Computer Networks Project 4

Moein Karami 810198540
Aryan Soltani Mohammadi 810198558

Part 1:

1-

Go back N protocol use a timer for receiving the first packet ACK. If the acknowledgment of a frame is not received within an agreed upon time period, all frames starting from that frame are retransmitted.

Pros: It just needs a single timer
Cons: not efficient, waste bandwidth when a packet is lost/broken.

Selective Repeat protocol It uses two windows of equal size: a sending window that stores the frames to be sent and a receiving window that stores the frames received by the receiver. The size is half the maximum sequence number of the frame. The receiver records the sequence number of the earliest incorrect or un-received frame. It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with every acknowledgement frame.

Pros: efficient, only lost/broken packets need retransmission
Cons: complicated with multiple timers, receiver needs a buffer to buffer out-of-order packets.

2-

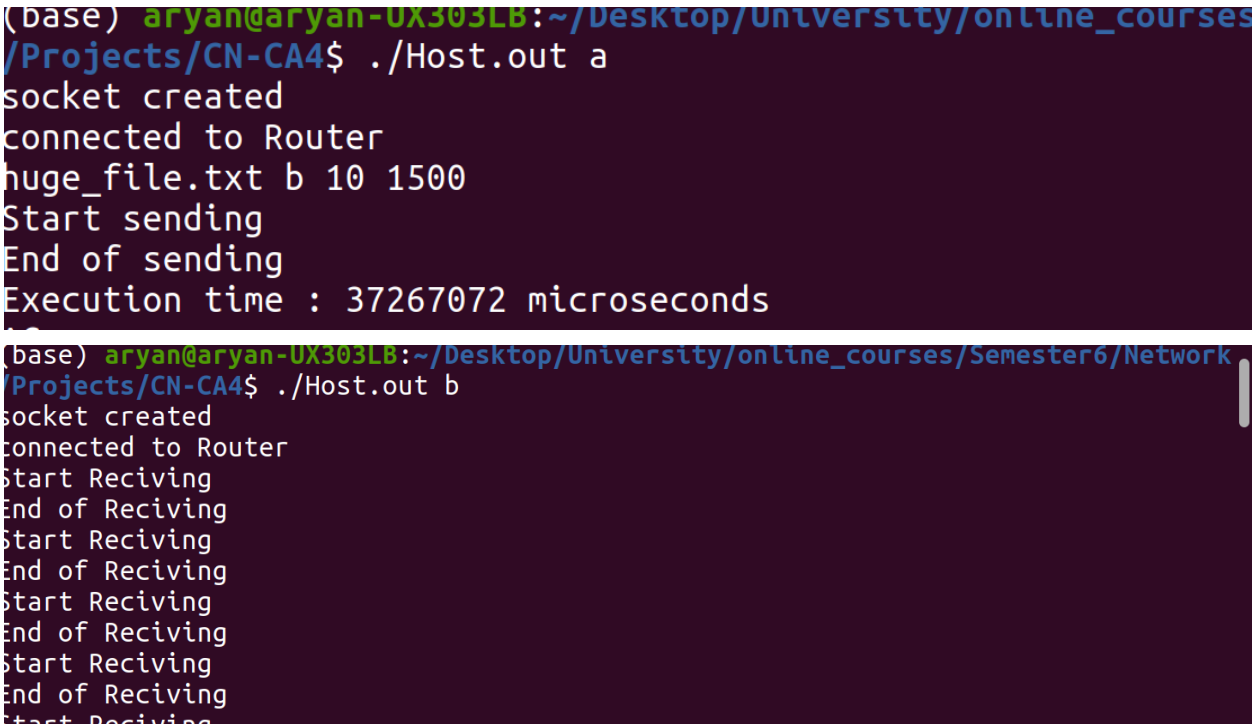
The protocol that we have used in this project is Go back N.

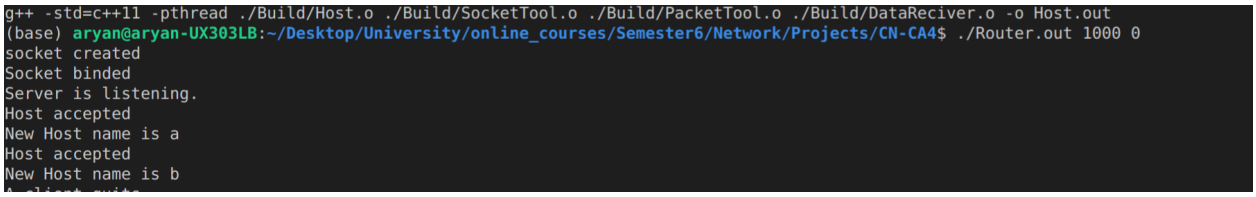
```
int last = cur_seq_num;
char buf[MAX_DATA_SIZE];

std::map<byte, bool> acks;

while (cur_seq_num < packets.size())
{
    while(last < std::min((int) packets.size(), cur_seq_num + window_size))
    {
        Msg msg(packets[last++]);
        send(fd, msg.msg, msg.len, 0);
        std::cerr << "Packet " << (int)packets[last-1].seq_num << " sent" << std::endl;
        std::cerr <<"sender, reciver, seqnum: " << (char)packets[last-1].sender << " " << (char)packets[last-1].reciver << " " << (char)packets[last-1].seq_num << std::endl;
        std::cerr << "msg sender reciver : " <<  (char)msg.msg[2] << " " << (char)msg.msg[0] << std::endl;
        std::cerr << "msg len " << msg.len << std::endl;
        std::cerr << "packet size : " << packets[last - 1].data_size << std::endl;
        std::cerr << "SHOw _____________"<<std::endl;
        for (int i = 0; i < msg.len; i++)
            std::cerr << msg.msg[i];
        std::cerr << "_________________"<<std::endl;
    }

    std::cerr << "wait for ack" << std::endl;

    int len = recv(fd, buf, MAX_DATA_SIZE, 0);
    if (len == -1)
    {
        std::cerr << "Time out" << std::endl;
        send_packets(packets, cur_seq_num, window_size);
        return;
    }
    std::vector<Packet> res = PacketTool::parse_packet(buf, len);
    for (auto packet : res)
    {
        if (packet.type == ACK_TYPE)
        {
            acks[packet.seq_num] = true;
            std::cerr << "ACK " << (int)packet.seq_num << " recived" << std::endl;
        }

    }


    while (cur_seq_num < last)
    {
        if (acks[cur_seq_num])
        {
            acks[cur_seq_num] = false;
            cur_seq_num++;
        }
        else
            break;
    }
}
```

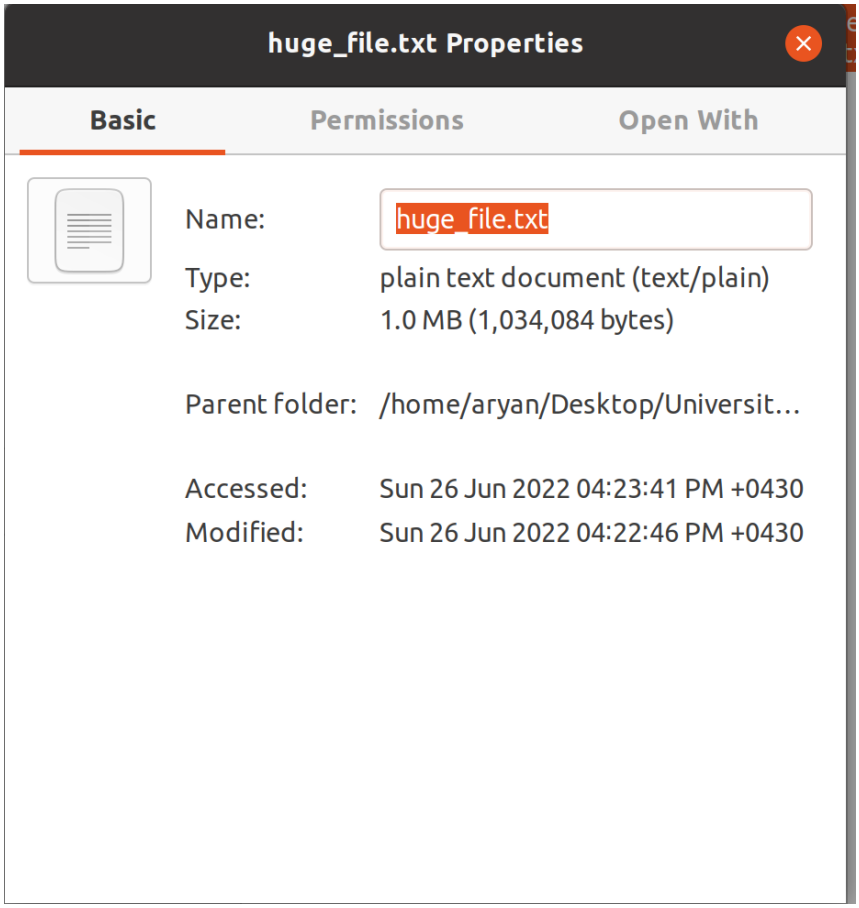
The above code is the implementation for the Go back N protocol using sliding window as we can see if we time out for receiving the first ACK, we invoke the function again and retransmit all the sliding window packets.

3-

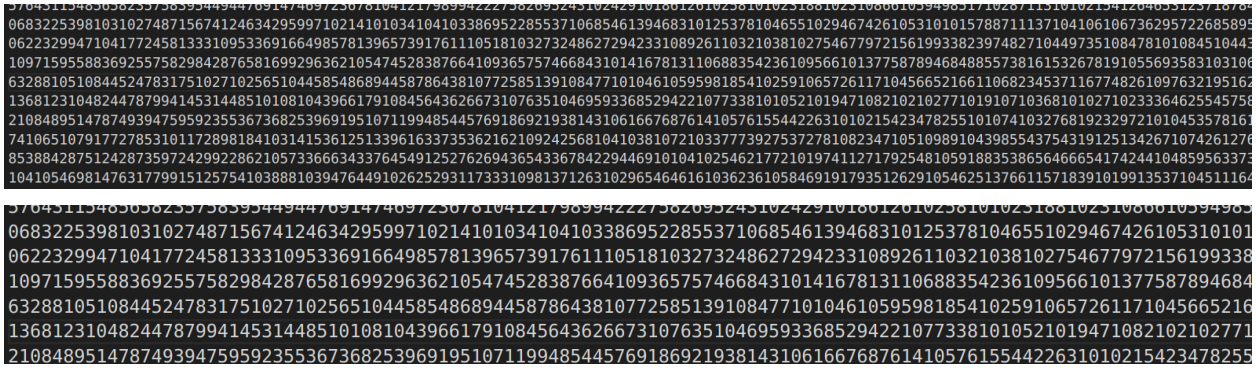
For packets in this project the first byte shows the receiver and the second byte is for the type(ACK or data type) and the third is the sender and the fourth the seq num so seq num is a number within 0 to 256. Also after that the data comes and the end a ‘\0’ character to find out the end of file.







As we can see in the above pictures we have made a huge file which its size is 1 Mb and it had been sent from host A to host B and the execution time was about 38 seconds.



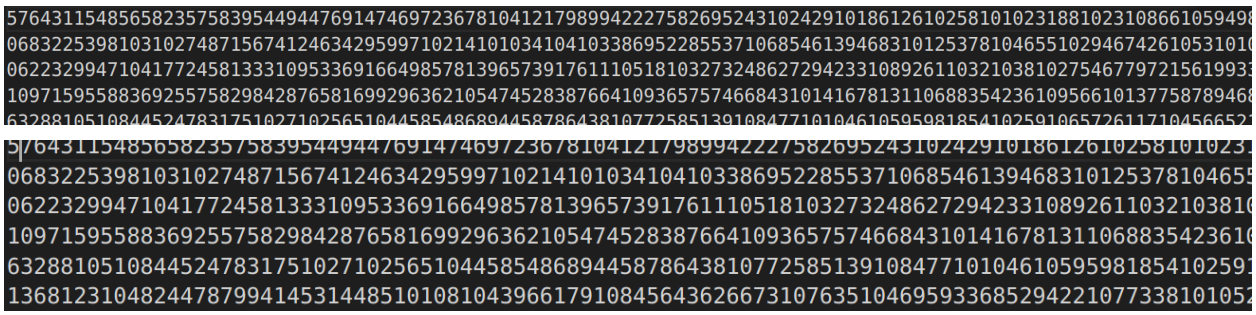
Some parts of output that are identical

4-

After running this

```
socket created
connected to Router
huge_file.txt b 10 1500
Start sending
End of sending
Execution time : 119966439 microseconds
```

Because the buffer size is 10 the time taken has been increased and it is about 100 seconds.



As we can see the file has built correctly

Part 2:

1-

RED protocol is a protocol for avoiding packet congestion in the router. It operates during the enqueue time, which when a new packet is going to be added in the queue it decides whether it should be dropped or added to the queue. It calculates a probability based on the average packet size for deciding to drop a packet.
It works in three stages:

1-Calculation of average queue length:

2- Calculation of drop probability, whether the packet will be enqueued or dropped depends on this probability.
3-Decision-making logic (helps to decide whether the incoming packet should be enqueued or dropped).

We always calculate the probability based on the average queue size and also two threshold have been held for the queue size and if the queue size was below the min threshold we would always send the packet if it was above the max-threshold we always drop the packet otherwise we would calculate the probability.

```
void RouterRed::add_new_packets_red(std::vector<Packet> new_packets)
{
    for(auto packet : new_packets)
    {
        double prob = calculate_new_probablility();
        std::cerr << "Probability for packet is: " << prob << std::endl;
        std::cerr << "Average packet is: " << new_avg << std::endl;
        if(prob > 0.5)
        {
            std::cerr << "From Router packet sent " << std::endl;
            router_queue.push(packet);
        }
        else
        {
            std::cerr << "Packet drop due to RED protocl" << std::endl;
        }
        calculate_avg();
    }
}
```

- - - > This is the part for running red protocol

```
double RouterRed::calculate_new_probablility()
{
    if(new_avg <= min_th)
    {
        return 1;
    }
    if(new_avg >= max_th)
    {
        return 0;
    }
    return 0.5 * ((new_avg - min_th) / (max_th - min_th));
}
```

- - - > part for calculating the probability

```
void RouterRed::calculate_avg()
{
    new_avg = (1 - w_q) * new_avg + w_q * router_queue.size();
}
```

- - - > part for calculating the new average.

```
Probability for packet is: 1
Average packet size is: 2.42377
From Router packet sent
Probability for packet is: 1
Average packet size is: 2.41892
From Router packet sent
```

Changes in the queue size can be seen in the image above.

```
(base) aryan@aryan-UX303LB:~/Desktop/University/online_courses/Semester6/Network/Projects/CN-CA4$ ./Router.out 1000 1
socket created
Socket binded
Server is listening.
```

```
int main()
{
    pid_t process_id;
    int return_val = 1;
    int state;
    process_id = fork();
    if(process_id == -1)
    {
        printf("can't fork, error occured\n");
        exit(0);
    }
    else if (process_id == 0)
    {
        int id = getpid();
        char tmp = char(id) + 'a';
        char name_host[2];
        name_host[0] =  tmp;
        name_host[1] = '\0';
        cout << id << endl;
        char * argv_list[] = {"./Src/Host.out", name_host, "temp", NULL};
        execv("./Host.out",argv_list); // the execv() only return if error occured.
        exit(0);
    }
    else
    {
        for(int i = 0; i < 20; i++)
        {
            fork();
        }
        for(int i  = 0; i < 20; i++)
        {
            wait(NULL);
        }
```

Code for creating 20 A host.

```
/Projects/CN-CA4$ ./Host.out a
socket created
connected to Router
./huge_file.txt b 10 1000
Start sending
End of sending
Execution time : 3205335 microseconds
^C
```