Inspera candidate number: 316

Project: UDP Server and client application with DRTP

Project document:

in this project I aimed to write a file transfer protocol that uses User Datagram Protocol (UDP) in transport layer. DRTP (Datagram reliable transfer protocol) is used to transfer a file reliably from client to server.

Application's functionality:

1. Initialization: The application can be run in two modes - as a server or as a client. This is determined by the arguments passed in the command line.

2. Connection Establishment: The client first initiates a connection to the server using a three-way handshake (SYN, SYN-ACK, ACK). This is a common method used in protocols like TCP to establish a connection.

3. File Transfer: Once the connection is established, the client starts transferring the file using a Go-Back-N scheme. It's a method used for error recovery and flow control, where the sender transmits multiple packets (as per the window size) without waiting for an acknowledgement (ACK), but retransmits all packets in the window if ack is not received.

4. Error Handling: The application has functionality to handle packet loss or errors. If the server does not receive a particular packet (determined by sequence number), it discards all following packets within the sliding window until it receives the missing one. The client, on the other hand, retransmits all packets in the window if it doesn't receive an ACK within a certain timeout.

5. Connection Termination: After the file transfer finished completely, the client sends a FIN packet to the server to give a signal that client wants to terminate connection. The server replies this with a FIN-ACK and the connection is terminated.

6. Throughput Calculation: After the termination of the connection, the server calculates the throughput and prints it. Throughput is the amount data transfered over a communication channel per second.

To ensure transfer datagrams reliabily, some concepts including Three-way handshake, sequence number (seq_num), acknowledgment number (ack_num) and Go-Back-N (send_file_gbn()) is used.

Source code documentation:

DRTP.py

DRTP class:

Class representing DRTP (Datagram Reliable Transport Protocol). This class contains a constructor and three functions.

def __init__(self):

This is the constructor of the DRTP class. It initializes the sequence number and confirmation number to 1.

self.seq_num = 1
self.ack_num = 1

def create_packet(self, data, syn=False, ack=False, fin=False, rst=False):

This function creates a flag based on the input parameters syn, ack, fin, rst. After that creates a header which includes the sequence number, acknowledgment number and flags. Finally function sets the header and the data in a packet. Sequence number is increased at the end.

arguments:
Data: The payload of the packet.
syn, ack, fin, rst: flags for packet headers.

Returns: packet with headers and data.

def parse_packet(self, packet):

This function extracts the header and data from the packet, unpacks the header into a sequence number, acknowledgment number, and flags. It returns all extracted values.

Arguments: self, packet.

Returns: Sequence number, acknowledgment number, flags and data.

def send_ack(self, socket, addr, fin=False, rst=False)

This function creates an acknowledgment packet with the optional Fin and Rst flags and sends the packet to the specified address via socket.

Arguments:
socket: The socket used to send acknowledgment.
Address: The address to which the acknowledgment will be sent.
fin, rst: Optional flags.

application.py

def server_program(ip, port, discard_seq):

This function creates a socket, binds the socket to the provided IP and port, and listens for incoming connections. It has the connection establishment, data transfer and connection termination phases. In connection establishment and termination phase it uses three-way handshake procedure. Server excepts packets only if they are in order. In case of receiving a out of order packet, server has ability to handle the situation and takes packets in order.

Arguments:

ip: The IP address of the server.
Port: The port number of the server.
Discard_seq: Sequence number of the packet to be discarded.

def send_file_gbn(socket, host, port, file name, window size):

The duty of this side function is sending files using the Go-Back-N aproach. It reads the file which is going to be sent and divides the data into 994 bytes pieces. Then by using the functions of drtp class sets the data into packet which also includes header. Packets created are sent within a window of a specified size. Managing lost/not acknowledged packets is also duty of this function. If a packet lost or not acknowledged, function retransmits all the packets within the window-size with starting that packet which is lost. Function sends relevant/necessary feedback for each situation.

Arguments:

socket: The socket used to send files.
Host: The IP address of the host.
Port: The port number of the host.
Filename: The name of the file to be sent.
window_size: The size of the sliding window of the GBN protocol.

def client_program (host, port, file name, window size):

This function performs client operations. As server does, this function has connection establishment and termination phase and transfer phase. In connection establishment and termination phase, function uses three-way handshake approach which is including sending SYN, waiting SYN-ACK and sending ACK again. In transfer phase function calls send_file_gbn() function to handle the situations may occur while a packet being transfered.

Arguments:

Host: The IP address of the host.
Port: The port number of the host.
Filename: The name of the file to be sent.
window_size: The size of the sliding window.

Discussion:

1) Running the application with window size 3, 5, 10 (RTT 100 ms):

window_size = 3:

```
19:30:31.665668 -- packet 1836 is received
19:30:31.666026 -- sending ack for the received 1836
19:30:31.693280 -- packet 1837 is received
19:30:31.693764 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.24 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

window-size = 5:

```
19:27:05.763715 -- packet 1836 is received
19:27:05.764203 -- sending ack for the received 1836
19:27:05.785798 -- packet 1837 is received
19:27:05.786171 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.39 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

window-size = 10:

```
19:23:39.417079 -- packet 1836 is received
19:23:39.417179 -- sending ack for the received 1836
19:23:39.424700 -- packet 1837 is received
19:23:39.424783 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.78 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

Test evaluation:

Results

With -w 3: throughput = 0.24 MB/s
With -w 5: throughput = 0.39 MB/s
With -w 10: throughput = 0.78 MB/s

Throughput increases with increment in window size. This is expected in my custom Go-Back-N implementation. Larger window size allows more packets to be in flight before waiting for acknowledgment, so with a larger window size can cause to use the network more efficiently.

As a curious user I tested my application even way larger window sizes such 50, 100, 200. Out of this test I concluded that increment of throughput by increasing the window size has a limit. After a point application reaches the limit of the network and packets started to be lost.

If we take a closer look at the test results:

Window size 3: The sender has ability to send 3 packets before starting to wait for an ACK. This causes a lower throughput (0.24 MB/s). As the sender using time mostly to wait an ack instead of sending packets, resulting in underutilization of network bandwidth.

Window size 5: In this scenario throughput inproved to 0.39 MB/s, because the sender can send more packets before waiting for an ACK. This reduces waste of time and improves network utilization.

Window size 10: With window size 10 the increment in throughput (0.78 MB/s) is more significant. This indicates a more effective usage of network resources because the more packets are in flight, the less time the sender waits for an ACK.

Conclusion:

According to the test results, it is obvious that increasing the window size and with the RTT of 100 mis will significantly improve the throughput until a limit. The limit is bandwidth, network and hardware circumstances. If the window size is set over the bandwidths limit, it will cause congestion and packet loss and effects throughput negatively.
Besides that, with way larger window sizes, in case of a packet loss sender has to retransmit more packets. This has a negative effect on effectiveness of the process.
For example if we set the window size 1837 which is the total number of packets I tried to send in this test, client would try to send all packets in once if one them for example 10. packet is lost the client has to retransmit 1927 packets again.
As a conclusion I may say that increment on window size has positive effect on throughput until a point.

2) Modifying the RTT

RTT 50 ms

window size 3:

```
19:48:41.818377 -- packet 1836 is received
19:48:41.818569 -- sending ack for the received 1836
19:48:41.835148 -- packet 1837 is received
19:48:41.835416 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.47 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

window size 5:

```
19:50:38.356307 -- packet 1836 is received
19:50:38.356513 -- sending ack for the received 1836
19:50:38.367454 -- packet 1837 is received
19:50:38.367677 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.78 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

window size 10:

```
19:51:57.837729 -- packet 1836 is received
19:51:57.838011 -- sending ack for the received 1836
19:51:57.840513 -- packet 1837 is received
19:51:57.840661 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 1.56 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

RTT 200:

window-size = 3:

```
19:59:12.300325 -- packet 1836 is received
19:59:12.300675 -- sending ack for the received 1836
19:59:12.468816 -- packet 1837 is received
19:59:12.469167 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.12 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

window-size = 5:

```
20:01:43.942254 -- packet 1836 is received
20:01:43.942542 -- sending ack for the received 1836
20:01:43.976586 -- packet 1837 is received
20:01:43.976912 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.20 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

window-size = 10:

```
20:04:16.151168 -- packet 1836 is received
20:04:16.151507 -- sending ack for the received 1836
20:04:16.163419 -- packet 1837 is received
20:04:16.163833 -- sending ack for the received 1837
....

FIN packet is received
FIN ACK packet is sent

The throughput is 0.39 Mbps
Connection Closes
root@harun-Surface-Pro-4:/home/harun/HomeExam#
```

Evaluation of test results:

Results:

Round-Trip time 200 milliseconds

-w 3: throughput = 0.12 Mbit/s
-w 5: throughput = 0.20 Mbit/s
-w 10: throughput = 0.39 Mbit/s

RTT = 100 ms

-w 3: throughput = 0.24 Mbit/s
-w 5: throughput = 0.39 Mbit/s
-w 10: throughput = 0.78 Mbit/s

RTT = 50 ms

-w 3: throughput = 0.47 Mbit/s
-w 5: throughput = 0.78 Mbit/s
-w 10: throughput = 1.56 Mbit/s

Same as the previous test scenario, throughput inproves with window size regardless of RTT. This confirms that larger window sizes allow more packets to be in flight simultaneously, thus helping to improve the throughput.

Impact of Round Trip Time (RTT):

In this scenario the impacts of modifying RTT on throughput can observed clearly. With the decrement of the RTT from 200 ms to 100 ms then 50, the improvement on throughput is observed clearly. Although it also depends other circumstances, according to results of this test it will not be incorrect to say that the throughput is doubled while RTT is halved. But it would be wrong to say that largest window-size and smallest RTT is the best combination in every circumstances. As I mentioned previous test, bandwidth, network and hardware circumstances are determining factors.

If we need to discuss only results of this test scenario and result of it, it is obvious that a higher RTT results in lower throughput. Example: window size 10:

◦ RTT 50ms: 1.56Mbps
◦ RTT 100 ms: 0.78 Mbps
◦ RTT 200ms: 0.39Mbps

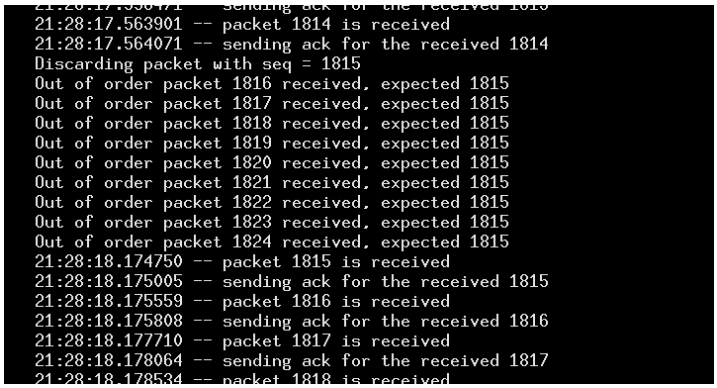
if we evaluate the spesific results of the test conducted with using -w 3, -w 5 and -w 10:

-w 3: With a RTT of 200 ms and a window size of 3, the throughput is 0.12 Mbps. This low throughput is caused by the high latency of acknowledgments, which limits the number of packets that can be sent before the sender has to wait for ack.

-w 5: Increase window size to 5 to increase throughput to 0.20 Mbps. While this is an improvement, it is still relatively low due to the higher RTT.

-w 10: A window size of 10 results in a throughput of 0.39 Mbps. Although this is the highest throughput for an RTT of 200 ms, it is significantly lower than the throughput observed for the same window size at lower RTT.

3. -d discarding a packet in server:



My custom DRTP (Datagram Reliable Transport Protocol) ensures correct and orderly transmission of data from sender to receiver over unreliable networks.

My transport protocol includes the following specifications to ensure reliability:

A. Acknowledgment (ACK): When the receiver receives a packet, it sends an acknowledgment back to the sender. If the sender does not receive an acknowledgment within a certain period of time (timeout), the packet is considered lost and resent. This happens in my application when catching socket.timeout exception.

B. Sequence Number: Each packet is assigned a unique sequence number. The receiver uses these numbers to put the data back in the correct order and to identify and discard duplicate packets.

C. Window/Flow Control: The sender maintains a window of packets that have been sent but have not yet received an acknowledgment. Once confirmation is received, this window will move forward. In my application this is achieved using "sliding_window". The size of the window determines the number of packages that can "fly" at the same time. This helps control the flow of data so that the receiver is not overloaded.

D**. Retransmission:** When the sender detects (via a lost ACK) that a packet has been lost, it retransmits the packet. In my application this is handled in the “except socket.timeout” block of the “send_file_gbn()” function.

Conclusion:

Briefly, in my code, when the server drops a packet (specified by the -d command line parameter), it sleeps more than timeout period which is defined in client side of the code (0.5s). This cause client not to take an ack for that packet within the timeout. If the client does not receive an ACK within the timeout period, the packet will be retransmitted as well as the packets within the window size. This mechanism ensures that data can be transmitted reliably even if packets are lost.

4. Test with tc-netem

A. test with a loss rate %2 and 5% with RTT 100 ms.

No Packet Loss

Window Size -w 3: Throughput = 0.24 Mbps
Window Size -w 5: Throughput = 0.39 Mbps
Window Size -w 10: Throughput = 0.78 Mbps

2% Packet Loss

Window Size -w 3: Throughput = 0.18 Mbps
Window Size -w 5: Throughput = 0.24 Mbps
Window Size -w 10: Throughput = 0.40 Mbps

5% Packet Loss

Window Size -w 3: Throughput = 0.15 Mbps
Window Size -w 5: Throughput = 0.16 Mbps
Window Size -w 10: Throughput = 0.21 Mbps

Evalution of test results

Impact of packet loss:

Packet loss has a significant impact on throughput. Throughput is reduced for all window sizes compared to the case without packet loss. The higher the packet loss rate, the more obvious the throughput decrease due to increased retransmission requirements.

Window size and packet loss rate

With a window size of 3, the throughput is effected negatively when we conducted test with 2% loss rate and 5% loss rate (from 0.24Mbit/s (no packet loss) to 0.18Mbit/s (2% packet loss rate) and 0.15Mbit/s (5% packet loss rate)). With the smaller window sizes fewer packets being in flight and the negative effect of time loss while retransmissions of lost packets being more pronounced.

If we test the application with a window size of 5, change in the throughput was like: from 0.39 (no loss) to 0.24 Mbps (2% loss) and 0.16 Mbps (5% loss). Although enlarging the window sizes help to minimize the effects of packet loss it has limit. But in our case throughput still drops obviously as packet loss rate increases.

When the same test conducted with window size 10, the throughput decreases again from 0.78 Mbps with no loss, to 0.40 mbps with 2% loss rate and finally to 0.21 Mbps with loss rate of 5%. Although a larger window size allows more packets to be in flight, packet loss can still negatively effect the throughput because of retransmissions.

Overall conclusions and findings:

The tests performed in this project included modifying the window sizes, RTT, and packet loss rates due to observe the effects of the changes in RTT, window size and packet loss rate. Th etest results provide clear insight upo the performance of the file transfer application which is the main subject of this project. After the series of tests performed its easier to understand the relationship between RTT, window size and effects of the packet loss cases on performance of the application.

Here are some findings across the test process:

Larger window size can effect throughput positively to an extent. This is because a larger window-size allows more packet to be transmitted at same time, besides reducing the time loss experienced by the sender while waiting for acknowledgment.

As the network conditions worsen, such as higer RTT and packet loss rate, the benefits of increasing the window-size diminish.

A lower RTT can improve throughput because acknowledgments and packets are transferred faster.

A higher RTT has negative impact on throughput, because it takes longer for a packet to reach form one to another. Affects are more visible when the window size smaller.

Packet loss has negative impact on throughput. As the packet loss rate increases, more packets are lost and need to be retransmitted, causing additional delays and reducing the effectiveness of data transfer.

Larger window sizes minimize the impact of packet loss to some extent by keeping more packets in flight. However the overall throughput still decreases with higher loss rate. For instance, a window size of 10 maintains better throughput under packet loss conditions compared to smaller window sizes, but the throughput reduction is still significant.