

گزارش پروژه چهارم درس شبکه‌های کامپیوتری

کیمیا محمدطاهری (۸۱۰۱۹۸۵۳۵)

دانشور امراللهی (۸۱۰۱۹۷۶۸۵)

4	مقدمه
4	شرح پروژه
5	ساختار کلی برنامه
6	پیاده‌سازی ارتباط: UDP
6	پیاده‌سازی: Go-Back-N
7	کلاس‌های کمکی:
7	کلاس: Message
7	کلاس: Socket
7	کلاس‌های اصلی:
7	کلاس: Sender
11	کلاس: Receiver
13	پیاده‌سازی: RED
15	نمونه اجرای برنامه
15	پروتوکول: Go-Back-N
16	تحلیل فرستنده:
16	تحلیل گیرنده:
16	پروتوکول: Random Early Detection
17	محاسبه زمان اجرا با فایل حجیم:
18	پروتوکول: Go-Back-N
18	پروتوکول: Random Early Detection

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 دقیقا 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 نسبتا 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.

کلاس‌های کمکی:

کلاس Message:

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

⁵ Constructor

- `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 متغییر 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

⁶ Last Acknowledge Received

⁷ Last Frame Sent

⁸ Sender Window Size

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;
}
```

```
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_rcv_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_rcv_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
<< 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++;
        }
    }
}
}

```

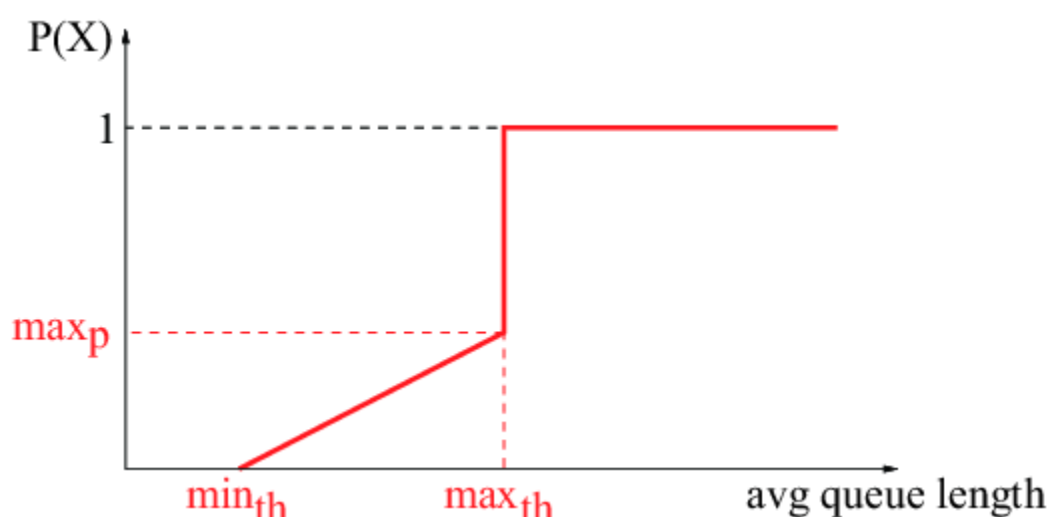
پیاده سازی کامل این کلاس در فایل زیر انجام شده است:

⁹ Receiver Window Size

¹⁰ Discard

پیاده‌سازی RED¹¹:

این پروتوکول، بر مبنای بخش قبلی (پروتوکول Go-Back-N) پیاده‌سازی می‌شود. بخش اصلی که در پیاده‌سازی این پروتوکول در مقایسه با بخش قبلی تغییر می‌کند، مربوط به منطق buffer در کلاس router است. همچنین برای امکان تست کردن با ۲۰ فرستنده¹²، فرمت پیغام 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)



شکل (۲): الگوریتم اجرای RED

مقدار این ثابت‌ها در فایل defs.h تعریف شده‌است:

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

در اینجا RED_DROP_RATE همان شیب خط بازه مورد نظر در شکل (۲) می‌باشد.

¹¹ Random Early Detection

¹² Sender

همانطور که در شکل (۲) نشان داده شده است، این پروتوکول به صورت خطی¹³ احتمال دور انداختن¹⁴ بسته¹⁵ دریافتی را با توجه به تعداد پکت‌هایی که در حال حاضر buffer شده‌اند زیاد می‌کند.

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;
    }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;
}
```

¹³ Linear

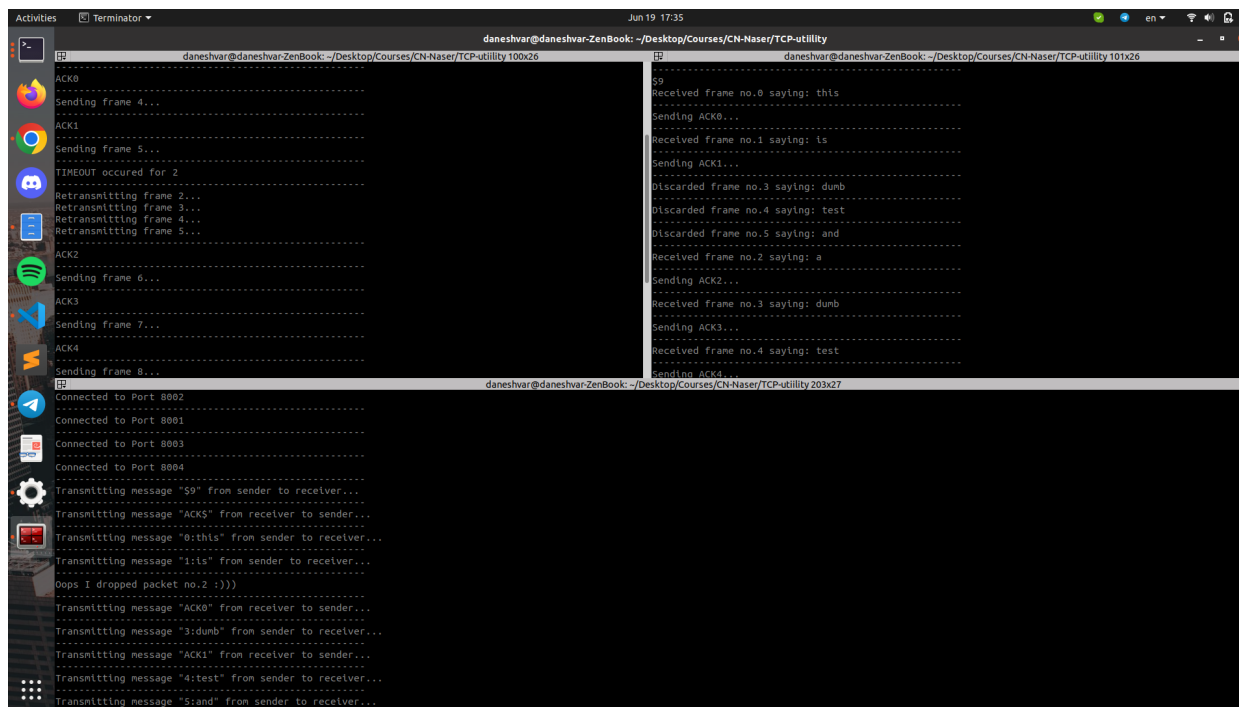
¹⁴ Drop

¹⁵ Packet

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...
Connected to Port 8002
Connected to Port 8001
Connected to Port 8003
Connected to Port 8004
Transmitting message "S9" from sender to receiver...
Transmitting message "ACK5" from receiver to sender...
Transmitting message "R: this" from sender to receiver...
Transmitting message "I: is" from sender to receiver...
Oops I dropped packet no.2 :)))
Transmitting message "ACK0" from receiver to sender...
Transmitting message "R: dumb" from sender to receiver...
Transmitting message "ACK1" from receiver to sender...
Transmitting message "R: test" from sender to receiver...
Transmitting message "S: 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 چاپ شده است. 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.

¹⁶ Retransmit

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:

The image shows a Linux desktop environment with two terminal windows open. The desktop background is a dark theme with various application icons on the left sidebar. The top of the screen displays the system date and time as 'Sun 19:05'.

The left terminal window, titled 'daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/RED-Protocol 100x2', shows a sequence of messages and acknowledgments. It starts with 'rcv_message = ACKS:1', followed by '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...', and 'Retransmitting frame 6...'. The right terminal window, titled 'daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/RED-Protocol 101x2', shows a similar sequence but with a 'TIMEOUT occurred for 1' and 'Retransmitting frame 1...', 'Retransmitting frame 2...', 'Retransmitting frame 3...', 'Retransmitting frame 4...', and 'rcv_message = ACK:1:2'.

The right terminal window is currently displaying a red banner with the text 'daneshvar@daneshvar-ZenBook: ~/Desktop/Courses/CN-Naser/RED-Protocol 101x2'. Below this, it shows a sequence of messages and acknowledgments, including 'Received (sender=1, seq_num=0, data=this)', 'Sending ACK:0:1', 'Received (sender=1, seq_num=1, data=1s)', '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', 'Discarded (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=1, seq_num=1, data=1s)', 'Sending ACK:1:2', and '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 دریافت کننده می باشند.

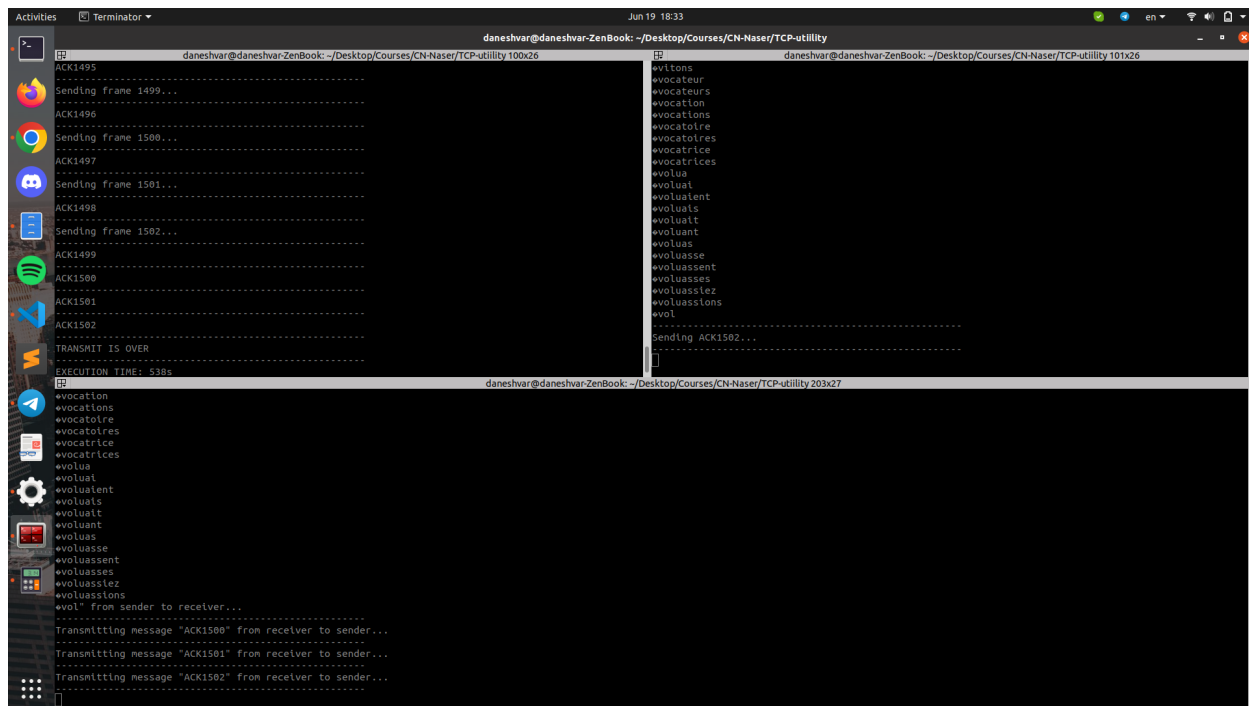
در مثال شکل (۴)، ظرفیت بافر ۳ می‌باشد. بنابراین با دریافت بسته (sender=2, seq_num=1, data=is)، ظرفیتی برای ذخیره آن ندارد (قبلا ۳ تا buffer شده‌اند) و آن را drop می‌کند. در نتیجه این عملیات drop، فرستنده ۲ (بالا راست) متوجه TIMEOUT آن شده و عملیات بازارسال را انجام می‌دهد. مشابه همین اتفاق برای (sender=1, seq_num=3, data=dumb) نیز رخ می‌دهد و فرستنده اول نیز مجبور به بازارسال آن می‌شود.

محاسبه زمان اجرا با فایل حجیم:

برای تعیین اندازه chinkهای 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/français.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):

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 ترمینال‌ها و ساختن فرستنده‌ها، روتر و دریافت‌کننده را دارد.

مدت زمان اجرای این بخش از برنامه بیشتر از ۱۰ دقیقه بود.