Concurrent and Distributed Programming

Homework 1 - Using and Measuring TCP and UDP transfers of various amounts of data

> Amarandei Matei Alexandru - MISS1

Introduction 🌍

The program is designed to measure the time required to transfer data of various sizes, under various network conditions, using various combinations of protocols and methods.

The program can be run on a **Linux** system, or any **Unix**-based Operating System (*e.g. macOS*). It currently supports two protocols, **TCP** and **UDP**, and two mechanisms for data transfer, namely **streaming** and **stop-and-wait**.

Task Breakdown 📝

1. The program **must** run on a Linux system
2. The supported protocols must be **parameters** for both client and server
3. Message size must be between **1 and 65535** bytes
4. **Streaming** and **Stop-and-Wait** mechanisms for data transfer
5. Output must contain:
    a. After each server session, the server will print the following information:
        - Used protocol
        - Number of messages read
        - Number of bytes read

    b. At the end of execution, the client will print the following information:
        - Transmission time
        - Number of sent messages
        - Number of bytes sent
6. A document presenting the process and giving an overview

Implementation 💻

Technologies used:

- **Language:** *Python 3.10*
- **Operating System:** *macOS Ventura*
- **IDE: Visual Studio Code**

Modules used:

- **Sockets:** *socket*
- **Time Measurement:** *time*
- **Configuration:** *JSON*
- **Data Generation:** *os*

Files 📑

1. **client.py:** *The client script that allows for:*
   a. *choosing a protocol between **TCP** or **UDP***
   b. *choosing a transfer method between **Stop-and-Wait** and **Streaming***
   c. *choosing the amount of data to be sent between **1MB**, **10MB**, **50MB**, **100MB**, **500MB**, **1GB***
2. **server.py:** *The server script that allows for:*
   a. *choosing a protocol between **TCP** or **UDP***
   b. *choosing a transfer method between **Stop-and-Wait** and **Streaming***
3. **data:**
   a. **common_config.json:** *Holds common data, such as **local host** and **message sizes***
   b. **config.json:** *Contains **port** and **buffer size** information with regards to both the protocol and method used*
   c. **data.txt:** *The file containing the data to be transferred*
4. **utils:**
   a. **generate_file.py:** *Generates a file of a specified **name** and **size**, comprised of **0 bytes***
   b. **json_utils.py:** *Reads and returns a **JSON** object as a **dictionary***

Measurement Approach 🤔

1. Used a stable and reliable network connection for the test
2. Closed all other programs and processes that may affect network performance during the test
3. Ran the program multiple times to get an average measurement and reduce the impact of outliers
4. Used multiple combinations of message sizes and clients
5. Monitored network traffic during the test to detect any anomalies that may affect the accuracy of the measurements

Measurements 📊

Protocol + Method	TCP + Stop and Wait	Messages	TCP + Streaming	Messages	UDP + Stop and Wait	Messages	UDP + Streamin
1MB	1.00881099	*23*	0.00340986	*17*	0.04556584	*161*	0.03210
10MB	1.02543187	*208*	0.11290597	*161*	0.173675727	*1601*	0.12236
50MB	1.04338097	*931*	2.76955986	*801*	0.717549429	*8001*	0.50554
100MB	1.08053302	*2155*	10.17102503	*1601*	1.555606007	*16002*	1.09599
500MB	1.32335710	*9525*	349.042908	*8001*	7.802975678	*80008*	5.49752
1GB	1.71180009	*19068*	821.764993	*16002*	14.33305622	*163856*	10.0982
BUFFER	**65535**		**65535**		**6553**		**6553**

Conclusion 🔚

TCP and UDP

1. TCP with stop-and-wait is the least efficient protocol, with the lowest throughput and longest latency. This is because packets are sent one at a time, and the sender must wait for each packet to be acknowledged before sending the next one.
2. UDP with stop-and-wait can provide moderate throughput, but reliability is also compromised due to the lack of error checking and flow control.

3. TCP with streaming is generally the most efficient protocol, providing the highest throughput and reliability. This is because TCP uses a sliding window protocol that allows for multiple packets to be sent and acknowledged in parallel, reducing the amount of time that the sender has to wait for acknowledgments. In this specific case, however, there were some inconsistencies.
4. UDP with streaming can provide high throughput, but reliability may be compromised due to the lack of error checking and flow control.

Buffer and Message Sizes

In terms of the buffer size and message size, larger buffer sizes generally allow for more efficient data transfer, as they can store more data for processing and transmission. However, large buffer sizes can also lead to higher latency and delay in data transmission. Similarly, larger message sizes can provide higher throughput, but also increase the likelihood of errors and packet loss, especially in networks with high congestion or limited bandwidth.

All things considered, the proper combination of protocol, transfer method, buffer size and message size can be found by carefully analysing the use-case, but also by means of experimenting and benchmarking, which may reveal some interesting insights.