, &receivedSegment, sizeof(struct segment));

  // Receive acknowledgment for 3-way handshake
```

```
  bytes_received = read(client_socket, &receivedSegment, sizeof(struct
segment));
  if (bytes_received <= 0) {
    printf("Client disconnected.\n");
  }
  printf("\nReceived:\nSource Port - %s\nDestination Port - %s\nAcknowledgment
"
         "Number - %s\nControl Field - %s\n\n",
         receivedSegment.head.source_port_address,
         receivedSegment.head.destination_port_address,
         receivedSegment.head.acknowledgment_number,
receivedSegment.head.type);

  // Continuous communication
  printf("Connection Established...Data Transfer Begins...\n\n");
  while (1) {
    // Receive data from the client
    ssize_t bytes_received = read(client_socket, &receivedSegment, sizeof(struct
segment));
    if (bytes_received <= 0) {
      printf("Client disconnected.\n");
      break;
    }

    printf("\nReceived:\nSource Port - %s\nDestination Port - %s\nSequence"
           " Number - %s\nData - %s\n\n",
           receivedSegment.head.source_port_address,
           receivedSegment.head.destination_port_address,
           receivedSegment.head.sequence_number, receivedSegment.data);

    // Check for the termination condition
    if (strcmp(receivedSegment.data, "exit") == 0) {
      printf("Received 'exit' from client. Terminating server.\n");
      break;    }

    // Update acknowledgment based on received data
    int new_ack = atoi(receivedSegment.head.sequence_number) +
```

```
                (strlen(receivedSegment.data) * 50);
    char temp[50];
    snprintf(temp, sizeof(temp), "%d", new_ack);
    strcpy(receivedSegment.head.acknowledgment_number, temp);

    // Prompt for input and send it back to the client
    printf("Enter a window size to send back to the client: ");
    fgets(receivedSegment.head.window_size,
sizeof(receivedSegment.head.window_size), stdin);

receivedSegment.head.window_size[strcspn(receivedSegment.head.window_siz
e, "\n")] = '\0';

    // Send the entire structure (header + data)
    write(client_socket, &receivedSegment, sizeof(struct segment));
  }

  // Close sockets
  close(client_socket);
  close(server_socket);
  return 0;
}
```

Client:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
```

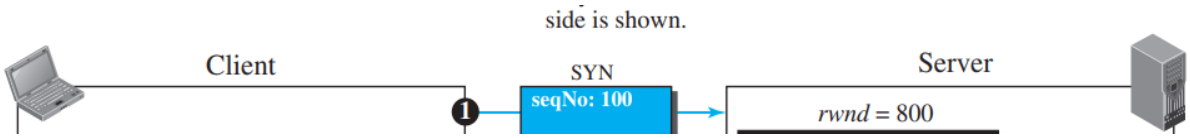
```
// Data structure representing the header of a network segment
struct header {
  char source_port_address[16];
  char destination_port_address[16];
  char sequence_number[32];
  char acknowledgment_number[32];
  char HLEN[4];
  char reserved[6];
  char type[7];
  char window_size[16];
  char checksum[16];
  char urgent_pointer[16];
};

// Data structure representing a network segment
struct segment {
  struct header head;
  char data[1024];
};

// Function to initialize the header with default values
void initializeHeader(struct header *hdr, char *random_str) {
  strcpy(hdr->source_port_address, "12345");
  strcpy(hdr->destination_port_address, "54321");
  strcpy(hdr->sequence_number, random_str);
  strcpy(hdr->acknowledgment_number, "0");
  strcpy(hdr->HLEN, "5");
  strcpy(hdr->reserved, "000000");
  strncpy(hdr->type, "000010", sizeof(hdr->type) - 1);
  hdr->type[sizeof(hdr->type) - 1] = '\0';
  strcpy(hdr->window_size, "8192");
  strcpy(hdr->checksum, "0000");
  strcpy(hdr->urgent_pointer, "0000");
}

int main(int argc, char *argv[]) {
  srand(time(0));
```

```
  char random_number_str[4]; // At least 3 digits for a random number
  snprintf(random_number_str, sizeof(random_number_str), "%d", rand() %
1000);

  // Check for correct number of command-line arguments
  if (argc != 2) {
    fprintf(stderr, "Usage: %s <port>\n", argv[0]);
    exit(1);
  }

  // Extract port number from command-line arguments
  int Port = atoi(argv[1]);

  int client_socket;
  struct sockaddr_in server_addr;

  // Create a socket for the client
  client_socket = socket(AF_INET, SOCK_STREAM, 0);
  if (client_socket == -1) {
    perror("Socket Creation Failed!");
    exit(1);
  }

  // Initialize server address structure
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(Port);
  server_addr.sin_addr.s_addr = INADDR_ANY;

  // Connect to the server
  if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr))
== -1) {
    perror("Socket Connect Failed!");
    exit(1);
  }

  struct segment tcpSegment;
  initializeHeader(&tcpSegment.head, random_number_str);
```

```
  // Initialization
  // Send the entire structure (header + data)
  write(client_socket, &tcpSegment, sizeof(struct segment));

  // Receive acknowledgment for the initialization
  ssize_t bytes_received = read(client_socket, &tcpSegment, sizeof(struct
segment));
  if (bytes_received <= 0) {
    printf("Server disconnected.\n");
  }
  char new_seq[50];
  strcpy(new_seq, tcpSegment.head.acknowledgment_number);
  printf(
      "\nReceived:\nSource Port - %s\nDestination Port - %s\nSequence Number "
      "- %s\nAcknowledgment Number - %s\nControl Field - %s\nWindow Size - "
      "%s\n\n",
      tcpSegment.head.source_port_address,
      tcpSegment.head.destination_port_address,
tcpSegment.head.sequence_number,
      tcpSegment.head.acknowledgment_number, tcpSegment.head.type,
      tcpSegment.head.window_size);

  int ackno = atoi(tcpSegment.head.sequence_number) + 1;
  char temp2[50];
  snprintf(temp2, sizeof(temp2), "%d", ackno);
  strcpy(tcpSegment.head.acknowledgment_number, temp2);

  // Send the entire structure (header + data)
  write(client_socket, &tcpSegment, sizeof(struct segment));

  // Continuous communication
  printf("Connection Established...Data Transfer Begins...\n\n");
  strcpy(tcpSegment.head.sequence_number, new_seq);
  while (1) {
    // Prompt user for a message
    printf("Enter a message : ");
```

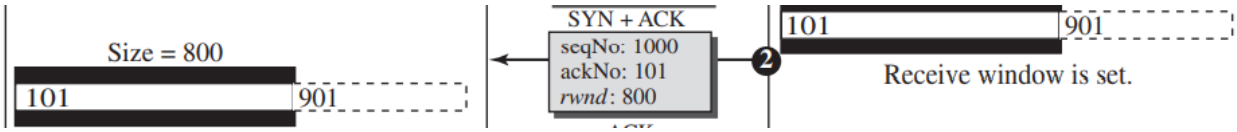
```
    fgets(tcpSegment.data, sizeof(tcpSegment.data), stdin);
    tcpSegment.data[strcspn(tcpSegment.data, "\n")] = '\0';

    // Check for the exit condition
    if (strcmp(tcpSegment.data, "exit") == 0) {
      printf("Terminating client.\n");
      break;
    }

    // Send the entire structure (header + data)
    write(client_socket, &tcpSegment, sizeof(struct segment));

    // Wait for the server's response
    ssize_t bytes_received = read(client_socket, &tcpSegment, sizeof(struct
segment));
    if (bytes_received <= 0) {
      printf("Server disconnected.\n");
      break;
    }

    printf("\nReceived:\nSource Port - %s\nDestination Port - "
           "%s\nAcknowledgment Number - %s\nWindow"
           " Size - %s\n\n",
           tcpSegment.head.source_port_address,
           tcpSegment.head.destination_port_address,
           tcpSegment.head.acknowledgment_number,
tcpSegment.head.window_size);

    // Update the sequence number
    strcpy(tcpSegment.head.sequence_number,
           tcpSegment.head.acknowledgment_number);
  }

  // Close the client socket
  close(client_socket);
  return 0;
}
```

OUTPUT




```
Server listening on port 1234...
Connection accepted from 127.0.0.1:55472

Received:
Source Port - 12345
Destination Port - 54321
Sequence Number - 53
Control Field - 000010
```

Client Sending Initial Sequence Number Along With SYN Field Set



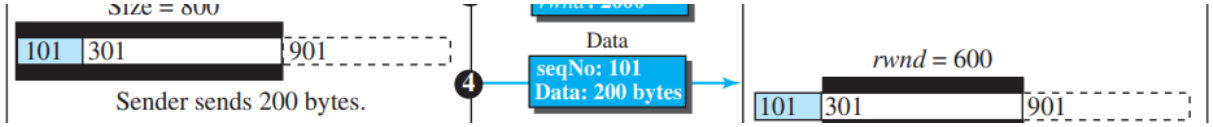

```
Received:
Source Port - 12345
Destination Port - 54321
Sequence Number - 63
Acknowledgment Number - 54
Control Field - 010010
Window Size - 200
```

Server Sends Back seqNo, ackNo, and Window Size Along With SYN + ACK


```
Received:
Source Port - 12345
Destination Port - 54321
Acknowledgment Number - 64
Control Field - 010000
```

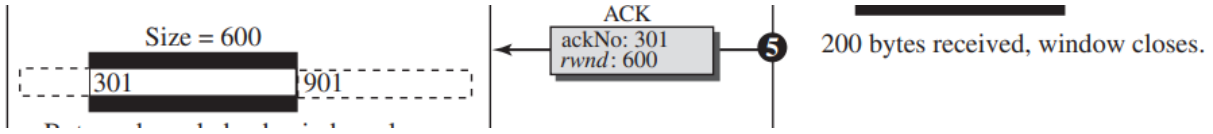
Client Send Acknowledgement By Incrementing Received Sequence Number

Data Transmission Begins



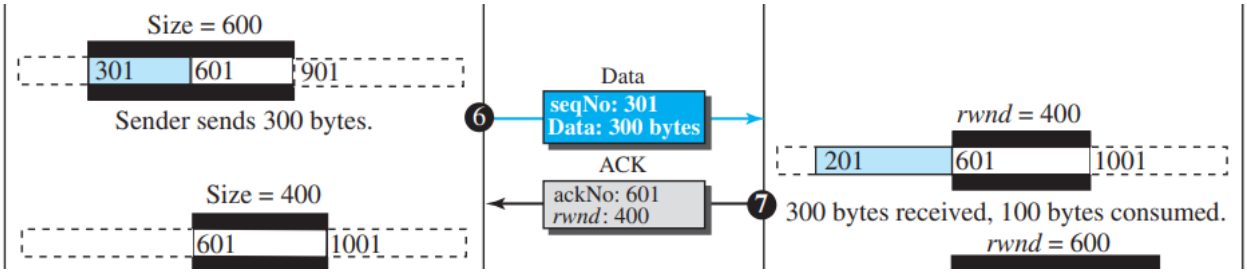

```
Received:
Source Port - 12345
Destination Port - 54321
Sequence Number - 54
Data - hi
```

The Sequence Number is Set to Received Acknowledgement Number




```
Received:
Source Port - 12345
Destination Port - 54321
Acknowledgment Number - 154
Window Size - 200
```

Acknowledgment Number is determined by multiplying the Length of Data by 50 and Adding To the Sequence Number




```
Received:
Source Port - 12345
Destination Port - 54321
Sequence Number - 154
Data - cat
```

Server's Side

```
Received:
Source Port - 12345
Destination Port - 54321
Acknowledgment Number - 304
Window Size - 200
```

Client's Side

LEARNING OUTCOMES

- **Understanding TCP Protocol:**

Gain a deep understanding of the Transmission Control Protocol (TCP), including its key features, header structure, and the role of sequence numbers and acknowledgment numbers in reliable data transmission.

- **Socket Programming Proficiency:**

Develop proficiency in socket programming using the C language. Understand how to create client-server applications for communication over a network.

- **Segment Number Allocation Algorithm:**

Design and implement an algorithm for the allocation of segment numbers based on the number of bytes transmitted. Consider the implications of sequence numbers and acknowledgment numbers in the TCP header.

- **Data Transmission Simulation:**

Simulate the process of data transmission between a client and server, including the initiation of communication, segment allocation, acknowledgment, and error handling.

- **User Interface Design:**

Design a simple and user-friendly interface for both the client and server sides, facilitating input of parameters and displaying relevant information about allocated segment numbers and acknowledgments.

- **Documentation Skills:**

Develop documentation skills by creating a comprehensive report that covers the design choices, algorithms used, implementation details, and instructions for running the simulation.

README FILE

Overview

The goal of this project is to design and implement a simplified Transmission Control Protocol (TCP) Segment Communication Simulation. TCP is a widely used transport layer protocol responsible for reliable data transmission over a network. In this simulation, we will focus on the allocation of the segment numbers based on the number of bytes transmitted and acknowledgment numbers.

Requirements

- C compiler (e.g GCC)
- Operating System with Socket Support (Linux)

Getting started

1. Clone the repository

git clone <project repository link>

2. Navigate to the project directory

cd tcp-segment-communication-simulation

Alternatively

Download The Files Locally And Follow As Given Below

3. Compile the client and server programs

gcc server.c -o server
gcc client.c -o client

4. Run the server

./server 1234

5. Run the client

./client 1234

Usage

- The client and server programs have interactive interfaces for inputting parameters and displaying allocated segment numbers and acknowledgments.

- Follow the on-screen prompts to initiate communication, request segment allocation, and simulate data transmission.

Algorithm

An algorithm that determines how segment numbers are allocated based on the number of bytes transmitted by the client. Considered factors such as sequence numbers and acknowledgment numbers in the TCP header.

User Interface

Created a simple user interface for both the client and server sides to input necessary parameters and displayed the allocated segment numbers and acknowledgment information.

Documentation

Provided comprehensive documentation that includes the design rationale, algorithms used, implementation details, and instructions for running the simulation.

GITHUB LINK

GitHub Link