CSC 573 INTERNET PROTOCOLS

PROJECT 1

Team members:

Shardul Khare (srkhare) 50%

Shrishty Singh (ssingh67) 50%

Subtasks	Shardul	Shrishty
Setup Requirements: connecting our laptops through local area network, and connected 4 laptops for BitTorent	50%	50%
Experiment design: number of files to be sent, which files to be sent, TCP connection limitations	50%	50%
Defining libraries: The libraries and packages to be used for each protocol	50%	50%
Writing code: writing the code for all the protocols	50%	50%
Testing code: testing, debugging and fixing the code for all the protocols	50%	50%
Measurement Criteria: measuring the average throughput and standard deviation	50%	50%
Report making and Analysis: making the report, analyzing the performance of protocols	50%	50%

Libraries used

Http 1.1

a) socket for basic tcp connections and it is included in python

Http 2.0

a) socket for basic tcp connections, it is included in python
b) h2. connection and h2.config are part of the h2 library (HTTP/2) in python, it can be installed using this command
   pip install h2
c) os library is used for file and directory operations, such as reading from or saving files to disk, it is included in python.

GRPC

a) grpcio for implementing grpc services, it can be installed using this command
   pip install grpcio
b) grpcio-tools for compiling protocol buffers (.proto), it can be installed using this command
   pip install grpcio-tools
c) concurrent.futures for running asynchronous tasks (important in grpc to handle concurrent service requests)
d) file_transfer_pb2 and file_transfer_pb2_grpc are generated from .proto files using protocol buffers compiler (grpcio-tools and grcpcio). These files are locally locally generated by running this command:
   python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. your_proto_file.proto
e) os library is used for file and directory operations, such as reading from or saving files to disk, it is included in python.

BitTorrent

a) Libtorrent for BitTorent implementation, it can be installed using this command (For mac os only):
   pip install python-libtorrent
b) os library is used for file and directory operations, such as reading from or saving files to disk, it is included in python.

Additional libraries:

a) Numpy used for its statistical functions, such as calculating the standard deviation of throughput values, it is included in python.
b) time library is used to measure the duration of file transfers, which is critical for calculating throughput (bytes per second), it is included in python.
c) Statistics for calculating the standard deviation among throughput values in simpler scripts or when adhering to minimal external dependencies, it is included in python.

Analyzing the observations:

The discussion below goes deep into the nuances of each protocol's performance and their optimal use cases, grounded in the statistics for average throughput, standard deviation, and the total application layer data transferred. This exploration aims to justify the observed performance differences and suggests scenarios where each protocol excels.

HTTP 1.1:

Efficiency and Overhead:

HTTP 1.1, with its sequential handling of requests, shows a decline in efficiency as file sizes increase. This protocol's design necessitates a new TCP connection for each request-response cycle, introducing significant overhead for large files or numerous small file transfers. The protocol's throughput decreases with larger files due to the cumulative latency from connection setup and teardown, and the overhead becomes more pronounced, evidenced by lower throughput figures for large file transfers. This overhead, however, is somewhat mitigated for smaller files, where the simplicity of HTTP 1.1 can be advantageous.

Reliability and Use-case Applicability:

Despite its limitations in handling concurrent requests efficiently, HTTP 1.1 remains widely used due to its universality and simplicity, making it a dependable choice for web applications with standard request-response interactions. It is particularly suited for applications where long-lived connections or high throughput are not critical. However, for modern applications requiring rapid content delivery and real-time interaction, HTTP 1.1's limitations become apparent.

HTTP 2.0:

Efficiency and Overhead:

The introduction of multiplexing in HTTP 2.0 allows multiple requests and responses to be interconnected on a single TCP connection, drastically reducing the overhead associated with HTTP 1.1's one-request-per-connection model. This capability not only enhances throughput for large file transfers but also improves the performance of web applications that require numerous resource fetches to render a page. The binary framing layer in HTTP 2.0 further reduces overhead, contributing to the protocol's superior throughput performance across all file sizes.

Reliability and Use-case Applicability:

HTTP 2.0's features, including stream prioritization and server push, are designed to optimize the performance of complex web applications. These features make HTTP 2.0 an excellent choice for high-performance web applications, APIs, and services requiring efficient use of network resources and fast delivery of diverse content types.

GRPC:

Efficiency and Overhead:

Built on top of HTTP 2.0, GRPC inherits its transport efficiency and adds benefits through protocol buffers, a mechanism for serializing structured data. This combination results in reduced payload sizes and, consequently, lower transmission overhead. GRPC's support for bidirectional streaming further enhances its efficiency, allowing for continuous data exchange without the need to reestablish connections. This protocol demonstrates high throughput and low overhead, particularly in environments where the exchange of small, frequent messages is common.

<u>Reliability and Use-case Applicability</u>:

GRPC is particularly suited for microservices architectures, where services frequently communicate with each other over the network. Its performance characteristics make it ideal for internal service communication, real-time data services, and mobile application backends, where efficiency and low latency are paramount. The structured nature of protocol buffers also aids in defining clear and robust service interfaces.

BitTorrent:

<u>Efficiency and Overhead:</u>

BitTorrent's peer-to-peer file-sharing model offers a distinct approach to data transfer, distributing the load across multiple peers rather than centralizing it on a server. This approach significantly reduces the overhead on any single point in the network, as file pieces are simultaneously uploaded and downloaded among peers. The protocol's efficiency improves as more peers participate, demonstrating a scalable solution for large file transfers. BitTorrent's performance, particularly for large files, highlights its capacity to handle high volumes of data transfer with reduced overhead, as evidenced by the throughput calculated when multiplying the file size by three for the peer-to-peer transfers.

<u>Reliability and Use-case Applicability</u>:

Ideal for scenarios requiring the distribution of large files to many users, such as software distributions, media files, and content delivery networks (CDNs). BitTorrent's resilience and efficiency in these scenarios stem from its decentralized nature, which not only speeds up transfers but also provides a robust mechanism against failures or bottlenecks commonly seen in client-server models.

Conclusion

The comparative analysis of HTTP 1.1, HTTP 2.0, GRPC, and BitTorrent protocols reveals distinct advantages and limitations of each, emphasizing the importance of selecting the right protocol based on specific application needs. While HTTP 1.1 offers simplicity and broad compatibility for traditional web applications, HTTP 2.0 and GRPC provide the efficiency, speed, and modern features required for today's dynamic web environments and microservices architectures. On the other hand, BitTorrent excels in scenarios that demand scalable, distributed file sharing, showcasing its unique strengths in peer-to-peer data transfer.