

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 << " recieved" << 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.

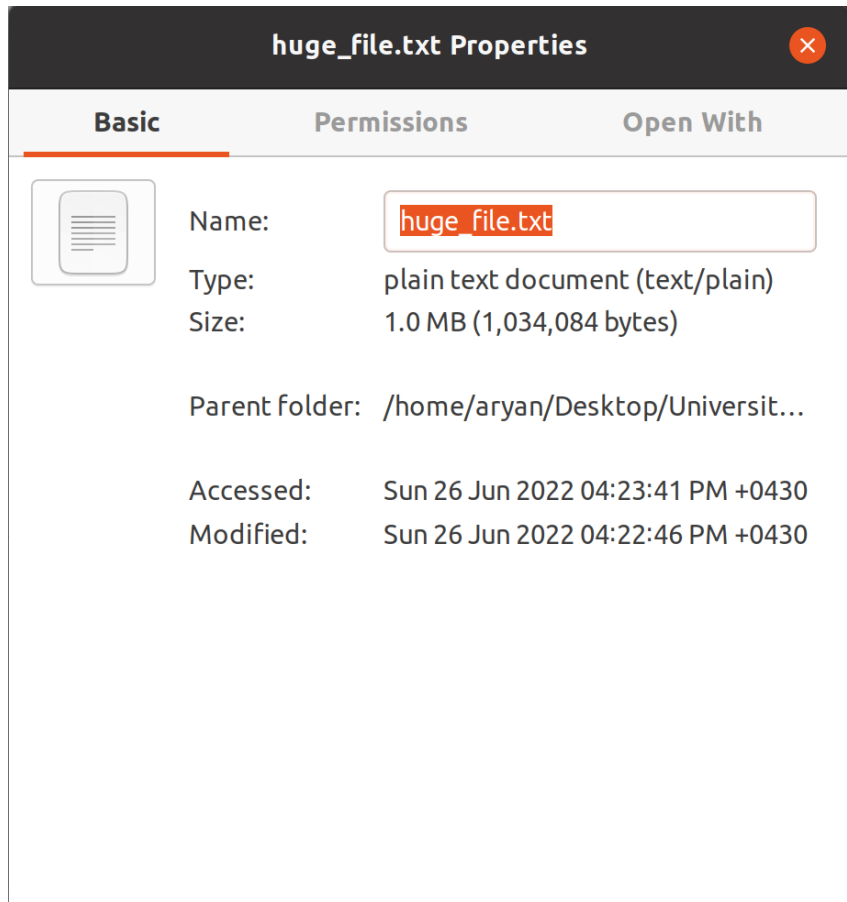
3-

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.

```
(base) aryan@aryan-UX303LB:~/Desktop/University/online_courses/Projects/CN-CA4$ ./Host.out a
socket created
connected to Router
huge_file.txt b 10 1500
Start sending
End of sending
Execution time : 37267072 microseconds
```

```
(base) aryan@aryan-UX303LB:~/Desktop/University/online_courses/Semester6/Network/Projects/CN-CA4$ ./Host.out b
socket created
connected to Router
Start Reciving
End of Reciving
Start Reciving
End of Reciving
Start Reciving
End of Reciving
Start Reciving
End of Reciving
Start Reciving
End of Reciving
Start Reciving
```

```
g++ -std=c++11 -pthread ./Build/Host.o ./Build/SocketTool.o ./Build/PacketTool.o ./Build/DataReciver.o -o Host.out
(base) aryan@aryan-UX303LB:~/Desktop/University/online_courses/Semester6/Network/Projects/CN-CA4$ ./Router.out 1000 0
socket created
Socket binded
Server is listening.
Host accepted
New Host name is a
Host accepted
New Host name is b
Host accepted
```



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.

```
370431134836382357363594494470914740972307810412179699422736269524310242910186126102361010231881023106801059498317102671151010213412048512371878
068322539810310274871567412463429599710214101034104103386952285537106854613946831012537810465510294674261053101015788711137104106106736295722685895
062232994710417724581333109533691664985781396573917611105181032732486272942331089261103210381027546779721561993382397482710449735108478101084510443
109715955883692557582984287658169929636210547452838766410936575746684310141678131106883542361095661013775878946848855738161532678191055693583103106
632881051084452478317510271025651044585486894458786438107725851391084771010461059598185410259106572611710456652166110682345371167748261097632195162
136812310482447879941453144851010810439661791084564362667310763510469593368529422107733810105210194710821021027710191071036810102710233364625545758
210848951478749394759592355367368253969195107119948544576918692193814310616676876141057615544226310102154234782551010741032768192329721010453578161
741065107917727853101172898184103141536125133961633735362162109242568104103810721033777392753727810823471051098910439855437543191251342671074261276
853884287512428735972429922862105733666343376454912527626943654336784229446910104102546217721019741127179254810591883538656466654174244104859563373
104105469814763177991512575410388810394764491026252931173331098137126310296546461610362361058469191793512629105462513766115718391019913537104511164
370431134836382357363594494470914740972307810412179699422736269524310242910186126102361010231881023106801059498317102671151010213412048512371878
068322539810310274871567412463429599710214101034104103386952285537106854613946831012537810465510294674261053101015788711137104106106736295722685895
062232994710417724581333109533691664985781396573917611105181032732486272942331089261103210381027546779721561993382397482710449735108478101084510443
109715955883692557582984287658169929636210547452838766410936575746684310141678131106883542361095661013775878946848855738161532678191055693583103106
632881051084452478317510271025651044585486894458786438107725851391084771010461059598185410259106572611710456652166110682345371167748261097632195162
136812310482447879941453144851010810439661791084564362667310763510469593368529422107733810105210194710821021027710191071036810102710233364625545758
210848951478749394759592355367368253969195107119948544576918692193814310616676876141057615544226310102154234782551010741032768192329721010453578161
741065107917727853101172898184103141536125133961633735362162109242568104103810721033777392753727810823471051098910439855437543191251342671074261276
853884287512428735972429922862105733666343376454912527626943654336784229446910104102546217721019741127179254810591883538656466654174244104859563373
104105469814763177991512575410388810394764491026252931173331098137126310296546461610362361058469191793512629105462513766115718391019913537104511164
370431134836382357363594494470914740972307810412179699422736269524310242910186126102361010231881023106801059498317102671151010213412048512371878
068322539810310274871567412463429599710214101034104103386952285537106854613946831012537810465510294674261053101015788711137104106106736295722685895
062232994710417724581333109533691664985781396573917611105181032732486272942331089261103210381027546779721561993382397482710449735108478101084510443
109715955883692557582984287658169929636210547452838766410936575746684310141678131106883542361095661013775878946848855738161532678191055693583103106
632881051084452478317510271025651044585486894458786438107725851391084771010461059598185410259106572611710456652166110682345371167748261097632195162
136812310482447879941453144851010810439661791084564362667310763510469593368529422107733810105210194710821021027710191071036810102710233364625545758
210848951478749394759592355367368253969195107119948544576918692193814310616676876141057615544226310102154234782551010741032768192329721010453578161
741065107917727853101172898184103141536125133961633735362162109242568104103810721033777392753727810823471051098910439855437543191251342671074261276
853884287512428735972429922862105733666343376454912527626943654336784229446910104102546217721019741127179254810591883538656466654174244104859563373
104105469814763177991512575410388810394764491026252931173331098137126310296546461610362361058469191793512629105462513766115718391019913537104511164
```

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.

```
5764311548565823575839544944769147469723678104121798994222758269524310242910186126102581010231881023108661059498
0683225398103102748715674124634295997102141010341041033869522855371068546139468310125378104655102946742610531010
0622329947104177245813331095336916649857813965739176111051810327324862729423310892611032103810275467797215619933
1097159558836925575829842876581699296362105474528387664109365757466843101416781311068835423610956610137758789468
632881051084452478317510271025651044585486894458786438107725851391084771010461059598185410259106572611710456652
5764311548565823575839544944769147469723678104121798994222758269524310242910186126102581010231
0683225398103102748715674124634295997102141010341041033869522855371068546139468310125378104655
0622329947104177245813331095336916649857813965739176111051810327324862729423310892611032103810
1097159558836925575829842876581699296362105474528387664109365757466843101416781311068835423610
6328810510844524783175102710256510445854868944587864381077258513910847710104610595981854102591
1368123104824478799414531448510108104396617910845643626673107635104695933685294221077338101052
```

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 ocured\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 ocured.
        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
```