Raw WebSocket Implementation (RFC 6455)

A from-scratch implementation of the WebSocket protocol without using any WebSocket libraries. This project demonstrates the low-level details of WebSocket handshake, frame parsing, and bidirectional communication.

📋 Table of Contents

🎯 Overview

This project implements the WebSocket protocol (RFC 6455) from scratch using Node.js's raw TCP sockets. It includes:

- A **WebSocket server** that handles the upgrade handshake and frame parsing
- An **HTML/JavaScript client** using the browser's native WebSocket API
- Complete frame encoding/decoding for text messages
- Support for different payload lengths (7-bit, 16-bit, 64-bit)
- Message masking/unmasking as per protocol specification

✨ Features

Server (`server.mjs`)

- ✅ HTTP to WebSocket upgrade handshake
- ✅ SHA-1 hash computation for handshake acceptance
- ✅ Frame parsing with opcode detection
- ✅ Payload length detection (7-bit, 16-bit, 64-bit)
- ✅ XOR unmasking of client messages
- ✅ Frame construction for server responses
- ✅ Echo functionality
- ✅ Ping/Pong support
- ✅ Graceful error handling
- ✅ Connection state management

Client (`index.html`)

- ✅ Modern, responsive UI

- ✅ Real-time message display
- ✅ Connection status indicator
- ✅ Message input with keyboard shortcuts
- ✅ Timestamped message log
- ✅ Automatic reconnection support
- ✅ Color-coded message types

📡 WebSocket Protocol Basics

The Handshake

The WebSocket protocol begins with an HTTP upgrade request:

Client Request:

```
GET / HTTP/1.1
Host: localhost:1338
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

Server Response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

The `Sec-WebSocket-Accept` is computed as:

```
Base64(SHA-1(Sec-WebSocket-Key + "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"))
```

Message Framing

After the handshake, data is exchanged in frames. Each frame has:

- **FIN bit**: Indicates if this is the final fragment
- **Opcode**: Type of frame (text, binary, close, ping, pong)
- **MASK bit**: Set to 1 for client-to-server messages
- **Payload length**: 7, 16, or 64 bits depending on size
- **Masking key**: 4 bytes used to XOR the payload (client→server only)
- **Payload data**: The actual message content

📁 Project Structure

```
nodejs-raw-websocket/
├── server.mjs          # WebSocket server implementation
├── index.html          # WebSocket client with UI
├── README.md           # This file
└── package.json        # Project metadata
```

🚀 Installation

Prerequisites

- Node.js (v14 or higher)
- A modern web browser (Chrome, Firefox, Edge, Safari)

Setup

1. Clone or download this project:

```
cd nodejs-raw-websocket
```

2. No dependencies to install! This is a pure Node.js implementation.

💻 Usage

Start the Server

```
node server.mjs
```

You should see:

```
🚀 WebSocket Server is listening on port 1338
📡 Connect via: ws://localhost:1338
```

Open the Client

Open `index.html` in your web browser:

```
# On Linux
xdg-open index.html

# On macOS
open index.html
```

```
# On Windows
start index.html
```

Or simply drag and drop `index.html` into your browser.

Interact

1. The client automatically connects to `ws://localhost:1338`
2. Type a message in the input field
3. Click "Send" or press Enter
4. Watch the server echo your message back!

🔧 Implementation Details

Server Architecture

1. HTTP Server Setup

```
const server = createServer((request, response) => {
  response.writeHead(200);
  response.end("Hello, World!");
}).listen(PORT);
```

2. Upgrade Handler

```
server.on("upgrade", onSocketUpgrade);
```

Intercepts HTTP upgrade requests and performs WebSocket handshake.

3. Frame Parser

```
function onSocketreadable(socket) {
  // Read frame header
  const [firstByte] = socket.read(1);
  const [secondByte] = socket.read(1);

  // Extract FIN, opcode, MASK, and payload length
  const isFinalFrame = Boolean(firstByte & 0x80);
  const opcode = firstByte & 0x0f;
  const isMasked = Boolean(secondByte & 0x80);

  // Read payload based on length indicator
  // Unmask data using XOR with masking key
  // Process the message
}
```

4. Frame Constructor

```
function sendMessage(message, socket) {
  // Create frame header
  // Encode payload length
  // Send header + payload
}
```

Client Architecture

1. WebSocket Connection

```
const socket = new WebSocket("ws://localhost:1338");
```

2. Event Handlers

- `onopen`: Connection established
- `onmessage`: Data received from server
- `onerror`: Error occurred
- `onclose`: Connection closed

3. Sending Messages

```
socket.send(message); // Browser handles framing automatically
```

📊 Frame Structure

Client to Server Frame

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-------+-+-------------+-------------------------------+
|F|R|R|R| opcode|M| Payload len |    Extended payload length    |
|I|S|S|S|  (4)  |A|     (7)     |             (16/64)           |
|N|V|V|V|       |S|             |   (if payload len==126/127)   |
| |1|2|3|       |K|             |                               |
+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +
|     Extended payload length continued, if payload len == 127  |
+ - - - - - - - - - - - - - - - +-------------------------------+
|                               | Masking-key, if MASK set to 1 |
+-------------------------------+-------------------------------+
| Masking-key (continued)       |          Payload Data         |
```

```
+------------------------------ - - - - - - - - - - - - - - - +
:                     Payload Data continued ...               :
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                     Payload Data continued ...               |
+---------------------------------------------------------------+
```

Opcodes

Opcode	Meaning	Description
0x0	Continuation	Continuation of fragmented message
0x1	Text	UTF-8 text message
0x2	Binary	Binary data
0x8	Close	Connection close
0x9	Ping	Heartbeat from sender
0xA	Pong	Heartbeat response

Payload Length

Value	Meaning
0-125	Actual payload length
126	Next 2 bytes contain length
127	Next 8 bytes contain length

Masking Algorithm

Client messages must be masked using XOR operation:

```
for (let i = 0; i < payload.length; i++) {
  decoded[i] = encoded[i] ^ maskKey[i % 4];
}
```

The masking key is 4 random bytes generated by the client.

🧪 Testing

Manual Testing

1. **Connection Test**
    - Start server
    - Open client
    - Verify "Connected" status appears

2. **Message Echo Test**
    - Type "Hello WebSocket"
    - Click Send
    - Verify you receive "Echo: Hello WebSocket"
3. **Long Message Test**
    - Send a message > 125 characters
    - Verify it's handled correctly with 16-bit length
4. **Rapid Fire Test**
    - Send multiple messages quickly
    - Verify all are echoed back

Using WebSocket Command-Line Tools

wscat (Node.js)

```
npm install -g wscat
wscat -c ws://localhost:1338
> Hello from wscat!
< Echo: Hello from wscat!
```

websocat (Rust)

```
websocat ws://localhost:1338
Hello from websocat!
Echo: Hello from websocat!
```

Using Browser DevTools

1. Open browser DevTools (F12)
2. Go to Network tab
3. Filter by "WS" (WebSocket)
4. Click on the connection
5. View frames being sent/received

📚 Resources

Official Specifications

- RFC 6455 - The WebSocket Protocol
- MDN WebSocket API

Useful Tools

- WebSocket King - Online WebSocket tester
- WebSocket.org Echo Test
- Chrome DevTools Protocol

Frame Visualization

- WebSocket Frame Parser

🎓 Learning Points

This implementation teaches:

1. **Binary Protocol Handling**
    - Reading/writing bytes directly
    - Bit manipulation (masks, shifts, XOR)
    - Endianness (Big-endian for WebSocket)
2. **Network Programming**
    - TCP socket handling
    - HTTP upgrade mechanism
    - Stateful connection management
3. **Cryptography Basics**
    - SHA-1 hashing
    - Base64 encoding
    - XOR masking
4. **Protocol Design**
    - Handshake mechanisms
    - Frame-based communication
    - Length prefixing

🐛 Debugging Tips

Server Not Starting

```
# Check if port is in use
lsof -i :1338
netstat -an | grep 1338

# Kill the process
kill -9 <PID>
```

Connection Refused

- Ensure server is running

- Check firewall settings
- Verify correct port number

Messages Not Received

- Check browser console for errors
- Verify frame parsing in server logs
- Use DevTools to inspect WebSocket frames

🔒 Security Considerations

This is an **educational implementation**. For production use:

- ⚠️ Add TLS/SSL support (WSS protocol)
- ⚠️ Implement proper authentication
- ⚠️ Add rate limiting
- ⚠️ Validate message size limits
- ⚠️ Handle malformed frames gracefully
- ⚠️ Implement connection timeouts
- ⚠️ Add CORS handling
- ⚠️ Sanitize user input

📝 License

This project is provided for educational purposes. Feel free to use and modify as needed.

🙏 Acknowledgments

- RFC 6455 specification authors
- Node.js and browser WebSocket implementations
- The open-source community

Happy WebSocket coding! 🚀

For questions or improvements, feel free to open an issue or submit a pull request.