Communication using Web Socket

1. Introduction

This report documents the using Web Socket as a communication protocol to enable reliable, bi-directional communication between a client and a server. The primary objective of this testing is to verify that the system can handle high message volumes without any loss, ensuring data integrity and robustness in real-time communication scenarios.

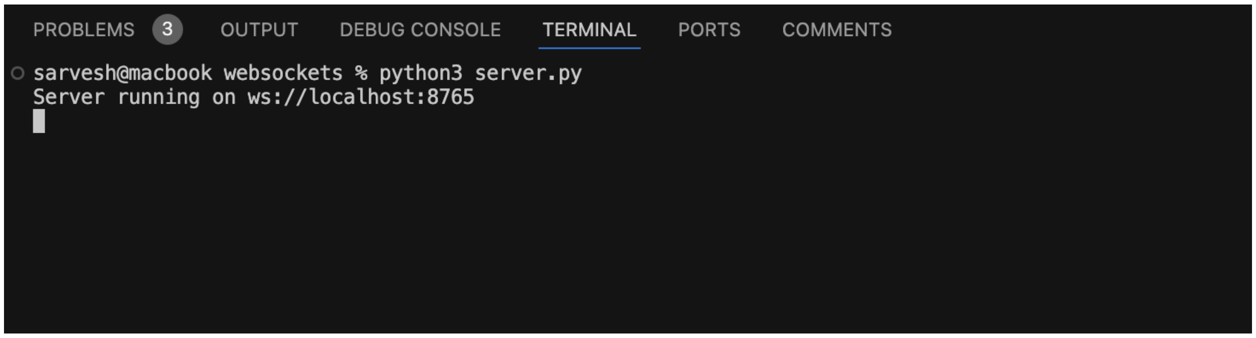
2. Setup and Configuration

The Web Socket application consists of two main components:

- **Server**: Listens for incoming messages, logs each message received, and responds to the client.
- **Client**: A web-based interface that allows users to send individual messages or batches of 10,000 messages.

Server Configuration

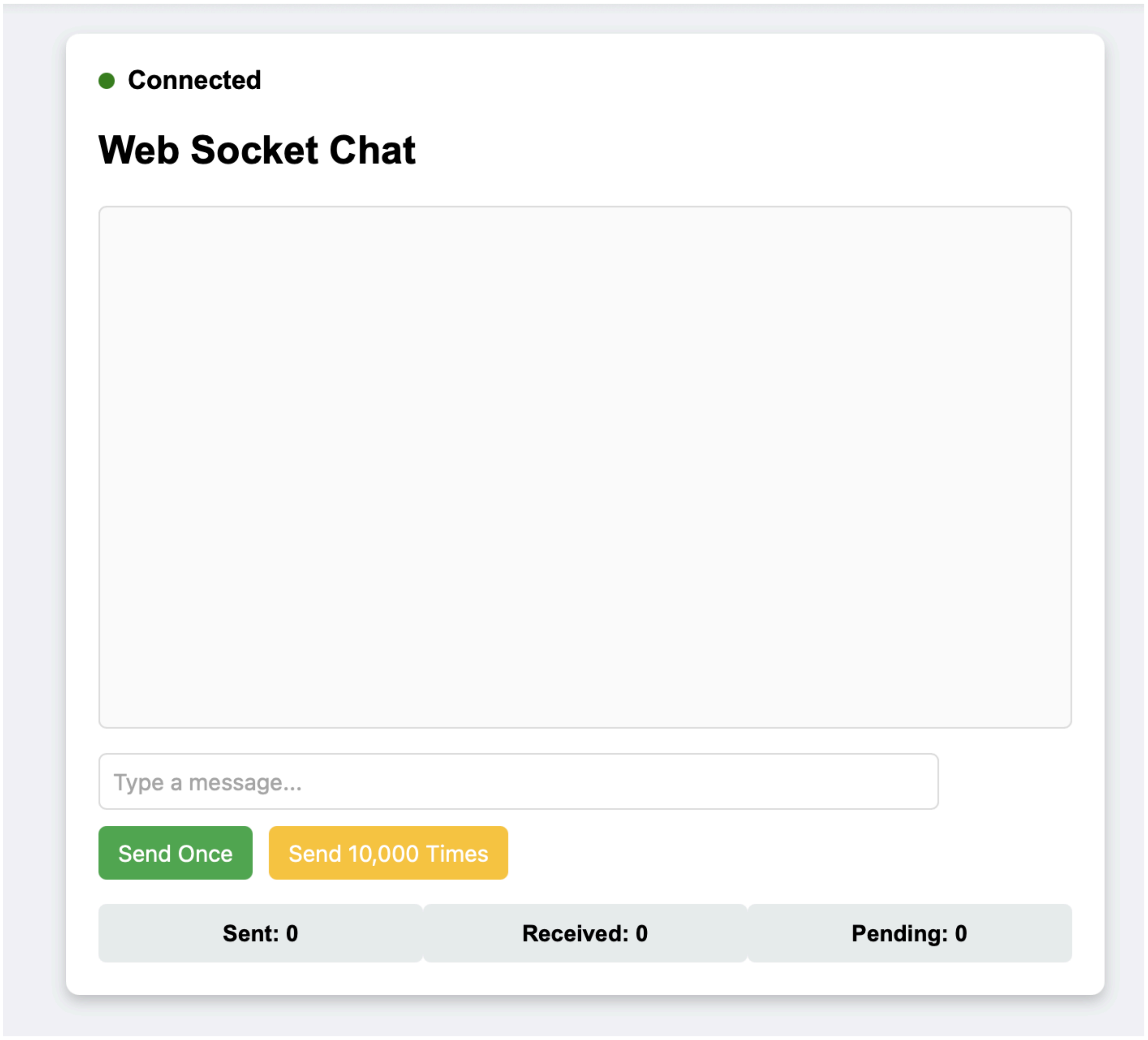
The server was set up to run on `ws://localhost:8765`. It logs each message in real-time to a file (`server.log`) to verify receipt and respond with a message appended with a random number.



```
sarvesh@macbook websockets % python3 server.py
Server running on ws://localhost:8765
```

Client Configuration

The client interface displays a dashboard showing the number of messages sent, received, and any pending acknowledgments. Users can send a single message or 10,000 messages at once.



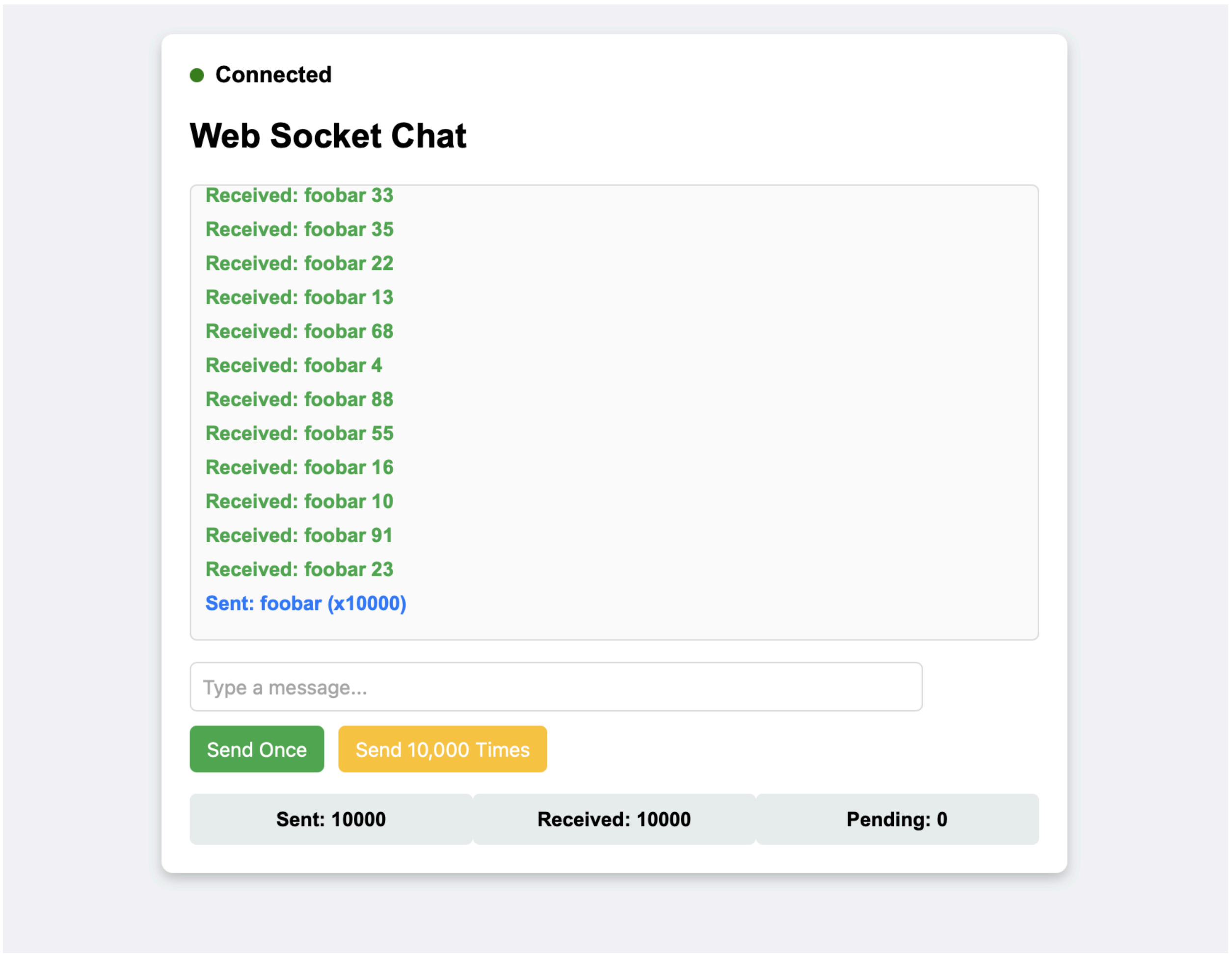
Screenshot showing the initial setup with the client interface and server connection confirmation

3. Testing Procedure

To evaluate the WebSocket application's robustness and reliability, two key scenarios were tested:

1. **Single Message**: A single message was sent from the client to the server to confirm basic connectivity and functionality.
2. **Batch of Messages:** A batch of 10,000 messages was sent to evaluate the system's capacity for high-frequency data without any loss.

The client interface provided real-time statistics on message transmission and receipt, while the server logged each message to `server.log`.



```
Received: foobar 33
Received: foobar 35
Received: foobar 22
Received: foobar 13
Received: foobar 68
Received: foobar 4
Received: foobar 88
Received: foobar 55
Received: foobar 16
Received: foobar 10
Received: foobar 91
Received: foobar 23
Sent: foobar (x10000)
```

Sent: 10000	Received: 10000	Pending: 0

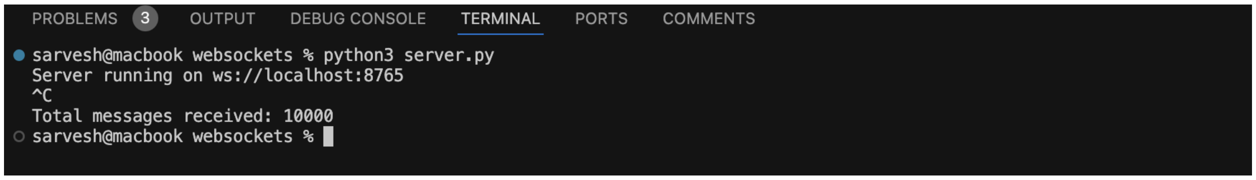
Screenshot showing the client statistics after sending 10,000 messages, with `Sent: 10000`, `Received: 10000`, and `Pending: 0`

4. Results

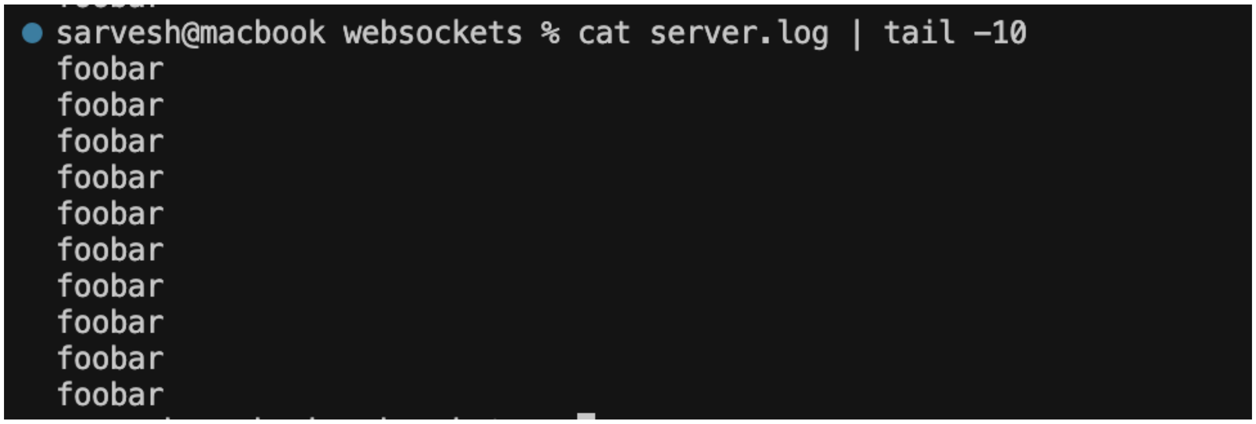
The test results confirmed the WebSocket application's ability to handle large volumes of messages without any loss, demonstrating both reliability and integrity in data transmission.

Key Findings

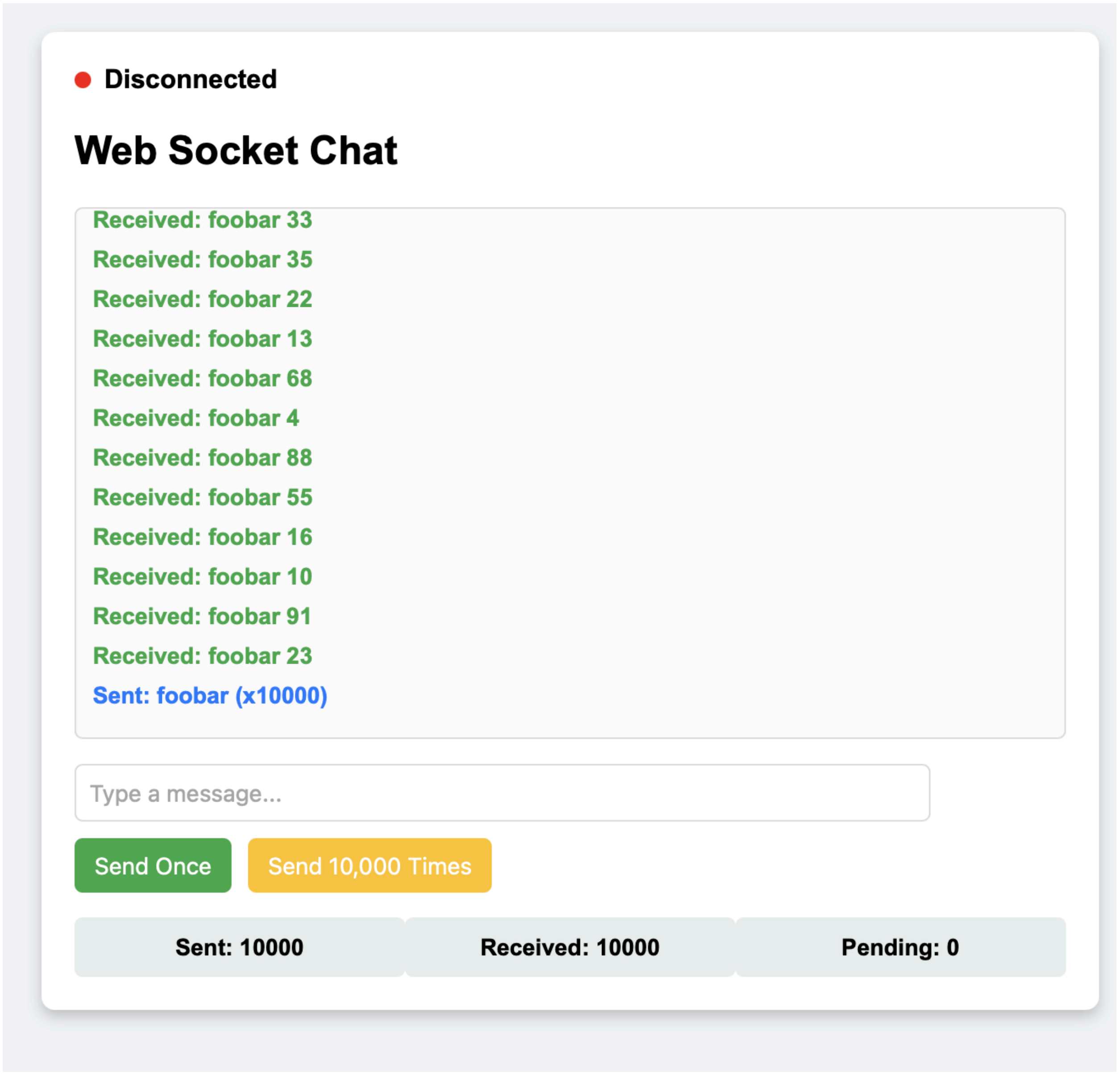
- **Message Integrity**: All 10,000 messages sent from the client were received by the server without any loss.
- **Accurate Logging**: The server's log file contained exactly 10,000 entries, matching the total sent messages.
- **Real-Time Statistics**: The client interface accurately displayed `Sent`, `Received`, and `Pending` counts, confirming that all messages were processed and acknowledged.



```
sarvesh@macbook websockets % python3 server.py
Server running on ws://localhost:8765
^C
Total messages received: 10000
sarvesh@macbook websockets % 
```



```
sarvesh@macbook websockets % cat server.log | tail -10
foobar
foobar
foobar
foobar
foobar
foobar
foobar
foobar
foobar
foobar
```



```
Received: foobar 33
Received: foobar 35
Received: foobar 22
Received: foobar 13
Received: foobar 68
Received: foobar 4
Received: foobar 88
Received: foobar 55
Received: foobar 16
Received: foobar 10
Received: foobar 91
Received: foobar 23
Sent: foobar (x10000)
```

Sent: 10000 | Received: 10000 | Pending: 0

---

5. Conclusion

The WebSocket application has demonstrated a high level of robustness, managing high message throughput effectively without any message loss. The successful handling of 10,000 messages highlights the reliability of WebSocket for real-time, bidirectional communication, making it suitable for applications where data integrity is essential.

This testing validates that the application can sustain large message volumes, making it well-suited for real-time communication needs, such as live chat or data streaming applications.