

گزارش پروژه چهارم درس شبکه‌های کامپیوتری

کیمیا محمدطاهری (۸۱۰۱۹۸۵۳۵)

دانشور امراللهی (۸۱۰۱۹۷۶۸۵)

فهرست مطالب

2	فهرست مطالب
3	مقدمه
3	شرح پروژه
4	ساختار کلی برنامه
5	پیاده‌سازی ارتباط: UDP
5	پیاده‌سازی: Go-Back-N
6	کلاس‌های کمکی:
6	Message: کلاس
6	Socket: کلاس
7	کلاس‌های اصلی:
7	Sender: کلاس
10	Receiver: کلاس
12	پیاده‌سازی: RED
14	نمونه اجرای برنامه
14	پروتوکول: Go-Back-N
15	پروتوکول: Random Early Detection
16	اندازه‌گیری زمان اجرا با فایل حجیم:
17	پروتوکول: Go-Back-N
18	پروتوکول: Random Early Detection
18	پاسخ سوالات صورت پروژه
18	قسمت اول
19	قسمت دوم

مقدمه

در این تمرین کامپیوتری، به پیاده‌سازی پروتوکول‌های Go-Back-N و Random Early Detection روی یک شبکه ساده بر پایه UDP پرداختیم. Go-Back-N یک حالت خاص از پروتوکول پنجره لغزان¹ می‌باشد که اندازه پنجره دریافت‌کننده دقیقاً ۱ می‌باشد. Random Early Detection یا همان RED، یک پروتوکول کنترل ازدحام² می‌باشد که متناسب با اندازه buffer، احتمالی را برای دور انداختن³ بسته⁴ در نظر می‌گیرد. کدهای پیاده‌سازی این پروژه در آدرس‌های زیر موجود می‌باشند:

<https://github.com/kymmt80/GoBackN-Protocol>

<https://github.com/daneshvar-amrollahi/RED-Protocol>

شرح پروژه

توپولوژی در نظر گرفته برای این پروژه به شکل زیر است:



شکل(۱): توپولوژی شبکه

در این توپولوژی کامپیوتر A از طریق روتر R فایل نسبتاً بزرگی را به کامپیوتر B ارسال می‌کند. روتر R بافری دارد که پیام‌های ورودی در آن ذخیره می‌شوند و به صورت FIFO به مقصد ارسال می‌کند. برای پیاده‌سازی این تمرین، هکدام از کامپیوترهای A و B و روتر R را به صورت یک پردازش مستقل در نظر گرفتیم. کامپیوتر A برای ارسال فایل، آن را به صورت بسته‌ها 1.5KBی از طریق مکانیسم Sliding Window ارسال می‌کند. برای ارتباطات اشاره‌شده میان پردازش‌ها از سوکت از نوع UDP استفاده می‌شود.

¹ Sliding Window

² Congestion Control

³ Drop

⁴ در طول این گزارش واژه‌های frame و packet (بسته) به جای هم به کار رفته‌اند.

ساختار کلی برنامه

```
.
├── doc
│   └── description.pdf
├── Makefile
├── README.md
├── receiver
│   ├── include
│   │   └── receiver.hpp
│   ├── Makefile
│   └── src
│       ├── main.cpp
│       └── receiver.cpp
├── router
│   ├── include
│   │   └── router.hpp
│   ├── Makefile
│   └── src
│       ├── main.cpp
│       └── router.cpp
├── sender
│   ├── include
│   │   └── sender.hpp
│   ├── Makefile
│   └── src
│       ├── main.cpp
│       └── sender.cpp
├── test
│   └── francais.txt
└── utils
    ├── defs.hpp
    ├── include
    │   ├── message.hpp
    │   └── socket.hpp
    └── src
        ├── message.cpp
        └── socket.cpp
```

کلاس‌های پیاده‌سازی شده در این پروژه شامل router، sender و receiver می‌باشند که به ترتیب متناظر با A، R و B در شکل (۱) می‌باشند.

پیاده‌سازی ارتباط UDP:

برای پیاده‌سازی ارتباط از طریق سوکت، یک کلاس به نام Socket تعریف کردیم که در سازنده⁵ خود مقدار ip و port را دریافت می‌کند و یک سوکت UDP جدید در این محل ایجاد می‌کند. این کلاس دو متد send و receive دارد که مربوط به نوشتن و خواندن از سوکت است. header مربوط به این کلاس در ادامه آمده است:

```
class Socket
{
private:
    struct sockaddr_in address;
public:
    int fd;
    Socket(char *ip, int port);
    int send(std::string data);
    std::string receive();
};
```

پیاده‌سازی این کلاس در فایل زیر انجام شده است:

<https://github.com/kymmt80/GoBackN-Protocol/blob/main/utils/src/socket.cpp>

پیاده‌سازی Go-Back-N:

در پروتکل پیاده‌سازی شده، ابتدا فرستنده پیغامی به فرمت *\$frameCount* ارسال می‌کند که frameCount نشان دهنده تعداد کل فریم‌هایی است که قرار است ارسال بشود. زمانی که گیرنده این پیغام را دریافت کند یک window به این فرستنده اختصاص می‌دهد. حال فرستنده frame ها را به فرمت *seqNum: Data* ارسال می‌کند که seqNum شماره frame و Data محتوای frame است. گیرنده با دریافت این پیغام آن را در محل مرتبط قرار می‌دهد. فرستنده همچنین یک timer با فرستادن هر پیام set می‌کند و در صورتی که timeout اتفاق بیوفتد مجدداً کل پنجره را ارسال می‌کند.

⁵ Constructor

کلاس‌های کمکی:

کلاس Message:

برای نگه داری frame ها کلاس Message را تعریف کردیم. متدهای مهم این کلاس شامل موارد زیر است:

- `read_file`: این متد از فایل می‌خواند از فایل و آن را به frame های 1.5 بایتی تقسیم می‌کند.
- `get_frame`: این متد frame با شماره `seq_num` را باز می‌گرداند.
- `store_frame`: این متد frame با شماره `seq_num` را ذخیره می‌کند.
- `store_file`: این متد frame ها را کنار هم قرار می‌دهد و در فایل ذخیره می‌کند.

```
class Message
{
private:
    std::vector<frame> content;
    std::vector<bool> received_frames;
public:
    frame get_frame(int seq_num);
    void store_frame(int seq_num, frame input);
    void read_file();
    void store_file();
    int get_size();
    int get_frame_size();
    void set_size(int sz);
    bool is_frame_received(int seq_num);
};
```

پیاده سازی این کلاس در فایل زیر انجام شده است:

<https://github.com/kymmt80/GoBackN-Protocol/blob/main/utls/src/message.cpp>

کلاس Socket:

از این کلاس برای برقراری ارتباط UDP استفاده می‌شود. توضیح این کلاس در بخش «پیاده‌سازی ارتباط UDP» آمده است.

کلاس های اصلی:

کلاس Sender:

این کلاس وظیفه خواند فایل ورودی و ارسال آن را دارد.

```
class Sender {
public:
    Sender(
        char* ip,
        int port_from_router,
        int port_to_router
    );
    void run();
    frame create_frame(int seq_num);
    frame get_next_frame();
    bool all_frames_sent();
    void send_new_frames();
    bool time_out();
    void retransmit();
private:
    char* ip;
    int port;
    int send_fd;
    int receive_fd;
    std::vector<Socket*>sockets;
    Message message;
    int LFS;
    int LAR;
    std::vector<clock_t> sent_times;
};
```

در ابتدا متد run این کلاس صدا زده می‌شود. این متد با دریافت ورودی شروع به ارسال فایل می‌کند. در ابتدا به عنوان پیام اول، تعداد frame ها به receiver ارسال می‌شود. در ادامه با دریافت هر ACK متد send_new_frames صدا می‌شود.

```
void Sender::run() {
    int max_sd;
    int bytes;
    fd_set master_set, read_set;
```

```

FD_ZERO(&master_set);
max_sd = max(receive_fd, send_fd);
FD_SET(STDIN_FILENO, &master_set);
FD_SET(send_fd, &master_set);
FD_SET(receive_fd, &master_set);
int starting_time;
while (1)
{

read_set = master_set;
bytes=select(max_sd + 1, &read_set, NULL, NULL, NULL);
if (FD_ISSET(receive_fd, &read_set))
{
    string recv_message = sockets[receive_fd]->receive();
    if (recv_message == FIRST_ACK)
        send_new_frames();
    else
    {
        int seq_num = get_seq_num(recv_message);
        LAR = seq_num;
        cout << recv_message << endl<<LOG_DELIM;
        if(!all_frames_sent()){
            send_new_frames();
        }
    }
}
if(FD_ISSET(STDIN_FILENO, &read_set)){
    starting_time = clock();
    sockets[send_fd]->send("$" +
to_string(message.get_size()));
    FD_CLR(STDIN_FILENO,&master_set);
}
if (time_out())
{
    cout << "TIMEOUT occured for " << LAR + 1 << endl<<LOG_DELIM;
    retransmit();
}
if(LAR==message.get_size()-1){
    cout<<"TRANSMIT IS OVER"<<endl<<LOG_DELIM;
    break;
}
}
}

```



```

int exec_time = (clock()-starting_time)/CLOCKS_PER_SEC;
cout << "EXECUTION TIME: " << exec_time << "s" << endl;
}

```

دو متغیر LAR و LFS به ترتیب شماره آخرین ACK دریافت شده⁶ و آخرین frame ارسال شده⁷ را نگه می‌دارند. همواره فاصله این دو به اندازه SWS⁸ است که مقدار آن در `defs.h` تعریف شده است. در هر مرحله با صدا شدن `send_new_frames`, ابتدا LAR با توجه به ACK به روز می‌شود، سپس آنقدر frame ارسال می‌کنیم تا مقدار LFS در تساوی $LFS - LAR = SWS$ صدق بکند.

```

void Sender::send_new_frames()
{
    while (LFS - LAR <= SWS)
    {
        cout << "Sending frame " << LFS << "..." << endl<<LOG_DELIM;
        sent_times[LFS] = clock();
        sockets[send_fd]->send(get_next_frame());
    }
}

```

لازم است که در صورتی که مدتی پس از ارسال فریم، ACK مربوط به آن را دریافت نکردیم، تمام window را مجدداً ارسال کنیم. برای این کار، آرایه `sent_times` را تعریف کردیم که زمان ارسال هر frame را نگهداری می‌کند. در هر مرحله اجرای حلقه اصلی `run`، در مدت `time_out` زمان سپری شده پس از ارسال اولین frame پنجره کنونی را بررسی می‌کنیم و اگر بیشتر از `PACKET_LOST_THRESHOLD` که در `defs.hpp` تعریف شده است بود، با صدا کردن مدت `retransmit` تمام frame های پنجره را مجدداً ارسال می‌کنیم.

```

bool Sender::time_out() {
    if(LAR + 1 >= message.get_size())
        return false;
    clock_t current_time = clock();
    float time_elapsed = (current_time - sent_times[LAR + 1]) /
        CLOCKS_PER_SEC;
    return time_elapsed > PACKET_LOST_THRESHOLD;
}

```

⁶ Last Acknowledge Received

⁷ Last Frame Sent

⁸ Sender Window Size

```

void Sender::retransmit() {
    for (int i = LAR + 1; i < LFS; i++)
    {
        sent_times[i] = clock();
        cout << "Retransmitting frame " << i << "..." << endl;
        sockets[send_fd]->send(create_frame(i));
    }
    cout<<LOG_DELIM;
}

```

پیاده سازی کامل این کلاس در فایل زیر انجام شده است:

<https://github.com/kymmt80/GoBackN-Protocol/blob/main/sender/src/sender.cpp>

کلاس Receiver:

این کلاس وظیفه دریافت فایل ارسالی و ذخیره آن را دارد.

```

class Receiver {
public:
    Receiver(
        char* ip,
        int port_to_router,
        int port_from_router
    );
    void run();
    void handle_recv_msg(std::string message);

private:
    char* ip;
    int port;
    int send_fd;
    int receive_fd;
    int LFR;
    std::vector<Socket*>sockets;
    Message message;
};

```

در ابتدا متد run این کلاس صدا زده می‌شود. این متد منتظر دریافت پیام از طرف فرستنده می‌ماند و با دریافت هر پیام از طرف فرستنده متد handle_recv_msg را صدا می‌کند.

```

void Receiver::run() {

```

```

int max_sd;
int bytes;
fd_set master_set, read_set;
FD_ZERO(&master_set);
max_sd = receive_fd;
FD_SET(STDIN_FILENO, &master_set);
FD_SET(receive_fd, &master_set);
FD_SET(send_fd, &master_set);
while (1)
{
    read_set = master_set;
    bytes=select(max_sd + 1, &read_set, NULL, NULL, NULL);
    if (FD_ISSET(receive_fd, &read_set))
    {
        handle_rcv_msg(sockets[receive_fd]->receive());
    }
}
}

```

متد `handle_rcv_msg` در صورتی که پیام دریافتی اولین پیام ارسالی توسط فرستنده باشد(که با کاراکتر `$` در ابتدای پیام مشخص می‌شود)، تعداد `frame` های مورد انتظار را ذخیره می‌کند. سپس در مراحل بعدی با دریافت هر پیام، با توجه به این که اندازه پنجره گیرنده⁹ در Go-Back-N برابر ۱ است، در صورتی که `seq_num` فریم دریافتی برابر مقدار مورد انتظار گیرنده بود، آن را ذخیره می‌کند و در غیر این صورت دور می‌اندازد¹⁰.

```

void Receiver::handle_rcv_msg(std::string message) {
    if (message[0] == '$')
    {
        this->message.set_size(stoi(message.substr(1, (int)(message.size()) -
1)));
        cout<<message<<endl;
        sockets[send_fd]->send("ACK$");
    }else{
        int seq_num = get_seq_num(message);
        string data = get_data(message);
        if(seq_num!=LFR){
            cout<<"Discarded frame no." <<seq_num << " saying: " << data <<
endl<<LOG_DELIM;
        }else{
            cout << "Received frame no." <<seq_num << " saying: " << data

```

⁹ Receiver Window Size

¹⁰ Discard

```

<< endl<<LOG_DELIM;
    this->message.store_frame(seq_num, data);
    sockets[send_fd]->send("ACK" + to_string(LFR));
    cout<<"Sending ACK"<<LFR<<"..."<<endl<<LOG_DELIM;
    LFR++;
}
}
}

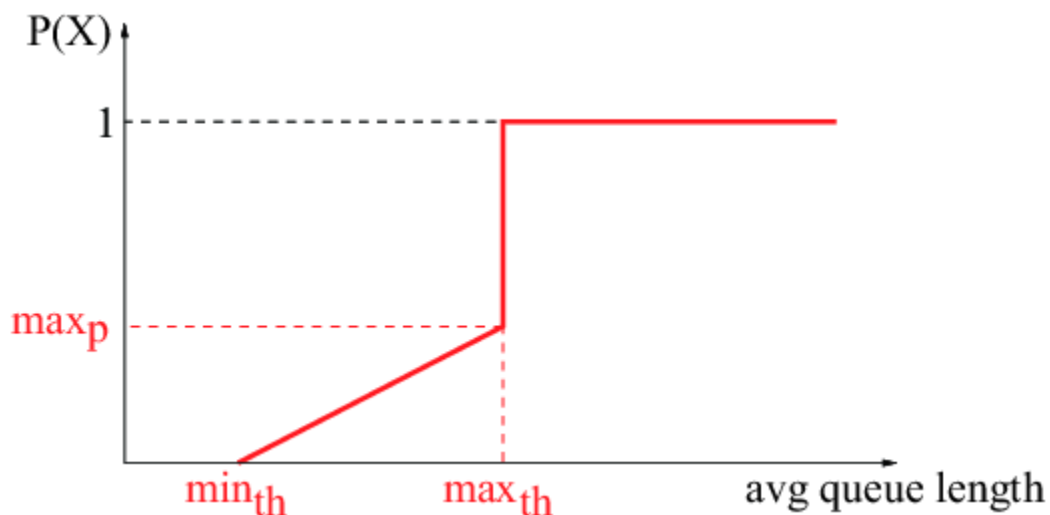
```

پیاده سازی کامل این کلاس در فایل زیر انجام شده است:

<https://github.com/kymmt80/GoBackN-Protocol/blob/main/receiver/src/receiver.cpp>

پیاده سازی RED¹¹:

این پروتوکول، بر مبنای بخش قبلی (پروتوکول Go-Back-N) پیاده سازی می شود. بخش اصلی که در پیاده سازی این پروتوکول در مقایسه با بخش قبلی تغییر می کند، مربوط به منطق buffer در کلاس router است. همانطور که در شکل (۲) نشان داده شده است، این پروتوکول به صورت خطی¹² احتمال دور انداختن¹³ بسته¹⁴ دریافتی را با توجه به تعداد پکت هایی که در حال حاضر buffer شده اند زیاد می کند.



شکل(۲): الگوریتم اجرای RED

¹¹ Random Early Detection

¹² Linear

¹³ Drop

¹⁴ Packet

همچنین برای امکان تست کردن با ۲۰ فرستنده^{۱۵}، فرمت پیام ارسالی از فرستنده‌ها اندکی تغییر می‌کند (فرستنده آیدی خود را در frame قرار می‌دهد تا بین ACK‌های دریافتی با sequence numberهای یکسان قابلیت تمایز وجود داشته باشد)

مقدار ثابت‌های مربوط به این پروتوکول در فایل defs.h تعریف شده‌است:

```
#define MIN_DROP_THRESHOLD 1
#define MAX_DROP_THRESHOLD 3
#define RED_DROP_RATE 0.1
```

در اینجا MIN_DROP_THRESHOLD، RED_DROP_RATE و MAX_DROP_THRESHOLD به ترتیب \min_{th} ، \max_{th} و \max_{th} می‌باشند.

منطق این بخش در متد Router::add_to_buffer مطابق شکل زیر پیاده‌سازی شده است:

```
int seq_num = get_seq_num(message);
string data = get_data(message);
int sender_id = get_sender_id(message);
if (buffer.size() <= MIN_DROP_THRESHOLD)
{
    buffer.push(message);
    cout << "Buffered (" << "sender=" << sender_id << ", seq_num=" <<
seq_num << ", data=" << data << ")" << endl << LOG_DELIM;
    return;
}
else if (buffer.size() >= MIN_DROP_THRESHOLD && buffer.size() <
MAX_DROP_THRESHOLD)
{
    float prob = float(rand())/RAND_MAX;
    float drop_prob = RED_DROP_RATE * ((int)buffer.size() -
MIN_DROP_THRESHOLD);
    if(prob <= drop_prob){
        cout<<"Oops I dropped ("<< "sender=" << sender_id << ",
seq_num=" << seq_num << ", data=" << data << ")" << "
:)))"<<endl<<LOG_DELIM;
```

¹⁵ Sender

```

    }else{
        buffer.push(message);
        cout << "Buffered (" << "sender=" << sender_id << ", seq_num="
<< seq_num << ", data=" << data << ")" << endl << LOG_DELIM;
    }
}
}else{
    cout<<"Oops the buffer is full -> dropped ("<< "sender=" <<
sender_id << ", seq_num=" << seq_num << ", data=" << data << ")" << "
:)))"<<endl<<LOG_DELIM;
}
}

```

نمونه اجرای برنامه

پروتوکول Go-Back-N:

جهت سهولت اثبات کارکرد درست برنامه در این بخش، تعداد ۸ فریم از فرستنده به گیرنده ارسال می‌کنیم:

```

daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/TCP-utility 100x26
ACK0
Sending frame 4...
ACK1
Sending frame 5...
TIMEOUT occurred for 2
Retransmitting frame 2...
Retransmitting frame 3...
Retransmitting frame 4...
Retransmitting frame 5...
ACK2
Sending frame 6...
ACK3
Sending frame 7...
ACK4
Sending frame 8...
daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/TCP-utility 101x26
S9
Received frame no.0 saying: this
Sending ACK0...
Received frame no.1 saying: is
Sending ACK1...
Discarded frame no.3 saying: dumb
Discarded frame no.4 saying: test
Discarded frame no.5 saying: and
Received frame no.2 saying: a
Sending ACK2...
Received frame no.3 saying: dumb
Sending ACK3...
Received frame no.4 saying: test
Sending ACK4...
daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/TCP-utility 203x27
Connected to Port 8002
Connected to Port 8001
Connected to Port 8004
Transmitting message "S9" from sender to receiver...
Transmitting message "ACK$" from receiver to sender...
Transmitting message "0:this" from sender to receiver...
Transmitting message "1:is" from sender to receiver...
Oops I dropped packet no.2 :)))
Transmitting message "ACK0" from receiver to sender...
Transmitting message "3:dumb" from sender to receiver...
Transmitting message "ACK1" from receiver to sender...
Transmitting message "4:test" from sender to receiver...
Transmitting message "5:and" from sender to receiver...

```

شکل(۳): نمونه اجرای برنامه با پروتوکول Go-Back-N - ترمینال بالا سمت چپ، بالا سمت راست و پایین به ترتیب فرستنده، گیرنده و روتر می‌باشند.

تحلیل فرستنده:

به دلیل drop شدن frame شماره ۲ توسط روتر، TIMEOUT occurred for 2، بنابراین، با وجود این که قبل از آن تا بسته ۵ ارسال شده بود، دوباره از بسته شماره ۲ همه را باز ارسال¹⁶ می‌کند.

تحلیل گیرنده:

چون frame شماره ۲ توسط روتر drop شده بود، frame‌های دریافتی شماره ۳ و ۴ و ۵ را discard می‌کند و تنها منتظر frame شما ۲ می‌ماند.

پروتوکول Random Early Detection:

جهت سهولت اثبات کارکرد درست برنامه در این بخش، تعداد ۲ تا فرستنده را به جای ۲۰ فرستنده در نظر می‌گیریم:

```
daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/RED-Protocol 100x26
rcv_message = ACK5:1
Sending frame 0...
Sending frame 1...
Sending frame 2...
Sending frame 3...
rcv_message = ACK:0:1
Sending frame 4...
rcv_message = ACK:1:1
Sending frame 5...
rcv_message = ACK:2:1
Sending frame 6...
TIMEOUT occurred for 3
Retransmitting frame 3...
Retransmission frame 3...

daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/RED-Protocol 100x27
Transmitting message "S1:0" from sender to receiver...
Receiving (sender=1, seq_num=0, data=this)
Buffered (sender=1, seq_num=0, data=this)
Buffered (sender=1, seq_num=1, data=ls)
Transmitting message "S2:0" from sender to receiver...
Transmitting message "ACK5:2" from receiver to sender...
Buffered (sender=2, seq_num=0, data=this)
BUFFER-PROCESSING DONE
Transmitting (sender=1, seq_num=0, data=this) from buffer...
Transmitting message "ACK:0:1" from receiver to sender...
Buffered (sender=1, seq_num=2, data=a)
Oops the buffer is full -> dropped (sender=2, seq_num=1, data=ls) )))
Oops the buffer is full -> dropped (sender=1, seq_num=3, data=dumb) :)))

daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/RED-Protocol 101x26
rcv_message = ACK5:2
Sending frame 0...
Sending frame 1...
Sending frame 2...
Sending frame 3...
rcv_message = ACK:0:2
Sending frame 4...
TIMEOUT occurred for 1
Retransmitting frame 1...
Retransmitting frame 2...
Retransmitting frame 3...
Retransmitting frame 4...
rcv_message = ACK:1:2

daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/RED-Protocol 101x27
Received (sender=1, seq_num=0, data=this)
Sending ACK:0:1
Received (sender=1, seq_num=1, data=ls)
Sending ACK:1:1
Received (sender=2, seq_num=0, data=this)
Sending ACK:0:2
Received (sender=1, seq_num=2, data=a)
Sending ACK:2:1
Received (sender=1, seq_num=4, data=test)
Discarded (sender=2, seq_num=4, data=test)
Discarded (sender=1, seq_num=6, data=1)
Received (sender=2, seq_num=1, data=ls)
Sending ACK:1:2
Received (sender=2, seq_num=2, data=a)
```

شکل(۴): نمونه اجرای پروتوکول RED - ترمینال بالا چپ، بالا راست، پایین چپ و پایین راست به ترتیب فرستنده اول، فرستنده دوم، روتر و دریافت‌کننده می‌باشند.

در مثال شکل(۴)، ظرفیت بافر ۳ می‌باشد. بنابراین با دریافت بسته (sender=2, seq_num=1, data=ls)، ظرفیتی برای ذخیره آن ندارد (قبلا ۳ تا buffer شده‌اند) و آن را drop می‌کند. در نتیجه این عملیات drop،

¹⁶ Retransmit

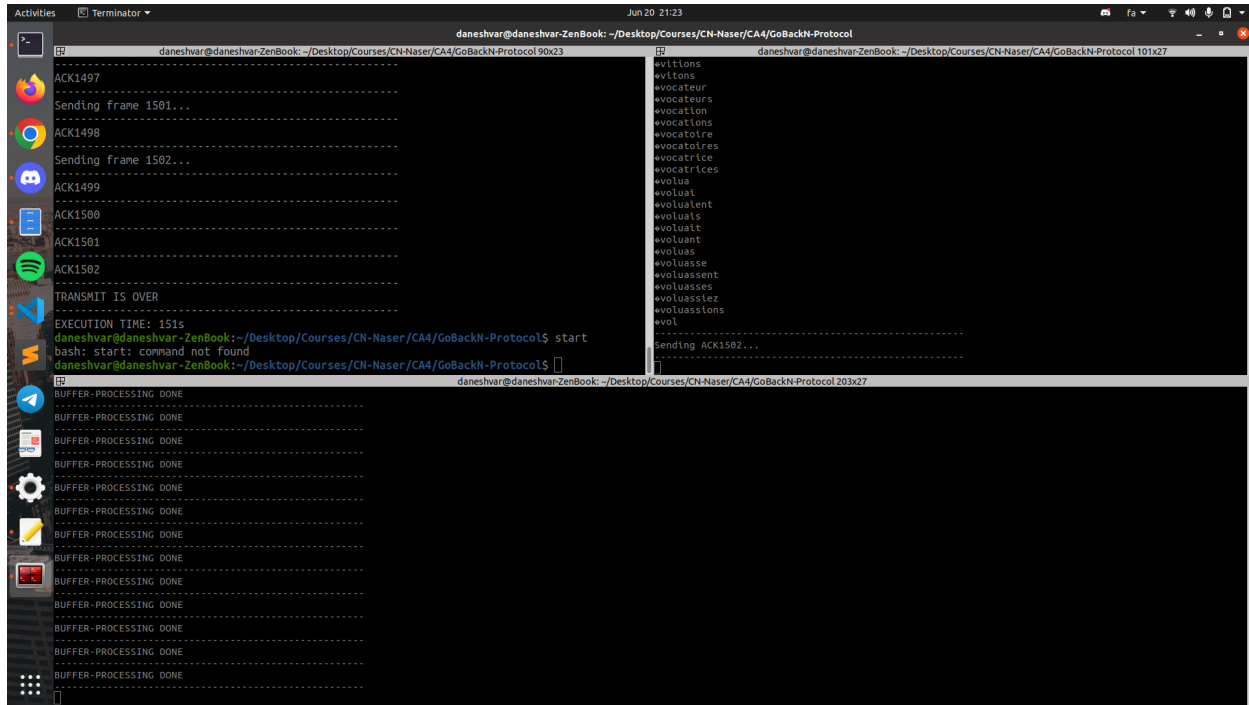
فرستنده ۲ (بالا راست) متوجه TIMEOUT آن شده و عملیات بازارسال را انجام می‌دهد. مشابه همین اتفاق برای (sender=1, seq_num=3, data=dumb) نیز رخ می‌دهد و فرستنده اول نیز مجبور به بازارسال آن می‌شود.

اندازه‌گیری زمان اجرا با فایل حجیم:

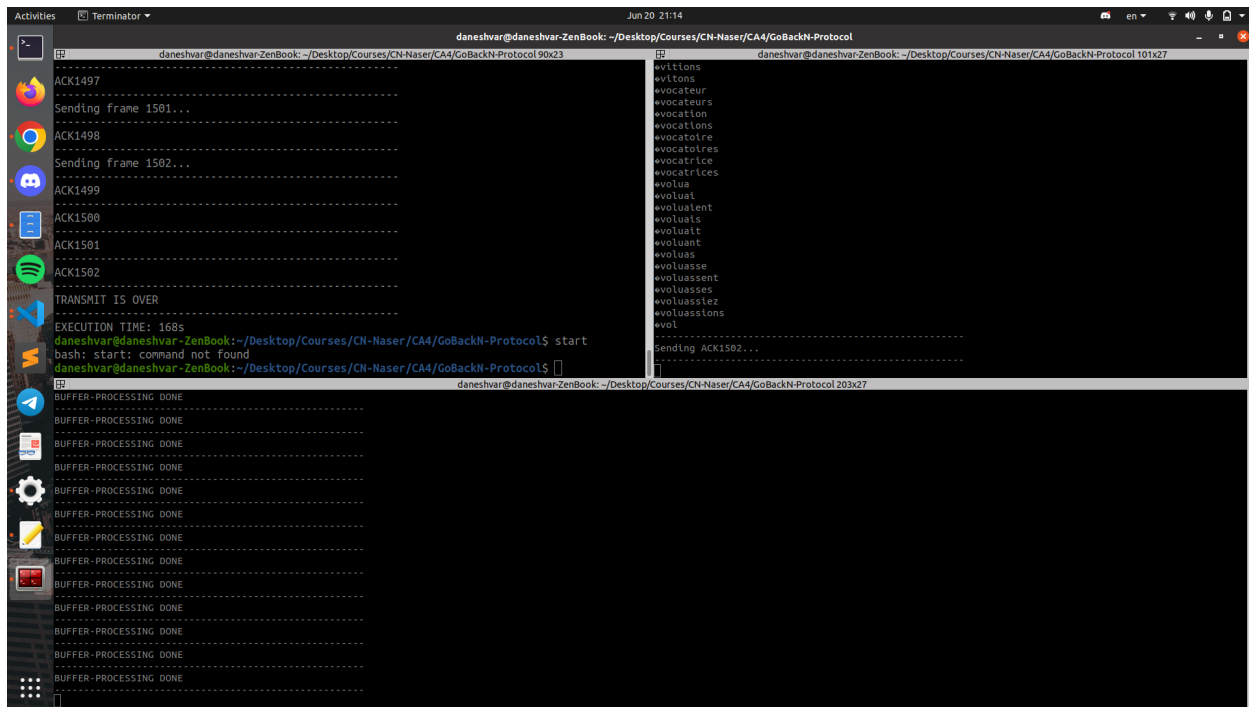
برای تعیین اندازه چنک‌های ارسالی مقدار ثابت CHUNK_SIZE در defs.h را برابر 1500B که همان 1.5KB است تعیین می‌کنیم. فایلی که برای تست انتخاب شده (tests/francais.txt) حجم 2.3MB دارد. محتوای این فایل در متد Message::read_file خوانده می‌شود:

```
void Message::read_file(){
    // content={"this","is","a","dumb","test","and","I","am","bored"};
    ifstream fin(FILE_ADDRESS);
    char c[CHUNK_SIZE];
    while (fin.read(c, CHUNK_SIZE))
    {
        string chunk = "";
        for (int i = 0; i < CHUNK_SIZE; i++)
            chunk += c[i];
        content.push_back(chunk);
    }
}
```


پروتوکول Go-Back-N:



شکل(۵): ترمینال بالا چپ، بالا راست و پایین به ترتیب فرستنده، دریافت‌کننده و روتر هستند - حالت بافر نامحدود



شکل(۶): ترمینال بالا چپ، بالا راست و پایین به ترتیب فرستنده، دریافت‌کننده و روتر هستند - حالت بافر با اندازه ۱۰

پروتوکول Random Early Detection:

برای ایجاد ۲۰ تا فرستنده، از یک اسکریپت bash به اسم run_tests.sh در پوشه test استفاده شده است که وظیفه باز کردن ترمنیال‌ها و ساختن فرستنده‌ها، روتر و دریافت‌کننده را دارد. مدت زمان اجرای این بخش ۱۶۵۱ ثانیه شد. البته مقدار زمان اندازه‌گیری شده به سرعت پردازش buffer که در صورت پروژه ذکر نشده بود نیز بستگی دارد. برای مثال در حالتی که سرعت پردازش buffer از حدی بیشتر باشد، هیچگاه اندازه صف آن به \min_{th} نمی‌رسد و در نتیجه هیچ dropی اتفاق نخواهد افتاد.

پاسخ سوالات صورت پروژه

قسمت اول

• سوال ۱ :

در جدول (۱) ویژگی‌های Selective-Repeat و Go-Back-N با یکدیگر مقایسه شده‌اند:

Go-Back-N Protocol	Selective Repeat Protocol
در این پروتوکول در صورتی که یک frame مشکوک به گم شدن باشد، تمام frame های پنجره مجددا ارسال می‌شود.	در این پروتوکول در صورتی که یک frame مشکوک به گم شدن باشد، فقط همان frame مجددا ارسال می‌شود.
اندازه پنجره فرستنده برابر N است	اندازه پنجره فرستنده هم برابر N است.
اندازه پنجره گیرنده برابر ۱ است	اندازه پنجره گیرنده برابر N است.
پیاده‌سازی ساده است.	پیاده‌سازی پیچیده است.
نیازی به مرتب سازی frame های دریافتی نیست.	گیرنده باید frame های دریافتی را مرتب کند.
نوع Acknowledgement ها تجمعی ¹⁸ است.	نوع Acknowledgement ها تکی ¹⁷ است.

¹⁷ Individual

¹⁸ Cumulative

در صورتی که packet خارج از ترتیب برسد، دور ریخته می‌شود و تمام پنجره مجددا ارسال می‌شود.	packet های خارج از ترتیب هم اگر در بازه پنجره گیرنده باشند پذیرفته می‌شوند.
در صورت رسیدن یک بسته ²⁰ خراب ²¹ ، تمام پنجره بازارسال ²² می‌شود.	در صورت رسیدن یک بسته خراب، بلافاصله یک NACK ¹⁹ ارسال می‌کند. بنابراین تنها همان بسته بازارسال می‌شود.

جدول (۱): مقایسه پروتوکول‌های Go-Back-N و Selective Repeat

مشاهده می‌شود که مزیت پروتوکول GBN²³ نسبت به SR²⁴ این است که نیازی به مرتب سازی ندارد و پیاده سازی به مراتب ساده تر دارد. عیب GBN این است که زمان اجرای آن و تعداد ارسال‌ها به دلیل ارسال مجدد تمام پنجره در صورت گم شدن یک بسته و همچنین کوچکتر بودن اندازه پنجره گیرنده، طولانی‌تر است.

• سوال ۲

در این پروژه پروتوکول Go-Back-N پیاده‌سازی شده است. توضیحات تکمیلی این پروتوکول در پاراگراف اول بخش «پیاده‌سازی Go-Back-N» نوشته شده است.

• سوال ۳

مطابق شکل (۵)، مدت زمان اجرا ۱۵۱ ثانیه بود.

• سوال ۴

مطابق شکل (۶)، مدت زمان اجرا ۱۶۸ ثانیه بود. همانطور که انتظار می‌رفت، مدت زمان انتقال در حالتی که اندازه بافر نامحدود بود، کمتر شد.

قسمت دوم

• سوال ۱

توضیحات این پروتوکول در پاراگراف اول بخش «پیاده‌سازی RED» نوشته شده است.

• سوال ۲

مطابق «پروتوکول Random Early Detection» از بخش اصلی «محاسبه زمان اجرا با فایل حجیم»، مدت زمان اجرا ۱۶۵۱ ثانیه بود.

¹⁹ Negative Acknowledge

²⁰ Packet

²¹ Corrupted

²² Re-transmit

²³ Go-Back-N

²⁴ Selective Repeat