به نام خدا

شبکه های کامپیوتری

گزارش پروژه چهارم

علی ممتحن ۸۱۰۱۰۰۲۱۳

امیرحسین راحتی ۸۱۰۱۰۰۱۴۴

مقدمه

در این تمرین چند مورد شبیه سازی الگوریتم های **go back n ,tcp new reno , tcp** انجام میشود.

بخش اول

در این بخش حالت ساده شده پروتکل **tcp** پیاده سازی میشود که به صورت **three-way handshake** و دارای سه نوع پکت **syn , ack , syn-ack** است.

یک پکت دارای ساختار زیر است که در آن **sequence** و جنس و نوع آن و دیتایی که در آن قرار دارد ب صورت رشته ذخیره شده است. همچنین در فایل **defs.hpp** توابعی برای ساخت بسته های با جنس متفاوت وجود دارد که به این فیلد ها مقدار میدهد.

```
Your Name, yesterday | 1 author (Your Name)
10  struct Packet{
11      unsigned int syn_seq;
12      unsigned int ack_seq;
13      unsigned short syn;
14      unsigned short ack;
15      unsigned short psh;
16      unsigned int data_size;
17      char data[1024];
18  };
```

دو کلاس کلاینت و سرور داریم که توابعی برای ارسال و دریافت پیام ، کانفیگ کردن سوکت و کانکشن و پاسخ به پیام ها دارند.

```
Server(int port) {
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        std::cerr << "Failed to create socket" << std::endl;
        return;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(port);

    if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        std::cerr << "Failed to bind socket" << std::endl;
        return;
    }

    std::cout << "Server started on port " << port << std::endl;
}
```

در بخش **client** هم ارتباط با سرور برقرار میشود و شروع به ارسال بسته و **handshaking** می کند.

```
Client(const std::string& serverIP, int serverPort) {
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        std::cerr << "Failed to create socket" << std::endl;
        return;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(serverIP.c_str());
    server_addr.sin_port = htons(serverPort);
}
```

مراحل **handshaking** در کلاینت به صورت زیر انجام میشود:

```
void handshake(){
    int seq = 10 , ack_seq =0;
    Packet *syn_packet = make_syn_packet(seq , 0);
    std::cout << syn_packet->syn << " " << syn_packet->syn_seq << syn_packet->ack << std::e
    Packet syn_ack_packet;
    sendto(sockfd, (void *) syn_packet, sizeof(Packet), 0, (struct sockaddr*)&server_addr,
    sockaddr_in recv_addr;
    socklen_t recv_len = sizeof(recv_addr);
    int len = recvfrom(sockfd,(void *) &syn_ack_packet, sizeof(Packet), 0, (struct sockaddr
    if(syn_ack_packet.ack & (syn_ack_packet.syn)){
        printl("SYN-ACK recived");
        seq++;
        ack_seq = syn_ack_packet.syn_seq +1;
    }else{
        printl("connection failed!");
        return;
    }
    Packet *ack_packet = make_ack_packet(seq ,ack_seq);
    sendto(sockfd, (void *) ack_packet, sizeof(Packet), 0, (struct sockaddr*)&server_addr,
```

ارسال بسته در کلاینت به صورت زیر انجام میشود.

```
sendto(sockfd, message.c_str(), sizeof(Packet), 0, (struct sockaddr*)&server_addr,
std::cout << "Sent message: " << message << std::endl;

char buffer[BUFFER_SIZE];
sockaddr_in recv_addr;
socklen_t recv_len = sizeof(recv_addr);

int recv_bytes = recvfrom(sockfd, &ans, sizeof(Packet), 0, (struct sockaddr*)&recv
if (recv_bytes < 0) {
    std::cerr << "Failed to receive ACK" << std::endl;
    return;
}
if(ans.ack && ans.psh)
std::cout << "Received ACK: "<< std::endl;
```

Handshaking در سمت سرور: در این بخش بعد از انجام handshaking ، کلاینت به یک ترد جداگانه منتقل میشود و پیام هایی که از سمت آن میاید جداگانه هندل میشود.

```
void handle_handshake(Packet *packet ,sockaddr_in *client_addr ,socklen_t client_len){
    int ack_seq = packet->syn_seq +1;
    Packet *syn_ack_packet =make_syn_ack_packet(10 , ack_seq);
    Packet ack_packet;
    sendto(sockfd,(void *)syn_ack_packet, sizeof(Packet), 0, (struct sockaddr*)client_addr,
    int recv_len = recvfrom(sockfd,(void *) &ack_packet, sizeof(Packet), 0, (struct sockadd
    if(ack_packet.ack & (!ack_packet.syn)){
        printl("ACK recived");
    }
    std::thread client(&Server::handle_client ,this ,std::ref(*client_addr), client_len);
    client.join();

}
```

بخش دوم

در این بخش از کلاینت و سرور بخش قبل استفاده میکنیم و الگوریتم TCP new Reno را روی آن پیاده سازی میکنیم.

این الگوریتم یک threshold و یک cwnd دارد که ابتدا شروع به ارسال بسته با سرعت اولیه میکند. پس از فرستادن بسته ها و دریافت کامل ack توسط مقصد ، مقدار cwnd را دوبرابر میکند تا به threshold برسد . بعد از رسیدن cwnd به threshold ، مقدار cwnd به صورت خطی رشد میکند .در این حین اگر بسته ای به درستی به مقصد نرسیده باشد ، مقصد بویسله seqnumber میتواند این موضوع را متوجه شود و در این وضعیت ، این موضوع را به مبدا با ارسال 3 تا ack پشت سر اطلاع میدهد. مبدا با دریافت این ۳ پیام ، به حالت fast recovery می رود و آن بسته گمشده را به مقصد می رساند. سپس با مقدار threshold را به نصف مقدار آخرین cwnd تغییر میدهد و مقدار cwnd به مقدار اولیه اش بر میگرداند .سپس دوباره به حالت اولیه برمیگردد و همین روند را ادامه میدهد.

کلاس Reno در این تمرین مقادیر cwnd و threshold را در خود ذخیره میکند .

```
class Reno
{
private:
    int threshold;
    int first_speed;
public:
    Reno();
    Reno(int ther , int _first_speed){threshold = ther;first_speed=_first_speed;}

    void run(Client& client);
    ~Reno();
};
```

در تابع **run** در کلاس **Reno** عملیات ارسال پیام و پیاده سازی پروتکل انجام میشود

```
void Reno::run(Client &client){
    client.handshake();
    int rec_ack[1000];
    int pn = 1 , ans_num=-1;

    int speed_rev = first_speed;
    while (pn < 1000) {

        for(int s =0 ; s< 3; s++){
            std::string message = std::to_string(pn);
            client.sendMessage(message , pn);
            std::thread w(&wait_time , speed_rev);
            auto ans = client.recieveMessage(speed_rev);
            w.join();


            if(ans == "TIMEOUT"){
                    printl("speed drop :" + std::to_string(speed_rev));
                    threshold = speed_rev * 2;
                    speed_rev = first_speed * 2;
                    break;
            }
            try{
                ans_num = std::stoi(ans);
```

در ادامه هم مقدار شرایط مربوط به **packet lost** بررسی میشود و مقدار **threshold** و **cwnd** آپدیت میشود.

```
    }catch(...){
        // send again a packet that droped
        ans_num = -1;
        int without_ack_num = std::stoi(ans.substr(1 , ans.size()-1));
        std::string message = std::to_string(without_ack_num);
        client.sendMessage(message , without_ack_num);
        auto ans = client.recieveMessage(20000 * 20000);//long time wait

    }
    rec_ack[ans_num]++;
    printl("ACK:" + ans + " " +std::to_string(pn));
    pn++;
    if(pn >= 1000)
        break;
        Your Name, 2 hours ago • add lost packet hanle part
}
if(speed_rev <= threshold){
        speed_rev -=512;
}else
    speed_rev /=2;
```

بخش سوم

در این بخش از کلاینت و سرور روی پروتکل **udp** استفاده میکنیم. هدف ایجاد ارتباط بین دو کلاینت است که از طریق یک سرور به هم متصل شده اند.

الگوریتم **go back N** :

در این الگوریتم ، فرستنده شروع به ارسال بسته ها در یک **window** مشخص میکند و در پایان **ack** هم هرل دریافت میکند. اگر در این بین **ack** پکتی به مبدا برنگردد یعنی به مقصد نرسیده بوده و فرستنده از آن پکت به بعد را دوباره ارسال میکند. در غیر اینصورت به روند قبلی ادامه میدهد.

در این بخش تمرین ، یک موجودیت سرور داریم که پیام هارا در یافت میکند و به مقصد میفرستد. سرور یک لیست از مپ بین آیپی کلاینت ها سوکت مربوط به آن ها نگه میدارد تا بعدا بتواند به آنها بسته ارسال کند . یک موجودیت کلاینت داریم که دو نوع کلاینت **sender** و **receiver** از آن ارث بری میکنند.

یک موجودیت **gbn** هم وجود دارد که وظیفه ذخیره استیت و متغیر های الگوریتم **gobackN** را دارد.

بقیه مقادیر مثل کانفیگ پورت و سوکت و دریافت و ارسال پیام مثل بخش های قبل انجام میشود.

برای مدیریت تایم اوت دریافت پیام هم یک آرگومان ثانیه به تابع **receive** میدهیم که به اندازه آن منتظر دریافت پیام بماند. در کد **sender** ارسال پیام و دریاف **ack** به صورت زیر انجام میشود.

```
void Sender::startSendingTask()
{
    while(gbn->getCurrentIndex() <= gbn->getTotalFrames())
    {
        for(int k=gbn->getCurrentIndex(); k < gbn->getCurrentIndex() + gbn->getSeqNum() && k <=
        {
            cout << "Sending Frame " << k << " ..." << endl;
            sendMessage("MESSAGE 192.168.1.1 192.168.1.3 -"+ to_string(k) + "-hello") ;
            gbn->increaseTotalTransaction();
        }
        string recievedAcks = recieveMessage(5);
        cout << "ACKS: " << recievedAcks << endl;

        gbn->updateIndex(splitString(recievedAcks , '-'));
        sleep(1);
    }
    cout << "program finnished. number of Transactions: " << gbn->getTotalTransaction() << endl
    return;
}       amirhossein, 2 days ago • Inheritance in clients
```

درگیرنده هم با دریافت یک سری از پکت ها و منتظر ماندن به اندازه تایم اوت ، بسته **ack** را ارسال میکند.

```
void Reciever::handleRecievingTask()
{
    while (true)
    {
        recievedSequences.clear();
        for(int i = 0 ; i < 16 ; i++)
        {
            string recievedMessage = recieveMessage(3);
            if(recievedMessage!="TIMEOUT")
            {
                vector<string> rec = splitString(recievedMessage, ' ');
                handleIncomingMessage(rec);
            }
            else
                break;
        }
        sendAcknowledgeMessage();
    }
}
```

سپس در مبدا لیست **ack** ها بویسله تابع **updateIndex()** بررسی میشود تا اگر جایی مشکلی وجود دارد ، دوباره آن بسته به بعد را بفرستیم.هرجا که **ack** دریافتی با ایندکس فعلی مطابقت نداشته باشد متوقف میشویم و ایندکس را آپدیت نمیکنیم.

```
void SenderGoBackN::updateIndex(std::vector<std::string> AckList)
{
    for(int j = 1 ;j < AckList.size();j++)
    {
        if(currentIndex == stoi(AckList[j]))
            currentIndex++;
        else
            break;
    }
}       amirhossein, 2 days ago • Inheritance in clients
```

نحوه کارکرد برنامه :

در این بخش اندازه **window** را ۱۶ و کل بسته ها را ۱۰۰ تا در نظر میگیریم . عملیات ارسال با توجه به اینکه در این شبیه سازی همه بسته ها میرسند، ۷ مرحله متشکل از ۶ مرحله ۱۶ تایی و یک مرحله ۴ تایی انجام میگیرد .

الگوریتم Selective repeat:

در این الگوریتم فرستنده چندین بسته را بدون انتظار برای تایید ارسال می‌کند. گیرنده هر بسته را جداگانه تایید می‌کند.اگر بسته ای به درستی نرسد، فقط بسته های از دست رفته یا خراب شده مجدداً ارسال می‌شوند.گیرنده می‌تواند بسته هارا را خارج از ترتیب بپذیرد و ذخیره کند.

تفاوت با:Go-Back-N

در Go-Back-N ، اگر یک بسته از دست برود، تمام بسته بعدی دوباره ارسال می‌شوند. اما selective repeat فقط بسته های مشکل‌دار را مجدداً ارسال می‌کند، که باعث کارایی بیشتر می‌شود.