Bilal Yağız Gündeğer

Websocket Performance Comparison – The Case

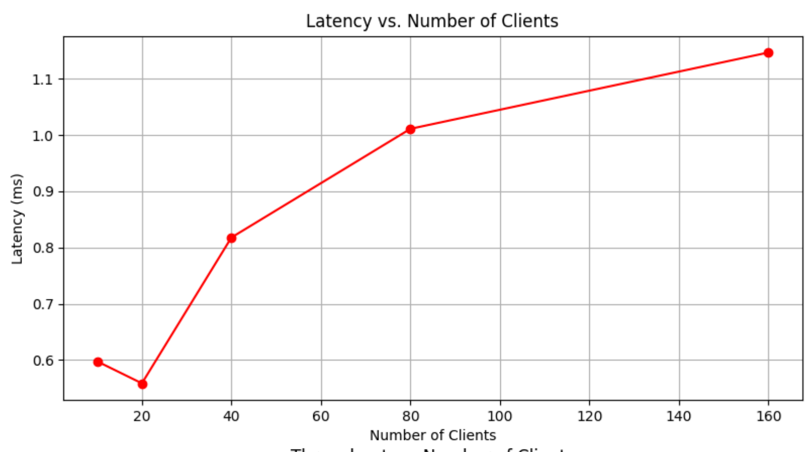
In this case study, our objective is to investigate and select two distinct Python libraries, focusing on their capability to manage fast-changing, real-time financial data. Therefore, the libraries chosen must demonstrate low latency and high throughput, especially under conditions of high demand. For this purpose, I have specifically chosen two libraries: **WebSockets** and the **Socket.IO** WebSocket Python library. I assessed their performance under various payload conditions, including alterations to the server's message emission rate and the number of connected clients. Such adjustments are crucial to mimic real-world scenarios, where handling real-time financial data often presents similar challenges.

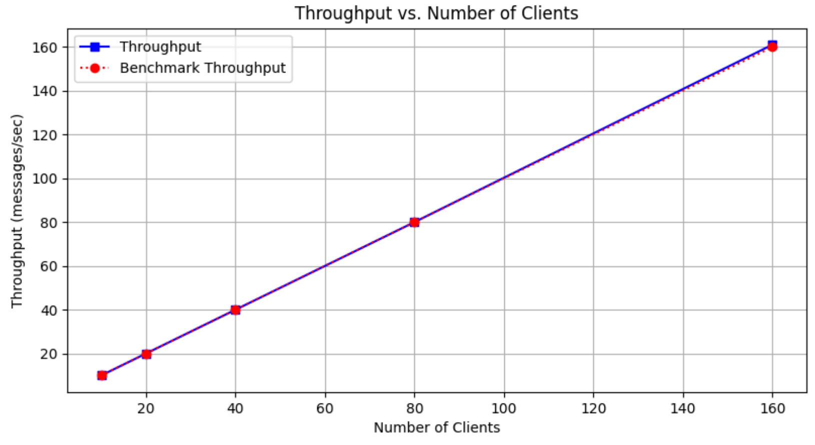
The libraries were tested with the following range of values:

- The frequency of data transmission was experimented with at [1, 0.5, 0.1] seconds per instance by the server.
- The number of connected clients varied among [10, 20, 40, 80, 160].

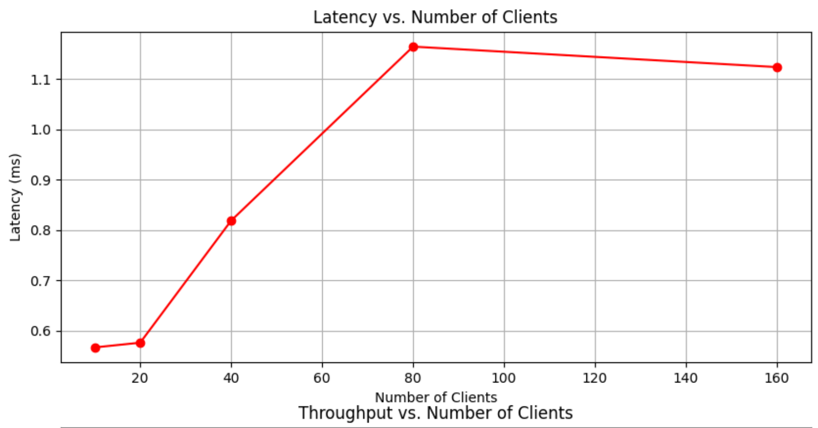
Websockets Library plots with respect to these parameters:

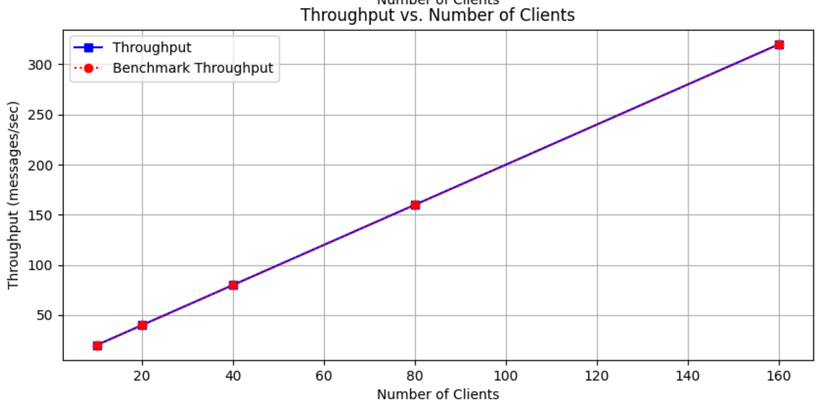
Data transmission frequency: once every second



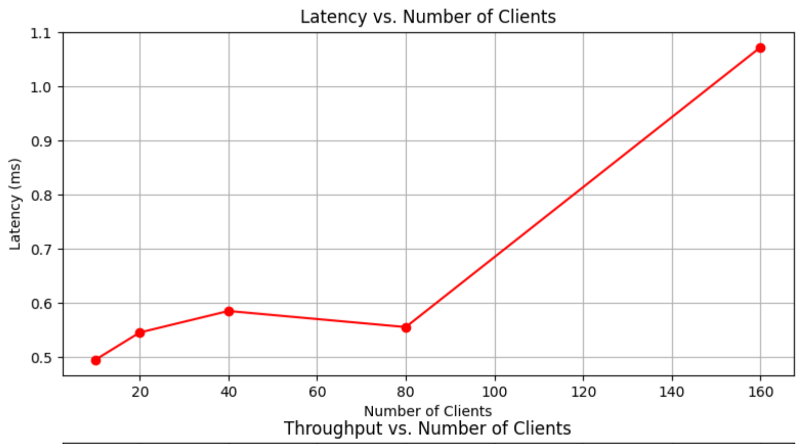


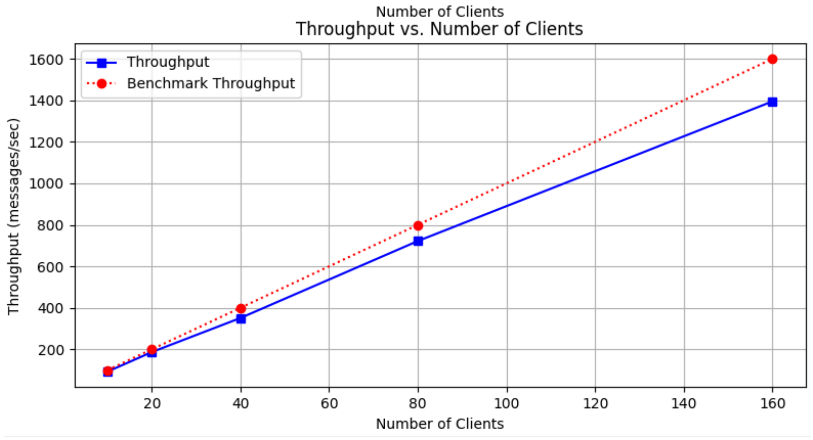
Data transmission frequency: once every 0.5 seconds.





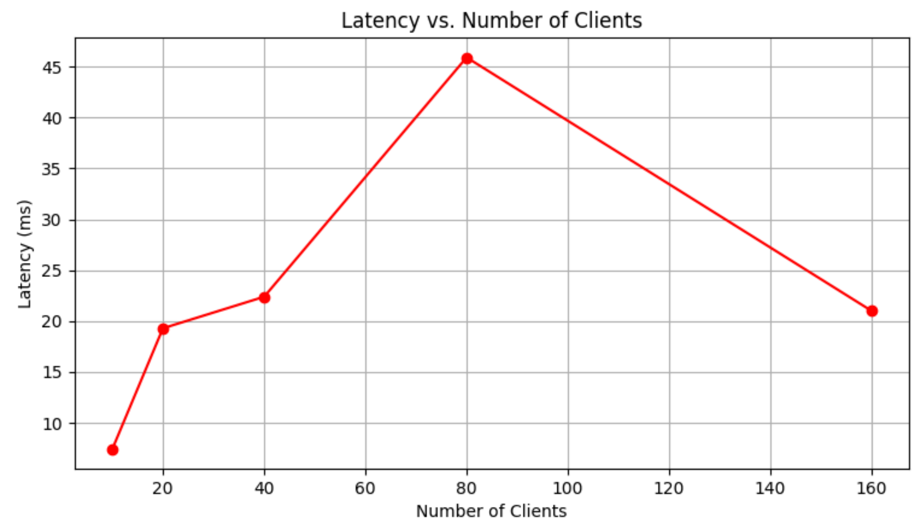
Data transmission frequency: once every 0.1 seconds.

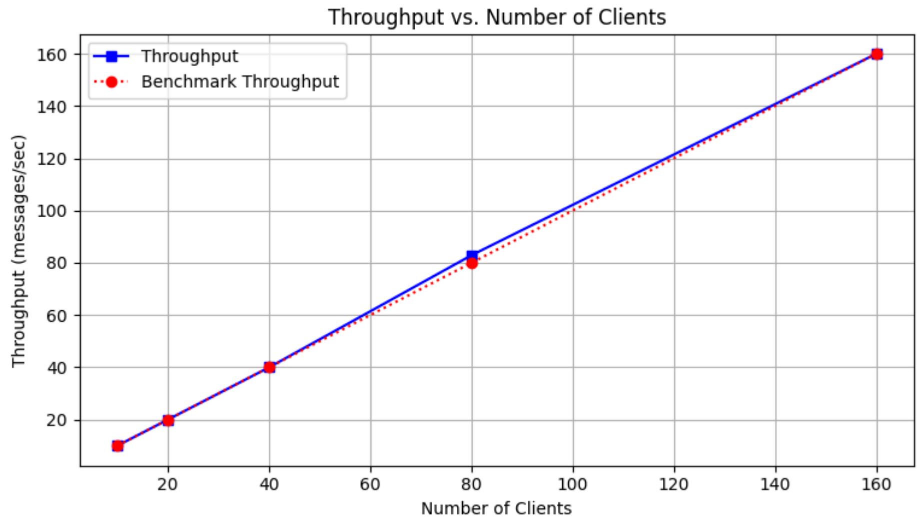




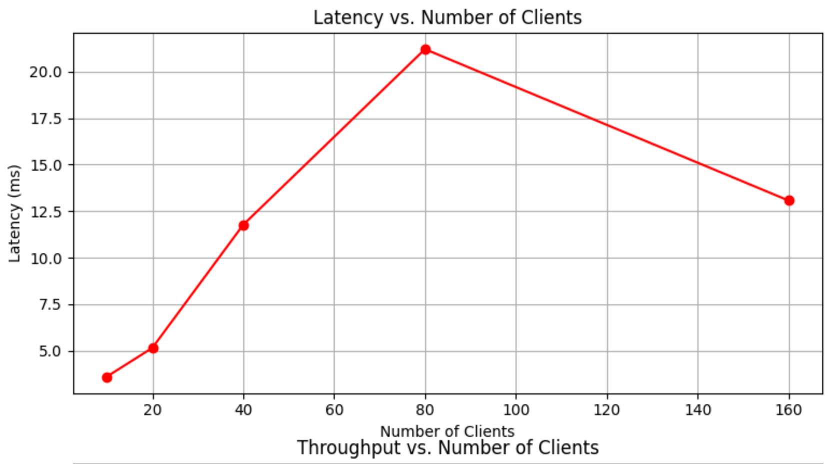
Socket.io Library plots with respect to these parameters:

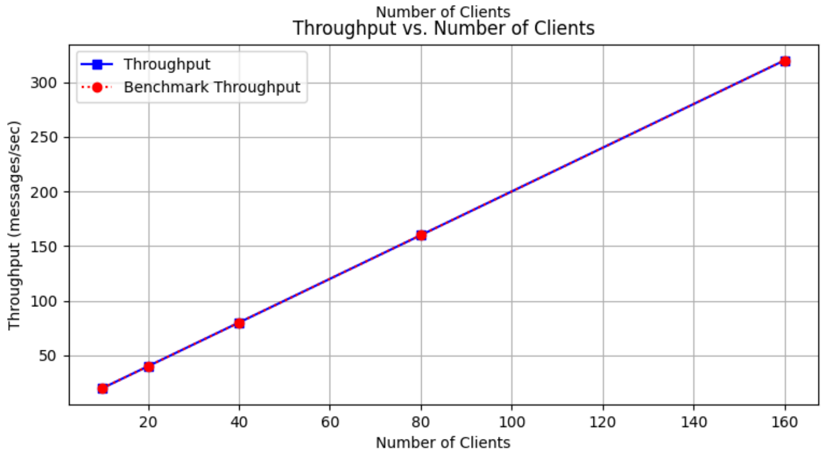
Data transmission frequency: once every second



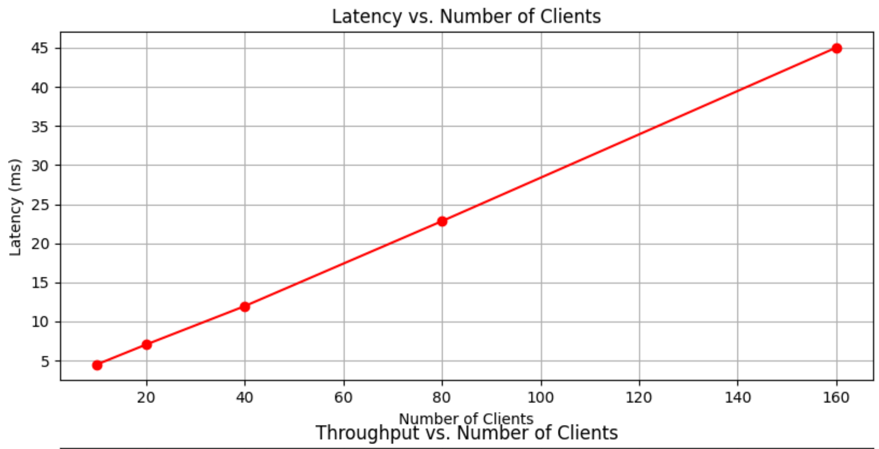


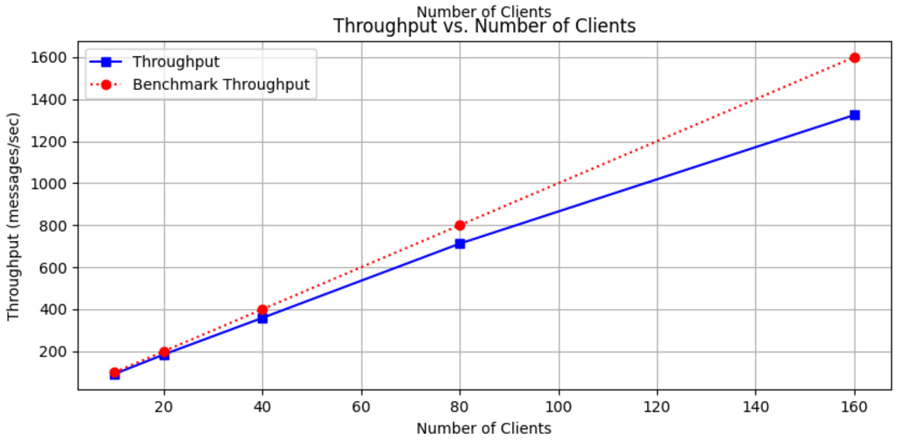
Data transmission frequency: once every 0.5 second





Sending frequency 0.1 second:





Test Environment Specifications:

Machine	CPU	MEMORY	OS
Workstation	**i5-9300H CPU @ 2.40GHz**	**8 GB of DDR4 2400 MHz**	**Windows 10**

Key Findings from Comparing These Results

Latency:

When comparing latency values under various numbers of connected clients, the **WebSockets** library outperforms the **Socket.IO** library. This is likely because **Socket.IO** adds additional functionalities, such as handshake (**Socket.IO** always initiates with an HTTP connection) and multiplexing (managing multiple virtual connections could slightly increase the processing time for delivering individual messages) [1].

Therefore, for the fast delivery of messages, the **WebSockets** library is more suitable for our case.

Throughput:

When comparing throughput under various numbers of connected clients, the **WebSockets** library outperforms the **Socket.IO** library. In benchmarking throughput, which indicates the number of messages that should be received if all sent messages are taken by the clients, increasing the number of clients, as expected, leads to an increase in lost messages. However, for **Socket.IO**, the difference in throughput from the benchmark increases. Thus, the WebSockets library is superior in terms of throughput comparison.

Scalability:

Both libraries exhibit scalability, albeit in distinct manners. Socket.IO offers a wider range of built-in functionalities, facilitating easier scalability compared to the WebSockets library. Research indicates that a single server can manage up to 10,000 concurrent client connections with Socket.IO[2]. In contrast, utilizing the WebSockets library requires careful consideration of various aspects, such as maintaining session continuity between clients and the server, among others[1]. The WebSockets library demands a more hands-on approach to address these challenges, providing a tailored solution for sophisticated needs.

References:

[1]https://ably.com/topic/socketio-vs-websocket#:~:text=Performance%20of%20Socket.IO%20vs%20WebSocket&text=However%2C%20lower%20performance%20from%20Socket,in%20exchange%20for%20additional%20functionality.

[2]https://ably.com/topic/scaling-socketio#socket-io-performance-review