به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه های کامپیوتری

پروژه چهارم

نام و نام خانوادگی	دانیال سعیدی(810198571) محمد قره حسنلو(810198461)
تاریخ ارسال گزارش	سه شنبه – ۳۱ خرداد ۱۴۰۱

بخش اول

۱. مزایا و معایب GBN و SR

مزایای **GBN**:

- فرستنده میتواند فریم های زیادی را در هر لحظه بفرستد.
- تایمر می تواند برای گروهی از فریم ها ست شود.
- یک ACK میتواند بیشتر یک فریم را Acknowledge کند.
- بهینه تر بودن

معایب GBD:

- نیاز به بافر
- فرستنده باید N پکت اخر را ذخیره کند.
- این طرح بهینه است هر موقع تاخیر زیاد است و نرخ ارسال داده بالاست.
- بازارسال غیر ضروری پکت های بدون خطا

مزایای **SR**:

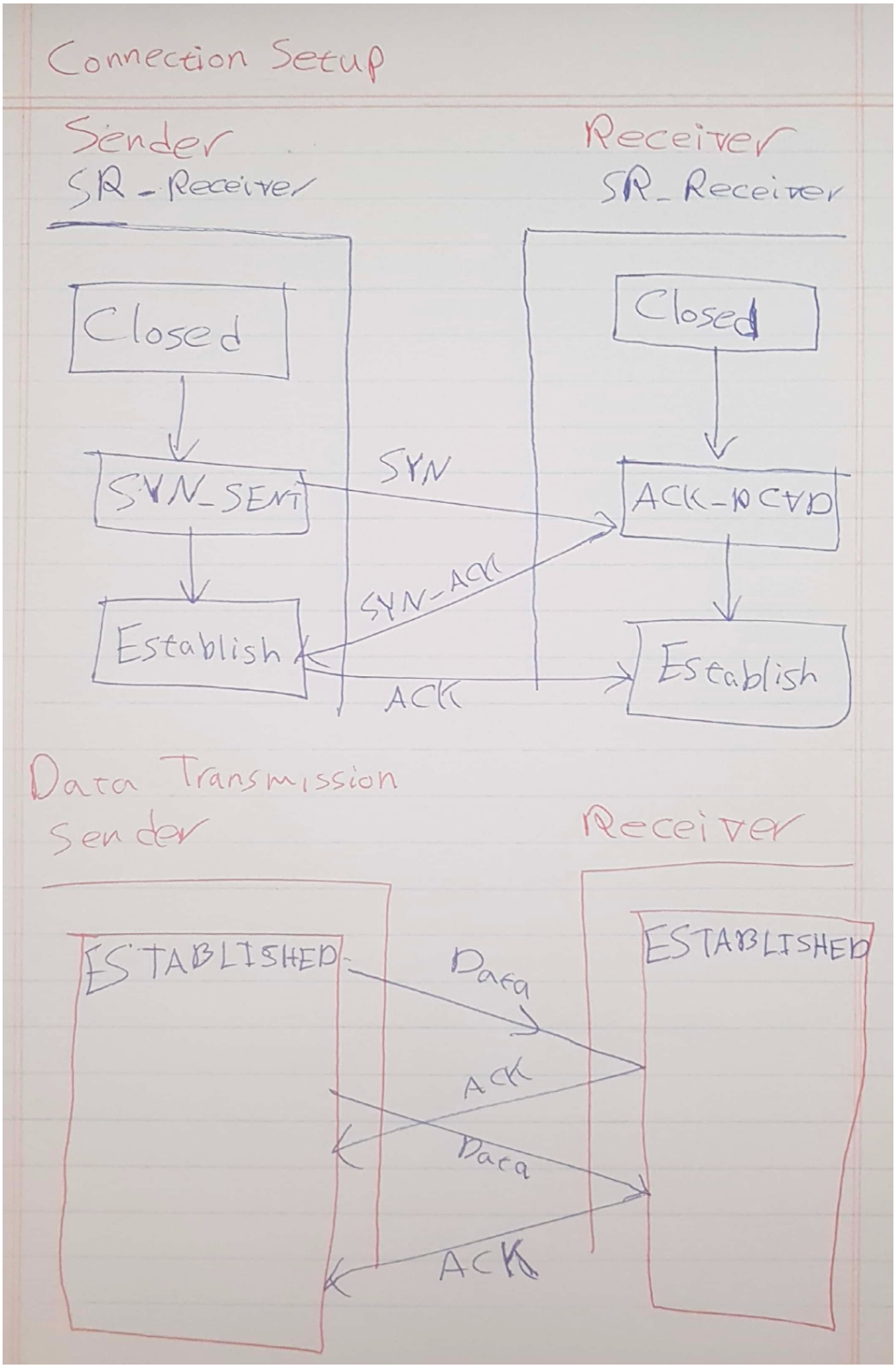
- در SR تنها فریم هایی دوباره ارسال میشوند که مشکوک به ارور هستند.

معایب GBD:

- پیچیده تر بودن نسبت GBN

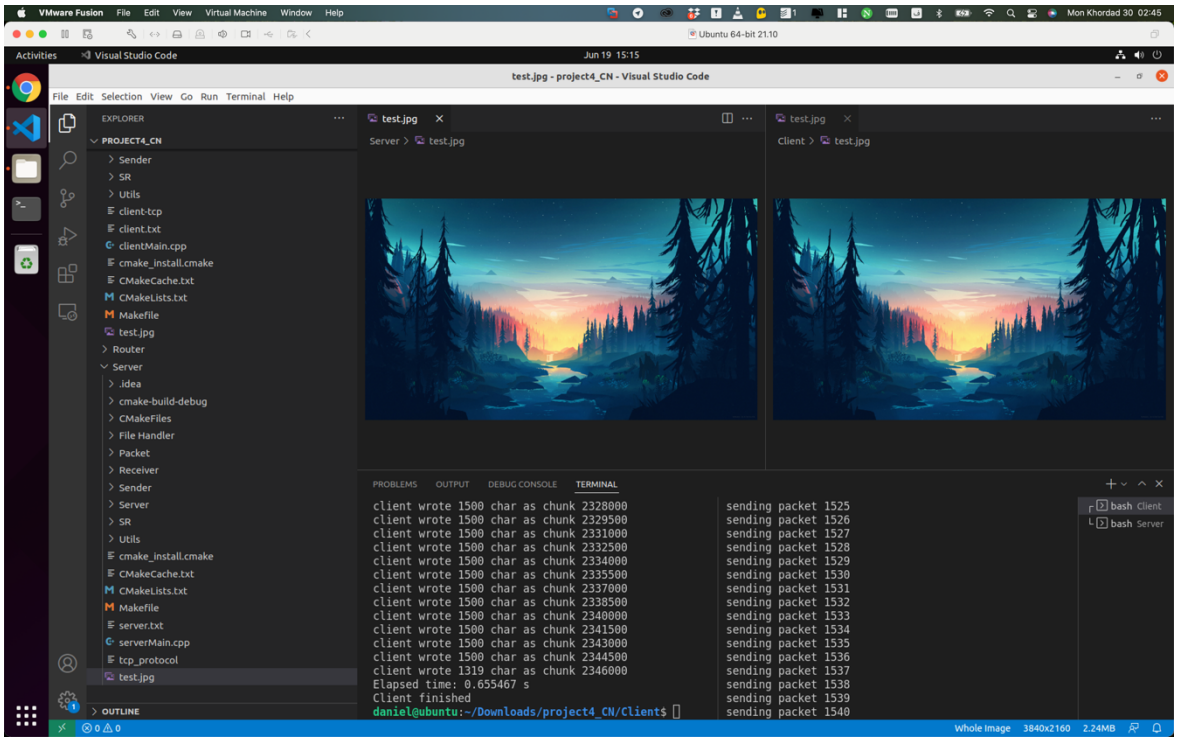
۲. پروتکل طراحی شده برای پروژه

برای Error Detection از Checksum استفاده شده است. از ایده 3-way handshaking برای connection ها بین فرستنده و گیرنده استفاده شده است که در شکل زیر میبینید:



۳. بخش ۳

یک فایل عکس jpg به حجم 2MB بین دو گره مبادله شده. مدت زمانی هم که طول کشیده بسته ها همه برسند 0.65s بوده است.



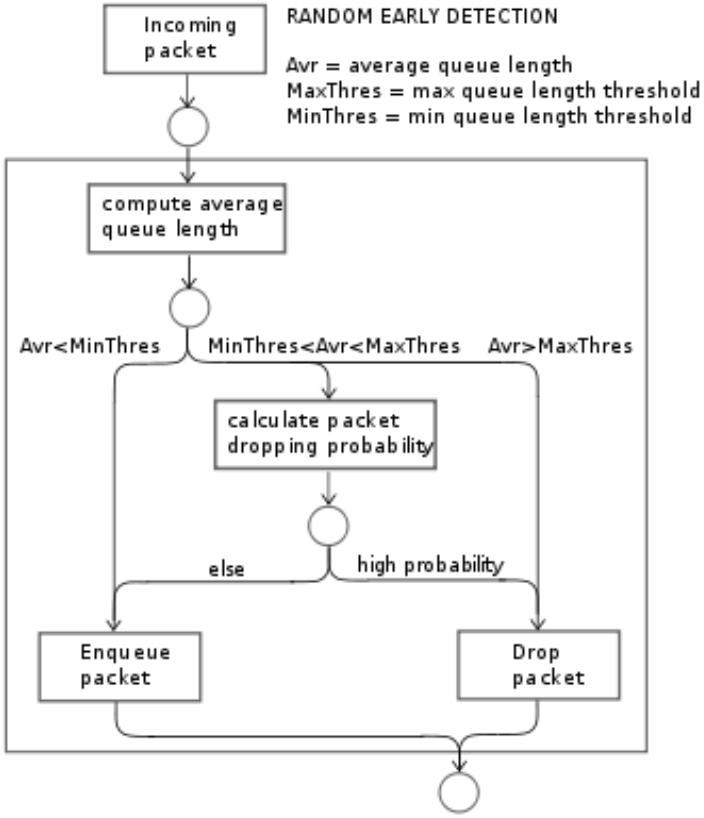
بخش دوم

۱. نحوه کارکرد RED

RED مخفف Random early detection یا به عبارت دقیق‌تر حذف زودهنگام تصادفی بسته‌ها یکی از الگوریتم‌های مدیریت فعال صف است. علاوه بر این الگوریتم یاد شده یکی از الگوریتم‌های معروف در زمینه Control Congestion است. در الگوریتم رایج droptail یک روتر یا هر مولفه شبکه تا حد امکان بسته‌ها را بافر می‌کند و بعد از پر شدن بافر بسته‌های جدید را حذف می‌کند. اگر بافر همیشه پر باشد، شبکه دچار ازدحام می‌شود. کاری که الگوریتم droptail تخصیص و تقسیم غیرهوشمند فضای بافر میان جریان ترافیکی است.

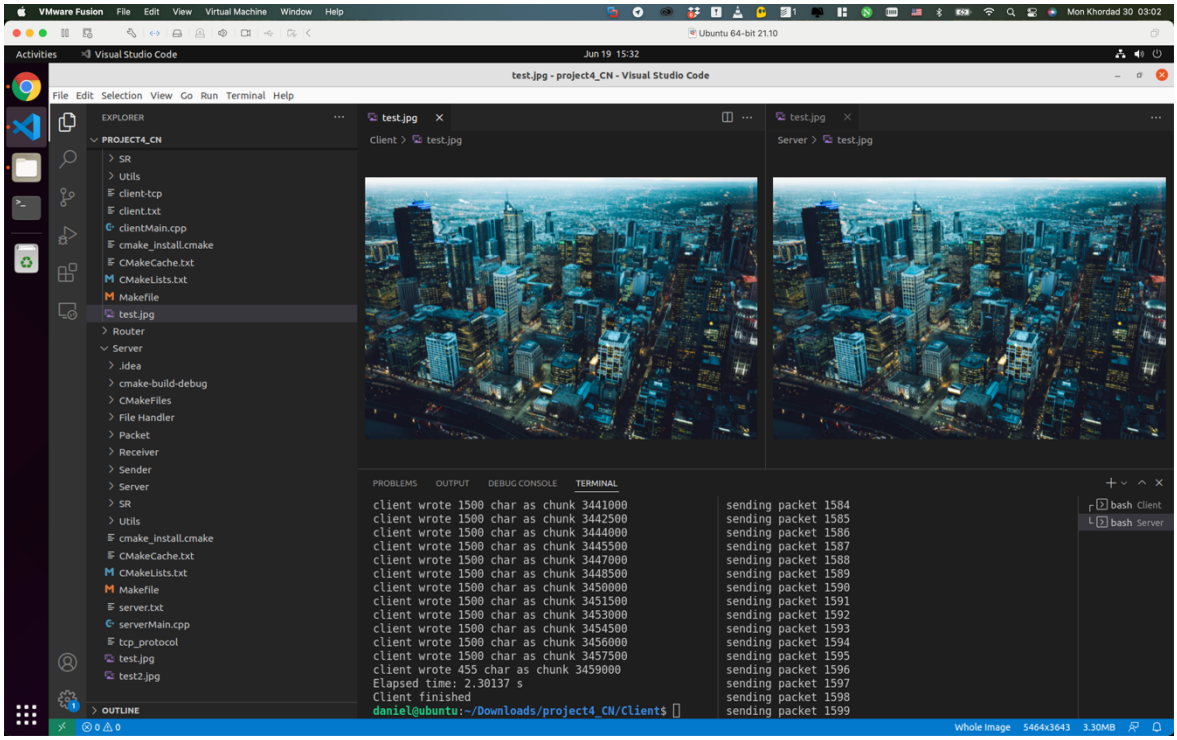
علاوه بر این، droptail گاهی اوقات مشکل پدیده همزمانی همگانی tcp را به وجود می‌آورد، زیرا همه اتصالات TCP به صورت هم‌زمان عقب‌نشینی و به صورت هم‌زمان شروع به ارسال ترافیک می‌کنند. به این ترتیب شبکه ها به صورتی نوبتی کار می‌کنند که در نهایت مشکل ازدحام را پدید می‌آورند. الگوریتم RED برای غلبه بر این مشکلات استفاده می‌شود. عملکرد RED به این‌گونه است که متوسط طول صف را پایش می‌کند و بسته‌ها را بر اساس احتمالات آماری حذف یا علامت‌گذاری می‌کند. اگر بافر نسبتا خالی باشد، تمام بسته‌های ورودی وارد صف می‌شوند. با افزایش طول صف، احتمال حذف شدن بسته‌های ورودی نیز بیشتر می‌شود.

وقتی بافر تقریباً پر شود، این احتمال به 1 میل می‌کند و تمام بسته‌های دریافتی حذف می‌شوند. RED عادلانه تر از droptail عمل می‌کند؛ چرا که تمایلی علیه ترافیک انفجاری که تنها از بخشی از پهنای باند استفاده می‌کند، ندارد. هرچه یک هاست ترافیک بیشتری ارسال کند، احتمال اینکه بسته‌هایش حذف شوند بیشتر می‌شود، زیرا احتمال حذف بسته یک هاست خاص به نسبت حجم داده‌ای است که در صف دارد. شناسایی زودهنگام به پیشگیری از پدیده همزمانی همگانی TCP کمک می‌کند.


Incoming packet
RANDOM EARLY DETECTION
Avr = average queue length
MaxThres = max queue length threshold
MinThres = min queue length threshold
compute average queue length
Avr<MinThres
MinThres<Avr<MaxThres
Avr>MaxThres
calculate packet dropping probability
else
high probability
Enqueue packet
Drop packet

۱. بخش دوم

یک عکس ۵ مگابایتی بین دو گره مبادله شده است. مدت زمان ارسال **2.30s** بوده است.



فولدر File Handler

این فولدر شامل کلاس FileReader و FileWriter است که به ترتیب وظیفه خواندن و نوشتن فایل را بر عهده دارند.

کلاس FileReader

- Get_file_size: اندازه فایل را بر میگرداند.
- Is_finished: تعیین می کند که آیا خواندن فایل تمام شده است یا خیر.
- Get_current_chuck_data: این تابع chunk فعلی را بر میگرداند.(بر اساس current_chunk_index)
- Advance_total_packer_number: مقدار current_chunk_index را یک واحد اضافه میکند.
- Get_total_packet_number: تعداد کل پکت ها را بر میگرداند.
- Close: فایل را میبندد.

```
class FileReader {

private:
    FILE *file;
    int chunk_size;
    int current_chunk_index;
    char* file_path;

public:
    explicit FileReader(const char* file_path, int chunk_size=CHUNK_SIZE);
    explicit FileReader(string file_path, int chunk_size=CHUNK_SIZE);

    int get_file_size();
    bool is_finished();
    int get_current_chunk_index();
    Packet get_current_chunk_data();
    Packet get_chunk_data(int chunk_index);
    void advance_chunk_pointer();
    int get_total_packet_number();
    void close();
};
```

کلاس FileWriter

این کلاس وظیفه نوشتن فایل را بر عهده دارد.

- Write_chunk: بر اساس current_chunk_index داده را می نویسد.
- Write_chunk_data: بر اساس chunk_index که به عنوان ورودی داده میشود data را می نویسد.

```
class FileWriter {

private:
    FILE *file;
    int chunk_size;
    int current_chunk_index;

public:
    FileWriter(string file_path, int chunk_size=CHUNK_SIZE);
    FileWriter();
    ~FileWriter();

    void write_chunk(string data);
    void write_chunk_data(int chunk_index, string data);
};
```

فولدر Packet

این فولدر شامل دیتا استراکچر packet و packetHandler است.

داده ساختار پکت:

```
/* Data-only packets */
struct Packet {
    /* Header */
    uint16_t cksum;
    uint16_t len;
    uint32_t seqno;
    /* Data */
    char data[CHUNK_SIZE]; /* Not always 500 bytes, can be less */
};
```

داده ساختار پکت ACK:

```
/* Ack-only packets are only 8 bytes */
struct Ack_Packet {
    uint16_t cksum;
    uint16_t len;
    uint32_t ackno;
};


/* Ack from server including the number of packets of the desired file */
struct Ack_Server_Packet {
    uint32_t packets_numbers;
};
```

کلاس PacketHandler

- Create_packet پکت را با توجه به شماره sequence و طول داده و خود داده درست و return می کند.
- Calculate_packet_checksum: این تابع checksum یک پکت را محاسبه می کند.
- Compare_packet_checksum: این تابع checksum پکت را بررسی می کند.

بقیه متود ها برای پکت ACK به طور مشابه تعریف میشوند.

```
class PacketHandler {

public:
    /* for data packet */
    static Packet create_packet(char* data, int seqno, int len);
    static uint16_t calculate_packet_checksum(Packet packet);
    static bool compare_packet_checksum(Packet packet);

    /* for ack packet */
    static Ack_Packet create_ack_packet(uint32_t ackno, uint16_t len);
    static uint16_t calculate_ack_packet_checksum(Ack_Packet packet);
    static bool compare_ack_packet_checksum(Ack_Packet packet);
};
```

فولدر Reciever

- Receive_packet: این تابع با دریافت socket_fd و آدرس سوکت، پکت را دریافت می کند.
- تابع receive_ack_packet پکت ACK را دریافت می کند.
- Receive_ack_server_packet: پکت ACK را از سرور دریافت می کند.

```
class Receiver {

public:
    explicit Receiver();

    static Packet receive_packet(int socket_fd, struct sockaddr_in
socket_address);
    static Ack_Packet receive_ack_packet(int socket_fd, struct sockaddr_in
socket_address, int& status, int TIMEOUT=1000);
    static Ack_Server_Packet receive_ack_server_packet(int socket_fd, struct
sockaddr_in socket_address);
};
```

فولدر RED

برای کنترل ازدحام از RED استفاده میشود. که در تابع زیر توضیحاتی که در قسمت نحوه عملکرد RED گفته شده در اینجا پیاده سازی شده است:

```
void red(Queue *queue, char *buffer) {
    printf("Current packet : %c\n", buffer[0]);
    // Average queue length calculation
    if (queue->size == 0) {
        double m = (time(NULL) - qTime) / 0.001;
        avg = pow((1 - wq), m) * avg;
    } else {
        avg = ((1 - wq)*avg) + (wq*queue->size);
    }
    printf("Average queue length : %f\n", avg);
    // If the average length is in between minimum and maximum threshold,
    // Probabilistically drop a packet
    if(minThreshold <= avg && avg < maxThreshold) {
        count++;
        pb = avg - minThreshold;
        pb = pb * maxp;
        pb = pb / (maxThreshold - minThreshold);
        double pa = pb / (1 - (count*pb));
        if (count == 50) {
            printf("Count has reached 1/maxp; dropping the next packet\n");
            pa = 1.0;
        }
        float randomProb = (rand()%100)/100.0;
        if(randomProb < pa) {
            // Drop the packet with probability pa
            printf("Dropping packet : %c with probability : %f\n", buffer[0],
pa);
            // Since this packet was dropped, count is reinitialized to 0
            count = 0;
        } else {
            // Add the packet to the queue
            add(queue, buffer[0]);
        }
    } else if (maxThreshold <= avg) {
        // Drop the packet
        printf("Packet Dropped : %c\n", buffer[0]);
        // Since this packet was dropped, count is reinitialized to 0
        count = 0;
    } else {
        // Average queue length is below minimum threhold, accept all packets
        // Add packet to the queue
        add(queue, buffer[0]);
        // Since the average queue length is below minimum threshold, initialize
count to -1
```

```
            count = -1;
        }
    }
```

فولدر Sender

این فولدر شامل کلاس Sender است که وظیفه ارسال پکت داده و ACK را بر عهده دارد.

- تابع send_packet: پکت را ارسال می کند.
- Send_ack: سیگنال ACK را بعد از اینکه پکت را دریافت کرد میفرستد.
- Send_server_ack: سیگنال ACK سرور را بعد از دریافت پکت، میفرستد.

```
class Sender {

private:
    struct sockaddr_in socket_address;

public:
    explicit Sender(struct sockaddr_in socket_addres);
    explicit Sender();

    void send_packet(Packet packet, int socket_fd);
    void send_ack(Ack_Packet ack_packet, int socket_fd);
    void send_server_ack(Ack_Server_Packet ack_server_packet, int socket_fd);
};
```

فولدر SR

در این فولدر پیاده سازی Selective Repeat قرار دارد.Window Size در اینجا 10 است.

```
class SR_Receiver {

private:
    // socket fd
    int socket_fd;
    string file_path;
    // window
    int cwnd = 10;
    map<int, Packet> received;
    int start_window_packet;
    int end_window_packet;
    int total_packets;
    // helpers
    Sender sender;
    FileWriter writer;
    struct sockaddr_in server_addr;

public:
    explicit SR_Receiver(int socket_fd, string file_path, int total_packets,
struct sockaddr_in server_addr);

    void recevFile();

};
```

فولدر Utils

این فولدر شامل Constants هایی ست که برای مساله تعریف شده است:

```
#ifndef TCP_PROTOCOL_CONSTANTS_H
#define TCP_PROTOCOL_CONSTANTS_H

#define CHUNK_SIZE 1500
#define SR_TIMEOUT 5

#endif
```

فولدر Client

- Connect_to_server: با این متود کلاینت به سرور وصل میشود.
- Init_client_socket: این متود سوکت کلاینت را initialize می کند.
- Send_request_to_server: این تابع درخواست دریافت داده را به سرور ارسال می کند.
- Receive_file: فایل را دریافت میکند.

```
class Client {

private:
    string server_ip_address;
    int server_port_number;
    int client_port_number;
    string requested_file_name;
    int initial_window_size;
    int sock_fd;
    struct sockaddr_in serv_address;

public:
    explicit Client(string client_conf_file_dir);

    void connect_to_server();
    void receive_file(int strategy_option);

private:
    void init_client_socket();
    uint32_t send_request_to_server();
};
```

فولدر Server

- Start_Server: سرور را استارت میزند و به سوکت bind می کند.
- Init_server: سوکت سرور را initialize می کند.

```
class Server {

private:
    int server_port_number;
    int maximum_window_size;
    int random_seed;
    double packet_loss_prob;
    int active_clients;
    int server_socket_fd;

public:
    explicit Server(string server_conf_file_dir);
    void start_server(int strategy_option);

private:
    void init_server();
};
```

فولدر Router

کد های روتر از این ریپو کمک گرفتیم.

```
class Router {
public:
    Router();

    /**
    * @brief register path
    * @param path path (e.g. /routes/path, /routes/:some_var)
    * @returns path id
    */
    void registerPath(std::string const &path);

    /**
    * @brief match path
    * @param path (e.g. /routes/path)
    * @returns path id if matched, or -1 if no match detected
    */
    PathMatch matchPath(std::string const &path);

private:
    std::vector<Details::PathTemplate> _templates;
    std::unique_ptr<Details::RegexConverter> _regexConverter;
};

}
```