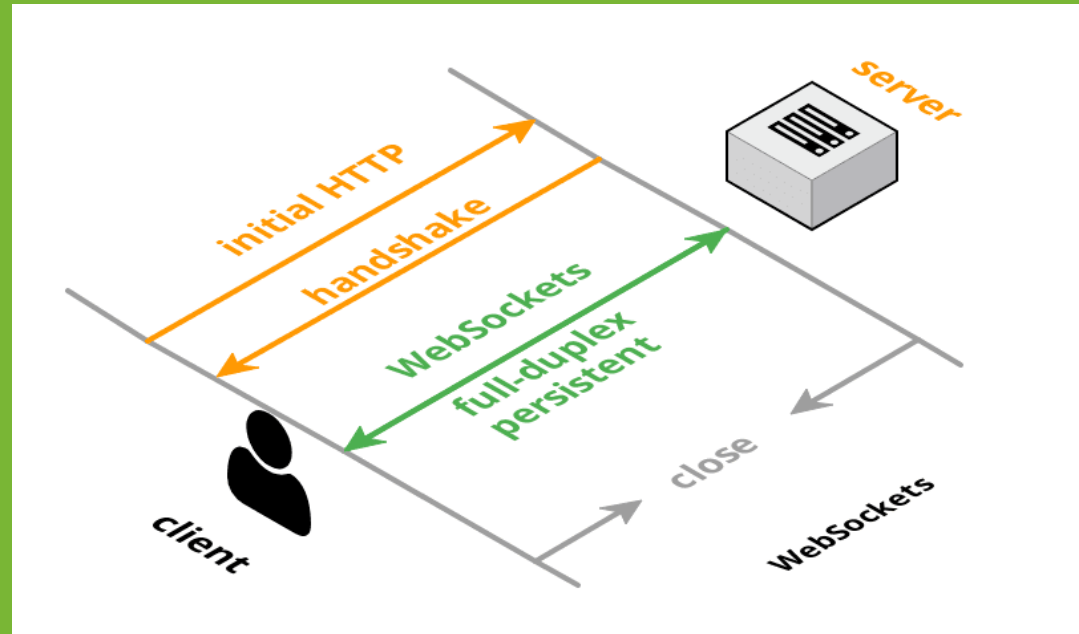


Web Sockets

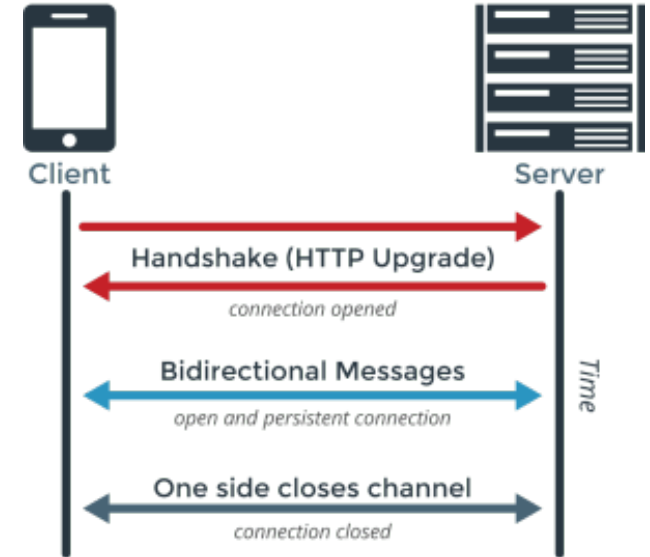
*Peer-to-peer
connections for
Real-time Web*



Feiner/Ulm/Schwab

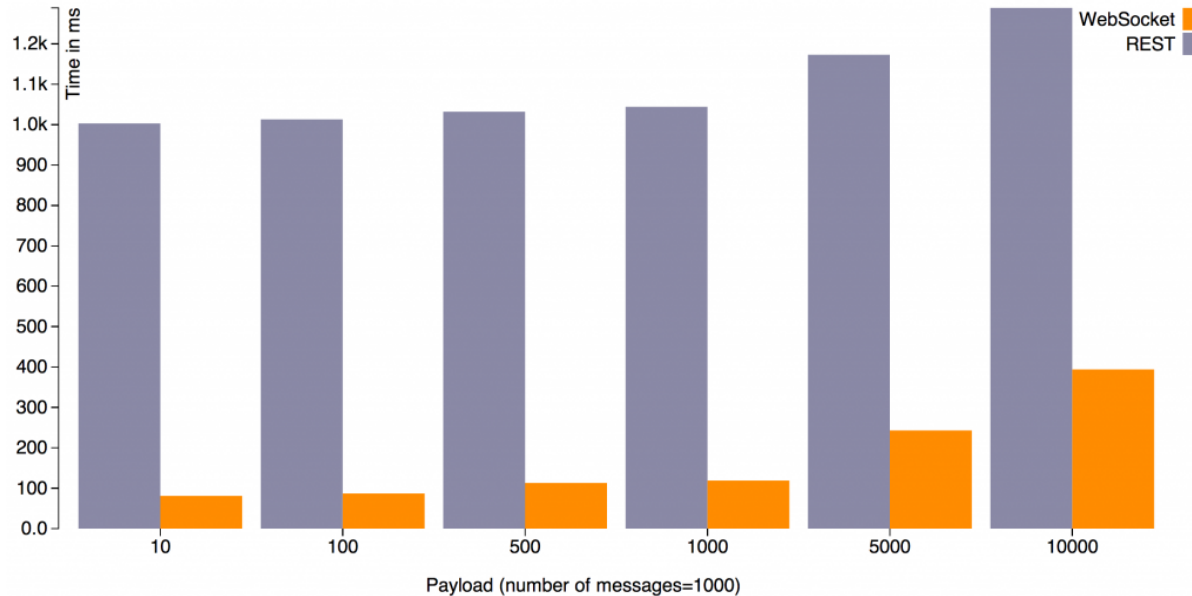
WebSocket for "Real-time Web"

- "Classic" HTTP is uni-directional with Request-Response
- WebSocket enables a **bi-directional communication** between WebApp and Server (compare WebRTC data channel)
- Mobile App initially connects to Server.
Reuse of existing connection, protocol upgrade.
- Server is able to "**push**" information to Client



<http://saurabh29july.blog/understanding-websockets-using-java-embedded-jetty-server-a-simple-html-javascript-client/>

Payload comparison REST vs. WebSocket



WebSocket readyState attribute

Identifies current connection state

(0) **CONNECTING**

Client establishes currently a connection to Server

(1) **OPEN**

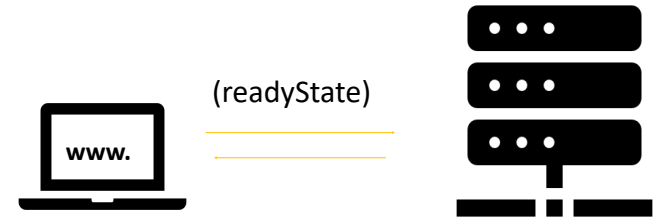
The Web Socket communication is open

(2) **CLOSING**

Current connection is currently closing

(3) **CLOSED**

The connection is closed.



WebSocket Events and Event Handlers

Each events will be bind with an Event Handler

open / onopen

Connection is established

message / onmessage

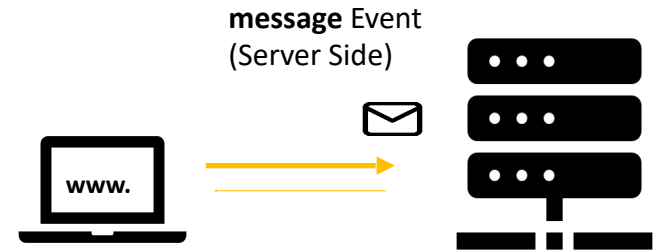
Client receives data from server

error / onerror

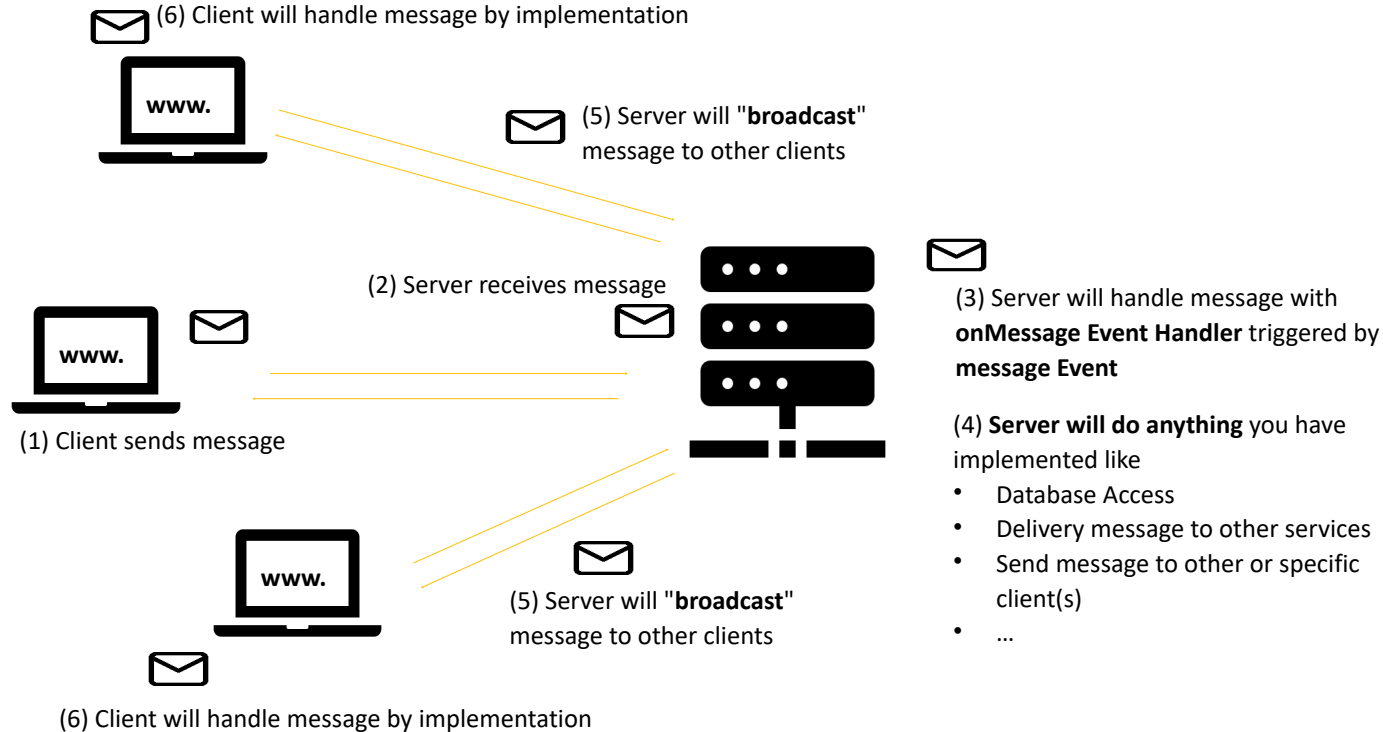
Occurs on any error in communication

close / onclose

Connection is closed



possible Infrastructure and Design



Handshake

- HTTP based
- Client computes **Key** and sends upgrade request to server
- Server calculates **Accept** key and sends back 101 response

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Sec-WebSocket-Accept>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/101>

<https://tools.ietf.org/html/rfc6454>

The screenshot displays the network tab of a web browser's developer tools, showing an HTTP 101 response. The status bar at the top indicates 'Status code: 101 Switching Protocols' and 'Version: HTTP/1.1'. The 'Response headers' section is expanded, showing 'Connection: Upgrade', 'Sec-WebSocket-Accept: WlvU68h1C7aJIWPM1OncNE3YEQ=' (highlighted with an orange box), and 'Upgrade: websocket'. The 'Request headers' section is also expanded, showing 'Accept: */*', 'Accept-Encoding: gzip, deflate', 'Accept-Language: de-AT,en;q=0.8,de;q=0.5,en-US;q=0.3', 'Cache-Control: no-cache', 'Connection: keep-alive, Upgrade', 'DNT: 1', 'Host: localhost:8081', 'Origin: http://localhost:8080' (highlighted with a purple box), 'Pragma: no-cache', 'Sec-WebSocket-Extensions: permessage-deflate', 'Sec-WebSocket-Key: ULrdri5tSMPWsY43Wu33g===' (highlighted with an orange box), 'Sec-WebSocket-Version: 13', and 'Upgrade: websocket'. The 'User-Agent' is 'Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/76.0'. Annotations on the right side of the image provide context: '101 response code' for the status code, 'key, computed by server' for the Sec-WebSocket-Accept header, 'Origin Header (SOP), RFC 6454' for the Origin header, and 'special key, computed by client' for the Sec-WebSocket-Key header.

Request URL: ws://localhost:8081/
Request method: GET
Remote address: [::1]:8081
Status code: 101 Switching Protocols 101 response code
Version: HTTP/1.1

Filter headers

Response headers (129 B)

Connection: Upgrade
Sec-WebSocket-Accept: WlvU68h1C7aJIWPM1OncNE3YEQ= key, computed by server
Upgrade: websocket

Request headers (484 B)

Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: de-AT,en;q=0.8,de;q=0.5,en-US;q=0.3
Cache-Control: no-cache
Connection: keep-alive, Upgrade
DNT: 1
Host: localhost:8081
Origin: http://localhost:8080 Origin Header (SOP), RFC 6454
Pragma: no-cache
Sec-WebSocket-Extensions: permessage-deflate
Sec-WebSocket-Key: ULrdri5tSMPWsY43Wu33g== special key, computed by client
Sec-WebSocket-Version: 13
Upgrade: websocket
User-Agent: Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/76.0

Message

- Event-Listener
 - `onmessage`
- Event-Object*
 - `message`
- Data**
 - `message.data`
- Typically serialised JSON
 - String
 - ArrayBuffer
 - Blob

```
const connection = new WebSocket('wss://example.com:1234');  
  
connection.onmessage = (message) => {  
  const msg = JSON.parse(message.data);  
  console.log(msg);  
};
```

Browser side example

- * Depends on WebSocket implementation!
- ** Depending on the implementation, the event object can already be the pure data stream

Message

```
connection.onmessage = (message) => {  
  console.log(message);  
};
```

```
connection.onmessage = (message) => {  
  const msg = JSON.parse(message.data);  
  console.log(msg);  
};
```

```
{  
  serverArray: (3) [-]  
    0: "hello"  
    1: "from"  
    2: "server"  
    length: 3  
    <prototype>: Array []  
    <prototype>: Object { - }
```

client.js:24:11

```
new message from server client.js:22:11  
message  
  bubbles: false  
  cancelBubble: false  
  cancelable: false  
  composed: false  
  currentTarget: null  
  data: "{\"serverArray\":[\"hello\",\"from\",\"server\"]}"  
  defaultPrevented: false  
  eventPhase: 0  
  explicitOriginalTarget: WebSocket { url: "ws://localhost:8081/", readyState: 1,  
    bufferedAmount: 0, - }  
  isTrusted: true  
  lastEventId: ""  
  origin: "ws://localhost:8081/"  
  originalTarget: WebSocket { url: "ws://localhost:8081/", readyState: 1,  
    bufferedAmount: 0, - }  
  ports: Array []  
  returnValue: true  
  source: null  
  srcElement: WebSocket { url: "ws://localhost:8081/", readyState: 1,  
    bufferedAmount: 0, - }  
  target: WebSocket { url: "ws://localhost:8081/", readyState: 1, bufferedAmount:  
    0, - }  
  timeStamp: 371  
  type: "message"  
  <get isTrusted(): function isTrusted()  
  <prototype>: MessageEventPrototype { initMessageEvent: initMessageEvent(), data:  
    Getter, origin: Getter, - }
```

<https://www.w3.org/TR/websockets/>

Selected references

[Docs] [txt|pdf] [draft-ietf-hybi...] [Tracker] [Diff1] [Diff2] [Errata]

Updated by: [7936](#), [8307](#), [8441](#)

PROPOSED STANDARD

Errata Exist

Internet Engineering Task Force (IETF)

I. Fette

Request for Comments: 6455

Google, Inc.

Category: Standards Track

A. Melnikov

ISSN: 2070-1721

Isode Ltd.

December 2011

The WebSocket Protocol

Abstract

The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the origin-based security model commonly used by web browsers. The protocol consists of an opening handshake followed by basic message framing, layered over TCP. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g., using XMLHttpRequest or <iframe>s and long polling).

<https://tools.ietf.org/html/rfc6455>