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

50.012 Networks

Cohort 1 Team 1

Team Members :

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

Table of Content

Introduction **2**
General Idea 2
Network Topics implemented 3
Transport Protocol **4**
Introduction 4
Literature Review on Reliable UDPs 4
Reliable User Datagram Protocol (RUDP) 4
Trivial File Transfer Protocol (TFTP) 5
Reliable Blast UDP (RBUDP) 5
Efficiency on the Reliable UDP 6
Experimental Procedure 6
Experimental Result 7
Multi-threaded downloader **8**
Introduction 8
How to run the program 10
Conclusion **11**
Reference **12**
Appendix **13**

1. Introduction

1.1. General Idea

While Transmission Control Protocol (TCP) is a connection-oriented transport protocol that provides a host of benefits through various features, such as congestion control and reliability delivery, it comes at a costly trade-off of significant latency due to connection setup and congestion control mechanism when loading web pages.[1] 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. The project github repository can be found here.

1.2. Network Topics implemented

This is a list of network topics that will be covered and implemented throughout this project.

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.[2] However, for this project, TCP is inefficient because of the increase in network traffic and delay. Thus, UDP will provide a simple transmission model without implicit handshaking techniques.

However, 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 explore various types of UDP available that guarantees reliability.

2.2. Literature Review on Reliable UDPs

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. In this protocol, senders and receivers will use acknowledgements and timeouts to ensure reliable data transfer. [3] A simple view of the protocol can be seen in the figure below.

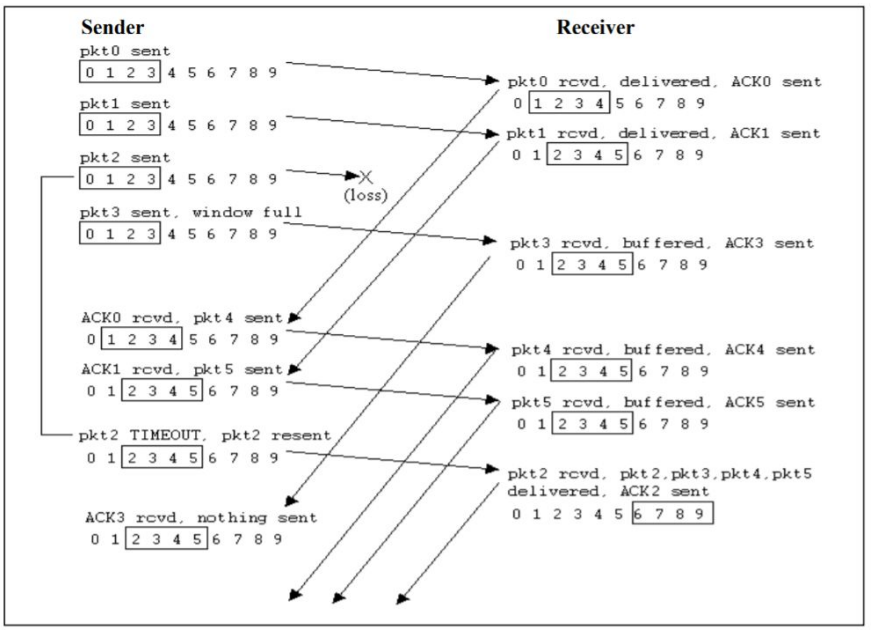


Figure 2.2.1. RUDP

2.2.2. Trivial File Transfer Protocol (TFTP)

TFTP is a 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. [4] When packets are lost, the intended recipient will timeout and resend that lost packet. All packets are sent with UDP.

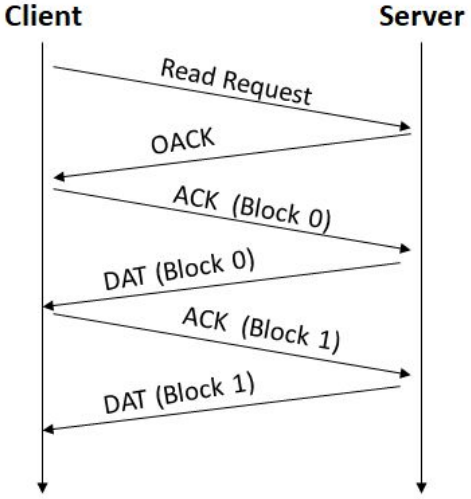


Figure 2.2.2 TFTP

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. [5] This is done by sending the entire payload at a user-specified sending rate using UDP datagrams and replying with aggregated acknowledgments at the end of a transmission phase which can be seen in the figure below. To minimize loss, the user-specified sending rate should not be larger than the bandwidth of the bottleneck link.

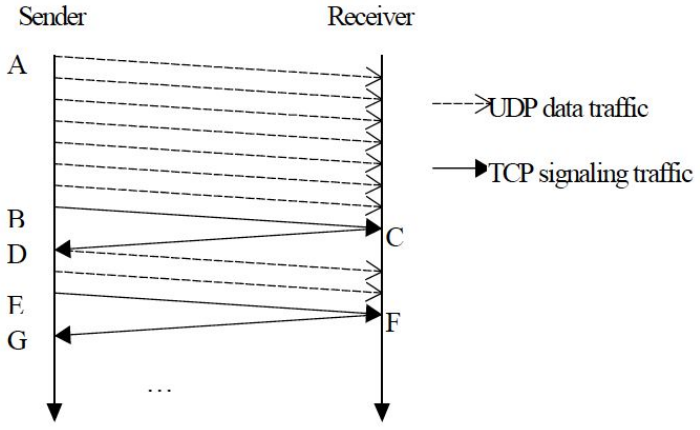


Figure 2.2.3 RBUDP

2.3. Efficiency on the Reliable 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.

We repeated this experiment 5 times for each protocol and recorded down the throughput and packet loss.

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
TCP	Small file: 0.01	No
	Large file: 0.07	No

It can be seen from the table above that RBUDP is slower compared to TCP due to factors such as the experiment is done through wireless environment and RBUDP thrive in wired environment and slower computer speed. Even though RBUDP has 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 for implementation on the multi-threaded downloader, where large files are downloaded, is RBUDP, since it is the most efficient among all the reliable UDP.

3. Multi-threaded downloader

3.1. Introduction

This is a multi-threaded downloader that manages parallel data download and merges file parts 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 overall process:

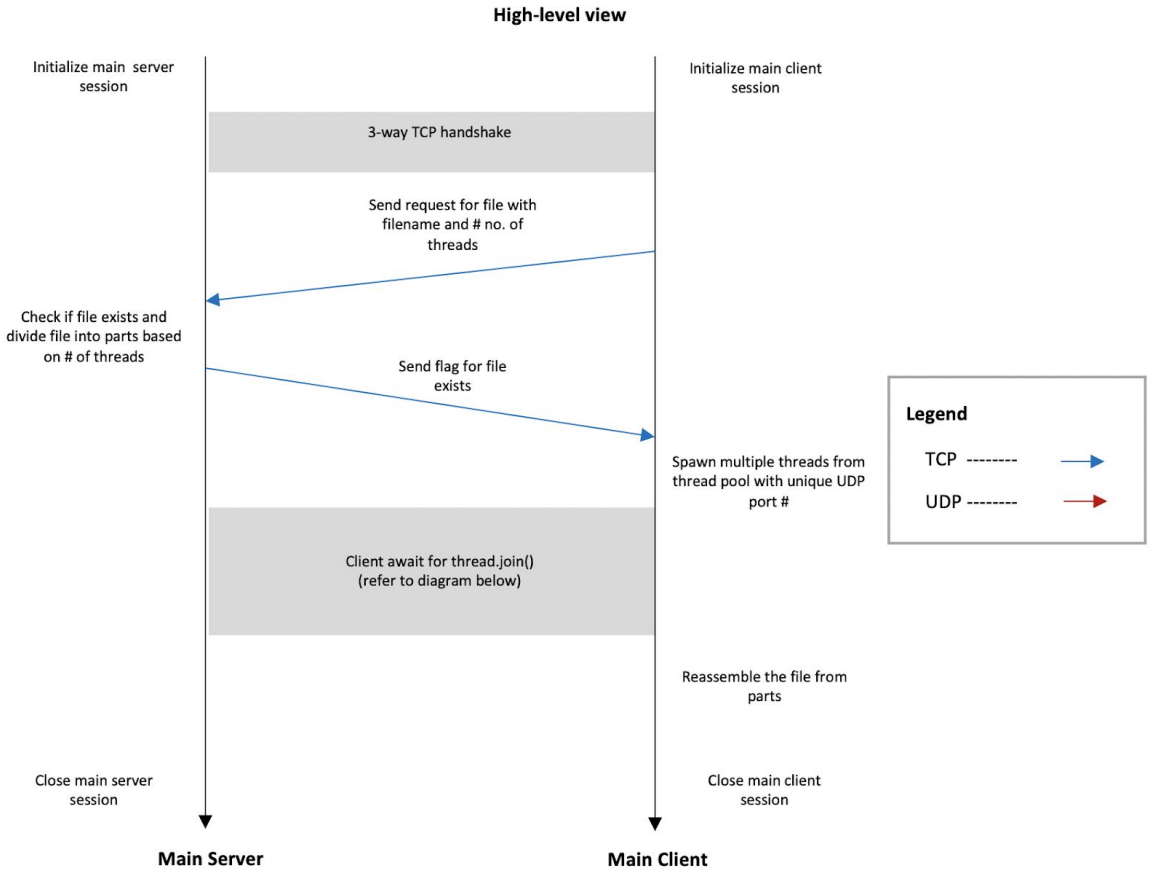


Figure 3.1.1 Client and Server Connection (Part 1)

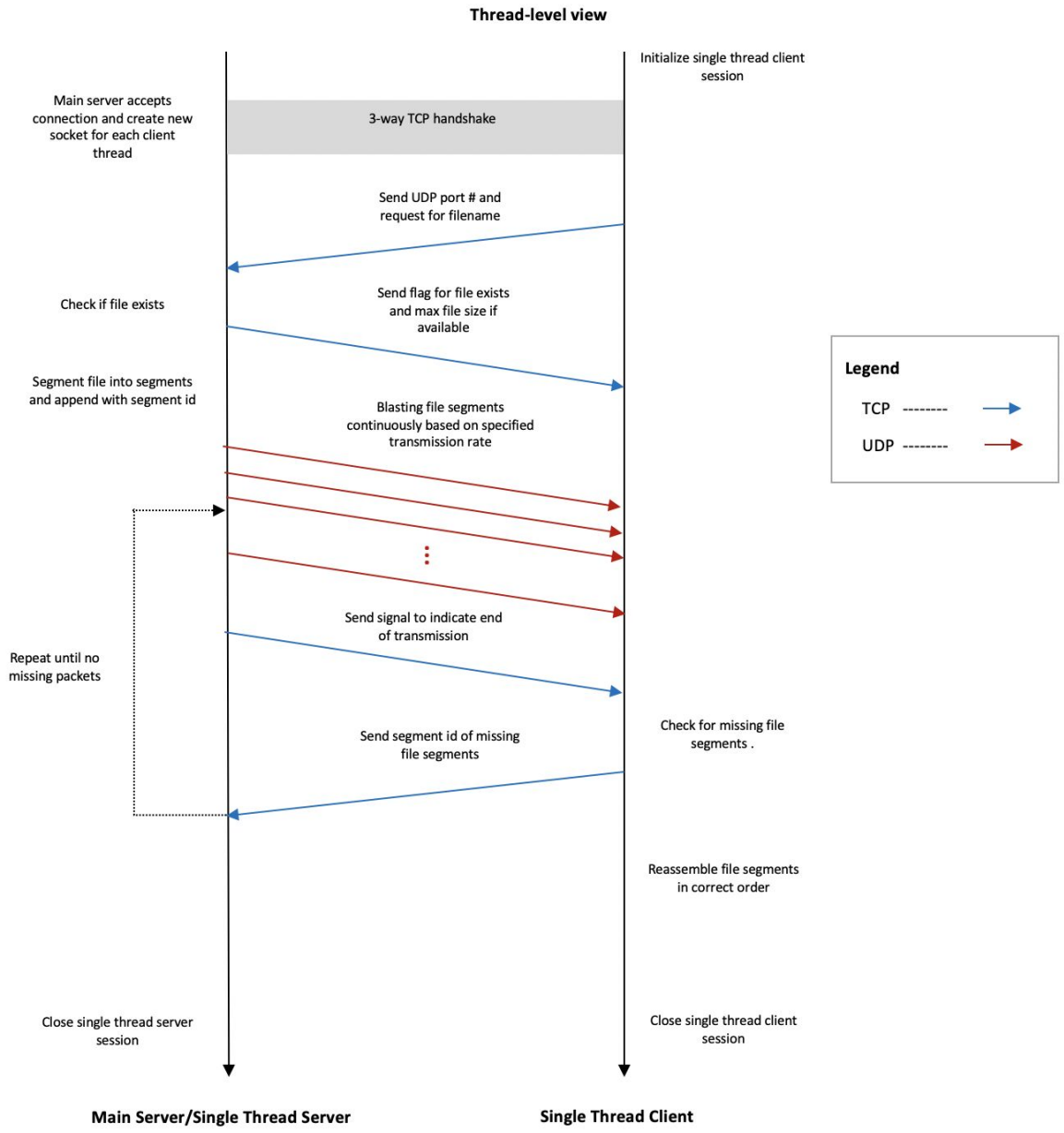


Figure 3.1.2 Client and Server Connection (Part 2)

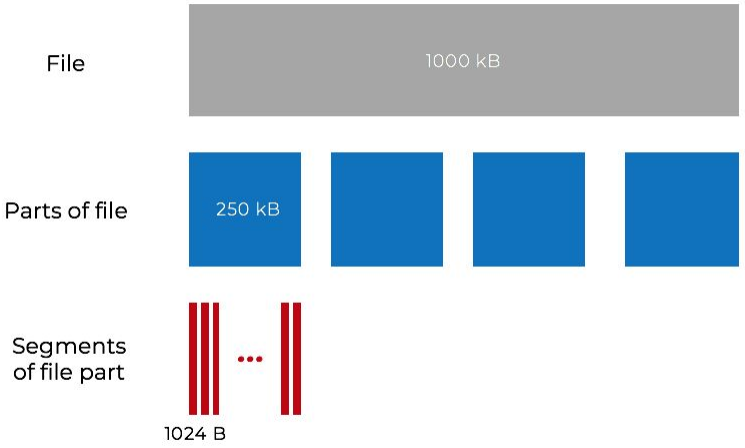


Figure 3.1.3 Details of file segmentation

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 parts accordingly based on the number of threads, as illustrated in Fig. 3.1.3 (E.g. if there are 4 threads, the fie will be divided into 4 parts). It then sends over the flag that indicates whether the file exists on the server.
3. The client creates threads from a thread pool. Each thread creates a TCP socket that establishes a connection with the TCP socket on the server side.
4. Upon receiving an incoming TCP connection, the server will create a new TCP socket and spawn a new thread to handle each client thread.
5. The client's thread then creates a UDP socket and sends over the UDP port number that this thread will use for receiving the file packets through its TCP connection. It also sends over the agreed filename of the particular part to download through the TCP connection.
6. The server's thread will send over a flag to indicate whether the file segment exists on the server for error detection and the max file size of the file segment.
7. Each server's thread will segment the file part, as illustrated in Fig. 3.1.3 and append the segment id. It then sends segments to the client's thread through the UDP connection at the user-specified rate. Once the transmission is done, the server's thread signals the client's thread through the TCP connection using a 'DONE' message.
8. The client's thread receives segments of the file part, and upon receiving the 'DONE' signal, it checks for missing file segments. If there are missing segments, it will send the segment id of the missing segments and step 7 will be repeated until there is no missing segments. It then assembles the file segments and saves them into a single file part and the client's thread will close.
9. Finally, the client combines parts 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 based on experimental result, deduced that RBUDP is the most optimal reliable UDP to be used in this project.

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

5. Reference

[1] Amit S. (2017). *Performance Analysis of QUIC Protocol under Network Congestion. (Master's thesis)*

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

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

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

[5] 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 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`

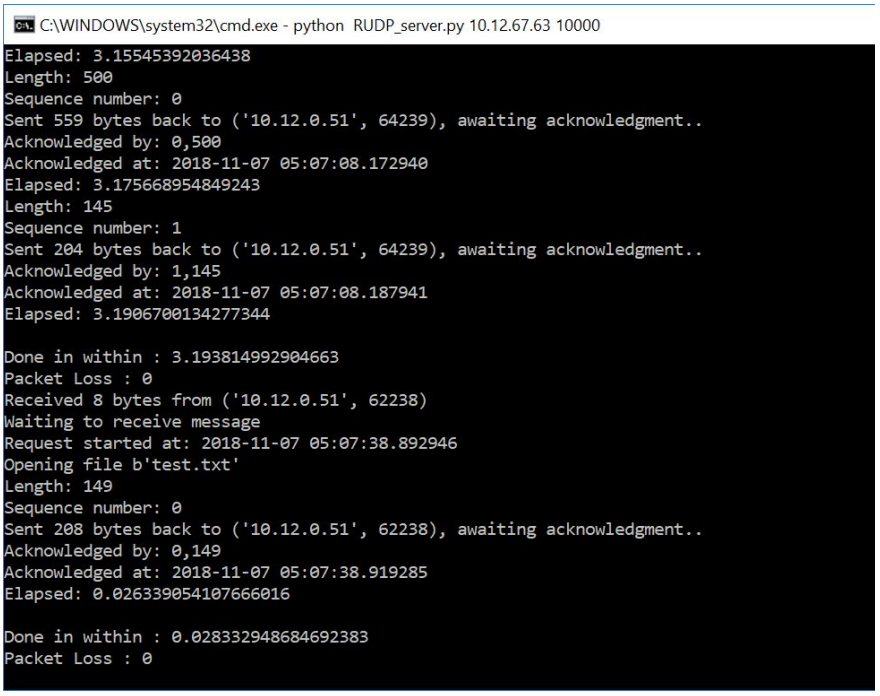
TFTP	`python3 TFTP_client.py -H [ip address of server] -D [file name to be requested]`
	
	E.g
	`python3 TFTP_client.py -H 127.0.0.1 -D test.txt`
		`python3 TFTP_server.py -r ./`
		
		*-r is the root location where the file to be download could be found
		
		E.g
		`python3 TFTP_server.py -r ./`

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`

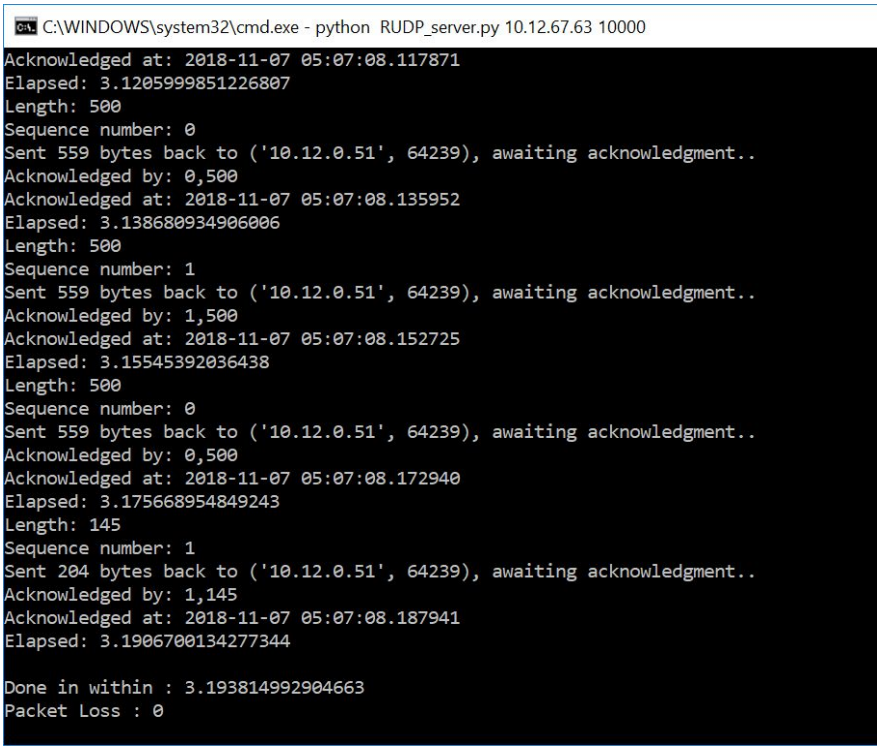
Result for each type of reliable UDP

➔ RUDP

- Small file size of 1kB

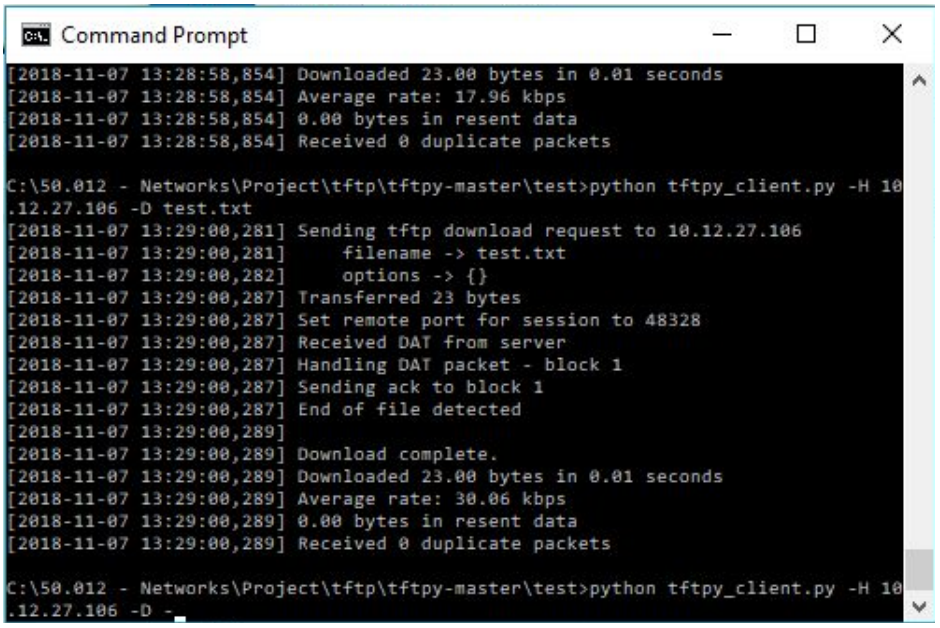


- Large file size of 109kB

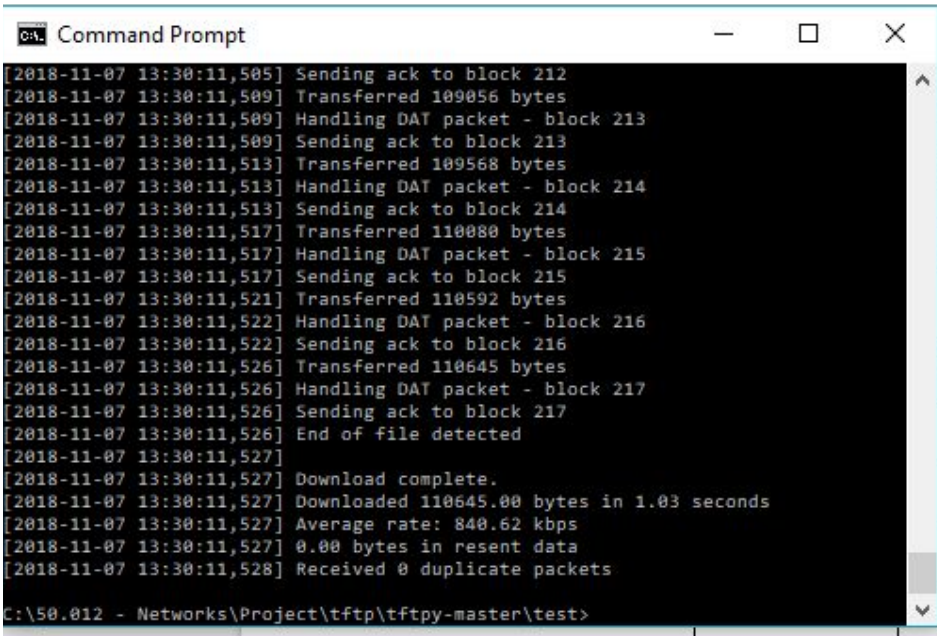


➔ TFTP

- Small file size of 1kB

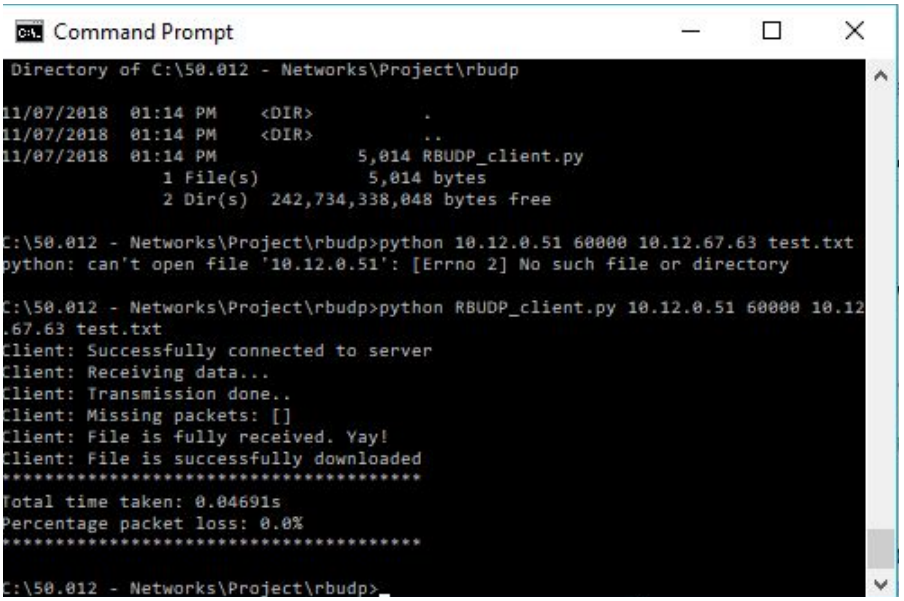


- Large file size of 109kB

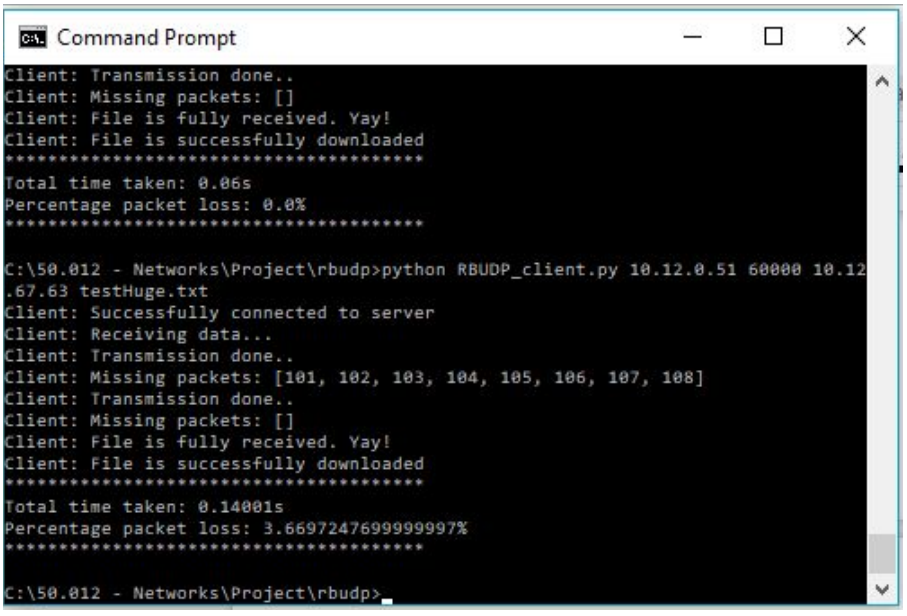


➔ RBUDP

- Small file size of 1kB



- Large file size of 109kB



➔ TCP

- Small file size of 1kB

```
File name: test_copy.txt
test_copy.txt Finish!
time elapsed = 0.013663053512573242
```

- Large file size of 1kB

```
File name: testHuge_copy.txt
testHuge_copy.txt Finish!
time elapsed = 0.05602407455444336
```