

CPE 400 Project: UDP-based reliable data transfer algorithm

Students: Brendan Aguiar, Nicholas Ang

Instructors: Dr. Engin Arslan, Md Tamjid Hossain

Date: 05/04/2022

Technical Report

Functionality

The program was implemented in Python 3. It is structured with a client-server architecture where the server receives the file over TCP and UDP channels. The TCP and UDP channels use the socket library in Python. The TCP channel is used for sending metadata such as file names and file sizes. The UDP channel is responsible for transporting the data of the file.

The UDP channel is modified to be able to reliably transfer data between the client and the server. On inputting a text file to send over on the client side, the program first establishes a TCP connection to send the metadata of the file. Then the TCP connection closes and the UDP connection opens to start transferring data.

The client of the UDP channel is responsible for creating packets with the file data and sending it to the server. The packets are formed using Python pickle which converts the packets into byte arrays. The packets consist of the sequence number, sequence number checksum, file contents checksum, and the actual contents of the file. The contents of the file is segmented into 200 Byte segments that are used in the packets. Both the sequence number checksum and the file contents checksum is hashed using SHA256.

The server of the UDP channel is responsible for accepting the file data and sending acknowledgements to the client. The verification of the data happens on the server side of the channel. When the server receives data, it separates the different parts of the packets into its appropriate variables. The sequence number and content parts of the packet are hashed with SHA256 and compared with their corresponding checksums that were received from the client. If they match, the data is accepted and outputted to the terminal. If they do not match, the server sends a message requesting a retransmit of the same packet and packet number. The server also simulates packet loss using random values. In the event of packet loss, the client automatically tries to resend the missing packet before continuing with the next packet. In the event of a timeout, the client will automatically resend the packet and throw away the information of the delayed packet.

At the end of the file transfer, the client sends a last message to close the UDP connection. It sends the sequence number which is compared to the file size given in the TCP connection as well as the checksums and closing delimiter. If the sequence

number matches with the file size, the download is complete. If it does not match, there was an error in downloading and the program is stopped.

Novel Contribution

Typically, UDP is an unreliable communications protocol and loses packets of data frequently. With our implementation, UDP reliably sends over the file data without losing any packets. Our implementation of reliable data transfer over UDP uses the novel idea of the stop-and-wait protocol for sending packets with checksums, sequence numbers, acknowledgements, and timeouts.

By itself, the stop-and-wait protocol is a simple flow control technique that sends packets one at a time and does not continuously send packets unless it receives acknowledgements of the previous packet. While the stop-and-wait protocol is capable of sending data over the channel, it does not account for corruption of packets. The stop-and-wait protocol only resends data in the case of timeout or lost acknowledgement. To increase the reliability of data transfer over UDP, the stop-and-wait protocol was augmented with checksums and makes sure that all packets are sent and received.

Two checksums are used to ensure that the data received by the server is correct. The sequence number checksum ensures that the packet being sent is consistent with how much data received. The contents checksum checks the actual data and compares it to what the server received. Whenever it fails the checksum or times out, the program will resend the data until it is entirely correct and in the proper sequence. This method of combining the stop-and-wait protocol with acknowledgements, sequence numbers, timeouts, and checksums maintains the reliability of the data transferred.

The timing diagram in Figure 1 visualizes the network and how the algorithm works. The TCP channel handles the first two packets for file name and file size. The UDP channel handles the rest of the packets, containing the file data, with handling with packet loss and time outs.

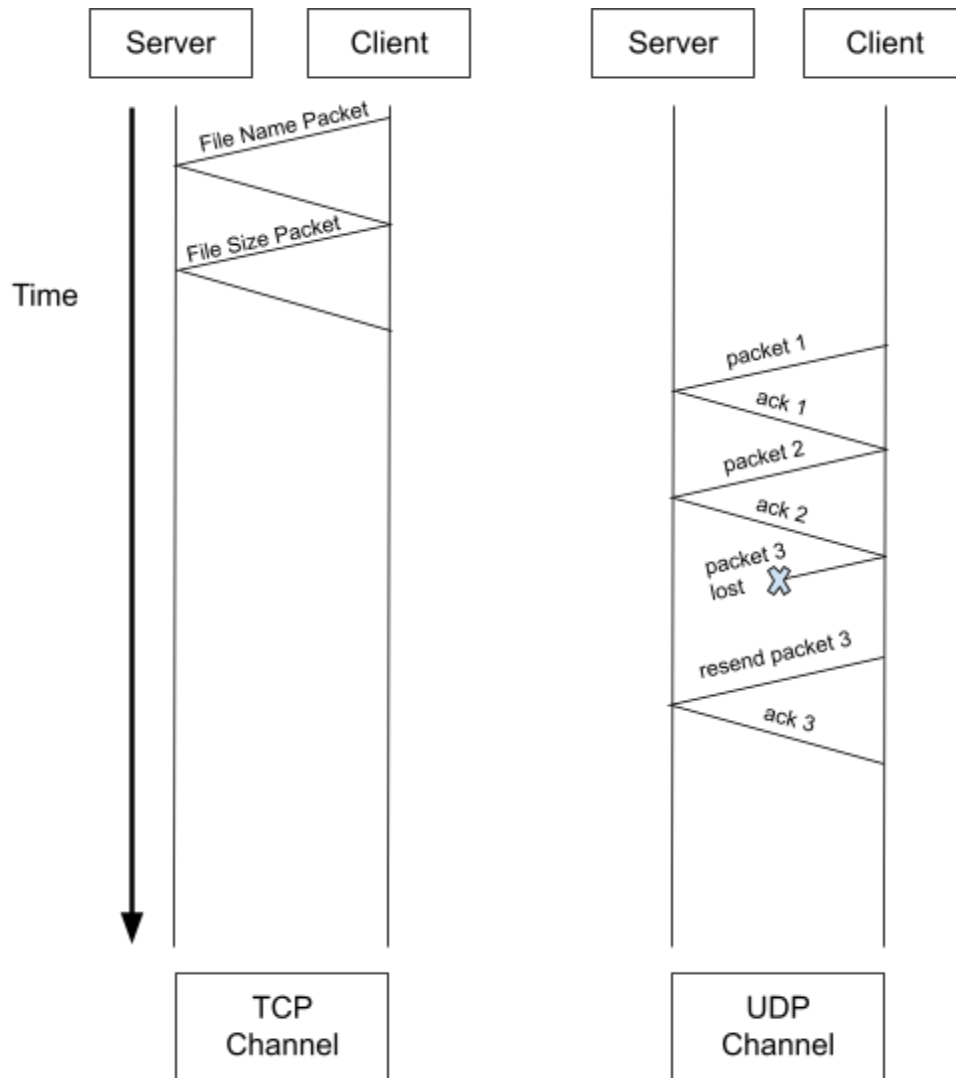
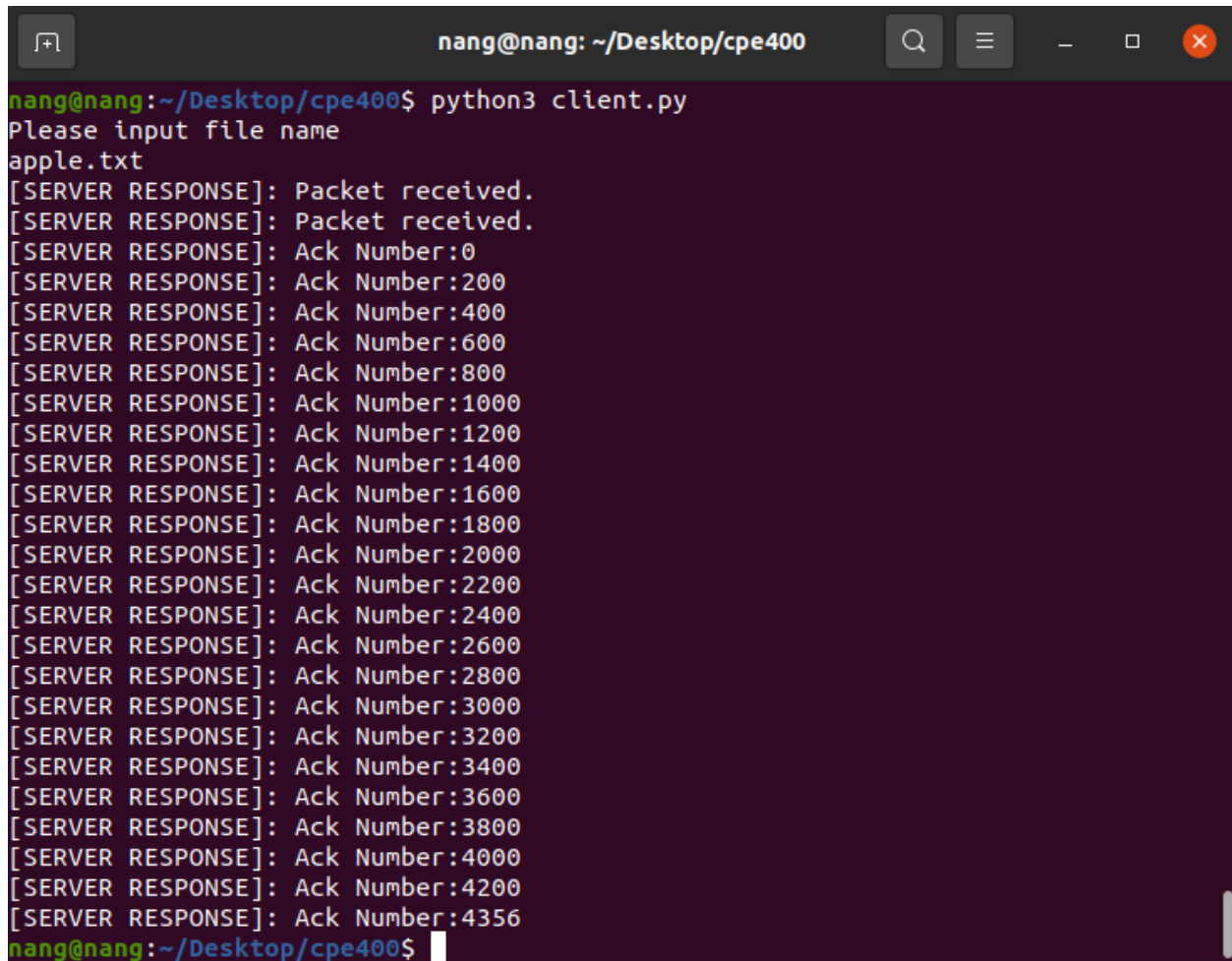


Figure 1 shows the timing diagram of the data transfer using our algorithm

Results and Analysis

The UDP client is able to successfully send the packets to the server. For every packet sent, the UDP server sends back an acknowledgement. As shown in Figure 2, the client keeps track of the acknowledgements as it prints the most recent one to the screen.

A terminal window titled 'nang@nang: ~/Desktop/cpe400' with standard window controls. The terminal shows the execution of 'python3 client.py'. The user enters 'apple.txt'. The program then prints a series of 21 lines of server responses, each starting with '[SERVER RESPONSE]:'. The first two lines are 'Packet received.'. The subsequent lines show 'Ack Number:' followed by values from 0 to 4356 in increments of 200. The terminal ends with the prompt 'nang@nang:~/Desktop/cpe400\$' and a cursor.

```
nang@nang:~/Desktop/cpe400$ python3 client.py
Please input file name
apple.txt
[SERVER RESPONSE]: Packet received.
[SERVER RESPONSE]: Packet received.
[SERVER RESPONSE]: Ack Number:0
[SERVER RESPONSE]: Ack Number:200
[SERVER RESPONSE]: Ack Number:400
[SERVER RESPONSE]: Ack Number:600
[SERVER RESPONSE]: Ack Number:800
[SERVER RESPONSE]: Ack Number:1000
[SERVER RESPONSE]: Ack Number:1200
[SERVER RESPONSE]: Ack Number:1400
[SERVER RESPONSE]: Ack Number:1600
[SERVER RESPONSE]: Ack Number:1800
[SERVER RESPONSE]: Ack Number:2000
[SERVER RESPONSE]: Ack Number:2200
[SERVER RESPONSE]: Ack Number:2400
[SERVER RESPONSE]: Ack Number:2600
[SERVER RESPONSE]: Ack Number:2800
[SERVER RESPONSE]: Ack Number:3000
[SERVER RESPONSE]: Ack Number:3200
[SERVER RESPONSE]: Ack Number:3400
[SERVER RESPONSE]: Ack Number:3600
[SERVER RESPONSE]: Ack Number:3800
[SERVER RESPONSE]: Ack Number:4000
[SERVER RESPONSE]: Ack Number:4200
[SERVER RESPONSE]: Ack Number:4356
nang@nang:~/Desktop/cpe400$
```

Figure 2: The acknowledgement received shows that 200 packets were sent at a time.

The server first receives the name of the file being sent and the size of the file from the TCP socket. The name could be used later when printing to a file on the server end. The file size is used by the UDP socket to check that all of the file has been sent before closing the connection. As shown in Figure 3, the address information of the client is given followed by the packet information and the packet itself in the TCP connection. After the UDP connection is started, it outputs the file contents received from the client. ASCII art of an apple labeled “apple.txt” was used to more easily determine if packet loss or out of order packet arrival occurred.


```
nang@nang: ~/Desktop/cpe400
plane flying in an insect-like pattern? Get your nose in there. Don't be afraid
. Smell it. Full reverse! Just drop it. Be a part of it. Ain for the center! Now
drop it in! Drop it in, woman! Come on, already. Barry, we did it! You taught m
e how to fly! - Yes. No high-five! - Right. Barry, it worked! Did you see the gi
ant flower? What giant flower? Where? Of course I saw the flower! That was geni
si - Thank you. - But we're not done yet. Listen, everyone! This runway is cover
ed with the last pollen from the last flowers available anywhere on Earth. That
means this is our last chance. We're the only ones who make honey, pollinate flo
wers and dress like this. If we're gonna survive as a species, this is our momen
t! What do you say? Are we going to be bees, or just Museum of Natural History k
eychains? We're bees! Keychain! Then follow me! Except Keychain. Hold on, Barry.
Here. You've earned this. Yeah! I'm a Pollen Jock! And it's a perfect fit. All
I gotta do are the sleeves. Oh, yeah. That's our Barry. Mom! The bees are back!
If anybody needs to make a call, now's the time. I got a feeling we'll be workin
g late tonight! Here's your change. Have a great afternoon! Can I help who's nex
t? Would you like some honey with that? It is bee-approved. Don't forget these.
Milk, cream, cheese, it's all me. And I don't see a nickel! Sometimes I just fee
l like a piece of meat! I had no idea. Barry, I'm sorry. Have you got a moment?
Would you excuse me? My mosquito associate will help you. Sorry I'm late. He's a
lawyer too? I was already a blood-sucking parasite. All I needed was a briefcas
e. Have a great afternoon! Barry, I just got this hu
- ge tulip order, and I can't get them anywhere. No problem, Vannie. Just leave
it to me. You're a lifesaver, Barry. Can I help who's next? All right, scramble.
- jocks! It's time to fly. Thank you, Barry! That bee is living my life! Let it g
o, Kenny. - When will this nightmare end?! - Let it all go. - Beautiful day to f
ly. - Sure is. Between you and me, I was dying to get out of that office. You ha
ve got to start thinking bee, my friend. - Thinking bee! - Me? Hold it. Let's ju
st stop for a second. Hold it. I'm sorry. I'm sorry, everyone. Can we stop here?
I'm not making a major life decision during a production number! All right. Tak
e ten, everybody. Wrap it up, guys. I had virtually no rehearsal for that.

Closing server
nang@nang:~/Desktop/cpe400$
nang@nang:~/Desktop/cpe400$ python3 client.py
Please Input file name
beemovie.txt
[SERVER RESPONSE]: Packet received.
[SERVER RESPONSE]: Packet received.
[SERVER RESPONSE]: Ack Number:0
[SERVER RESPONSE]: Ack Number:200
[SERVER RESPONSE]: Ack Number:400
[SERVER RESPONSE]: Ack Number:600
[SERVER RESPONSE]: Ack Number:800
[SERVER RESPONSE]: Ack Number:1000
[SERVER RESPONSE]: Ack Number:1200
[SERVER RESPONSE]: Ack Number:1400
[SERVER RESPONSE]: Ack Number:1600
[SERVER RESPONSE]: Ack Number:1800
[SERVER RESPONSE]: Ack Number:2000
[SERVER RESPONSE]: Ack Number:2200
[SERVER RESPONSE]: Ack Number:2400
[SERVER RESPONSE]: Ack Number:2600
[SERVER RESPONSE]: Ack Number:2800
[SERVER RESPONSE]: Ack Number:3000
[SERVER RESPONSE]: Ack Number:3200
[SERVER RESPONSE]: Ack Number:3400
[SERVER RESPONSE]: Ack Number:3600
[SERVER RESPONSE]: Ack Number:3800
[SERVER RESPONSE]: Ack Number:4000
[SERVER RESPONSE]: Ack Number:4200
[SERVER RESPONSE]: Ack Number:4400
[SERVER RESPONSE]: Ack Number:4600
[SERVER RESPONSE]: Ack Number:4800
[SERVER RESPONSE]: Ack Number:5000
[SERVER RESPONSE]: Ack Number:5200
[SERVER RESPONSE]: Ack Number:5400
[SERVER RESPONSE]: Ack Number:5600
[SERVER RESPONSE]: Ack Number:5800
[SERVER RESPONSE]: Ack Number:6000
[SERVER RESPONSE]: Ack Number:6200
[SERVER RESPONSE]: Ack Number:6400
[SERVER RESPONSE]: Ack Number:6600
[SERVER RESPONSE]: Ack Number:6800
[SERVER RESPONSE]: Ack Number:7000
[SERVER RESPONSE]: Ack Number:7200
[SERVER RESPONSE]: Ack Number:7400
[SERVER RESPONSE]: Ack Number:7600
[SERVER RESPONSE]: Ack Number:7800
[SERVER RESPONSE]: Ack Number:8000
[SERVER RESPONSE]: Ack Number:8200
[SERVER RESPONSE]: Ack Number:8400
[SERVER RESPONSE]: Ack Number:8600
```

Figure 4: The file “beemovie.txt” contains sample text from “Bee Movie.”

After successfully sending both the “apple.txt” and “beemovie.txt” files, the algorithm was updated to force scenarios of packet loss. When a packet loss or packet timeout occurs, the server sends the last successfully received acknowledgement. The client then is able to resend the packet with the subsequent sequence number. As shown in Figure 5, the client prints a retransmitting statement to notify that its resending the previous packet before sending the next acknowledgement number.

