Homework 1 - Socoteală Andrei-Alin

Structure

This program contains a unified application (`app.c`) that can act as both a server and a client, supporting three different network protocols: TCP, UDP, and QUIC. The structure of the project is as follows:

```
|── app.c            # Unified program for both server and client roles
|── app              # Compiled executable for app.c
|── quic_client.c    # QUIC client implementation
|── quic_client      # Compiled executable for QUIC client
|── quic_server.c    # QUIC server implementation
|── quic_server      # Compiled executable for QUIC server
|── tcp_client.c     # TCP client implementation
|── tcp_client       # Compiled executable for TCP client
|── tcp_server.c     # TCP server implementation
|── tcp_server       # Compiled executable for TCP server
|── udp_client.c     # UDP client implementation
|── udp_client       # Compiled executable for UDP client
|── udp_server.c     # UDP server implementation
|── udp_server       # Compiled executable for UDP server
```

How to Use the Application (`./app`)

For QUIC support, ngtcp2 must be installed before compiling. On Debian-based systems, this can be done with: `sudo apt install libngtcp2-dev` then compile it with: `gcc -o app app.c -lngtcp2`

The application can be used in two modes: server or client.

To start a server, use: `./app server <protocol>`
Where <protocol> can be:

- tcp
- udp
- quic

To start a client, use: `./app client <protocol> <mode> <size_mb> <msg_size>`
Where:

- <protocol> = tcp, udp, or quic
- <mode> = streaming or stop-and-wait
- <size_mb> = Total data size to send (500 or 1024)
- <msg_size> = Message size in bytes (1 - 655535)

Example: `./app client tcp streaming 500 1024`

If needed, the programs can also be executed separately without using ./app
Servers

- ./tcp_server
- ./udp_server
- ./quic_server — requires ngtcp2

Clients

- ./tcp_client streaming 500 1000
- ./udp_client stop-and-wait 500 1000
- ./quic_client streaming 500 1000 — requires ngtcp2

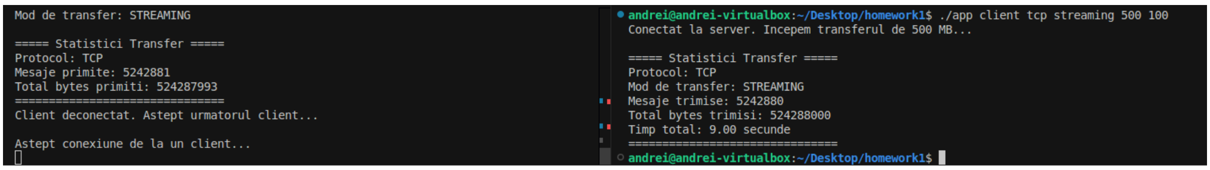
Statistics and Performance Comparisons for TCP

The performance of the TCP protocol has been evaluated in both streaming and stop-and-wait modes. Although TCP inherently operates with an acknowledgment mechanism (similar to stop-and-wait), both modes were implemented for a comprehensive comparison.

TCP Streaming Mode

Test 1: 500 MB Transfer with 100-Byte Messages

- In this test, 500 MB of data was sent using 100-byte messages.
- Observations showed that all messages were received correctly, along with an additional end message used for signaling completion.
- The total transmission time was 9 seconds.
- Increasing the message size to values above 10000 bytes led to an unexpected behavior where the server received more messages than were sent, this is probably due to the way TCP segmentation and buffering work



- When 1 GB of data was sent using 100-byte messages, the total transmission time increased, but it did not double compared to the 500 MB test, demonstrating TCP's efficiency in handling bulk transfers.

TCP Stop-and-Wait Mode

Test 2: 500 MB and 1 GB Transfers with Varying Message Sizes

- Regardless of the message size, all messages were received correctly
- The only varying factor was time, which increased when the message size was smaller.

- Smaller messages lead to increased transmission time, as each message requires an acknowledgment before proceeding.
- Larger messages reduce the number of acknowledgment cycles, improving overall performance.

Statistics and Performance Comparisons for UDP

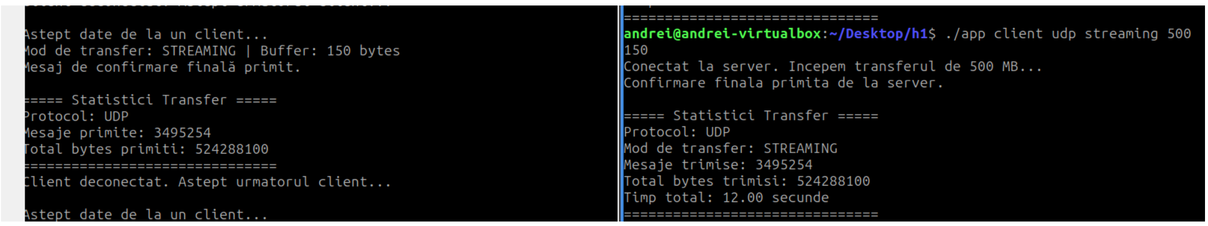
UDP has been evaluated in both streaming and stop-and-wait modes. Unlike TCP, UDP does not provide built-in mechanisms for acknowledgment, ordering, or retransmission, making it more prone to packet loss, especially in high-speed transmissions.

UDP Streaming Mode

Test 1: Message Loss Analysis

In streaming mode, the following results were observed:

- 524,288 messages sent (buffer size: 1000 bytes)
    - 524,090 messages received
    - Loss rate: ~0.04%
- 374,492 messages sent (buffer size: 1400 bytes)
    - 374,378 messages received
    - Loss rate: ~0.03%
- 104,858 messages sent (buffer size: 5000 bytes)
    - 104,815 messages received
    - Loss rate: ~0.04%
- 10,485,760 messages sent (buffer size: 50000 bytes)
    - 10,485,592 messages received
    - Loss rate: ~0.002%



- A larger message size reduces transmission time (e.g., 12 seconds for 150 bytes vs. 34 seconds for 50 bytes).
- 10486 messages sent (buffer size: 5000 bytes)
    - 10412 messages received
    - Loss rate: ~0.7%.
- The conclusion here is that the buffer should not be too large. Even if only a few messages are lost when the buffer is large, those messages contain a significant amount of data, resulting in a substantial data loss.

UDP Stop-and-Wait Mode

Test 2: Reliability vs. Performance

- Stop-and-wait mode ensures all messages are received correctly, independent of message size.
- The only difference is transmission time, which decreases with larger message sizes, so we can use any message size if we are interested in performance.
- This mode introduces significant overhead, making it much slower than streaming mode.

Statistics and Performance Comparisons for QUIC

Test 1: Message Loss and Performance

The following observations were made for 500 MB transfers:

Buffer Size (Bytes)	Messages Sent	Messages Received	Transmission Time (Seconds)	Loss Rate (%)
150	3,495,254	3,495,150	14	~0.003%
1000	524,288	524,202	2	~0.016%
2000	262,144	262,083	2	~0.023%
5000	104,858	104,782	~0	~0.07%
50000	10486	10432	~0	~0.5%

- The conclusion is that a small buffer increases transmission time but results in minimal data loss (as only a few small messages are lost). Increasing the buffer size significantly reduces transmission time (from 14 seconds to nearly 0). However, with a larger buffer, a similar number of messages are lost, but since they are larger, the total amount of data lost is greater.
- For 1 GB transfers, the trends are similar

QUIC Stop-and-Wait Mode

Test 2: Reliability vs. Performance

- All messages are correctly received, regardless of buffer size.
- Larger buffers improve performance, but transmission is still slower than streaming mode.

- The trade-off is between latency and guaranteed delivery—stop-and-wait ensures perfect accuracy but at the cost of speed.

Statistics and Performance Comparisons

These are for 500MB, if we use 1GB the time will increase

Message size → Protocol ↓	100 bytes	300 bytes	500 bytes	1000 bytes
TCP streaming (500MB)	23 seconds	10 seconds	5 seconds	2 seconds
UDP streaming (500MB)	17 seconds	6 seconds	3 seconds	1 second
QUIC streaming (500MB)	20 seconds	7 seconds	4 seconds	2 seconds

Message size → Protocol ↓	1000 bytes	100 bytes
TCP stop-and-wait(500MB)	28 seconds	262 seconds
UDP stop-and-wait(500MB)	25 seconds	237 seconds
QUIC stop-and-wait(500MB)	27 seconds	250 seconds

Analysis of Experimental Results

1. Impact of Message Size on Performance

- For TCP, increasing the message size improves throughput but can introduce unexpected behavior in streaming mode, where the server may receive more messages than expected due to TCP segmentation.
- For UDP and QUIC, increasing the message size reduces transmission time but leads to higher data loss

2. Protocol Efficiency for Large Transfers

- TCP in streaming mode achieves high efficiency and reliable delivery but may suffer from delays caused by congestion control.
- UDP streaming is fast but can experience significant packet loss, making it unsuitable for critical data transfers without additional mechanisms (e.g., error correction).

- QUIC streaming performs better than TCP, as it avoids TCP's retransmission delays while maintaining reliability through built-in mechanisms.

3. Transmission Speed vs. Reliability

- TCP stop-and-wait ensures 100% data integrity but is slower due to per-message acknowledgments.
- UDP stop-and-wait also ensures correct delivery but remains faster than TCP due to reduced overhead.
- QUIC stop-and-wait achieves a balance between the two, offering reliability similar to TCP but with lower latency.

4. Performance in Unstable Network Conditions

- TCP struggles in high-packet-loss environments due to its reliance on acknowledgments and retransmissions, which increase latency.
- UDP does not suffer from retransmission delays, but high packet loss can severely impact data integrity.

Conclusions: Advantages and Disadvantages of Each Protocol

TCP Protocol
Advantages:
- Guarantees 100% reliable data delivery.
- Optimized for large file transfers.
- Efficient congestion control, ensuring fair bandwidth distribution.

Disadvantages:
- High overhead due to acknowledgments and congestion control.
- Performance decreases in high-latency or lossy networks.
- Handshake delay increases connection setup time.

UDP Protocol

Advantages:
- Minimal overhead, allowing faster transmission.
- Ideal for real-time applications (e.g., video streaming, VoIP).
- No congestion control, making it suitable for burst transmissions.

Disadvantages:
- Unreliable delivery (packet loss can be significant).
- No built-in error correction or retransmission.
- Packet loss increases significantly with larger buffer sizes.

Quic Protocol

Advantages:
- Lower latency than TCP, as it avoids handshake delays.
- Handles packet loss better than UDP, maintaining high speed.
- Multiplexed streams prevent head-of-line blocking.

- Faster reconnections (useful for mobile devices switching networks).

Disadvantages:

- Higher computational overhead due to encryption (TLS 1.3).
- Less optimized than TCP for large-scale data center transfers.
- UDP-based, meaning firewall compatibility may be an issue.

Under different transfer conditions

Transfer Condition	Best Protocol	Why?
Large file transfers (500MB – 1GB)	TCP Streaming / QUIC Streaming	Reliable, optimized for bulk transfers
Real-time applications (VoIP, Video Streaming)	UDP Streaming / QUIC Streaming	Low latency, no retransmissions
Unstable or high-loss networks	QUIC Streaming	Handles loss better than TCP, maintains speed
Secure and encrypted transfers	QUIC / TCP+TLS	QUIC is faster due to built-in TLS
Mobile networks (Wi-Fi to 5G switching)	QUIC	Fast reconnections, better for roaming devices
Small, reliable data transfers (API, IoT)	TCP Stop-and-Wait / QUIC Stop-and-Wait	Ensures message integrity