CSC 573 -PROJECT 1

Team Member	Percentage contribution
Sahil Purohit(spurohi2)	50%
Shivangi Chopra(schopra4)	50%

HTTP 1.1		
Sub-Tasks	**Sahil**	**Shivangi**
Research, identifying, and understanding the operation of the packages that implement the protocol	50 %	50 %
Client implementation	30 %	70 %
Server implementation	40 %	60 %
Extend implementation to perform computations on both machines A and B	50 %	50 %
Perform the experiments to derive the results in Results_file.xlsx	50 %	50 %

HTTP 2		
Sub-Tasks	**Sahil**	**Shivangi**
Research, identifying, and understanding the operation of the packages that implement the protocol	50 %	50 %
Client implementation	50 %	50 %
Server implementation	50 %	50 %
Extend implementation to perform computations on both machines A and B	50 %	50 %
Perform the experiments to derive the results in Results_file.xlsx	50 %	50 %

gRPC		
Sub-Tasks	**Sahil**	**Shivangi**
Research, identifying, and understanding the operation of the packages that implement the protocol	50 %	50 %
Client implementation	60 %	40 %
Server implementation	65 %	35 %
Extend implementation to perform computations on both machines A and B	50 %	50 %

Perform the experiments to derive the results in Results_file.xlsx	50 %	50 %

Bittorrent		
Sub-Tasks	**Sahil**	**Shivangi**
Research, identifying, and understanding the operation of the packages that implement the protocol	50 %	50 %
Seeder implementation	60 %	40 %
Leecher implementation	40 %	60 %
Perform the experiments to derive the results in Results_file.xlsx	50 %	50 %

Observations

HTTP 1.1 Protocol –

Library :

File transfer using HTTP 1.1 protocol was tested using the python packages http which comes as a part of python3.

https://docs.python.org/3/library/http.html

Observations from experiment: This protocol is fast and simple to transfer files from server to client. From the results, it can be deduced that the protocol performs fairly badly in terms of throughput if the file is small, as for the small files the transmission time is extremely small but the propagation time can get relatively large. Therefore, it can be seen from the results, as the file size increases the throughput of the protocol also increases. But this increase is not linear as the propagation time is dominating which usually remains constant but the transmission time increases almost linearly and sum of these two times decides the throughput. This library provides a complete solution to implement HTTP 1.1 client server model as it does not require explicit socket implementation and takes care of other limiting factors implicitly.

HTTP 2 Protocol –

Library :

File transfer using HTTP 2.0 protocol was tested using the python packages the python package h2 which can be installed through these instructions :

https://pypi.org/project/h2/

Cmd - pip install h2

Observations from experiment: Similar to HTTP 1.1, this protocol is also relatively simple and fast. But the library which we utilized in addition to sockets does not implement flow control windows. We had to explicitly create sockets and write required operations for it. We explicitly wrote acknowledgement and flow control window messages to continue the transmission. Because of this, performance of this protocol seems to be poor in comparison to HTTP 1.1 and gRPC. This can be justified by the results. Results are expected to change for this protocol as the network changes fast and because of a lot of explicit operations performed. Hence, the throughput for different sizes of file is relatively small and the increase is not that significant in terms of HTTP 1.1.

GRPC Protocol –

Library :

gRPC protocol was tested using the python packages grpcio and grpcio-tools mentioned below -

https://pypi.org/project/grpcio/

https://pypi.org/project/grpcio-tools/

These can be installed using pip like this –

pip install grpcio grpcio-tools

Observations from experiment:

Through our experiments of sending files of sizes 10kB up to 10MB between the client and server, we observed a much higher throughput using gRPC – roughly 2x higher than HTTP 1.1 for file sizes below 10 MB.

The increase in throughput using gRPC can be attributed to the following reasons.

- Using HTTP/2 instead of HTTP/1.1

While HTTP/1.1 allows for processing just one request at a time, HTTP/2 supports multiple calls via the same channel. HTTP/2 communication is divided into smaller messages and framed in binary format, which unlike text-based HTTP/1.1, makes sending and receiving messages compact and efficient.

- Using protocol buffers as an alternative to XML and JSON.

Parsing with Protocol Buffers is less CPU-intensive because data is represented in a binary format which minimizes the size of encoded messages. At runtime, messages are compressed and serialized in binary format.

Bittorrent Protocol

Library :

The BitTorrent protocol was implemented and tested using the python package- libtorrent, py3createtorrent as mentioned below -

https://www.libtorrent.org/python_binding.html

https://py3createtorrent.readthedocs.io/en/latest/user.html

Since we implemented this on linux machine, the installation commands are as follows:

sudo apt install libtorrent

sudo apt install python-libtorrent

Observations from experiment:

Through our experiments of sending files of sizes 10kB up to 10MB between the peers, we observed the highest throughput using BitTorrent– roughly 2x higher than gRPC for the file sizes of 10MB.

The increase in throughput using BitTorrent for bigger files can be attributed to the following reasons:

- Adaptable Bandwidth: A fixed bandwidth is often used by HTTP1.1, HTTP2, and gRPC for the entirety of the file transfer process, which might result in inefficient bandwidth use. In contrast, BitTorrent allows each peer to change the upload and download rates according to the network conditions, so it can adapt to the available bandwidth.
- Robust: A single point of failure in the network can halt the entire file transfer process, making HTTP1.1, HTTP2, and gRPC vulnerable to network disruptions. BitTorrent, on the other hand, enables each peer to download portions of the file from numerous other peers, making it more robust to network disruptions.
- Increase in throughput: HTTP1.1, HTTP2, and gRPC uses client-server model where the server sends the entire file. This acts as a bottleneck that prevents the server from handling multiple requests and can be slow for large files. In contrast, BitTorrent splits the file into smaller chunks and enables each peer to simultaneously download chunks from a number of other peers. Parallel downloads are made possible as a result, greatly accelerating file transfers.

The standard deviation for all readings for bittorrent is low since we ran all peers locally on the same machine.

Observations of all the protocols combined

Overall, it can be concluded that for the client-server model, gRPC has performed better. In reality, HTTP 2.0's performance should be better than HTTP 1.1 and comparable to gRPC as gRPC relies on HTTP 2.0, providing it with better performance and low latency. gRPC with HTTP/2 is generally more efficient for smaller payloads due to its advanced features like multiplexing and header compression, while HTTP/1.1 might perform better for larger files due to its simpler framing mechanismBut as mentioned earlier, HTTP 2.0's performance is bad because of lack of feature implementation and requirement of implementing sockets, which costs additional time.