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*
- **IDE:** *Visual Studio Code*

Modules used:

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

Environments used:

- **Local:** *macOS Ventura*
- **Cloud:** *EC2 - Amazon Linux with t2 Micro*

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, 2GB***
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 📊

💡 TCP - Stop and Wait (Left: Local, Right: EC2) - Averages for 50 runs

Message Size	Buffer Size	Elapsed Time	Number of Messages	Bytes Received	Bytes Sent		Elapsed Time	Number of Messages	Bytes Received
1 MB	65535	0.00512	19	1048576	1048576		0.00919	16	1048576
50 MB	65535	0.02982	955	53116907	52428800		0.07425	822	53116907
100 MB	65535	0.16104	1618	106004445	104857600		0.19173	1621	106135513
1 GB	65535	1.51906	16486	1087372896	1073741824		1.75259	16703	1104313435
2 GB	65535	3.01172	34294	2204989733	2147483648		3.43012	33489	2178612298

💡 TCP - Streaming (Left: Local, Right: EC2) - Averages for 50 runs

Message Size	Buffer Size	Elapsed Time	Number of Messages	Bytes Received	Bytes Sent		Elapsed Time	Number of Messages	Bytes Received
1 MB	65535	0.00034	17	1048576	1048576		0.00101	17	1048576
50 MB	65535	0.01825	801	52428800	52428800		0.01802	801	52428800
100 MB	65535	0.02417	1601	104857600	104857600		0.02657	1601	104857600
1 GB	65535	0.38076	16385	1073741824	1073741824		0.47534	16385	1073741824
2 GB	65535	0.76141	32769	2147483648	2147483648		0.85420	32769	2147483648

💡 UDP - Stop and Wait (Left: Local, Right: EC2) - Averages for 50 runs

Message Size	Buffer Size	Elapsed Time	Number of Messages	Bytes Received	Bytes Sent		Elapsed Time	Number of Messages	Bytes Received
1 MB	32767	0.01133	33	1048576	1048576		0.02091	33	1048576
50 MB	32767	0.31614	1601	52428800	52428800		0.31917	1601	52428800
100 MB	32767	0.52752	3201	104857600	104857600		0.62540	3201	104857600
1 GB	32767	6.42834	32770	1073741824	1073741824		6.44350	32770	1073741824
2 GB	32767	12.21871	65539	2147483648	2147483648		13.18978	65539	2147483648

UDP - Streaming (Left: Local, Right: EC2) - Averages for 50 runs

Message Size	Buffer Size	Elapsed Time	Number of Messages	Bytes Received	Bytes Sent		Elapsed Time	Number of Messages	Bytes Received
1 MB	32767	0.00753	33	1048576	1048576		0.02796	33	1048576
50 MB	32767	0.16489	1601	52428800	52428800		0.17274	1601	52428800
100 MB	32767	0.3141	3201	99811482	104857600		0.31665	3201	98500802
1 GB	32767	3.14358	32770	1049297642	1073741824		3.30749	32770	1062469976
2 GB	32767	6.54604	65539	2119926601	2147483648		6.63104	65539	2122875631

Conclusion END

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.
4. UDP with streaming can provide high throughput, but reliability may be compromised due to the lack of error checking and flow control. Noticed significant losses.

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.

Environment

A Local Wi-Fi, with only one device connected and only this tasks running, seems to provide, empirically, but also logically, the best average performance. However, surprisingly, the performance did not decline as much as it was expected when running the server and client on two dedicated EC2 t2 micro instances.

Final Thoughts

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.