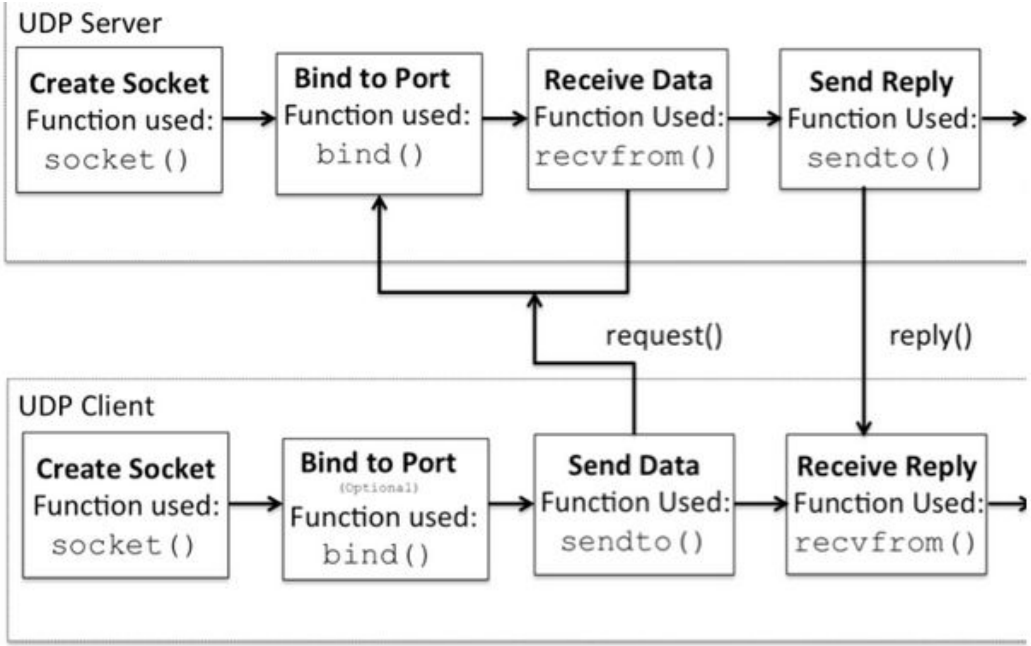
Reliable Transport Protocol with UDP

Class 2425II_INT2213_9
Student ID: 23020082
Full name: Nguyen Quoc Huy
Link Github

UDP structure and problem

- **bind()** associates a socket with a specific IP address and port on the local machine. It essentially "reserves" that port for listening to incoming data.
- Receiver (server in diagram): Listen for incoming data on this IP/port.
- UDP is not reliable -> Need a mechanism to handle error, integrity while transmit with UDP

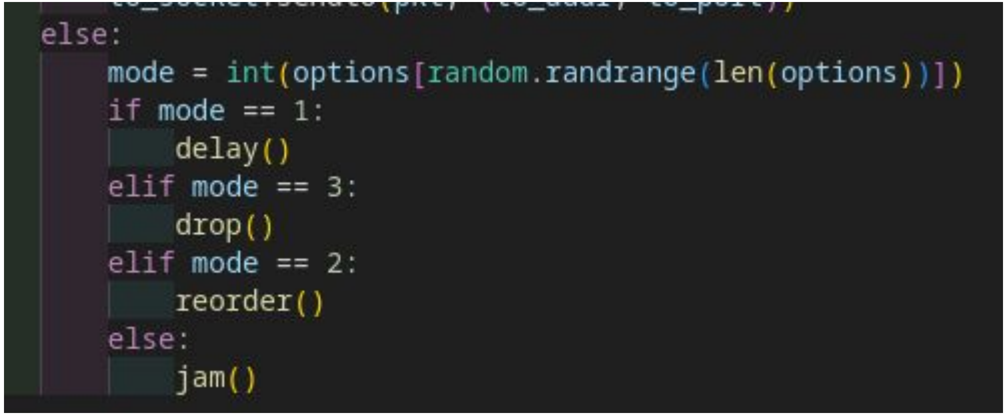


RTP Mechanism

- Reliable based on handshake (START and END), ACK (cumulative, individual later), checksum(ensure integrity), inorder delivery (in order structure)
- **Note**: I count Data seq_number from 1 -> n so end ACK have seq_num = n + 1

Proxy

- The **proxy** in RTP project acts as a **network emulator** between the sender and receiver to simulate real-world network issues



```
        to_socket.sendto(pkt, (to_addr, to_port))
    else:
        mode = int(options[random.randrange(len(options))])
        if mode == 1:
            delay()
        elif mode == 3:
            drop()
        elif mode == 2:
            reorder()
        else:
            jam()
```

Library - key note

- Socket: networking interface for Python [Link]
- OrderDick: Maintain the insertion order of keys [Link]
- Some other data structure in Python

Part 1 Sender implementation

Check list:

1. Send START packet (type 0, seq_num 0) and wait for ACK (type 3, seq_num 1).

2. Read input data, split into chunks of 1456 bytes each (since the max packet size is 1472, header is 16 bytes).

3. Use a sliding window mechanism. The window size is given as an argument.

4. Send packets within the window, track sent packets.

5. Handle ACKs from the receiver. For each ACK, update the window base.

6. Retransmit packets if ACKs aren't received within 500ms.

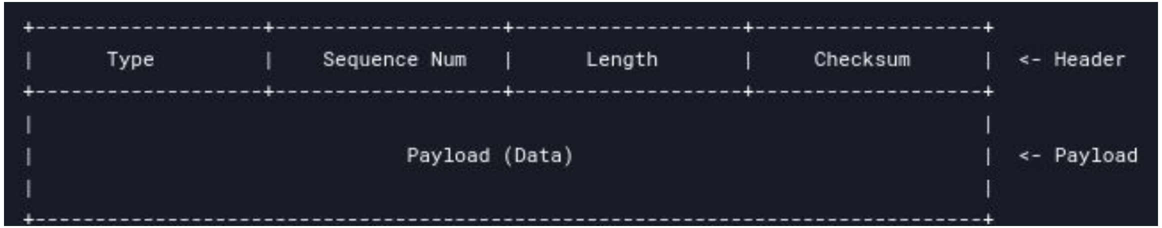
7. After all DATA packets are sent and ACKed, send END packet (type 1) and wait for its ACK.

Behavior

- Address have random port (arbitrary port for each time running)

Packet size = 1472

- Standard Ethernet networks have an MTU(Maximum Transmission Unit of **1500 bytes** [Link].
- IP header = 20 [Link]
- UDP head = 8 [Link]
- Packet size (UDP payload) = 1472 (RTP header (16: 4 int field) + Payload (max 1456))
- In my socket:



```
+-----------------+-----------------+-----------------+-----------------+
|      Type       |  Sequence Num   |     Length      |    Checksum     |  <- Header
+-----------------+-----------------+-----------------+-----------------+
|                                                                       |
|                            Payload (Data)                             |  <- Payload
|                                                                       |
+-----------------------------------------------------------------------+
```

Part 2 Receiver implementation

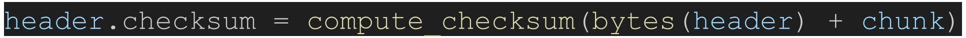
Check list

- 1. Handle START packet
- 2. Check connection is started (ignore START messages)
- 3. Check sum for each packet
- 4. Packet handling: Sliding window track expected sequence number and buffer within window
  - 4.1 Unlike expected: buffer
  - 4.2 Like expected: in-order deliver
  - 4.3 Drop all package outsize window
  - 4.4 Cumulative ACKs
- 5. Exit with END message

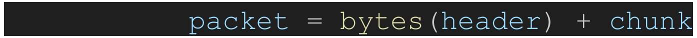
Drop ack at start handshake

- Receiver must send ack until sender catch
- Instead of using boolean flag, I use numeric represented for address of first sender connecting successfully

Invalid check sum at receive?

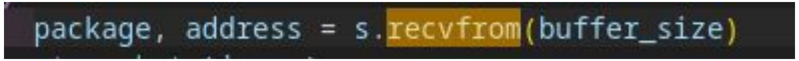


```
header.checksum = compute_checksum(bytes(header) + chunk)
```



```
           packet = bytes(header) + chunk
```

- This cause header.checksum always = 0
- Set buffer size = 1472 for recvfrom() function. Misunderstand buffer size vs packet size -> Set 1472 miss some information -> Wrong recalculate check sum



```
package, address = s.recvfrom(buffer_size)
```

Part 3 Optimize - ACK mechanism

Check list

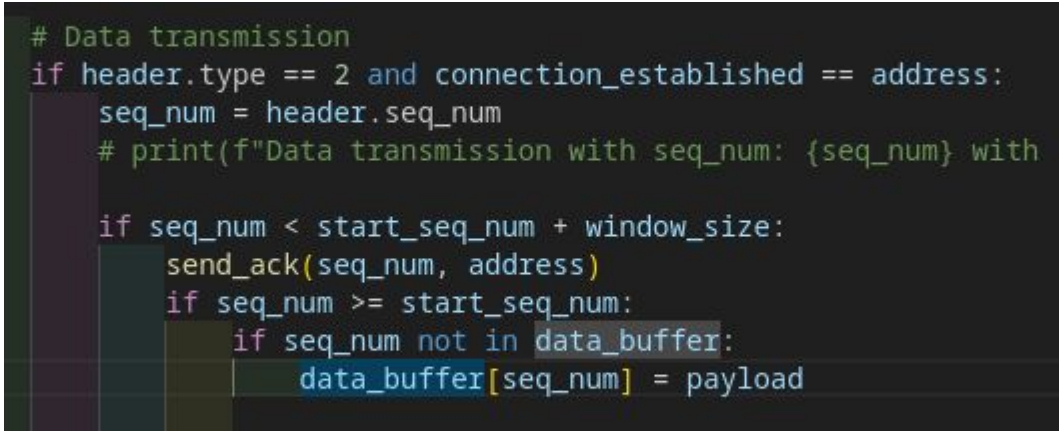
- Modify the receiver to send individual ACKs for each DATA/END packet instead of cumulative ACKs.
- Replace the sliding window logic to track individual ACKs

Key in my optimization

- Individual ACK Tracking
- On timeout, retransmit only unacknowledged packets in the window.
- Precomputed Packets (quite efficient)
-

ACK from receiver is dropped

- So i fix it with resend seq_num for each time receiver catch it
- If sender no catch ack from receiver (dropped), it will resend it. Because of that, receiver will resend an ack. Continue until sender catch satisfy ack.



```
# Data transmission
if header.type == 2 and connection_established == address:
    seq_num = header.seq_num
    # print(f"Data transmission with seq_num: {seq_num} with

    if seq_num < start_seq_num + window_size:
        send_ack(seq_num, address)
        if seq_num >= start_seq_num:
            if seq_num not in data_buffer:
                data_buffer[seq_num] = payload
```

Conclusion and Grading

1. **Building a reliable protocol** on UDP through sequence numbers, checksums, ACKs, and retransmissions.
2. **Implementing core mechanisms** (error detection, flow control, acknowledgment) to ensure data integrity and completeness.
3. **Contrasting sliding window strategies**:
   - Cumulative ACKs vs. individual ACKs(better or not? why better?).
   - How window size (what happen when small vice versa) and ACK strategies impact efficiency and reliability.

Complete:

60: RTP-base passes test

- 10: built on top of UDP (doesn't use TCP sockets)
- 15: correctly implement cumulative ACK
- 15: correctly implement timeout and retransmission
- 20: correct received message

40: RTP-opt passes test

- 15: doesn't send cumulative ACKs
- 15: correctly implement timeout and retransmission
- 10: correct received message

Some further work

I invest some way to improve this project:

- Adaptive Timeout Mechanisms [Link]
- Support Selective ACKs (SACK) [Link]
- Multiple Concurrent Connections (Just my idea, use thread?)
- Dynamic Window Sizing (Possible idea like Adaptive timeout)
- Testing and Metrics (like packet loss, etc…)

Challenging

- Need a better proxy. Through my work, I see this project’s proxy is quite simple and not allowed to do some further work.
- Event is random and not consistent -> Dynamic timeout and window size will have no effect
- My lack of knowledge and time to build up a better proxy

Thank you

- This project took me over six hours to complete due to my lack of experience and knowledge.
- This tour offers an exciting opportunity for newcomers like me to dive into the world of computer networking
- Thank you for taking the time to read my report. I hope you can provide feedback on my work.