DRTP

Zulfeqar Shirzadeh

16 Mai 2023

<small>Oslo Metropolitan University</small>

Contents

1 Introduction

In this portfolio we have implemented a file transfer protocol called DRTP. The application have the functionality to transfer photo and text from one end to other end. The user should specify what type of file they are going to transfer both in the client and server side.

The server can be run by writing the following in the command line.

> python3 application.py -s -i 10.x.x.x -p xxxx -f xxxx.jpg -r reliable method

On the other hand the client side can be run by writing the following in the command line.

> python3 application.py -c -i 10.x.x.x -p xxxx -f xxxx.jpg -r reliable method

If the user pleases to choose the reliability GO-back-N or Selective repeat, The uesr have the freedom to choose the window for the transfer. The user can just put a specific number after -w , but on default its 5.

At the end of every transfer info over the period of time for the transfer, the amount of data transferred and the bandwidth for the connection will be displayed on the client side. The user can choose what format for the amount of data. it can be secified after -f.

2 Background

DRPT is a reliable file transfer protocol which is build over UDP. UDP in its simplest form is a network protocol used for sending data over the internet. it establishes no connection between sender and receiver before sending data, so it simply sends information's.

The use for UDP comes more forward when the speed is more important then reliability. Since it doesn't have any overhead of establishing a connection nor verifying that the data has been delivered correctly or not. UDP transfer data faster then other protocols such as TCP since it doesn't have have any error checking or re-transmission mechanism where it can guarantee that the data will arrive in its destinations.

DRTP on the other hand will use UDP on its core but it will be a reliable transfer protocol. DRTP will always establish a three way handshake connection, where the client send sends a packet with synchronized(SYN) flag first. The server then sends an acknowledgment(ACK) for the packet received. The client will also an acknowledgment(ACK) to complete the three way handshake.

There are three functions build where it guarantee the reliability in every one of them.it means that if a packet is lost on its way to the receiver, it will be re-transmitted.

3 Implementation

For the implementation of this application there were many ways to choose, we first wanted to do it in object oriented way. But we decide not since we already had good experience from the first portfolio. Through implementation we have tried to use the **Dry** method for optimization as much as possible.

The application has two mode to run. We can run the application in the server mode or the client mode.If the user want to run both at the same time with one command, it will show an error. We will explain the server side first and then move on to the client side.

3.1 Three way handshake

Three way handshake is a process used in TCP to establish connection between a clients in a network. The three way handshake involves a message exchanged between a client and a server where it helps to synchronize the sequence and acknowledgment number.

As its mentioned previously we are building the three reliability methods over a UDP. There are no overhead establishment control in UDP that's why we should build our own three way handshake.

To initiates the connection with three way handshake. We start by sending a packet with SYN flag first where it synchronized the connections. The server will respond with A SYN-ACK packet that indicates that it has received the packet with SYN flag and is ready to communicate. Then the client will also sends a packet with ACK flag where the server indicate that it has received the packet with SYN-ACK flag.

After the three-way handshake is complete , both server and client will be synchronized and will start exchanging data using the agreed upon sequence and acknowledge numbers. This is the start of our reliable communication over UDP.

3.2 server

The server of the application will be run with the following command.

python3 application.py -s -i 10.x.x.x -p xxxx -f xxxx.jpg -r —

When the user wants to run the server, They have the freedom to choose the ip address and the port number where the application can be run. Now -f refers to what user want to call the file received. -r refers to the reliability type where the user can choose between 'stop_and_wait', 'gbn' or 'sr'.

We started with making a socket using UDP protocol. Then we bind the socket with the IP address and port number given by the user.

Before entering the while loop we creates a new object and assigns it to a variable. The open() function is a built-in Python function that is used to open files. The second argument, "wb", specifies the mode in which the file should be opened. In this case, the w stands for "write" and the b stands for "binary". This means that the file will be opened in binary mode, and any data that is written to the file will be treated as binary data.

After creating the object we enter an infinite while loop, where we constantly receive file from the client. Now when we receive the file we use a help method called receive_packet to parse the header and take out data from the received packet.

receive_packet will get the packet received with its address. Now the packets we receive has a size of 1472 bytes where first 12 of those are header. That's why we parse the header of every packet to see the sequence number, acknowledgment number, flags type and the window size. Then we again parse the flags as we have 3 types of flags.

While establishing connection with the client, We can receive packet with three types of falg, SYN, ACK and FIN. Now the client will according to flag.

If the client receives a packet with Syn flag, It will send an acknowledgment to the client. If the clients receives a packet with ACK flag. It will do nothing and continues. Now SYN and ACK is used for three way handshake establishment. If it receives packet with FIN flag it will jump out of the loop an closes the file that were opened.

The server side have three reliability methods. The user can choose which reliability it wants to run the server with. The reliability is set to stop_and_wait as a default where the client sends an ACK for every packet it receives.

If the clients wants to choose other reliability then the default, they should specify it with -r. For our server we have only gbn and sr left.if the user chooses gbn, we will check if the received packet has the expected sequence number. If it does, we writes the received data to the file object, sends an acknowledgement packet with the next expected sequence number, and updates the base variable. If the received packet is out-of-order, it will be ignored and no acknowledgement packet is sent.

Now if the user chooses sr as a server, we are gonna check if the packets are in order of not. If the packet come in order, it will then be checked if the test cases is triggered or not for that packet if yes, then we don't write or store the packet, we also wouldn't send any ACK for the packet either. But if its out of order it will be stored so that when the right time for the packet comes we can just pop it out. and send an ACK for the the packet.

3.3   Client

The client side is contained of 3 function where every function provides different type of reliability. The user can run the client by following command.

4

python3 application.py -c -i 10.x.x.x -p xxxx -f xxxx.jpg -r

User can specify the type of reliability by mentioning it after -r. If nothing is mentioned the the default is set to be stop_and_wait.

3.3.1 Stop and wait

Stop and wait is a method of communication between two devices in a network.It starts where the client sends a packet to the server and waits for the acknowledgment before sending the next message.

To implement this method we start by making a socket using UDP protocol. Since we don't have overhead establishment control for our connection we use the three way handshake to start with.

After we the three handshake is complete and the connection is established, we start by reading the file specified in the command line arguments and sends it to the server in small chunks of 1460 bytes. Each packet has a sequence number that identifies it and a data payload containing the file data.

After sending each packet, the client waits for an ACK packet from the server. If the ACK packet has the the correct sequence number, then we sends the next packet. Otherwise if we didn't receive any ACK in 5 seconds for the packet we will resend the packet again.

The code continues sending packets until it reaches the end of the file. Once all packets are sent, then we creates a packet with FIN flag and sends it to the server to indicate that the transmission is complete and the socket will will be closed.

3.3.2 Go back N

Go-back-N is a communication method used in data transmission, specifically in the Automatic Repeat Request(ARQ) protocol. In Go-back-N a sliding window protocol is used to manage the transmission and receipt of data between a client and server.

The idea behind Go-back-N is that the sender transmits multiple packets and wait for acknowledgment for each packet from the server. The client assign a unique sequence number to each packet, which allows the server to identify and send an ACK to the client.

Sliding window protocol is a reliable method which ensure the efficiency and error detection. The packets that client send to the server is in order. That's what the clients expect. So when the client receives and ACK for a packet the window will move and a new packet will be added to the window which will be send to the server. If a packet from the start of the window is missing ACK the window will not move, but if a packet which is in the middle or at last of the window, the window will move on.

For every connection we establish we start by sending a packet with SYN flag and wait for an ACK. While waiting for the ACK we can open the file we want to send in bytes mode. Now when the ACK is received and three way handshake is complete. We start a while loop where we check if we have come to the end of the file or not. We also use a queue here which follows FIFO data structure type. We make a queue of packets and send it to the server. As long as we have data to read we keep on making packet and pushing it to the queue while updating the variables.

We use the sliding window here where we remove a packet every time we receive an ACK. When we receive an ACK, we check the base and sequence number if it was correct then we remove it from the queue and add another packet in the queue.

If the client didn't receive an ACK by the timeout it will resend the packets in the window again. The re-sending depends to where in the window the ACK is missing. If everything goes as normal, then we go out of the loop as there is not data to read and send a packet with the FIN flag to indicate that the file is transferred.

3.3.3   Selective repeat

Selective repeat (SR) is an error control protocol which is used to guarantee the dependable transmission of data packets. The protocol is actually used in transmission control protocol (TCP) to make sure that the data sent over network is received by the receiver.

In SR, the sender divides the data into packets the same way as GO-back-N and sends it to the receiver. The receiver then sends and acknowledgment(ACK) for each packet received. If the sender doesn't receive an ACK for the a specific packet, it will then re-transmit only that packet instead of the entire set of packets.

In SR we uses the window the same way as GO-back-N to ensure that the sender doesn't overwhelm the receiver with too many packets.The sender and receiver maintain a window of packets based on the window size.

The implementation for SR is the same as GO-back-N. The only difference is when it come to the exception part. When the client is missing an ACK for a packet. We are not sending the whole window but the only packet which is missing. Apart from that there is not difference in the implementations.

4   Test cases

To check the efficacy of the application we are gonna run some tests with different values. We will start by changing the RTT for the stop-and-wait reliability function, but for the GO-back-N,we will be changing both the RTT and the window size for the connection.

4.0.1   Stop-and-wait

RTT	Elapsed Time	Bandwidth
25ms	3.1 sec	0.44 Mbps
50ms	6.2 sec	0.22 Mbps
100ms	12 sec	0.12 Mbps

To start for stop-and-wait we are gonna start by changing RTT from 20ms which was the default for the topology to 25ms. We see that the amount of time that took for the file to be transferred was 3.1 ms and when we raise the value for the RTT the time that the file can be transferred will also raise Which is something that's normal. But on the other hand the bandwidth gets lower and lower, that's because the bandwidth is calculated when its divided by the time elapsed which will raise proportional to the RTT.

To check if the code handle the packets loss or not. We can run the server by the following command precising that it will skip the pack with the specified number so that the client must re-transmit the packet again.

Python3 Application.py -s -f Photo.jpg -t -seqNr

The user can specify which pack should be dropped by the server so that it should be easier to track if its working properly or not.

Now there is another way we can check if its working or not. We can use netem to emulate 10% packet loss. but before that we must remove the existing rules on the network with the following command.

sudo qdisc del dev h1-etho0 root

Now we can set new rules by the following command.

sudo tc qdisc add dev h3-eth0 root netem loss 10%

This will simulates a 10% packet loss rate on h3 outgoing transmissions. Consequently, the client is required to re-transmit packets. This re-transmission process will be handled by the server and it doesn't impact the output.

Figure 1: stop-and-wati-loss

4.0.2 Go-back-N

RTT	window size	Bandwidth	Elapsed Time
	5	2.04 Mbps	0.7 sec
25ms	10	3.17 Mbps	0.4 sec
	15	3.88 Mbps	0.4 sec
	5	1.09 Mbps	1.3 sec
50ms	10	1.93 Mbps	0.7 sec
	15	2.2 Mbps	0.7 sec
	5	0.55 Mbps	2.5 sec
100ms	10	1.03 Mbps	1.3 sec
	15	1.50 Mbps	0.9 sec

To test the efficacy with the Go-back-N. We are gonna change the RTT and window size for the connection. We start by changing the RTT to 25 and run it with different window size. We see the the amount of time which is elapsed is starting getting a lower when we move the window size. That means the when we send the packet as in a window it will be buffered to the server side and will be processed very quickly. when it wasn't the case for stop and wait. Because stop and wait had to wait for an ACK before sending the next packet.
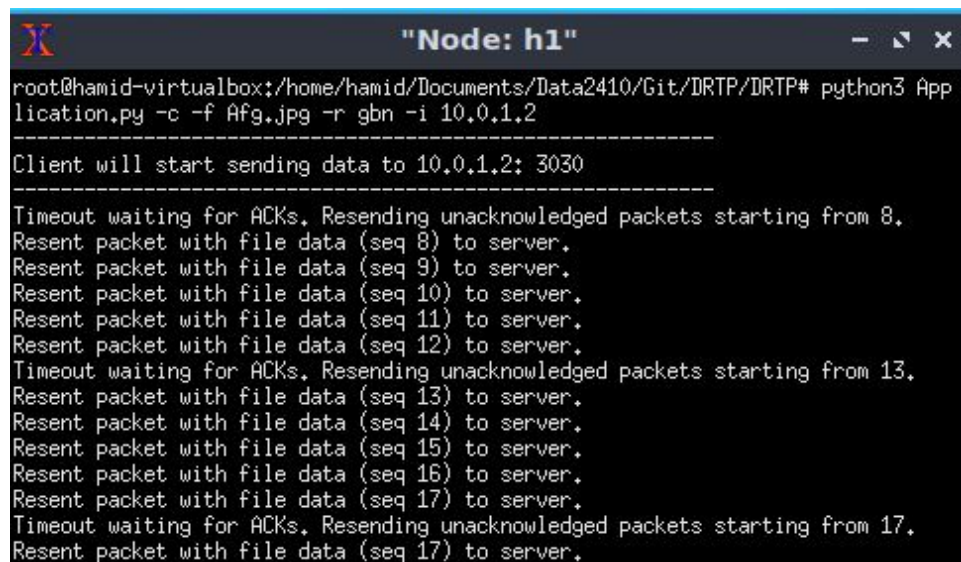
The results repeat itself for the other cases too where the the elapsed time get lower when we increase the window size. Now by that the probability for the the dropped packets will also raise as if the buffer will be fulled, then the packet will be dropped.

8

To check if we the GO-back-N can handle the re-transmission of the window. We can run server to skip an ACK with -t. That way we will see that the whole window will be re-transfers and the amount of transfer data will also get higher from the original as we send the same packet multiple times.

We can also emulate the packet loss here too with netem. We check if the it can handle the packet loss first. To check the packet loss we remove the rules from h1 and sett new rules on it with the following command.

sudo tc qdisc add dev h1-eth0 root netem loss 10%

With the command above, we are dropping 10% of the outgoing packet from the client side. When the client doesn't receive an ACK , It will start re-sending starting from the last ACK received.

Figure 2: Go-back-N-loss

We can check if the server also can handle if the packets come out of order. We remove the existing rules with command above and sett a new rule so that 50% of the packets will get a delay of 40ms.

sudo tc qdisc add dev h3-eth0 root netem delay 40ms reorder 50% 50%

Now we see that some of the packets arrive to the server with 40ms delay. That's why the server will discard those that comes out of order and wait for it.

Figure 3: Go-back-N-reorder

4.0.3 Selective repeat

For the selective repeat we run the same tests as for GO-back-N. We change the
RTT and run it with different window-size. Now the we expect that the result
in selective repeat should be similar to go-back-N as they are following the same
rules aside from the exception part.

RTT	window size	Bandwidth	Elapsed Time
25ms	5	2.15 Mbps	0.6 sec
	10	3.33 Mbps	0.4 sec
	15	4.26 Mbps	0.3 sec
50ms	5	1.09 Mbps	1.3 sec
	10	2.03 Mbps	0.7 sec
	15	2.77 Mbps	0.5 sec
100ms	5	0.54 Mbps	2.5 sec
	10	1.04 Mbps	1.3 sec
	15	1.49 Mbps	0.9 sec

The difference here is that when a packet is missing, the re-transmission will
happen only for that packets which will result to a lower elapsed time. When
the server receive a packet and its out of order, it will send the ACK for the
packets recived but it wont be written on the file. They will be stored until its
time comes, it will removed be from where its stored to make place for the other
packets.

The result we got for the test cases are not surprising. It again important to
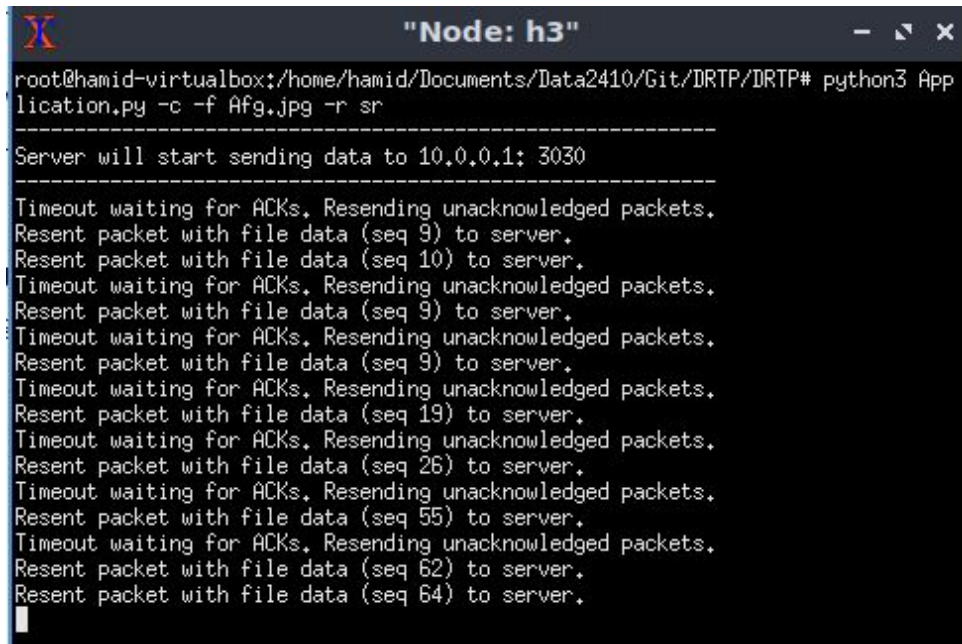know that the higher the window size is, the higher the probability for the packet

loss will be.

To check the efficacy of the code, we can again skip and ACK for a packet number with -t. The client wont receive the ACK and it will go the the timeout which will result to re-sending only the packet that misses the ACK, not the whole window.

we can also emulate the packet loss here with netem to see if the whole window will be send or only the missing packet. Now for the client side we can see it by using wire-shark, but here we can use the client side throughput.

We remove again the existing rules from the network and sets a new rule with 10% loss with the following command.

sudo tc qdisc add dev h3-eth0 root netem loss 10%

Figure 4: Selective-repeat-loss

5 Conclusions

We see that the server re-transmit the packet with the missing ACK and then it will move on.

In light of the evidence presented, it is clear that the protocol we have implemented supports the re-liabilities such as stop-and-wait, go-back-N and selective repeat while using the three way handshake. Since we don't have an overhead

11

connection establishment, We use the three way handshake to make the connection reliable.

The application is implemented using functions. We have two main functions in core, the server and the client. This two main functions will call the other functions when the right flag is triggered from the terminal.

From the implementation of this application we have come to the following conclusion. We know that Stop-and-wait works, but its too slow as we wait ever-time for an ACK. it take RTT time for every packet to send. We know that Go-back-N is better then Stop-and-wait, but we are still very slow as if a packet is lost on its way to the server, the whole window will be re-transmitted. Now with the selective repeat which a better version of Go-back-N as if a packet is lost in the window. Then the only missing packet will be resent to the server.