Computer Networking

Assignment 4

RDT App Programming

Class: F1403028

Name: Yi Jiang

Student ID: 5140219152

Table of Contents

1 Abstract

This project implements a simple file transfer application using UDP Socket APIs and also implements RDT 3.0 at the application layer. Both are implemented in a client uploading to server way. Together with file transfer using TCP Socket APIs, the time overhead is compared with each other.

2 Unreliable File Transfer

In this part, unreliable file transfer using UDP Socket APIs is implemented. Large files can be successfully transferred from client side to server side though transfer quality cannot be guaranteed. It can be applied to both text files and binary files. Total transmission delay is recorded.

The server listens to a certain port, while the client split a file into segments and send them one after another. The server side would receive each segment and construct the original file. After the file is sent, the client will stop and the server will continue running until "^C" is entered.

2.1 Main Functions

1）Server listens to a given port: 2680;

2）Client picks a file, split it into segments, and send them to localhost with port number 2680;

3）Server collects all segments received and reconstruct it into a file. Server continues to listen to the port, waiting for another file unless "^C" is pressed and quit.

2.2 How to Run

You need to open two terminals to run the program.

First, get into the folder of "udp_server.py" and then type "*python udp_server.py server_address server_port*". For example, to listen to port 2680 of your own PC, you can type "*python udp_server.py localhost 2680*". The terminal will display "Listening on port: 2680" if it runs correctly.

Then get into the folder of "udp_client.py" using the other terminal and type "*python udp_client.py server_address server_port filename_to_be_sent*". For example, to send "test.pdf" to port 2680 of your own PC, you can type "*python udp_client.py localhost 2680 test.pdf*". The time stamp of the moment transfer of the file is started will be displayed on the terminal if it runs correctly.

If the file transmission is successful, the file can be found in server's folder and the server side will continue to run unless *"^C*" is pressed. The terminal will display "*file filename received*". And the transmission delay, say the time it takes to transfer the file from the first to the last bit, will be displayed on the terminal after delta in seconds. The time stamp of the end of file transfer will also be displayed before them.

For any questions, you are welcome to contact elainejy@outlook.com.

2.3 Troubleshooting

2.3.1 Transmission Delay

To correctly record the transmission delay, two time stamps are used. As the transmission delay is the time interval between the first bit sent and the last bit received, it's hard to set up a timer on both the client and server side. Thus I record the time stamp on the client side when the first bit is being sent out, say the filename, as start and the time stamp on the server side when the end of file is received as end. The transmission delay is calculated as the difference between start and end.

3 Reliable File Transfer using RDT 3.0

In this part, reliable file transfer using RDT 3.0 protocol at the application layer is implemented. Large files can be successfully transferred from client side to server side reliably. It can be applied to both text files and binary files. Total transmission delay is recorded.

The server listens to a certain port, while the client split a file into segments and send them one after another. A new segment will be sent until the previous ACK is received correctly. The server side would receive each segment and construct the original file. After each file is sent, the client will stop and ask the user whether to continue or not. The server will continue running until "^C" is entered.

3.1 Main Functions

1）Server listens to a given port: 2680;

2）Client picks a file, split it into segments, and send them to localhost with port number 2680;

3）Server collects all segments received and reconstruct it into a file. Server continues to listen to the port, waiting for another file unless "^C" is pressed and quit.

4）Client asks user whether to continue send or quit.

3.2 How to Run

You need to open two terminals to run the program.

First, get into the folder of "rdt_server.py" and then type "*python rdt_server.py server_address server_port*". For example, to listen to port 2680 of your own PC, you can type "*python udp_server.py localhost 2680*". The terminal will display "Listening on port: 2680" if it runs correctly.

Then get into the folder of “rdt_client.py” using the other terminal and type “*python rdt_client.py*”. Then the terminal will display “*Please input the address, port num and filename you want to transfer in sequence separated by space:*”. Thus you can type the requested info, say “*localhost 2680 test.txt*”, which can send “test.txt” to port 2680 of your own PC. The time stamp of the moment transfer of the file is started will be displayed on the terminal and so are the “ACK0”s and “ACK1”s, if it runs correctly. After the file is sent, it will ask you whether to continue to send or not.

If the file transmission is successful, the file can be found in server’s folder and the server side will continue to run unless *“^C*” is pressed. The terminal will display “*file filename received*”. And the transmission delay, say the time it takes to transfer the file from the first to the last bit, will be displayed on the terminal after delta in seconds. The time stamp of the end of file transfer will also be displayed before them.

For any questions, you are welcome to contact elainejy@outlook.com.

3.3 Troubleshooting

3.3.1 ACK Message

To implement reliable file transfer application, the server need to send ACK message together with the sequence number back to the client side to let the client continue to send next segment. If the client side doesn’t receive the ACK message, it will wait a certain period of time and send that segment again until Timeout. Checksum is also implemented to check bit errors, though I don’t count exact number of that.

4 File Transfer using TCP

Details of this part are omitted as it is basically the previous assignment.

5 Transmission Delay Comparison

Two files, “test.txt” and “test.pdf”, are used for testing and comparing the transmission delay of different methods. The size of “test.txt” is 224KB and that of “test.pdf” is 158KB. Their transmission delay (in seconds) are as follows:

	test.txt	test.pdf
Unreliable file transfer	0.573008060455	0.405445098877
Reliable file transfer (RDT 3.0)	1.54669094086	1.09102416039
TCP	0.0166139603	0.0119290352

Basically, reliable file transfer takes longer than unreliable file transfer. Also TCP is quicker when the file is not very big and there is no congestion within the network. The larger the file is, the longer it takes to transfer.

6 Something Worth Noticing

Environment: macOS 10.12.2 Sierra

Language: Python 2.7.11

7 Conclusion

In this assignment, both reliable and unreliable file transfer using UDP Socket APIs are implemented. Their transmission delays are compared and I’ve got a far better understanding of the protocol. Many thanks to the professor and TA.