Multi-threaded downloader for parallel data download based on User Datagram Protocol (UDP)

50.012 Networks

Beta Version
Cohort 1 Team 1

Team Members :

Kimberlyn Loh	(1002221)
Tan Yi Long	(1001566)
Chen Wenshu	(1002291)
Delbert Felix Nurawan	(1002374)

Table of Content

1. Introduction

1.1. General Idea

While Transmission Control Protocol (TCP) provides a host of benefits through its various features, it comes at a costly trade-off of significant delays due to connection setup and congestion control when loading web pages. User Datagram Protocol (UDP), on the other hand is a no-frill protocol that has the potential to achieve similar or even better performance than TCP.

In addition, multithreaded downloading utilizes more bandwidth with more number of connections to the server compared to the conventional single threaded downloading. It is possible to download different segments of a large file from the server simultaneously, and merge them in the right sequence into the original file on the client.

This project seeks to develop a multi-threaded downloader based on UDP that can match or even outperform an equivalent TCP downloader. In addition, the project github repository can be found here.

1.2. Network Topic implemented

This type of network topic that will be thought of and implemented throughout this project is listed as below.

Network Topic	Description
Service provided by Transport Layer	The type of services provided by the transport layer are: ● Reliable delivery To ensure the data receive is complete (resending of lost packet) ● Same order delivery To ensure the data is in order ● Flow control Thus, the implementation of the multi-threaded downloader have to follow the services provided by the Transport layer and be reliable, ordered and efficient.
Segmentation and Reassembly	In a packet-switched telecommunication network, segmentation and reassembly is the process of breaking a packet into smaller units before transmission and reassembling them into the proper order at the receiving end of the communication. Packets are made smaller to speed them through the network and specifically because of specified packet size restrictions in a given path. Thus, the implementation would require packets to be segmented properly.
Congestion Control	The implementation would require congestion control to increase efficiency so as to prevent congestion or help mitigate congestion after it has occur.

2. Transport Protocol

2.1. Introduction

Transmission Control Protocol (TCP) provides a reliable, connection-oriented service to the invoking application, whereas User Datagram Protocol (UDP) provides an unreliable and does not require a connection.[1] However, for this project, TCP will become inefficient as the network bandwidth and delay increase. Thus, UDP will provide a simple transmission model without implicit handshaking techniques.

On the other hand, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. Thus, before commencing the main part of the project, which is the multi-threaded downloader, literature research was done to venture the types of UDP available.

2.2. Types of reliable UDP

2.2.1. Reliable User Datagram Protocol (RUDP)

Reliable User Datagram Protocol (RUDP) is a transport layer protocol where reliability is achieved using the Selective Repeat version of the Sliding Window protocol, whereby acknowledgements and timeouts to achieve the goal of reliable data transfer. [2] A simple view of the protocol can be seen in the figure below.

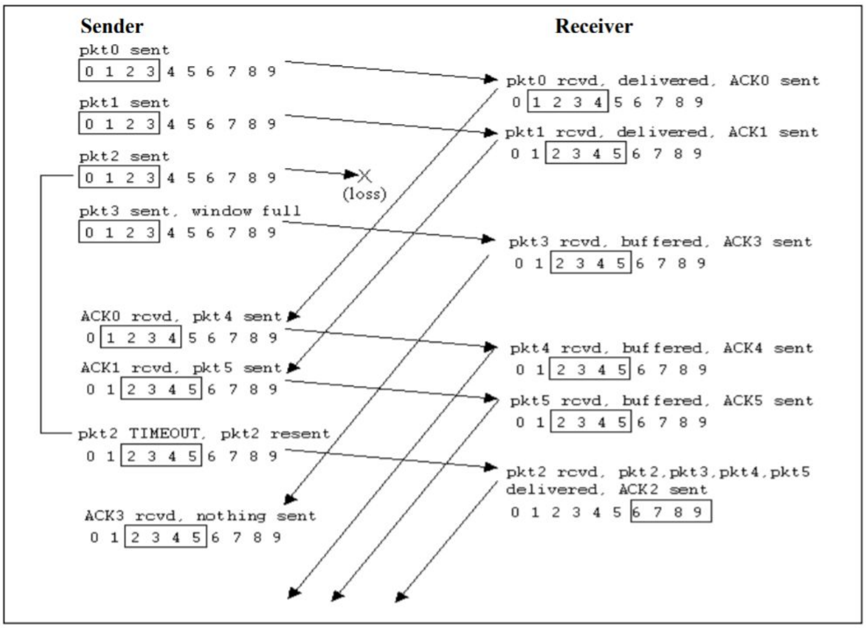


Figure 2.2.1. RUDP

2.2.2. Trivial File Transfer Protocol (TFTP)

TFTP is a very simple protocol used to transfer files. Each nonterminal packet is acknowledged separately. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. [3] When packets are lost, the intended recipient will timeout and resend that lost packet. All packets are sent with UDP.

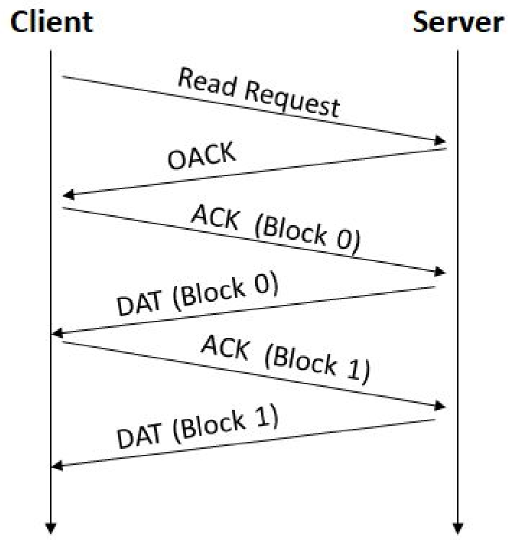


Figure 2.2.2 TFTP

2.2.3. Reliable Blast UDP (RBUDP)

RBUDP is a data transport tool and protocol specifically designed to move very large files over wide area high-speed networks. RBUDP keep the network pipe as full as possible during bulk data transfer and avoid TCP's per-packet interaction. [4] This is done by sending aggregated acknowledgments at the end of a transmission phase which can be seen in the figure below.

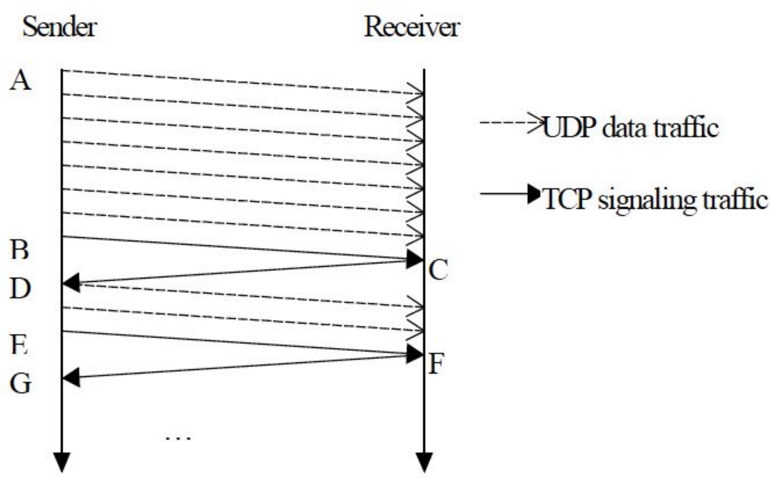


Figure 2.2.3 RBUDP

2.3. Efficiency on the type of UDP

2.3.1. Experimental Procedure

The code for the implementation of 2.2 can be found here and the command for running each code can be found in the Appendix.

Experiment conducted on each protocol, on a single thread, to determine best performance in terms of throughput and packet loss.

1. Setup server and client end-points.
   Server contains 2 files to be downloaded by client, test.txt (1 kb) and testHuge.txt (109 kb).

2. Client requests test.txt from server and downloads it.
   Record time taken for download and number of packets lost.

3. Repeat download process in step 2 for testHuge.txt.
   Record time taken for download and number of packets lost.

After conducting this experiment for the 4 transfer protocols, the throughput and packet loss recorded.

2.3.2. Experimental Result

The summary of the experimental result is as shown below and the screenshot result of each protocol can be found in the Appendix

.

Protocol	Average Time Taken/ s	Packet Loss
RUDP	Small file : 0.03	No
	Large file : 3.19	No
TFTP	Small file: 0.01	No
	Large file: 1.03	No
RBUDP	Small file: 0.04	No
	Large file: 0.14	No

Even though RBUDP have suffered packet loss while transmitting,, it retransmits back the lost packet at the end, which will result in no packet loss. In summary, RBUDP is efficient in transmitting large file but not small file. Hence, the protocol chosen to implement on the multi-threaded downloader, where large files are downloaded, is RBUDP as it is the most efficient among all the reliable UDP.

3. Multi-threaded downloader

3.1. Introduction

This is a multi-threaded downloader for parallel data transfer that would do parallel data download and merge in proper order on the client side. The downloader uses TCP connections to communicate between the server and the client for access control, and UDP connections to transfer the file. The diagram below illustrates the process.

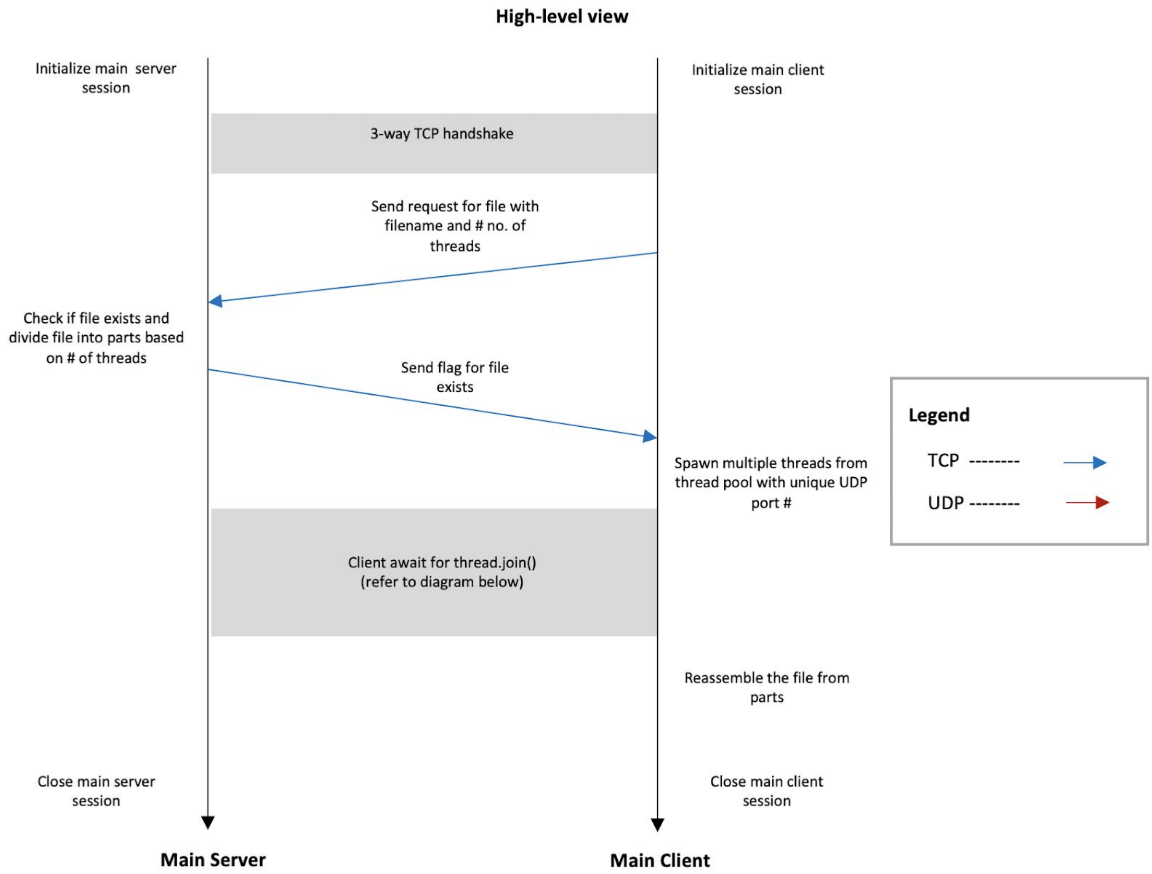


Figure 3.1.1 Client and Server Connection (Part 1)

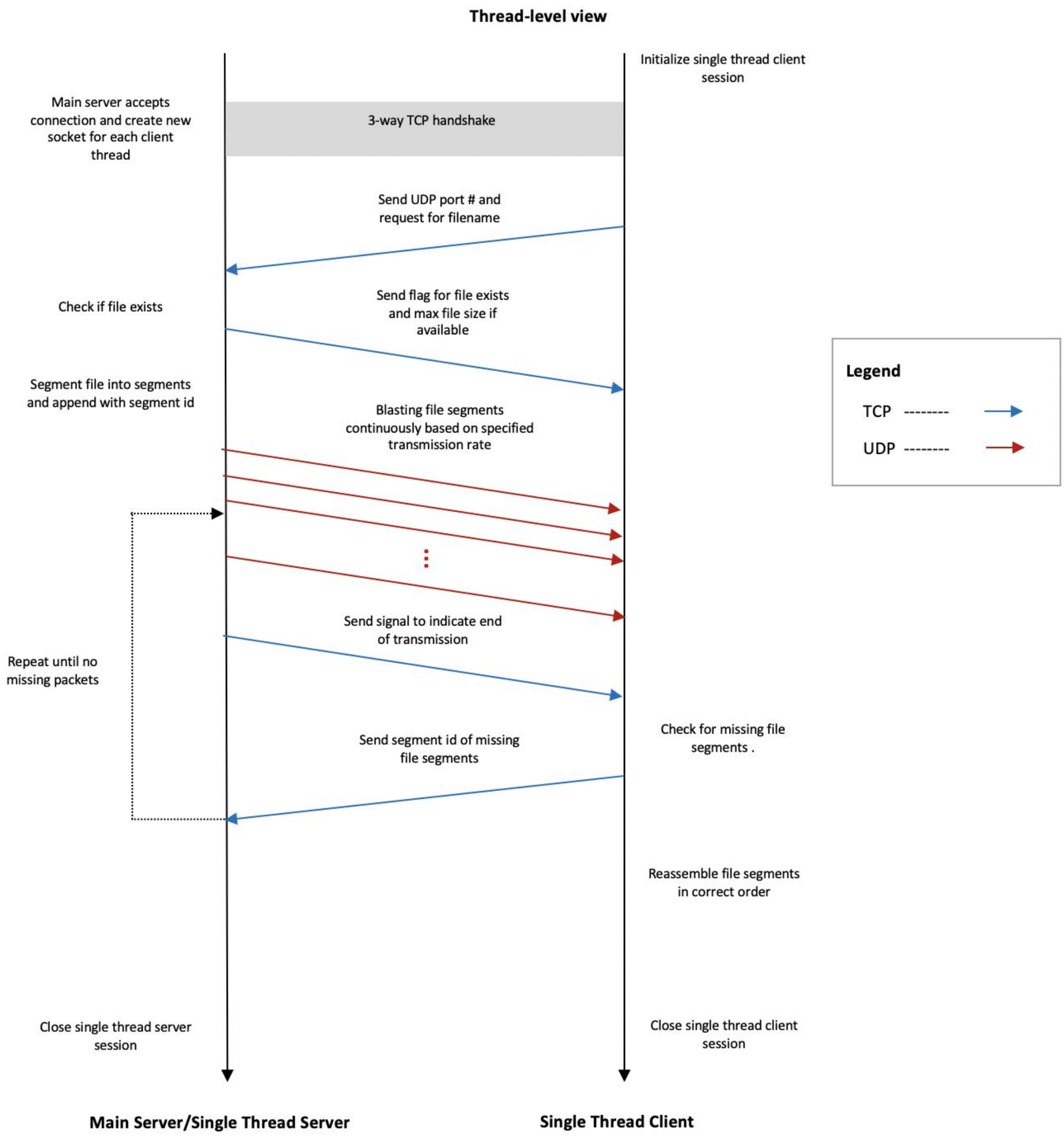


Figure 3.1.2 Client and Server Connection (Part 2)

The client and server connection can be described as followed

1. The client first sends over the number of threads and the name of the file to download through TCP.
2. The server receives and divides the file into smaller segments accordingly. It then sends over the flag that indicates whether the file exists on the server.
3. The client creates threads. Each thread creates a socket that establishes a TCP connection with the TCP socket on the server side.
4. The client's thread sends over the client's name and its UDP port number that this thread will use through its unique TCP connection. It also sends over the agreed filename of the particular segment to download through the TCP connection.
5. The server creates threads. Each thread sends blocks of the file segment to the client's thread through the UDP connection. Once the transmission is done, the server's thread signals the client's thread through the TCP connection.
6. The client's thread receives blocks of the file segment, and saves them into a single file after it receives the DONE signal from the server's thread.
7. Finally, the client combines the segments of the file in sequence and obtains the complete file.

3.2. How to run the program

A detailed guide can be found here in README.md.

4. Conclusion

With network bandwidth and delay increase in a multi-threaded downloader system, TCP will be inefficient and the usage of UDP is needed as it has a simple transmission model without implicit handshaking techniques. However, UDP is unreliable and there is a need for reliable UDP. Thus, the team researched on the various type of reliable UDP and deduce based on experimental result that RBUDP is the optimal reliable UDP to be used in this project.

Some of the future works that can be done is to experiment all the found types of reliable UDP protocols with multi-threaded and see if there is a significant difference. In addition, the project can be further implemented to become an extension to a browser, for faster multiple data download. .

Beta version of the multi-threaded downloader does not include a GUI and extensive unit tests. Execution of multi-threaded downloader is run from the command line interface.

5. Reference

[1] Kurose, J. F., & Ross, K. W. (2017). *Computer networking: A top-down approach*. Boston: Pearson.

[2] Thammadi, A. (2011). *Reliable user datagram protocol (RUDP)* (Unpublished master's thesis).

[3] Sollins, K. R. (1992). The TFTP Protocol (Revision 2)

[4] Eric, H., Jason, L., Oliver, Y., & Thomas, D. A. (2002). *Reliable Blast UDP : Predictable High Performance Bulk Data Transfer* (Tech.). Chicago, Illinois: IEEE Cluster Computing.

Appendix

Github repository : https://github.com/chenwenshu/multithreaded-downloader

Running of each type of reliable UDP

Type	Client side	Server side

RUDP	`python3 RUDP_client.py [ip address of server] [port number of server UDP socket] [file name to be requested]`
	
	E.g
	`python3 RUDP_client.py 127.0.0.1 60000 127.0.0.1 test.txt`
		`python3 RUDP_server.py [ip address of server] [port number of server UDP socket]`
		
		E.g
		`python3 RUDP_server.py 127.0.0.1 60000 127.0.0.1 test.txt`

TFTP	`python3 TFTP_client.py -H [ip address of server] -D [file name to be requested]`
	
	E.g
	`python3 TFTP_client.py -H 127.0.0.1 -D test.txt`
		`python3 TFTP_server.py -r ./`
		
		*-r is the root location where the file to be download could be found
		
		E.g
		`python3 TFTP_server.py -r ./`

RBUDP	`python3 RBUDP_client.py [ip address of client] [port number of client UDP socket] [ip address of server] [file name to be requested]`
	
	E.g
	`python3 RBUDP_client.py 127.0.0.1 60000 127.0.0.1 test.pdf`
		`python3 RBUDP_server.py [ip address of server] [transmission rate of UDP socket in Mbps]`
		
		*Note that the transmission rate should be lower than the overall throughput rate
		
		E.g
		`python3 RBUDP_server.py 127.0.0.1 1000.0`

Result for each type of reliable UDP

- RUDP
  - Small file size of 1kB

```
C:\WINDOWS\system32\cmd.exe - python  RUDP_server.py 10.12.67.63 10000
Elapsed: 3.15545392036438
Length: 500
Sequence number: 0
Sent 559 bytes back to ('10.12.0.51', 64239), awaiting acknowledgment..
Acknowledged by: 0,500
Acknowledged at: 2018-11-07 05:07:08.172940
Elapsed: 3.175668954849243
Length: 145
Sequence number: 1
Sent 204 bytes back to ('10.12.0.51', 64239), awaiting acknowledgment..
Acknowledged by: 1,145
Acknowledged at: 2018-11-07 05:07:08.187941
Elapsed: 3.1906700134277344

Done in within : 3.193814992904663
Packet Loss : 0
Received 8 bytes from ('10.12.0.51', 62238)
Waiting to receive message
Request started at: 2018-11-07 05:07:38.892946
Opening file b'test.txt'
Length: 149
Sequence number: 0
Sent 208 bytes back to ('10.12.0.51', 62238), awaiting acknowledgment..
Acknowledged by: 0,149
Acknowledged at: 2018-11-07 05:07:38.919285
Elapsed: 0.026339054107666016

Done in within : 0.028332948684692383
Packet Loss : 0
```

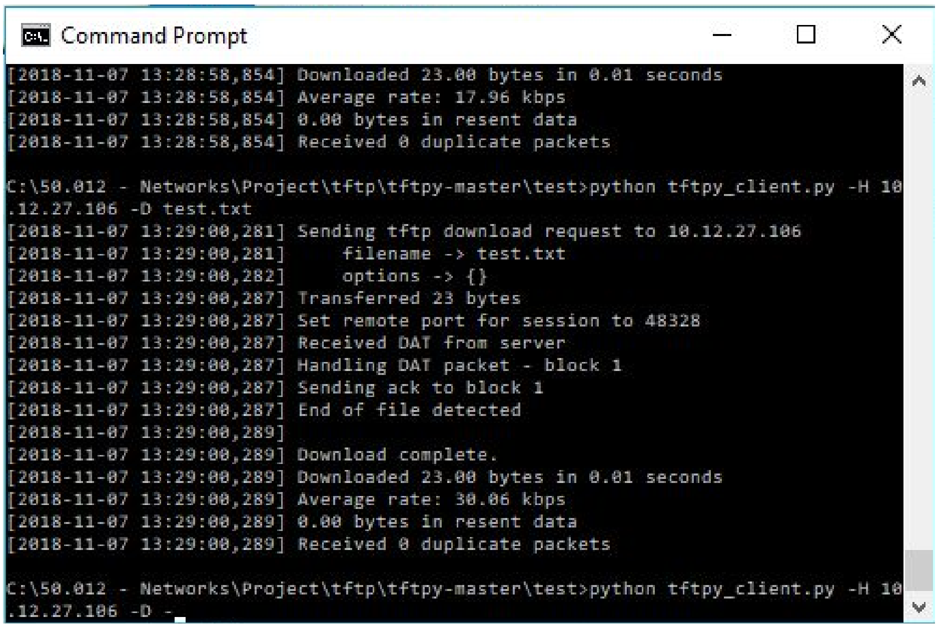
  - Large file size of 109kB

```
C:\WINDOWS\system32\cmd.exe - python  RUDP_server.py 10.12.67.63 10000
Acknowledged at: 2018-11-07 05:07:08.117871
Elapsed: 3.1205999851226807
Length: 500
Sequence number: 0
Sent 559 bytes back to ('10.12.0.51', 64239), awaiting acknowledgment..
Acknowledged by: 0,500
Acknowledged at: 2018-11-07 05:07:08.135952
Elapsed: 3.138680934906006
Length: 500
Sequence number: 1
Sent 559 bytes back to ('10.12.0.51', 64239), awaiting acknowledgment..
Acknowledged by: 1,500
Acknowledged at: 2018-11-07 05:07:08.152725
Elapsed: 3.15545392036438
Length: 500
Sequence number: 0
Sent 559 bytes back to ('10.12.0.51', 64239), awaiting acknowledgment..
Acknowledged by: 0,500
Acknowledged at: 2018-11-07 05:07:08.172940
Elapsed: 3.175668954849243
Length: 145
Sequence number: 1
Sent 204 bytes back to ('10.12.0.51', 64239), awaiting acknowledgment..
Acknowledged by: 1,145
Acknowledged at: 2018-11-07 05:07:08.187941
Elapsed: 3.1906700134277344

Done in within : 3.193814992904663
Packet Loss : 0
```

➔ TFTP

- Small file size of 1kB

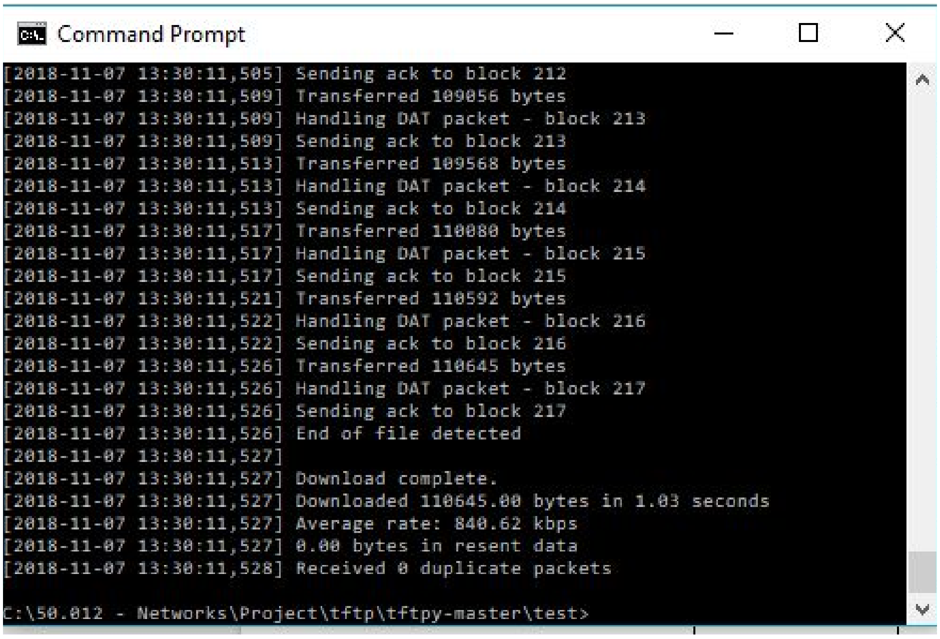


```
Command Prompt
[2018-11-07 13:28:58,854] Downloaded 23.00 bytes in 0.01 seconds
[2018-11-07 13:28:58,854] Average rate: 17.96 kbps
[2018-11-07 13:28:58,854] 0.00 bytes in resent data
[2018-11-07 13:28:58,854] Received 0 duplicate packets

C:\50.012 - Networks\Project\tftp\tftpy-master\test>python tftpy_client.py -H 10
.12.27.106 -D test.txt
[2018-11-07 13:29:00,281] Sending tftp download request to 10.12.27.106
[2018-11-07 13:29:00,281]     filename -> test.txt
[2018-11-07 13:29:00,282]     options -> {}
[2018-11-07 13:29:00,287] Transferred 23 bytes
[2018-11-07 13:29:00,287] Set remote port for session to 48328
[2018-11-07 13:29:00,287] Received DAT from server
[2018-11-07 13:29:00,287] Handling DAT packet - block 1
[2018-11-07 13:29:00,287] Sending ack to block 1
[2018-11-07 13:29:00,287] End of file detected
[2018-11-07 13:29:00,289]
[2018-11-07 13:29:00,289] Download complete.
[2018-11-07 13:29:00,289] Downloaded 23.00 bytes in 0.01 seconds
[2018-11-07 13:29:00,289] Average rate: 30.06 kbps
[2018-11-07 13:29:00,289] 0.00 bytes in resent data
[2018-11-07 13:29:00,289] Received 0 duplicate packets

C:\50.012 - Networks\Project\tftp\tftpy-master\test>python tftpy_client.py -H 10
.12.27.106 -D -
```

- Large file size of 109kB

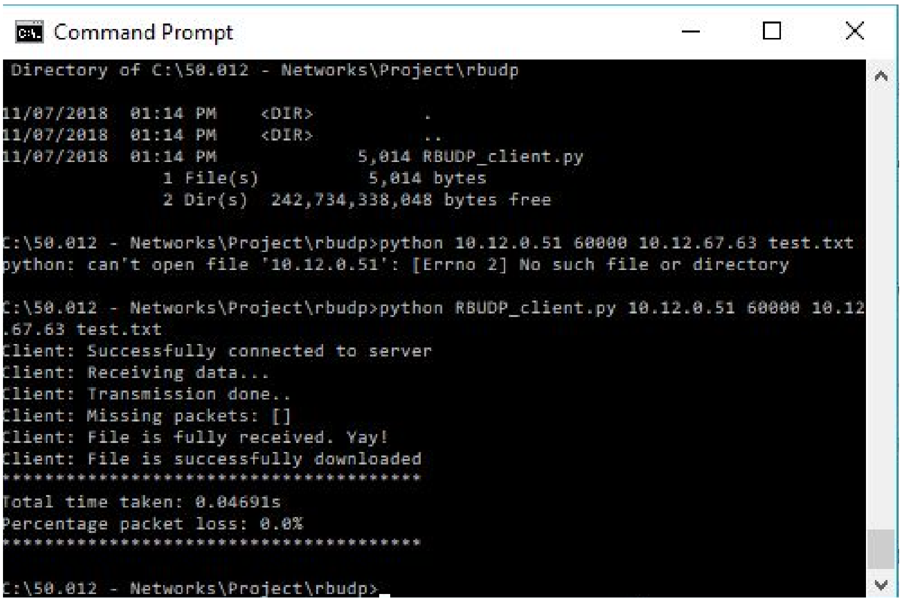


```
Command Prompt
[2018-11-07 13:30:11,505] Sending ack to block 212
[2018-11-07 13:30:11,509] Transferred 109056 bytes
[2018-11-07 13:30:11,509] Handling DAT packet - block 213
[2018-11-07 13:30:11,509] Sending ack to block 213
[2018-11-07 13:30:11,513] Transferred 109568 bytes
[2018-11-07 13:30:11,513] Handling DAT packet - block 214
[2018-11-07 13:30:11,513] Sending ack to block 214
[2018-11-07 13:30:11,517] Transferred 110080 bytes
[2018-11-07 13:30:11,517] Handling DAT packet - block 215
[2018-11-07 13:30:11,517] Sending ack to block 215
[2018-11-07 13:30:11,521] Transferred 110592 bytes
[2018-11-07 13:30:11,522] Handling DAT packet - block 216
[2018-11-07 13:30:11,522] Sending ack to block 216
[2018-11-07 13:30:11,526] Transferred 110645 bytes
[2018-11-07 13:30:11,526] Handling DAT packet - block 217
[2018-11-07 13:30:11,526] Sending ack to block 217
[2018-11-07 13:30:11,526] End of file detected
[2018-11-07 13:30:11,527]
[2018-11-07 13:30:11,527] Download complete.
[2018-11-07 13:30:11,527] Downloaded 110645.00 bytes in 1.03 seconds
[2018-11-07 13:30:11,527] Average rate: 840.62 kbps
[2018-11-07 13:30:11,527] 0.00 bytes in resent data
[2018-11-07 13:30:11,528] Received 0 duplicate packets

C:\50.012 - Networks\Project\tftp\tftpy-master\test>
```

➔ RBUDP

- Small file size of 1kB



- Large file size of 109kB