



Deep-Dive: WebSocket Performance Testing using Apache JMeter

Why WebSocket Testing Matters in Performance Engineering


WebSocket is a stateful, full-duplex communication protocol designed for real-time applications (e.g., stock trading, chat apps, multiplayer games). Unlike HTTP where each request-response is isolated, WebSocket establishes a persistent TCP connection, reducing overhead and improving latency.

In performance testing, simulating 1000s of clients maintaining long-lived socket connections with bi-directional data flow introduces **unique challenges** not seen in HTTP-based tests:

- **Memory pressure on clients and servers**
- **Thread & connection pool saturation**
- **Message ordering and latency jitter**
- **Concurrent message handling under burst load**

JMeter—when configured correctly with the right plugins—can be a powerful tool to emulate such load and validate system behavior under concurrency and message burst scenarios.

Plugin Setup: Peter Doornbosch WebSocket Sampler

JMeter doesn't natively support WebSocket, so we rely on:  [WebSocket Samplers Plugin by Peter Doornbosch](#)

Installation Steps:

1. Download JAR:
[WebSocket Sampler Releases](#)
2. Copy the .jar to:
3. JMeter_HOME/lib/ext/
4. Restart JMeter.
5. Samplers will appear under:

6. Sampler > WebSocket open / write / read / ping / close
-

Realistic WebSocket Load Test Plan Design

Scenario: Simulating Chat Application

Components:

1. Thread Group
2. WebSocket Open Connection
3. WebSocket Request Sampler (Send Message)
4. WebSocket Response Reader
5. WebSocket Ping Sampler (optional heartbeat)
6. WebSocket Close

WebSocket Open Connection

Server: chat.example.com


Port: 443

Protocol: wss

Path: /ws/chat

Timeout: 30000ms

Connection ID: conn_\${__threadNum}

 Tip: Use \${__threadNum} to maintain unique connection IDs across users.

Examples of Common WebSocket Payloads

JSON Chat Message (Text Frame)

```
{  
  "action": "sendMessage",  
  "channel": "perf-room",
```

```
"user": "perfUser01",  
"message": "Load test message ${__RandomString(10,abcdef,)}"  
}
```

Use the above in:

WebSocket Request Sampler > Request Data > Text

◆ Binary Frame Use-Case (File transfer, gaming)

Ensure sampler is set to Binary and input is Base64 encoded.

Key Metrics to Observe

Metric	Description
Connection Time	Time taken to establish WebSocket session
Response Latency	Time taken to get response for a sent message
Message Throughput (TPS)	Messages sent/received per second
Open Connections	Simultaneously active WebSocket sessions
CPU/Memory Usage	Monitor client (JMeter) and server under long-lived sockets
GC Pauses (Client + Server)	Long-lived sessions may hold memory, causing GC pressure

Common Issues & Their Resolutions

Issue	Cause & Resolution
Connection timeout / handshake failure	- Wrong path or SSL mismatch- Add proper Origin and Sec-WebSocket-Protocol headers if required
Server closes connection after a few seconds	- Lack of ping/pong- Add WebSocket Ping Sampler every 10-20 seconds
"Message not received" or read sampler timeout	- Async nature; Use WebSocket Read Sampler with retry or custom wait logic

Server sends fragmented messages not received fully	- Increase read timeout; ensure fragment assembly is supported
JMeter crashes or stalls with many connections	- Heap exhaustion- Run JMeter in non-GUI, set JVM - Xmx4g or more- Use distributed testing
Out-of-Order messages or concurrency issues	- Add timestamp/correlation ID in messages- Validate ordering manually in assertions
SSL handshake errors on wss://	- Install server's CA cert in JMeter's JVM truststore or disable strict validation (not recommended)
WebSocket 'protocol error' on server logs	- Wrong subprotocol (e.g., JSON, STOMP); must match server's expectation

Diagnostic Strategies

1. Correlate Message Timings

Use:

- Timestamp + Correlation ID inside payloads.
- Regex Extractor to pull message ID from server response.

2. Enable Debug Samplers + Logs

- Add **Debug Sampler** and **JSR223 PostProcessor** to log connection status, headers, and messages.

3. Monitor JMeter Client Health

Use jconsole or VisualVM:

- Track heap size, thread count, GC time.
- Check for thread leaks (long-lived threads in BLOCKED state).

Scaling Strategy for WebSocket Load in JMeter

A. JMeter Distributed Mode

If testing 5k+ concurrent users:

- Master JMeter

- Multiple slaves (2 CPU, 4 GB RAM at least)
- Ensure plugin installed on all slaves
- Synchronize clocks (for message timing accuracy)
- Set unique connectionId per thread

```
jmeter-server -Jserver_port=1099
```

B. Headless CLI Execution

```
jmeter -n -t websocket_test.jmx -l websocket_results.jtl -e -o report/
```

Assertion Examples

Check for Keyword in Response

Response contains: "status":"delivered"

Add Response Assertion > Pattern to Match.

Custom JSON Path Assertion (Using JSR223)

```
import groovy.json.JsonSlurper
def body = prev.getResponseDataAsString()
def json = new JsonSlurper().parseText(body)
assert json.status == 'delivered'
```

Specialized Test Scenarios






Scenario	Strategy
Broadcast to 10k users	Open 10k connections, send 1 message, verify all users receive it
Message Burst Test (Chat Flooding)	Ramp up 500 users in 30s, each sends 5 messages/second
Network Instability Simulation	Drop connections mid-test (e.g., using tc in Linux)

Authentication Token Expiry + Reconnect	Handle via re-auth + re-open WebSocket after expiry
Message Ordering Test	Send seq # in payload, validate ordering via scripting/assertion

Security Considerations

- Use wss:// always for production-like tests.
- Set Origin and custom headers if your server checks them.
- Token-based headers (JWT, OAuth) should be passed via Sec-WebSocket-Protocol or custom headers (add via WebSocket Open sampler).

Final Takeaways

-  WebSocket testing is not fire-and-forget like HTTP—requires long session management.
-  Proper read/write sampler sequencing, timeout settings, and message tracking is critical.
-  Simulating user behavior over WebSocket often includes *idle time*, *heartbeat*, *chat bursts*, and *disconnect/reconnect logic*.
-  Correlation, message validation, and frame timing are key to validating business flow.
-  Use distributed execution for high concurrency, monitor JMeter client and server infra continuously.