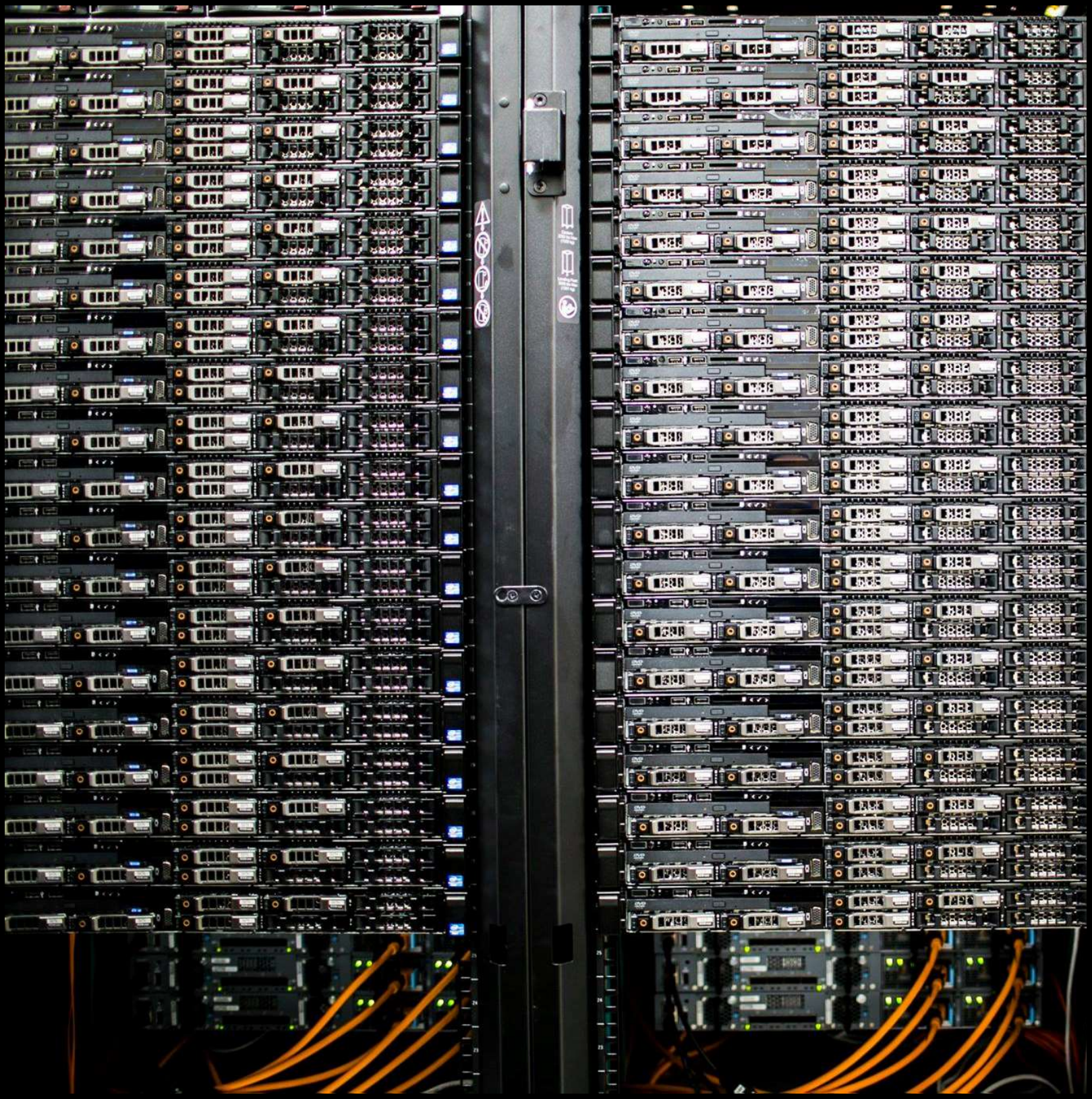
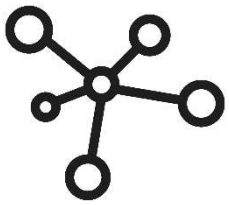


HTTP2 Server



Computer Networks

Ain Shams University
Faculty of Engineering
CESS
CSE351 Computer Networks
Fall 24
Proposal Document

Proposal Done by:

Youssef Mohamed AlSayed	21P0094
Youssef Mohamed Salah	21P0159
Saleh Ahmed ElSayed	21P0324

Table of Contents

ABSTRACT

An HTTP/2 web server design is presented in this proposal with the goal of enhancing the effectiveness and performance of contemporary web communication. By leveraging essential elements of the HTTP/2 protocol, such as header compression, flow control, and multiplexing, the server gets around the drawbacks of HTTP/1.1. The server can now manage several data streams over a single TCP connection thanks to these improvements, which lowers latency and optimizes resource utilization.

Specialized components at the core of the server's design cooperate to guarantee seamless operation. The Connection Handler manages incoming client connections and validates the initial HTTP/2 handshake. The Frame Processor takes care of parsing and validating HTTP/2 frames. The Stream Manager handles stream lifecycles, including creation, prioritization, and closure, ensuring the server follows multiplexing rules. The HPACK Module compresses and decompresses headers efficiently, helping to minimize data size and save bandwidth. The Flow Control Module manages data transmission to prevent overload, while the Error Handler gracefully addresses any protocol violations or errors that arise.

PROJECT SCOPE

Developing a reliable, high-performance web server that fully supports the HTTP/2 protocol is the goal of this project. By implementing HTTP/2, the server will offer faster connectivity, lower latency, and better efficiency in comparison to traditional HTTP/1.1 servers.

Managing several client connections at once, accurately processing incoming requests, and effectively managing numerous requests and responses over a single connection are the project's main goals. The server will also implement effective data flow management, reduce communication overhead with header compression, and ensure stable operation through robust error handling.

To further enhance performance, the server will support persistent connections to minimize the need for repeated handshakes, improving response times. The design document will provide more detailed technical information on these features.

COMPONENTS

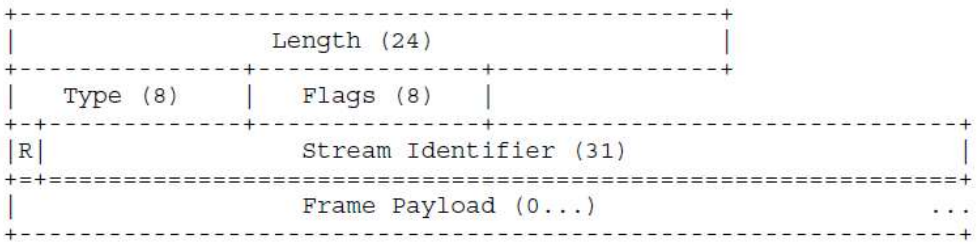
Connection Handler

- Manages TCP connections from clients.
- Validates HTTP/2 connection preface.
- Initializes client-specific settings.

The Connection Handler is responsible for managing TCP connections that clients set with the server. The connection handler's first task is to validate the HTTP/2 connection preface, and this is to ensure that the client sticks to the protocol's initial handshake requirements as specified in RFC7540. This involves checking for the specific magic string and correctly handling settings exchanged during the initial communication. Additionally, it initializes client-specific settings such as flow control parameters and stream limits, tailoring the server's behavior to each client's needs while ensuring compatibility with the HTTP/2 protocol.

Frame Processor

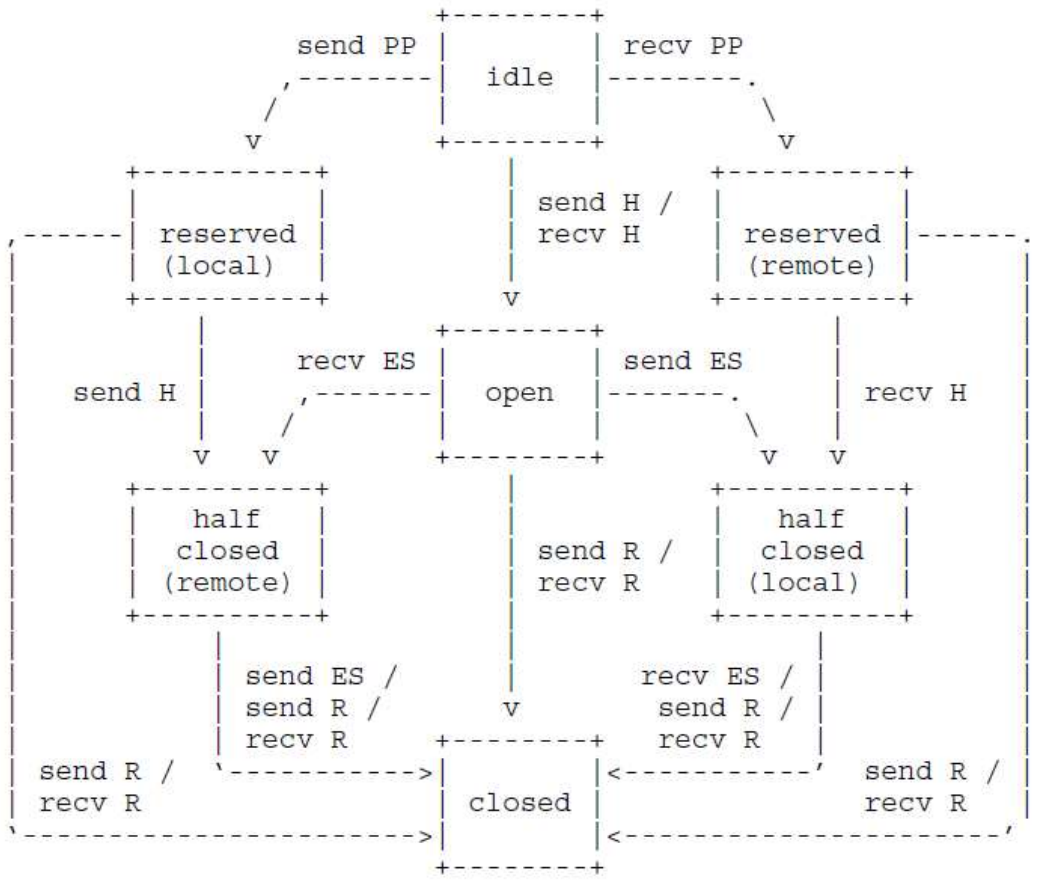
- Parses incoming HTTP/2 frames (HEADERS, DATA, SETTINGS,....).
- Validates frame types, flags, and lengths.
- Delegates frame processing to the appropriate handler.



The Frame Processor is responsible to handle the parsing and validating the incoming HTTP/2 frames. This include HEADERS, DATA, SETTINGS, and other types. The Frame Processor ensures that each frame complies with the protocol's structure, verifying frame types, flags, and lengths as outlined in the specification. Once a frame passes validation, the Frame Processor assigns its handling to the appropriate component, whether it's routing a HEADER frame to stream initialization or passing a DATA frame for further processing.

Stream Manager

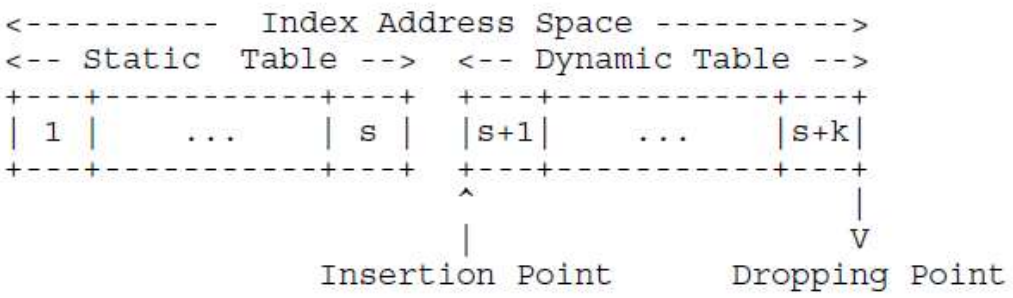
- Manages the lifecycle of streams (creation, activity, closure).
- Handles stream prioritization and dependencies.
- Ensures compliance with HTTP/2 multiplexing rules.



The Stream Manager manages the lifecycle of HTTP/2 streams, this includes their creation, activity and their closure. The Stream Manager handles complex tasks like stream prioritization and managing dependencies between streams, This ensures that the server sticks to HTTP/2's multiplexing rules. This includes respecting priority weights and dependencies to optimize resource allocation and stream scheduling. Additionally, the Stream Manager ensures compliance with protocol constraints, such as avoiding stream ID reuse and maintaining proper state transitions.

HPACK Compression/Decompression Module

- Implements header compression and decompression using static and dynamic tables.
- Encodes and decodes HTTP header blocks as specified in RFC 7541.



According to RFC 7541, using static and dynamic tables, the HPACK Compression/Decompression Module implements the compression and decompression of HTTP headers. It acts as a key component in reducing the size of HTTP header blocks. This reduces bandwidth usage and also helps in improving communication performance in HTTP/2. When compressing headers, it maps header field names and values to indexes in a pre-defined static table or dynamically maintained tables that are shared between the client and server. This process is reversed during decompression, translating the compressed representation back into human-readable HTTP headers.

Flow Control Module

- Manages connection-level and stream-level flow control.
- Processes WINDOW_UPDATE frames and adjusts flow control windows accordingly.

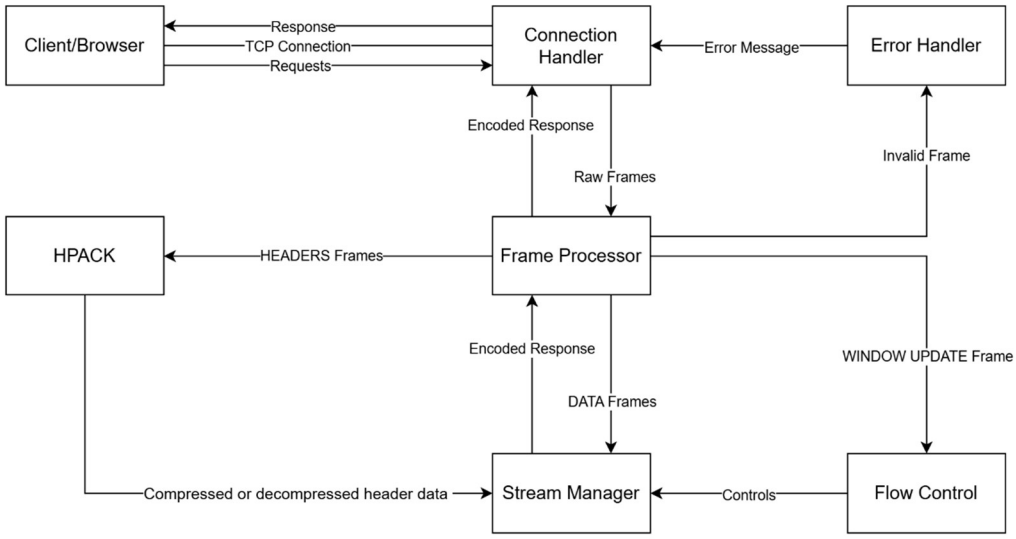
According to RFC 7540, the Flow Control Module is in charge of controlling data flow between the client and server at the connection and stream levels. By controlling the size of the flow control windows, the Flow Control Module makes sure that neither side is overloaded with incoming data. This module processes **WINDOW_UPDATE** frames sent by peers, which inform it of available space in their respective flow control windows. By dynamically adjusting these windows, the module prevents buffer overflows and ensures smooth and reliable data transmission. This flow control mechanism is essential for optimizing throughput and fairness in multi-stream scenarios where multiple streams share the same connection.

Error Handler

- Detects and handles protocol errors such as malformed frames or invalid stream states.
- Sends appropriate HTTP/2 error codes (RST_STREAM, GOAWAY, ...).

The Error Handler is designed to detect and respond to protocol errors that may happen during HTTP/2 communication. According to RFC 7540, these errors may be malformed frames, invalid stream states, or violations of HTTP/2's stateful protocol rules. If found an error, the handler sends the appropriate HTTP/2 error code, such as **RST_STREAM** to terminate a specific stream or **GOAWAY** to signal the end of the connection. This ensures that protocol errors are well defined without interrupting other active streams or degrading overall connection performance.

INTERACTION BETWEEN COMPONENTS



Connection Handler

Input	Incoming client TCP connection.
Tasks	Read and validate the HTTP/2 connection preface. Exchange initial SETTINGS frames with the client. Forward incoming frames to the Frame Processor.
Output	Validated and parsed frames passed to the Frame Processor.

Frame Processor

Input	Raw frames received from the Connection Handler.
Tasks	Parse frame headers and payloads. Dispatch frames to the appropriate modules Enforce protocol rules, such as stream ID validation.
Output	Processed frames forwarded to respective modules

Stream Manager

Input	HEADERS and DATA frames.
Tasks	Manage stream states (idle, open, closed). Handle prioritization and dependency hierarchies. Forward stream-specific data to the application layer.
Output	Encoded/decoded HTTP/2 responses sent to the Frame Processor.

HPACK Compression/Decompression Module

Input	Header blocks from HEADERS frames.
Tasks	Compress HTTP/2 response headers into header blocks. Decompress client header blocks and populate dynamic tables.
Output	Compressed or decompressed header data passed to the Stream Manager.

Flow Control Module

Input	WINDOW_UPDATE frames.
Tasks	Manage flow control windows for streams and the connection. Ensure that outgoing DATA frames respect flow control limits.
Output	Adjusted flow control windows passed to the Frame Processor.

Error Handler

Input	Detected protocol violations or internal errors.
Tasks	Log errors and send appropriate HTTP/2 error frames (e.g., GOAWAY). Gracefully terminate streams or the connection.

SYSTEM COMMUNICATION PROTOCOLS

HTTP/2 Frames

- The server will implement the frame structure defined in RFC 7540, with each frame comprising:
    - **9-byte header**: Includes length, type, flags, and stream ID.
    - **Variable-length payload**: Depends on the frame type.

Header Compression

- Use HPACK (RFC 7541) to compress and decompress HTTP/2 headers.
- Maintain static and dynamic tables for efficient header encoding/decoding.

Multiplexing and Flow Control

- Use the stream identifier to manage multiple streams over the same connection.
- Respect flow control windows to ensure efficient data transmission.

KEY FUNCTIONALITIES OF WEB SERVER

Handle Client Connections

The web server will be designed to efficiently accept incoming connections from clients, ensuring smooth communication. It will maintain persistent connections by implementing the "keep-alive" mechanism. By doing this, latency is decreased and performance is enhanced because there is no need to create new connections for each request. The server can manage several requests and responses over a single connection by maintaining the connection, which makes it appropriate for contemporary applications where resource optimization and performance are crucial.

Protocol Handling (HTTP/2)

Processing of the HTTP/2 protocol, including the management of HTTP/2 frames as specified in RFC7540, will be handled by the server. In HTTP/2, frames are the basic building blocks of communication, and the server will parse and interpret them to provide content in an exact and effective manner. Additionally, it will implement HPACK, a header compression mechanism detailed in RFC7541. HPACK reduces the size of HTTP headers by using a combination of static and dynamic header tables, which saves bandwidth and enhances performance, especially for repeated headers in subsequent requests. This capability ensures faster and more efficient data transmission, essential for modern web applications.

Error Handling and Custom Error Pages

In order to signify success, client errors, or server problems, the server shall provide comprehensive error handling and deliver the necessary HTTP status codes. For example, it will deliver codes such as 500 for "Internal Server Error" or 404 for "Not Found" when necessary. Additionally, it will enable a branded and user-friendly method of handling mistakes by enabling the serving of personalized error pages. These pages can provide more context about the problem and guide users on what to do next, making the server more polished and user centric.