Comparative Test Results for TCP, UDP, and QUIC Protocols

1. Introduction

This report compares the performance of three protocols: TCP, UDP, and QUIC, based on various message sizes and data amounts. The focus is on:

- Transmission time for each protocol.
- Data loss rates for UDP and QUIC.
- The impact of message size on performance.

The tests are conducted under two scenarios: **with streaming** and **without streaming**. The data sizes tested are **500 MB** and **1 GB**, and message sizes of **1024 bytes**, **4096 bytes**, **16384 bytes**, and **65535 bytes**.

2. Methodology

For each protocol, tests were run with the following parameters:

- **Data Sizes**: 500 MB, 1 GB
- **Message Sizes**: 1024 bytes, 4096 bytes, 16384 bytes, 65535 bytes
- **Streaming**: Tests with streaming (for protocols that support it) and without streaming.

The test scripts run a client-server communication where the client sends data and the server records the transmission time, data received, and any packet losses.

How to Use the Program (Client/Server Options)

To use the program, you will need to run both the **server** and **client** components. Below are the instructions for setting up and running both:

Server Setup:

- Run the following command to start the **TCP server** on your machine:

  ```
  python servers.py --protocol tcp --host 127.0.0.1 --port 5001
  ```

- To start the **UDP server**, use the following command:

  ```
  python servers.py --protocol udp --host 127.0.0.1 --port 5001
  ```
- For the **QUIC server**, run:

  ```
  python servers.py --protocol quic --host 127.0.0.1 --port 5002
  ```

Client Setup:

- To run the **TCP client**, use the following command:
  ```
  python clients.py --protocol tcp --host 127.0.0.1 --port 5001
  --data_size 1GB --msg_size 4096 --streaming
  ```
- To run the **UDP client**, use:
  ```
  python clients.py --protocol udp --host 127.0.0.1 --port 5001
  --data_size 1GB --msg_size 16384
  ```
- For the **QUIC client**, run:
  ```
  python clients.py --protocol quic --host 127.0.0.1 --port 5002
  --data_size 1GB --msg_size 65535
  ```

.Summary of Options:

- `--protocol <tcp|udp|quic>`: Specify the protocol to use for communication.
- `--host <host>`: Set the host address (typically `127.0.0.1` for local connections).
- `--port <port>`: Set the port for the connection (e.g., `5001` or `5002`).
- `--data_size <size>`: Define the data size (e.g., `500MB`, `1GB`).
- `--msg_size <size>`: Define the message size (e.g., `1024`, `4096`, `16384`, `65535`).
- `--streaming`: Enable streaming mode for the client (optional).

3. Results

TCP with Streaming vs. TCP with Stop-and-Wait

1. **Transmission Time**:
   - **TCP with Streaming** consistently shows faster transmission times across all data sizes and message sizes compared to **TCP with Stop-and-Wait**. For instance:
     - For **500MB**, TCP with streaming takes as little as 0.13 seconds (for 65535 message size), while TCP with stop-and-wait takes up to **13.95 seconds** (for 1024 message size).
     - Similarly, for **1GB**, TCP with streaming takes a maximum of **2.52 seconds** (for 1024 message size), while TCP with stop-and-wait takes up to **28.45 seconds**.
2. **Impact of Message Size**:
   - In both **TCP with Streaming** and **TCP with Stop-and-Wait**, larger message sizes reduce the transmission time. However, the effect is much more pronounced with streaming, where transmission times drop dramatically with larger message sizes (e.g., 1GB with message size 65535 takes **0.25 seconds** with streaming, versus **0.35 seconds** with stop-and-wait).
   - With stop-and-wait, transmission time reduction with larger message sizes is also observed, but it is less efficient compared to streaming.

3. **Efficiency**:

    - **TCP with Streaming** is much more efficient for larger data sizes, with faster transmissions and lower latency. It seems to handle data transfer in a more continuous flow, leading to less delay.
    - **TCP with Stop-and-Wait** suffers from significant delays, especially for larger data sizes, due to the need to wait for an acknowledgment after each message.

Conclusion:

TCP with streaming provides a clear advantage over TCP with stop-and-wait in terms of transmission time, especially for larger data sizes. The **stop-and-wait protocol**, while still functional, results in significantly slower data transfers and higher latency. Therefore, streaming is a more efficient and faster approach, particularly when large data sets and bigger message sizes are involved. This difference could have a considerable impact in scenarios requiring high-speed data transmission.

Protocol	Data Size	Message Size	Transmission Time	Data Loss
TCP-streaming	500MB	1024	1.20 sec	n/a
	500MB	4096	0.35 sec	n/a
	500MB	16384	0.16 sec	n/a
	500MB	65535	0.13 sec	n/a
	1GB	1024	2.52 sec	n/a
	1GB	4096	0.71 sec	n/a
	1GB	16384	0.34 sec	n/a
	1GB	65535	0.25 sec	n/a
TCP-stop and wait	500MB	1024	13.95 sec	n/a
	500MB	4096	3.56 sec	n/a
	500MB	16384	0.90 sec	n/a
	500MB	65535	0.17 sec	n/a
	1GB	1024	28.45 sec	n/a
	1GB	4096	7.34 sec	n/a

	1GB	16384	1.94 sec	n/a
	1GB	65535	0.35 sec	n/a

UDP with Streaming vs. UDP with Stop-and-Wait

Transmission Time:

- **UDP with Stop-and-Wait** takes significantly longer to transmit data compared to **UDP with Streaming**. For 500MB of data, the transmission time in Stop-and-Wait mode is much higher, especially with smaller message sizes (e.g., 1024 bytes takes 11.85 seconds, compared to just 0.82 seconds in Streaming mode).
- Even for larger data sizes (1GB), the difference is notable. **Stop-and-Wait** mode takes 24.30 seconds with a 1024-byte message size, while **Streaming** mode only requires 1.69 seconds.

Data Loss:

- **UDP with Stop-and-Wait** shows **0.00% data loss** across all configurations, which is expected as each message is acknowledged before the next is sent, ensuring reliable delivery.
- On the other hand, **UDP with Streaming** shows a noticeable amount of **data loss** that increases as the message size increases. For 500MB of data, loss ranges from 36.32% (1024 bytes) to 9.49% (65500 bytes). Similarly, for 1GB, the loss decreases from 36.75% to 5.25% as the message size increases.

Effect of Message Size:

- In both modes, larger message sizes reduce transmission time. However, the reduction in transmission time is far more pronounced in **UDP with Streaming**, where larger messages (e.g., 65500 bytes) lead to transmission times of just 0.07 seconds for 500MB and 0.14 seconds for 1GB.
- In **UDP with Stop-and-Wait**, increasing message sizes also reduce transmission time, but the improvement is less drastic, and the transmission times remain higher overall.

Conclusion:

- **UDP with Streaming** offers faster transmission times but sacrifices reliability, leading to higher data loss. It is suitable for scenarios where speed is prioritized over guaranteed delivery.
- **UDP with Stop-and-Wait** ensures 100% reliability with no data loss, but it has much slower transmission speeds, making it less efficient for large data transfers compared to Streaming mode. It is more suitable for applications requiring guaranteed message delivery.

Protocol	Data Size	Message Size	Transmission Time (sec)	Data Loss
UDP-stop and wait	500MB	1024	11.85	0.00%
	500MB	4096	3.07	0.00%
	500MB	16384	0.82	0.00%
	500MB	65500	0.26	0.00%
	1GB	1024	24.30	0.00%
	1GB	4096	6.17	0.00%
	1GB	16384	1.65	0.00%
	1GB	65500	0.57	0.00%
UDP-streaming	500MB	1024	0.82	36.32%
	500MB	4096	0.23	33.09%
	500MB	16384	0.09	15.61%
	500MB	65500	0.07	9.49%
	1GB	1024	1.69	36.75%
	1GB	4096	0.48	32.51%
	1GB	16384	0.20	7.28%
	1GB	65500	0.14	5.25%

Quic with Stop-and-Wait

- **Transmission Time**: QUIC experiences extremely long transmission times, especially with smaller message sizes. For **500MB** data, using **1024 bytes** results in **5347.52 sec**, and for **1GB**, the transmission time is approximately **10982.57 sec**. These times are significantly higher than both **TCP** and **UDP** protocols, indicating inefficiency when using **Stop-and-Wait** for QUIC.

- **Data Loss**: There is a substantial amount of data loss, especially for smaller message sizes. For example: At **500MB** with **1024 bytes**, the data loss is extreme (close to 100%). Even for larger message sizes like **16384 bytes** and **65535 bytes**, the data loss remains high, hovering around **99.95% to 97.15%**.

- **Performance Decrease**: Even though QUIC is typically favored for its low-latency features, its use in **Stop-and-Wait** here has led to poor results. The performance drop with larger message sizes still does not compensate for the high data loss, especially in comparison to **TCP** or **UDP**.

Conclusion:

- The poor results observed for **QUIC with Stop-and-Wait** are likely due to issues with the **implementation** of the protocol in this context, leading to **high transmission times** and **excessive data loss**. **QUIC's performance in this scenario could be improved significantly** if optimized

Protocol	Data Size	Message Size	Transmission Time (sec)	Data Loss
QUIC -stop and wait	500MB	1024	5347.52	99.99%
	500MB	4096	1334.23	99.98%
	500MB	16384	334.46	99.95%
	500MB	65535	84.04	97.15
	1GB	1024	10982.57	99.99%
	1GB	4096	2746.27	99.99%
	1GB	16384	685.72	99.99%
	1GB	65535	171.93	99.98%

4.Final Conclusions

- **TCP**:
  - **Advantages**: Reliable, with lower transmission times for larger message sizes, especially in streaming mode. Best suited for applications requiring guaranteed delivery.
  - **Disadvantages**: Slower transmission in Stop-and-Wait mode, especially with small message sizes. Higher latency compared to UDP and QUIC in certain scenarios.
- **UDP**:
  - **Advantages**: Faster transmission times, particularly in streaming mode. More efficient for large data transfers where speed is prioritized over reliability.
  - **Disadvantages**: Higher data loss in streaming mode. Stop-and-Wait mode ensures reliability, but at the cost of much slower transmission speeds.
- **QUIC**:

    - **Advantages**: Designed for low-latency performance, ideally suited for modern applications requiring fast data transfer.
    - **Disadvantages**: Poor performance with Stop-and-Wait, resulting in extremely high transmission times and significant data loss due to protocol implementation issues.

In summary, **TCP with Streaming** offers the best balance of reliability and speed, while **UDP with Streaming** is optimal for speed at the expense of reliability