SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

III year CSE – V Semester

UCS 2501 Computer Networks

Team Project

Title :SELECTIVE REPEAT PROTOCOL

Academic year : 2023-2024

Batch: 2021 - 2025

Faculty Incharge: Dr. S.V.Jansi Rani

Team members

Sham S	– 3122 21 5001 099
Sridhar K	– 3122 21 5001 105
Sanjay B	– 3122 21 5001 311
Roshan R	– 3122 21 5001 310

INDEX

PROBLEM STATEMENT:-

Design and implement the Selective Repeat Automatic Repeat reQuest (ARQ) protocol using any programming language of your choice. Selective Repeat ARQ is a sliding window protocol employed in data communication to ensure reliable and error-free transmission of data over a network. The objective is to create a simulation of the Selective Repeat ARQ mechanism, encompassing both the sender and receiver components.

Key requirement

1. Sender Module

Implement a sender module responsible for dividing the data into frames and transmitting them to the receiver.Utilize a sliding window approach for selective repeat, where the sender keeps track of sent frames and maintains a window of frames that can be transmitted without waiting for an acknowledgment.

2. Receiver Module

Develop a receiver module that acknowledges received frames and maintains a window of frames it is ready to accept.Handle out-of-sequence frames and ensure that they are properly reordered before passing them to the higher layers.

3. Error Handling

Simulate a realistic communication environment by introducing errors during data transmission. Implement error detection and correction mechanisms to ensure the reliability of the communication.

4. Timeout Mechanism

Implement a timeout mechanism in the sender module to retransmit frames that are not acknowledged within a specified time period.

5. User Interface

Create a user interface or command-line interface to allow users to input parameters such as the size of the sliding window, the transmission rate, and the probability of frame corruption.

6. Statistics and Logging

Provide functionality to collect and display statistics on the number of transmitted frames, acknowledged frames, retransmitted frames, and any other relevant metrics.Implement logging mechanisms to record the sequence of events during the simulation.

7. Documentation

Include detailed documentation describing the design choices, algorithms used, and instructions for running the simulation.Clearly specify how to set up the simulation environment and any dependencies required.

8. Testing

Develop a comprehensive testing suite to validate the correctness and reliability of the Selective Repeat ARQ implementation.Include scenarios with various window sizes, transmission rates, and error probabilities to assess the robustness of the protocol.

9. Code Quality

Write clean and well-documented code following best practices in software development.

What is selective repeat ARQ?

Selective Repeat Automatic Repeat reQuest (ARQ) is a type of sliding window protocol used in data communication to ensure reliable and error-free transmission of data over a network. The selective repeat ARQ protocol is an extension of the basic automatic repeat reQuest (ARQ) protocol, enhancing its efficiency by allowing the receiver to selectively acknowledge individual frames within a window.

FTP (File Transfer Protocol) is a standard network protocol used to transfer files from one host to another over a TCP-based network, such as the internet. Selective Repeat is a type of automatic repeat request (ARQ) protocol used in networking to manage the flow of data and ensure reliable delivery.

1. **FTP (File Transfer Protocol):**
    - FTP operates on the application layer of the OSI model.
    - It uses a client-server architecture, where the client initiates a connection to the server to request file transfers.
    - FTP can work in either active or passive mode, depending on how data connections are established.
2. **Selective Repeat Protocol:**
    - Selective Repeat is a specific type of automatic repeat request (ARQ) protocol used in networking to ensure reliable data transfer over unreliable channels.
    - In Selective Repeat, the sender is allowed to transmit multiple frames (blocks of data) before receiving an acknowledgment.
    - The receiver keeps track of the frames it has received correctly and sends selective acknowledgments for those frames.
    - If a frame is not received correctly, the receiver discards it and requests the sender to retransmit only that specific frame.

Now, let's see how FTP can use the Selective Repeat protocol:

- **Connection Establishment:**
    - The client establishes a connection with the FTP server.
    - The client and server negotiate parameters for data transfer, including the use of Selective Repeat for reliable data transfer.
- **File Transfer:**
    - When the client wants to upload or download a file, it breaks the file into smaller frames.
    - These frames are then transmitted to the server using the Selective Repeat protocol.
- **Selective Repeat Operation:**

    - The server receives the frames and acknowledges them selectively, indicating which frames were received successfully.
    - If the server detects a frame with errors, it requests the client to retransmit only that specific frame.
- **Flow Control:**
    - Selective Repeat helps in managing the flow of data between the client and server.
    - It allows for efficient utilization of network resources by enabling the sender to transmit multiple frames without waiting for individual acknowledgments.

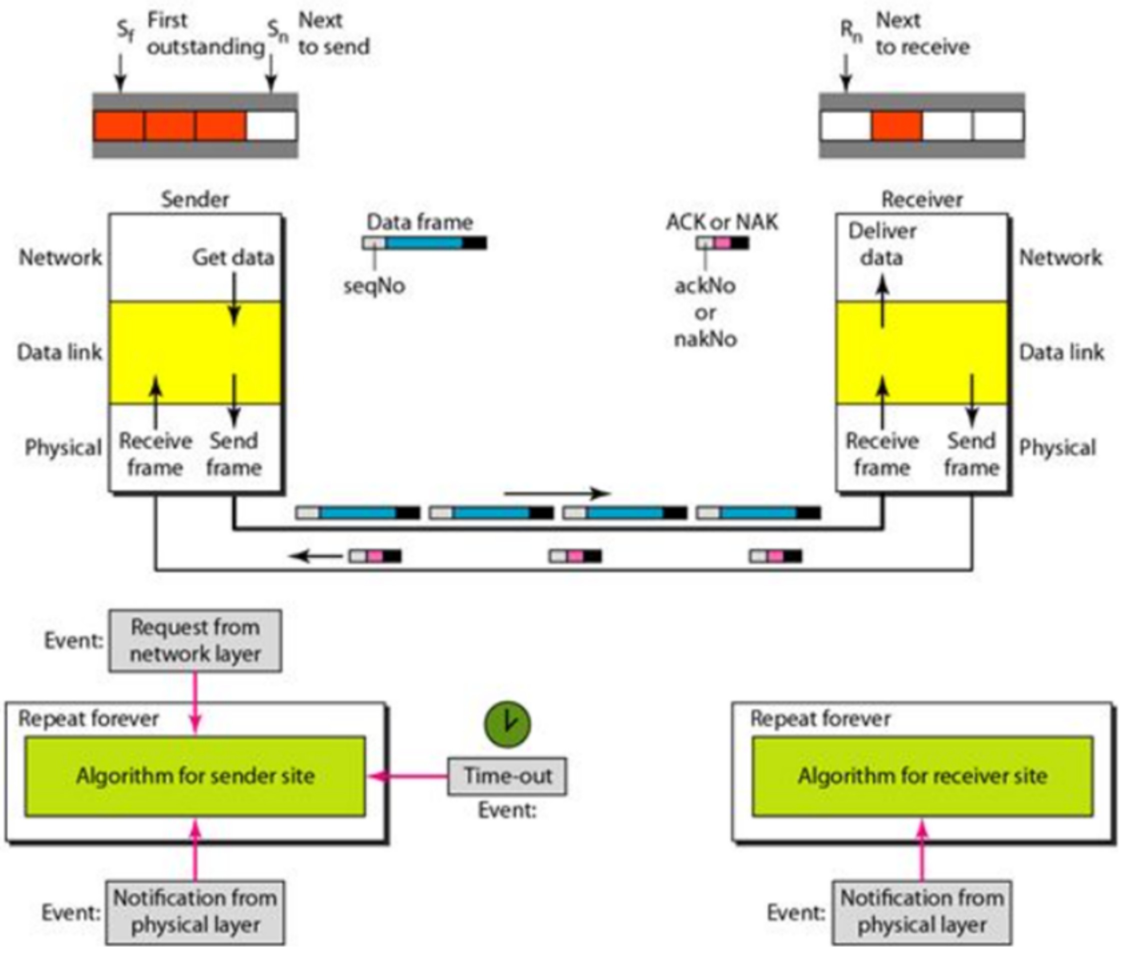
Working process of selective repeat

1. **Connection Establishment:**
    - **Client-Server Handshake:**
        - The client initiates a connection with the FTP server.
        - The server responds, and they negotiate parameters for the data transfer, including the use of Selective Repeat.
2. **File Transfer Initialization:**
    - **Client Request:**
        - The client requests to upload or download a file.
        - The server grants permission and prepares for the data transfer.
    - **Frame Division:**
        - The client divides the file into smaller frames for transmission.
3. **Selective Repeat Operation:**
    - **Frame Transmission:**
        - The client begins transmitting frames to the server.
        - The server receives the frames.
    - **Selective Acknowledgment:**
        - The server selectively acknowledges the frames it has successfully received without errors.
        - If a frame is received with errors, the server discards it.
    - **Error Detection and Retransmission:**
        - If the server detects an error in a frame, it requests the client to retransmit only that specific frame.
        - The client retransmits the requested frame.
4. **Flow Control:**
    - **Efficient Data Flow:**
        - Selective Repeat allows the client to continue transmitting frames without waiting for acknowledgments for each frame.
        - This improves the overall efficiency of data transfer.
5. **Completion and Closure:**
    - **File Assembly:**

        - The server, upon receiving all frames without errors, assembles the frames to reconstruct the original file.
    - **Final Acknowledgment:**
        - The server sends a final acknowledgment to the client, indicating the successful completion of the file transfer.
    - **Connection Closure:**
        - The client and server may then close the FTP connection.

In summary, FTP using the Selective Repeat protocol involves the establishment of a connection, initiation of file transfer with frame division, transmission of frames with selective acknowledgment, error detection and retransmission, efficient flow control, and finally, the completion and closure of the connection. This process ensures reliable and efficient file transfer over the network.

Block diagram of selective repeat ARQ



Selective repeat ARQ method explanation

Selective Repeat Automatic Repeat reQuest (ARQ) is a specific type of sliding window protocol used in data communication to ensure reliable and error-free transmission of data between a sender and a receiver over a network. The protocol allows for the selective retransmission of only those frames that are either lost or received with errors, rather than retransmitting the entire window of frames. Here is an explanation of how Selective Repeat ARQ works:

1. Sliding Window

- Both the sender and the receiver maintain a window that represents a range of sequence numbers for frames. This window slides as new frames are transmitted and acknowledged.

2. **Frame Transmission:**

- The sender divides the data into frames, each of which is assigned a unique sequence number. These frames are then transmitted to the receiver.

### 3. **Acknowledgment (ACK):**

- When the receiver successfully receives a frame without errors, it sends an acknowledgment (ACK) back to the sender. The ACK contains the sequence number of the next expected frame.

### 4. **Selective Acknowledgment:**

- In Selective Repeat ARQ, the receiver can individually acknowledge the correct receipt of specific frames, even if they are not in sequence. This is in contrast to other protocols like Go-Back-N, where the receiver acknowledges only the next expected frame.

### 5. **Negative Acknowledgment (NAK):**

- If the receiver detects an error in a received frame or if a frame is missing, it can send a Negative Acknowledgment (NAK) to the sender. The NAK indicates the sequence number of the frame that needs to be retransmitted.

### 6. **Sender Actions on ACK or NAK:**

- Upon receiving an ACK, the sender slides the window forward, acknowledging the successful transmission of frames up to the acknowledged sequence number.

- Upon receiving a NAK, the sender retransmits only the frame indicated by the NAK.

### 7. **Timeout and Retransmission:**

- The sender employs a timeout mechanism. If it does not receive an ACK within a specified time, it assumes that a frame was lost and retransmits the frames in the current window.

- The timeout mechanism helps in handling lost frames, and selective retransmission ensures efficiency.

### 8. Handling Out-of-Sequence Frames:

- The receiver buffers out-of-sequence frames until the missing frames are received. It then delivers the data to the higher layers in the correct order.

### 9. **Efficiency**

- Selective Repeat ARQ is more efficient than protocols like Go-Back-N in scenarios where the network conditions may lead to occasional frame loss or corruption. It minimizes unnecessary retransmissions, utilizing the network bandwidth more effectively.

Selective Repeat ARQ is widely employed in various communication protocols, such as the Transmission Control Protocol (TCP) in computer networks, to achieve reliable data transmission in the presence of errors.

CODE:-

Server:

```
// Server side C/C++ program to demonstrate Socket programming
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <stdbool.h>
#include <pthread.h>


#define VIDEO_FILE "files/outUDP.mp4"
#define PORT 8080
#define BUFFER_SIZE 500
#define WINDOW_SIZE 5


struct Packet {
   int seqNum;
   int size;
   char data[BUFFER_SIZE];
   bool sent;
   bool received;
   bool ack;
   bool eof;
};

struct ackPacket
{
  int seqNum;
  int size;
  bool eof;
};


int n_pkt_wait = 0;
int num_of_acked_pkts = 0;
int n;
int servSocket;
struct sockaddr_in address;
int addrlen = sizeof(address);
char buffer[BUFFER_SIZE];
FILE *output_file;

int seq_num = 0;
struct ackPacket ack;
int last_acked_in_sequence;
bool eof_reached;
bool eof_reached = false;

struct Packet window[WINDOW_SIZE];


void rescOverR_Udp();
void recvFileOverSR();
void threadSendAck();
void initWindowSeq();
void slideWindow(int steps);
void addPacket();



void socketConnection() {

   servSocket = socket(AF_INET, SOCK_DGRAM, 0);

   if (servSocket < 0) {
       perror("[-]socket failed");
       exit(EXIT_FAILURE);
   }
   else {
       puts("[+]Socket created");
   }

   // Client config
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = INADDR_ANY;
   address.sin_port = htons( PORT );

   // Binding
   if (bind(servSocket, (struct sockaddr *)&address, sizeof(address))<0) {
       perror("[-]bind failed");
       exit(EXIT_FAILURE);
   }

}

/**
* operations to receive udp reliably
*/
void rescOverR_Udp() {
   n_pkt_wait = WINDOW_SIZE;
   initWindowSeq();

   rescAgain:
   recvFileOverSR();

   // if all pkts acked else send acks
   if (num_of_acked_pkts < WINDOW_SIZE-1) {
       n_pkt_wait = WINDOW_SIZE - num_of_acked_pkts;
       goto rescAgain;
   }

   // if eof not reached then continue
   if (eof_reached == false) {
       printf("\t~Sliding window\n");
       slideWindow(WINDOW_SIZE);
       rescOverR_Udp();
   }

   // eof reached
   else {
       slideWindow(n_pkt_wait+1);
   }
}

/**
* init a pkt array to be received
*/
void initWindowSeq() {
   seq_num = 0;
   for (int i = 0; i < WINDOW_SIZE; i++) {
       struct Packet newPkt;
       newPkt.seqNum = seq_num;
       newPkt.received = false;
       seq_num++;
       window[i] = newPkt;
   }
}

/**
* send the acks for the packets
* that are received accordingly
* --> ignore the duplicate packets
*/
void recvFileOverSR() {
  for (int i = 0; i < n_pkt_wait; i++) {
       struct Packet newPacket;
       newPacket.eof = false;

       n = recvfrom(servSocket, &newPacket, sizeof(newPacket), MSG_WAITALL, ( struct sockaddr *)
&address, &addrlen);

       int num = newPacket.seqNum;

       // if the pkt is already recieved then ignore it
       if (!window[num].received)
       {
           printf("packet received: %d\n", num);
           window[num] = newPacket;
           window[num].received = true;
           struct ackPacket ack;
           ack.seqNum = num;
           ack.eof = newPacket.eof;

           int n = sendto(servSocket, &ack, sizeof(ack), MSG_CONFIRM, (const struct sockaddr *)
&address, addrlen);
           window[num].ack = true;
           printf("Ack sent: %d\n",num);
           num_of_acked_pkts++;
       }

       // send ack for duplicate packets
       else if (window[num].received) {
           struct ackPacket ack;
           ack.seqNum = num;
           ack.eof = newPacket.eof;
           int n = sendto(servSocket, &ack, sizeof(ack), MSG_CONFIRM, (const struct sockaddr *)
&address, addrlen);
       }

       if (newPacket.eof == true) {
           printf("eof seq num %d\n", newPacket.seqNum);
           eof_reached = true;
           n_pkt_wait = newPacket.seqNum;
           break;
       }

       if (newPacket.size < 0) // Error
       perror("[-]ERROR writing to socket");
  }
}

/**
* pushing a new packet to the window arr
*/
void addPacket() {
  /* shifting array elements */
  n = WINDOW_SIZE;
  for(int i=0; i < n-1; i++){
     window[i] = window[i+1];
  }

  struct Packet nullPkt;
  nullPkt.seqNum = -2;
  window[n-1] = nullPkt;
}

/**
* slides the window
* --> init with new packets
* --> write down the previous packets on the file
*/
void slideWindow(int steps) {
  for (int i = 0; i < steps; i++) {
       struct Packet packet = window[i];
       printf("writting: %d\n", packet.seqNum);

       size_t num_write = fwrite(&packet.data, 1, packet.size, output_file);

       if (num_write == 0) {
           packet.eof = true;
           break;
       }
   }
```

```
    for (int i = 0; i < WINDOW_SIZE; i++) {
        addPacket();
    }
}



int main(int argc, char const *argv[])
{
    puts("SELECTIVE REPEAT PROTOCOL WITH FTP");
    socketConnection();
    puts("\t~Client found");
    puts("\t~Receiving file from the clinet.");

    output_file = fopen(VIDEO_FILE, "wb");

    rescOverR_Udp();

    puts("\t~File is received");
    if (output_file != NULL)
        fclose(output_file);
    close(servSocket);
    puts("\t~Closing Connection.");

    return 0;
}
```

Client:

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/sendfile.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <pthread.h>
#include <time.h>


#define VIDEO_FILE "files/sampleVideo.mp4"
#define PORT 8080
#define SERVER_ADDR "127.0.0.1"
#define BUFFER_SIZE 500
```

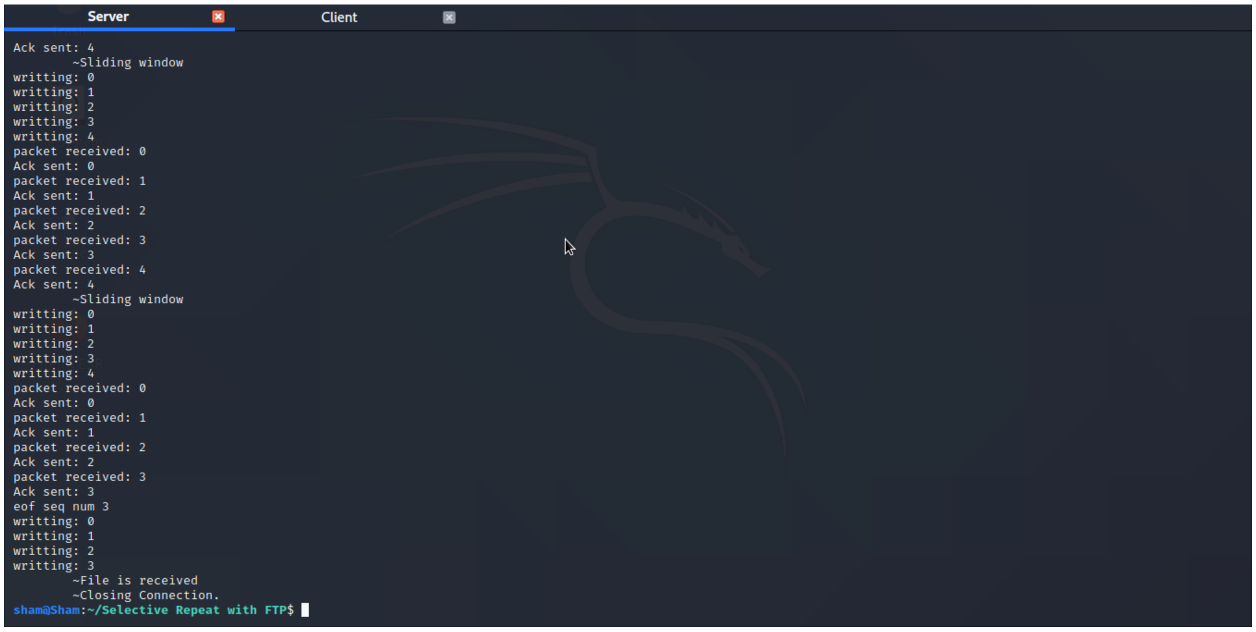
```c
#define WINDOW_SIZE 5




struct Packet {
  int seqNum;
  int size;
  char data[BUFFER_SIZE];
  bool sent;
  bool received;
  bool ack;
  bool eof;
};

struct ackPacket
{
  int seqNum;
  int size;
  bool eof;
};





FILE *fp;
int clientSocket;
struct sockaddr_in server;
int addrlen = sizeof(server);
char buffer[BUFFER_SIZE];
int num_of_ack_recieved = 0;

int ack_of_eof = 0;
int acks_to_wait = WINDOW_SIZE - 1;
int n;
int seq_num = 0;
bool eof_reached;
int eof_set = 0;

struct Packet window[WINDOW_SIZE];

/**
* Initializing functions
*/
```

```c
void sendOverR_UDP();
void sendOverUDP();
void thread_waiting();
void wait_ack();
void addPacket(struct Packet newPacket);
void slideWindow(int steps);



/**
* initializing socket connection
*/
int socketConnection() {

  clientSocket = socket(AF_INET, SOCK_DGRAM, 0);

  if (clientSocket < 0) {
     perror("[-]Couldn't creat socket!");
  }

  // server config
  server.sin_addr.s_addr = inet_addr(SERVER_ADDR);
  server.sin_family = AF_INET;
  server.sin_port = htons(PORT);
}

/**
* Reliable udp sender
*/
void sendOverR_UDP() {
  acks_to_wait = WINDOW_SIZE;

  RESEND:
  sendOverUDP();
  wait_ack();

  // if all acks not received send again
  if (num_of_ack_recieved < acks_to_wait) {
     goto RESEND;
  }

  // if eof then stop
  if (eof_reached == false) {
     printf("\t~Sliding window\n");
```

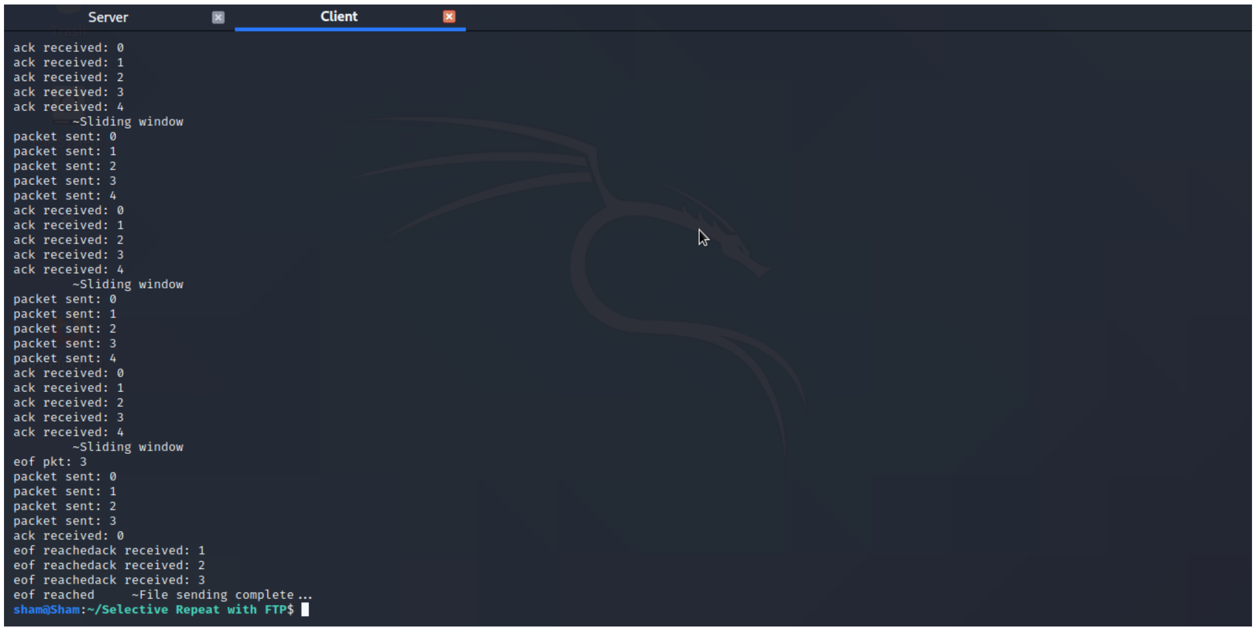
```
      slideWindow(WINDOW_SIZE);
      sendOverR_UDP();
   }
}


/**
* sending the data over udp socket
*/

void sendOverUDP() {
  for (int i = 0; i < WINDOW_SIZE; i++) {
     if (window[i].ack == false){

        n = sendto(clientSocket, &window[i], sizeof(window[i]), MSG_CONFIRM,
(const struct sockaddr *) &server, addrlen);

        printf("packet sent: %d\n", window[i].seqNum);

        if (window[i].size == 0) // end of file.
           break;

        else if (n == 0)
           break;
     }
  }
}


/**
* waiting for the acks of the
* packets that has been sent
*/
void wait_ack(){
  int n;
  for (int i = 0; i < acks_to_wait; i++)
  {
     wait_for_ack: ;
     struct ackPacket newAck;
     n = recvfrom(clientSocket, &newAck, sizeof(&newAck), 0,  (struct sockaddr
*) &server, &addrlen);
     if (n < 0)
           perror("ERROR reading from socket");
     int ack_num = newAck.seqNum;

     // matching ack with expected ack
     for (int i = 0; i < WINDOW_SIZE; i++)
     {
        if (window[i].seqNum == ack_num) {

           // if already acked
           if (window[i].ack == true) {
              goto wait_for_ack;
           }
           else {
              window[i].ack = true;
              // printf("eof: %d\n", newAck.eof);
              printf("ack received: %d\n", ack_num);

              //if eof reached
              if (eof_set > 0) {
                 newAck.seqNum = eof_set;
                 printf("eof reached");
                 eof_reached = true;
                 acks_to_wait = newAck.seqNum+1;
              }
              num_of_ack_recieved++;
           }
        }
     }
  }
}

/**
* pushing packet to the array
*/
void addPacket(struct Packet newPacket) {
  /* shifting array elements */
  n = WINDOW_SIZE;
  for(int i=0; i < n-1; i++){
     window[i] = window[i+1];
  }
  window[n-1] = newPacket;
}



/**
```

```
* slides the window
* --> add new pkts
* --> and fill them with new data
*/
void slideWindow(int steps) {

  for (int i = 0; i < steps; i++)
  {
     struct Packet packet;
     size_t num_read = fread(packet.data, 1, BUFFER_SIZE, fp);

     packet.seqNum = seq_num;
     packet.size = num_read;
     packet.sent = false;
     packet.ack = false;

     if (num_read == 0) {
        printf("eof pkt: %d\n", packet.seqNum);
        packet.eof = true;
        eof_set = packet.seqNum;
        addPacket(packet);
        break;
     }

     addPacket(packet);
     seq_num++;

     if (seq_num >= WINDOW_SIZE) {
        seq_num = 0;
     }
  }
}

/**
*
* TRANSFER FILE OVER UDP
*
*/
int main(int argc , char *argv[]) {

  //set timeout on the socket receiving end
  struct timeval timeout;
  timeout.tv_sec = 0;
  timeout.tv_usec = 100000;
  socketConnection();
  if (setsockopt (clientSocket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof
timeout) < 0)
       printf("setsockopt failed\n");

  fp = fopen(VIDEO_FILE, "rb");
  if (fp == NULL)
     printf("\t~File open failed!\n");
  else
     printf("\t~File successfully opened!\n");
  puts("\t~Sending file.");
  // ready the first packets
  slideWindow(5);
  // then start transfer
  sendOverR_UDP();

  printf("\t~File sending complete...\n");
  if (fp != NULL)
     fclose(fp);
  close(clientSocket);

  return 0;
}
```

Output:-



```
Ack sent: 4
        ~Sliding window
writting: 0
writting: 1
writting: 2
writting: 3
writting: 4
packet received: 0
Ack sent: 0
packet received: 1
Ack sent: 1
packet received: 2
Ack sent: 2
packet received: 3
Ack sent: 3
packet received: 4
Ack sent: 4
        ~Sliding window
writting: 0
writting: 1
writting: 2
writting: 3
writting: 4
packet received: 0
Ack sent: 0
packet received: 1
Ack sent: 1
packet received: 2
Ack sent: 2
packet received: 3
Ack sent: 3
eof seq num 3
writting: 0
writting: 1
writting: 2
writting: 3
        ~File is received
        ~Closing Connection.
sham@Sham:~/Selective Repeat with FTP$
```



```
ack received: 0
ack received: 1
ack received: 2
ack received: 3
ack received: 4
        ~Sliding window
packet sent: 0
packet sent: 1
packet sent: 2
packet sent: 3
packet sent: 4
ack received: 0
ack received: 1
ack received: 2
ack received: 3
ack received: 4
        ~Sliding window
packet sent: 0
packet sent: 1
packet sent: 2
packet sent: 3
packet sent: 4
ack received: 0
ack received: 1
ack received: 2
ack received: 3
ack received: 4
        ~Sliding window
eof pkt: 3
packet sent: 0
packet sent: 1
packet sent: 2
packet sent: 3
ack received: 0
eof reachedack received: 1
eof reachedack received: 2
eof reachedack received: 3
eof reached     ~File sending complete...
sham@Sham:~/Selective Repeat with FTP$ 
```

Learning Outcomes:-

1. Understanding of File Transfer Protocol (FTP):

- Describe the client-server architecture of FTP.
- Explain the role of FTP in transferring files over a network.

2. Knowledge of Selective Repeat Protocol:

- Define the Selective Repeat protocol in the context of data transmission.
- Understand the mechanisms of selective acknowledgment and retransmission.

3. Application Layer Protocols:

- Recognize that FTP operates at the application layer of the OSI model.
- Differentiate between application layer protocols and their functions.

4. Flow Control in Networking:

- Explain the concept of flow control in networking.
- Understand how Selective Repeat contributes to efficient flow control in file transfer.

Readme:-

~~~
# Selective Repeat

Implementing SR protocol of the udp to transfer a video file.

## Description

The protocol is made reliable using the following techniques:
* Sequence numbers
* Retransmission (selective repeat)
* Window size of 5-10 UDP segments (stop n wait)
* Re ordering on receiver side

## Getting Started

### Dependencies

* C++
* Linux

### Installing

* Clone the repo
```
```
* change dir to reliableUDP then specify the port number and the
video file you want to use in **client.c and server.c** . Also
change the server ip in case of LAN.
```
#define VIDEO_FILE "testFiles/sampleVideo.mp4"
#define PORT 2003
#define SERVER_ADDR "192.168.112.94"
```

### Executing program

* Start the server first
```
gcc server.c -o server
~~~

./server
```
* Then start the client
```
gcc client.c -o client
./client
```

Link:

https://github.com/Sham2003/SR-Protocol---Networks-Assignment.git

END
```