

# Java API for WebSocket

---

Arun Gupta · Red Hat · [@arungupta](https://twitter.com/arungupta)

**Arun Gupta**

- Director, Developer Advocacy, Red Hat Inc.
- O'Reilly and McGraw Hill author
- Fitness freak

## Agenda

- Primer on WebSocket
- **JSR 356: Java API for WebSocket 1.0**

# **Interactive Web Sites**

---

## Interactive Web Sites

- **HTTP is half-duplex**

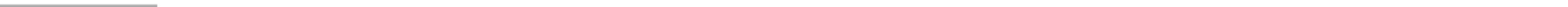
## Interactive Web Sites

- **HTTP is half-duplex**
- **HTTP is verbose**

## Interactive Web Sites

- HTTP is half-duplex
- HTTP is verbose
- Hacks for Server Push
  - Polling
  - Long Polling
  - Comet/Ajax
  - Complex, Inefficient, Wasteful

# WebSocket to the rescue

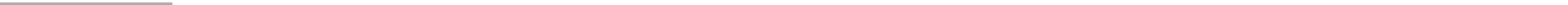


**WebSocket to the rescue**

- **TCP based, bi-directional, full-duplex messaging**

## WebSocket to the rescue

- TCP based, bi-directional, full-duplex messaging
- Originally proposed as part of HTML5



## WebSocket to the rescue

- TCP based, bi-directional, full-duplex messaging
- Originally proposed as part of HTML5
- IETF-defined Protocol: RFC 6455
  - Handshake
  - Data Transfer

## WebSocket to the rescue

- TCP based, bi-directional, full-duplex messaging
- Originally proposed as part of HTML5
- IETF-defined Protocol: RFC 6455
  - Handshake
  - Data Transfer
- W3C defined JavaScript API
  - Candidate Recommendation

**What's the basic idea ?**

---

**What's the basic idea ?**

- Upgrade HTTP to WebSocket (**single TCP connection**)
  - Transparent to proxies, firewalls, and routers

## What's the basic idea ?

- Upgrade HTTP to WebSocket (**single TCP connection**)
  - Transparent to proxies, firewalls, and routers
- Send data frames in both direction (**Bi-directional**)
  - No headers, cookies, authentication
  - No security overhead
  - “ping”/“pong” frames for keep-alive

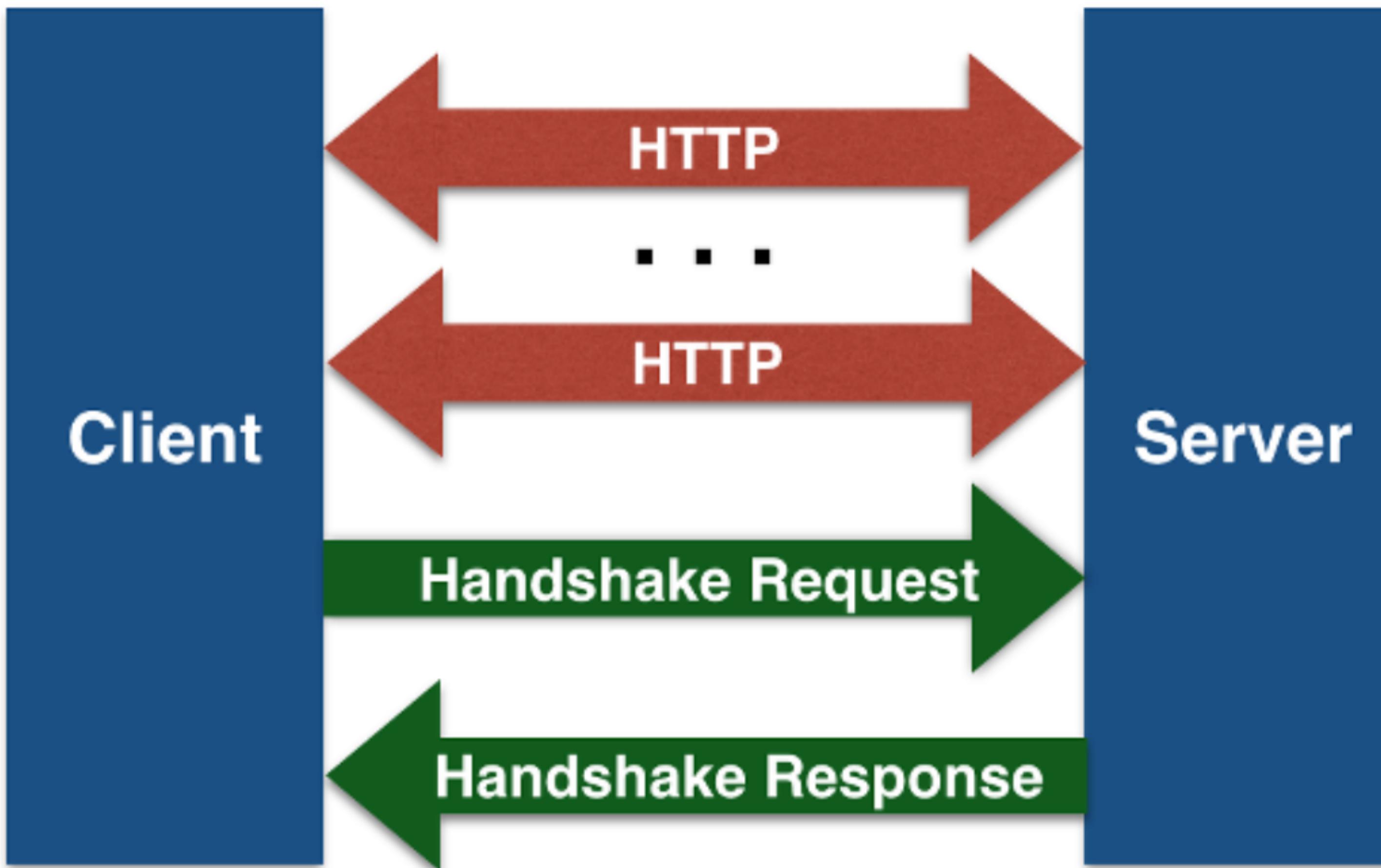
## What's the basic idea ?

- Upgrade HTTP to WebSocket (**single TCP connection**)
    - Transparent to proxies, firewalls, and routers
  - Send data frames in both direction (**Bi-directional**)
    - No headers, cookies, authentication
    - No security overhead
    - “ping”/“pong” frames for keep-alive
  - Send message independent of each other (**Full Duplex**)
-

## What's the basic idea ?

- Upgrade HTTP to WebSocket (**single TCP connection**)
    - Transparent to proxies, firewalls, and routers
  - Send data frames in both direction (**Bi-directional**)
    - No headers, cookies, authentication
    - No security overhead
    - “ping”/“pong” frames for keep-alive
  - Send message independent of each other (**Full Duplex**)
  - End the connection
-

## Establish a connection



# Handshake Request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket ①
Connection: Upgrade ②
Sec-WebSocket-Key: dGh1IHNhbXBsZSSub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

① HTTP upgrade header

② Upgrade to WebSocket

# Handshake Response

HTTP/1.1 101 Switching Protocols

Upgrade: websocket ①

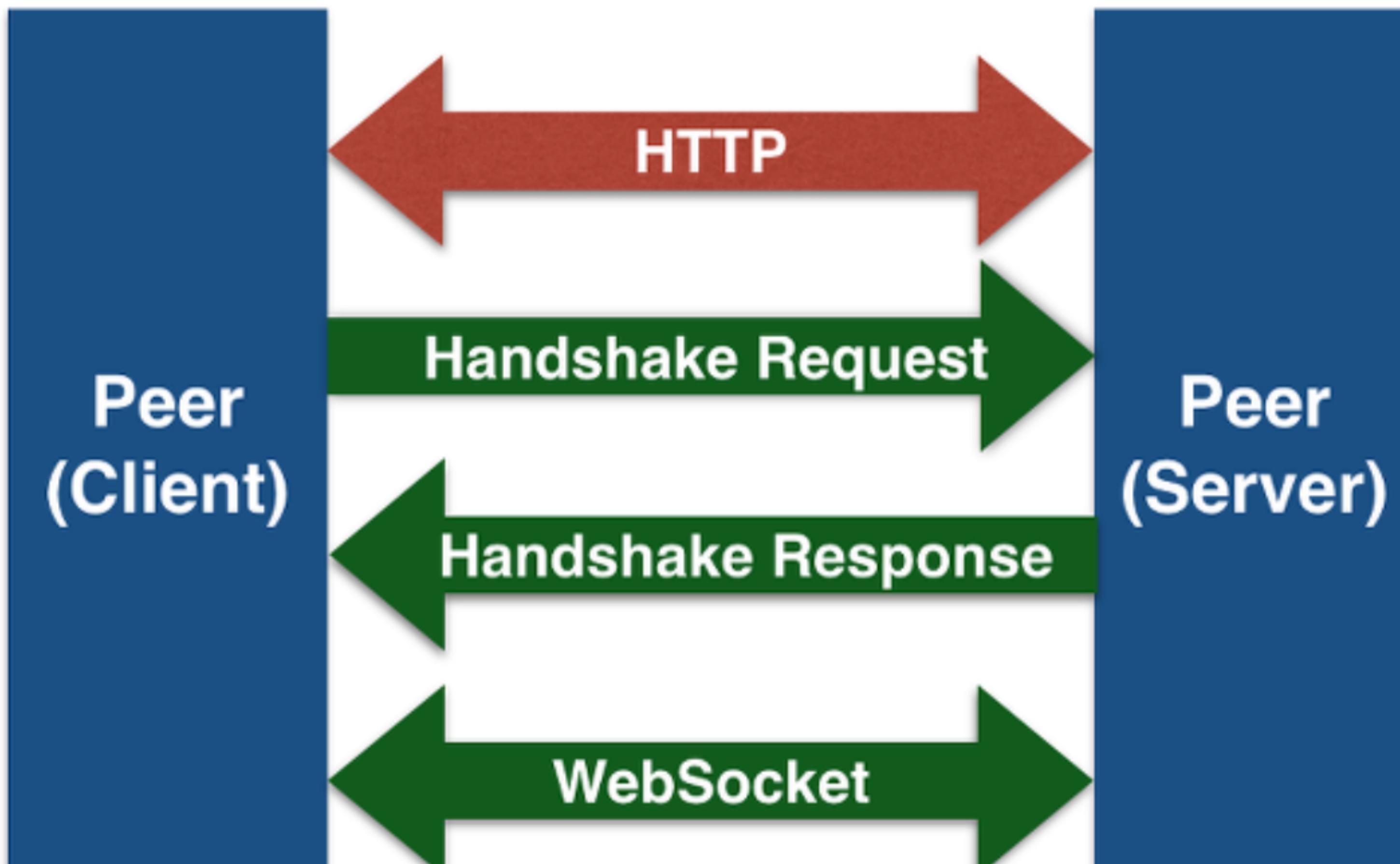
Connection: Upgrade ②

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzhZBk+xOo= ③

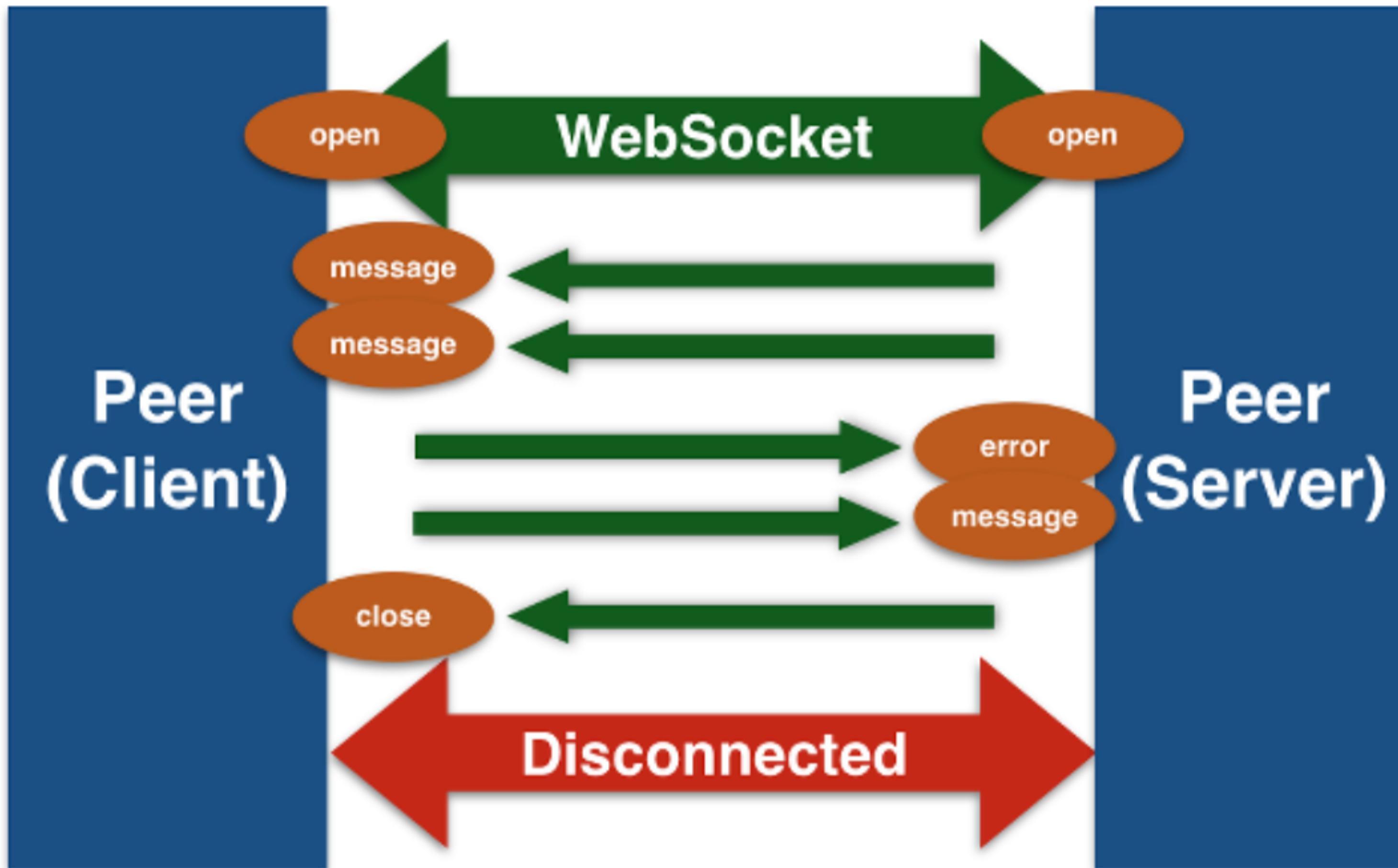
Sec-WebSocket-Protocol: chat

- ① Server echoes the header
- ② Server echoes the header, completes the handshake
- ③ Indicates server's acceptance of connection

**Connection established**



# WebSocket Lifecycle



# WebSocket API

```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
    readonly attribute DOMString url;

    // ready state
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSING = 2;
    const unsigned short CLOSED = 3;
    readonly attribute unsigned short readyState;
    readonly attribute unsigned long bufferedAmount;

    // networking
        attribute EventHandler onopen;
        attribute EventHandler onerror;
        attribute EventHandler onclose;
    readonly attribute DOMString extensions;
    readonly attribute DOMString protocol;
    void close([Clamp] optional unsigned short code, optional DOMString reason);

    // messaging
        attribute EventHandler onmessage;
        attribute DOMString binaryType;
    void send(DOMString data);
    void send(Blob data);
    void send(ArrayBuffer data);
    void send(ArrayBufferView data);
};
```

- <http://www.w3.org/TR/websockets/>

# WebSocket JavaScript support in browsers

## Can I use Web Sockets?

Compatibility table for support of Web Sockets in desktop and mobile browsers.

[View in interactive mode](#)

= Supported   = Not supported   = Partially supported   = Support unknown

### Web Sockets - Candidate Recommendation

Bidirectional communication technology for web apps

Global user stats \*:

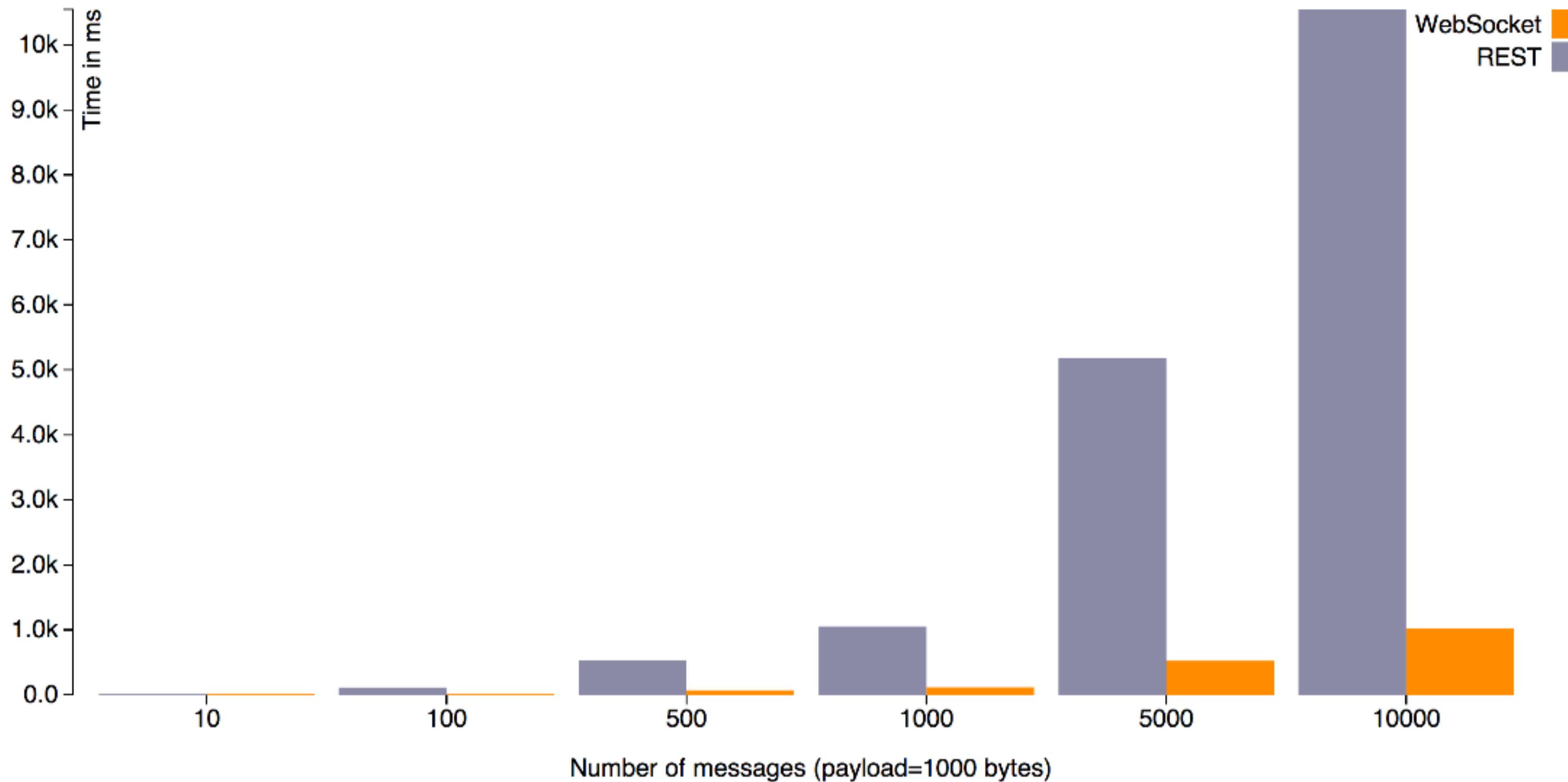
Support	70.61%
Partial support:	2.13%
Total:	72.74%

Resources: [WebPlatform Docs](#) [Wikipedia](#) [Details on newer protocol](#) [WebSockets Information](#) [has.js test](#)

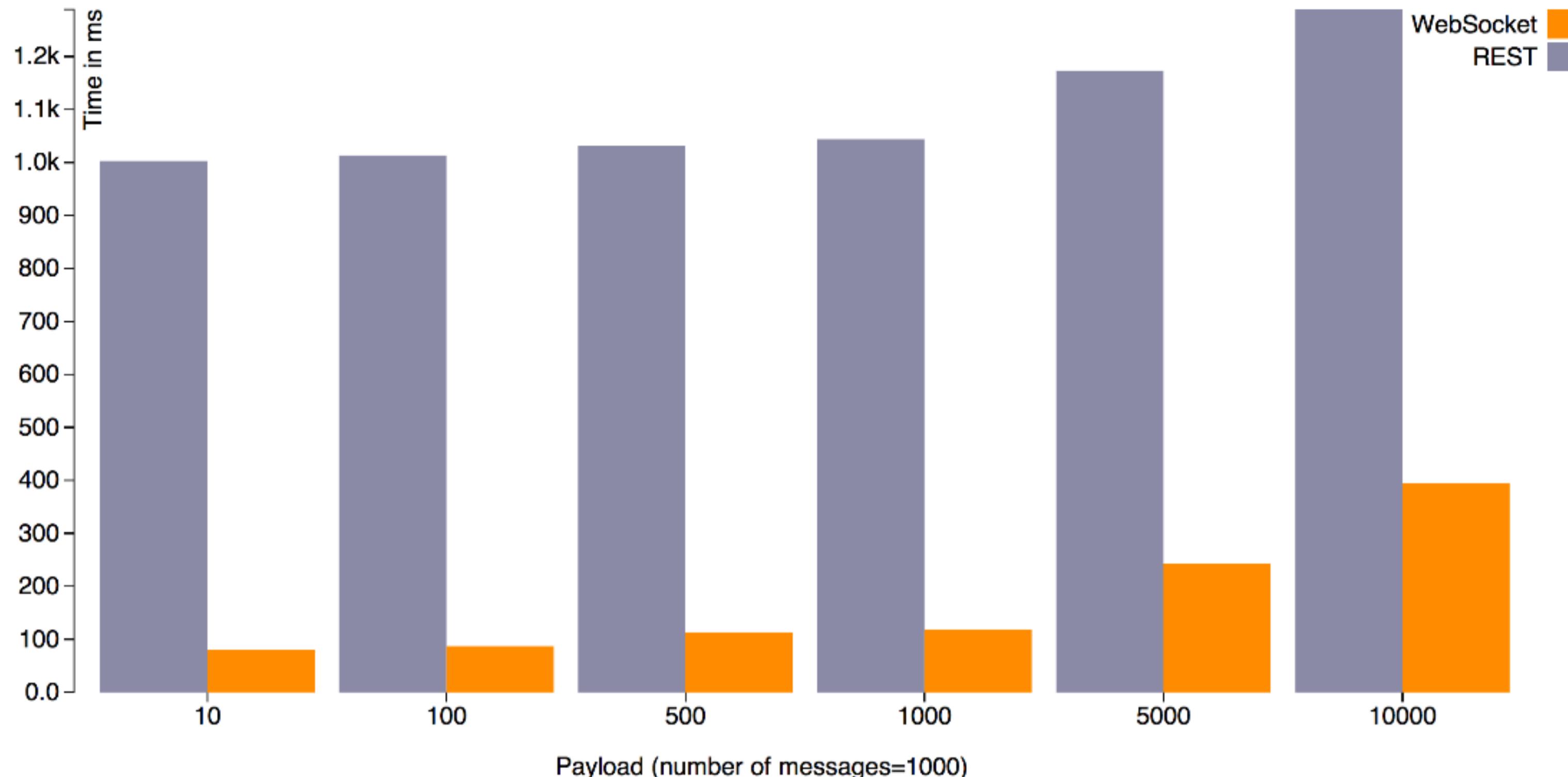
	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
28 versions back			4.0										
27 versions back		2.0	5.0										
26 versions back		3.0	6.0										
10 versions back		17.0	22.0		11.0								
9 versions back		18.0	23.0		11.1								
8 versions back		19.0	24.0		11.5								
7 versions back		20.0	25.0	3.1	11.6			2.1					
6 versions back	5.5	21.0	26.0	3.2	12.0			2.2		10.0			
5 versions back	6.0	22.0	27.0	4.0	12.1	3.2		2.3		11.0			
4 versions back	7.0	23.0	28.0	5.0	15.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	24.0	29.0	5.1	16.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	25.0	30.0	6.0	17.0	5.0-5.1		4.1		12.0			
Previous version	10.0	26.0	31.0	6.1	18.0	6.0-6.1	4.2-4.3	7.0	12.1				
Current	11.0	27.0	32.0	7.0	19.0	7.0	5.0-7.0	4.4	10.0	16.0	32.0	26.0	10.0
Near future		28.0	33.0		20.0								
Farther future		29.0	34.0		21.0								
3 versions ahead		30.0	35.0										

□ <http://caniuse.com/websockets>

# REST vs WebSocket



# REST vs WebSocket



- **Standard Java API for creating WebSocket Applications**
- **Transparent Expert Group**
  - <http://jcp.org/en/jsr/detail?id=356>
  - <http://java.net/projects/websocket-spec>
- **Included in Java EE 7**

## JSR 356 Implementations



**GlassFish 4 (RI), WildFly 8, Atmosphere,  
Tomcat 7.0.x/8.0.x, Jetty 9.1.x, Cauchon Resin**

# Java API for WebSocket Features

## Java API for WebSocket Features

- **API for WebSocket Server and Client Endpoint**
  - **Annotated:** `@ServerEndpoint`, `@ClientEndpoint`
  - **Programmatic:** `Endpoint`
    - **WebSocket opening handshake negotiation**

## Java API for WebSocket Features

- **API for WebSocket Server and Client Endpoint**
  - Annotated: `@ServerEndpoint`, `@ClientEndpoint`
  - Programmatic: `Endpoint`
    - WebSocket opening handshake negotiation
- **Lifecycle callback methods**

## Java API for WebSocket Features

- API for WebSocket Server and Client Endpoint
  - Annotated: `@ServerEndpoint`, `@ClientEndpoint`
  - Programmatic: `Endpoint`
    - WebSocket opening handshake negotiation
- Lifecycle callback methods
- Integration with Java EE technologies

# WebSocket Annotated Endpoint

```
import javax.websocket.*;  
  
@ServerEndpoint("/hello") ①  
public class HelloBean {  
    @OnMessage ②  
    public String sayHello(String name) {  
        return "Hello " + name;  
    }  
}
```

① `@ServerEndpoint` marks the POJO as WebSocket endpoint

② `@OnMessage` marks the method called when WebSocket message is received

# WebSocket annotations

---

- ### Class-level annotations

- **@ServerEndpoint** Turns a POJO in a server endpoint
- **@ClientEndpoint** Turns a POJO in a client endpoint

## WebSocket annotations

- **Class-level annotations**

- `@ServerEndpoint` Turns a POJO in a server endpoint
- `@ClientEndpoint` Turns a POJO in a client endpoint

- **Method-level annotations**

- `@OnMessage` Intercepts WebSocket message events
- `@OnOpen` Intercepts WebSocket open events
- `@OnClose` Intercepts WebSocket close events
- `@OnError` Intercepts WebSocket error events

# WebSocket annotations

- **Class-level annotations**

- `@ServerEndpoint` Turns a POJO in a server endpoint
- `@ClientEndpoint` Turns a POJO in a client endpoint

- **Method-level annotations**

- `@OnMessage` Intercepts WebSocket message events
- `@OnOpen` Intercepts WebSocket open events
- `@OnClose` Intercepts WebSocket close events
- `@OnError` Intercepts WebSocket error events

- **Parameter-level annotation**

- `@PathParam` Matches path segment of a URL-template

## **@ServerEndpoint attributes**

- **value** Relative URI or URI template e.g. '/hello' or '/chat/{subscriber-level}'
- **decoders** list of message decoder classnames
- **encoders** list of message encoder classnames
- **subprotocols** list of the names of the supported subprotocols

# Custom Payloads

```
@ServerEndpoint(  
    value="/hello",  
    decoders={MyMessageDecoder.class}, ①  
    encoders={MyMessageEncoder.class} ②  
)  
public class MyEndpoint {  
    . . .  
}
```

① Message decoder class

② Message encoder class

# Custom payloads: Text decoder

```
public class MyMessageDecoder implements Decoder.Text<MyMessage> { ①
    public MyMessage decode(String s) { ②
        JsonObject jsonObject = Json.createReader("...").readObject();
        return new MyMessage(jsonObject);
    }

    public boolean willDecode(String string) { ③
        . . .
        return true;
    }
    . . .
}
```

① Text decoder

② `decode` decodes a `String` to `MyMessage`

③ Returns `true` if payload can be processed

# Custom payloads: Text encoder

```
public class MyMessageEncoder implements Encoder.Text<MyMessage> { ①
    public String encode(MyMessage myMessage) { ②
        return myMessage.jsonObject.toString();
    }
    ...
}
```

① Text encoder

② `encode` encodes a `MyMessage` to `String`

# Custom payloads: Binary encoder

```
public class MyMessageDecoder implements Decoder.Binary<MyMessage> { ①
    public MyMessage decode(byte[] bytes) { ②
        . . .
        return myMessage;
    }

    public boolean willDecode(byte[] bytes) { ③
        . . .
        return true;
    }
    . . .
}
```

① Binary decoder

② `decode` decodes a `byte[]` to `MyMessage`

③ Returns `true` if payload can be processed

# **@OnMessage method signature**

---

## @OnMessage method signature

- Exactly one of the following
  - **Text:** String, Java primitive or equivalent class, String and boolean, Reader, any type for which there is a decoder
  - **Binary:** byte[], ByteBuffer, byte[] and boolean, ByteBuffer and boolean, InputStream, any type for which there is a decoder
  - **Pong messages:** PongMessage

## @OnMessage method signature

- Exactly one of the following
  - Text: String, Java primitive or equivalent class, String and boolean, Reader, any type for which there is a decoder
  - Binary: byte[], ByteBuffer, byte[] and boolean, ByteBuffer and boolean, InputStream, any type for which there is a decoder
  - Pong messages: PongMessage
- An optional Session parameter

## @OnMessage method signature

- Exactly one of the following
  - Text: String, Java primitive or equivalent class, String and boolean, Reader, any type for which there is a decoder
  - Binary: byte[], ByteBuffer, byte[] and boolean, ByteBuffer and boolean, InputStream, any type for which there is a decoder
  - Pong messages: PongMessage
- An optional Session parameter
- 0..n String parameters annotated with @PathParam

## @OnMessage method signature

- Exactly one of the following
  - Text: String, Java primitive or equivalent class, String and boolean, Reader, any type for which there is a decoder
  - Binary: byte[], ByteBuffer, byte[] and boolean, ByteBuffer and boolean, InputStream, any type for which there is a decoder
  - Pong messages: PongMessage
- An optional Session parameter
- 0..n String parameters annotated with @PathParam
- Return type: String, byte[], ByteBuffer, Java

# Sample Messages

---

## Sample Messages

- **void m(String s);**

## Sample Messages

- `void m(String s);`
- `void m(Float f, @PathParam("id")int id);`

## Sample Messages

- `void m(String s);`
- `void m(Float f, @PathParam("id")int id);`
- `Product m(Reader reader, Session s);`

## Sample Messages

- `void m(String s);`
- `void m(Float f, @PathParam("id")int id);`
- `Product m(Reader reader, Session s);`
- `void m(byte[] b); or void m(ByteBuffer b);`

## Sample Messages

- `void m(String s);`
- `void m(Float f, @PathParam("id")int id);`
- `Product m(Reader reader, Session s);`
- `void m(byte[] b); or void m(ByteBuffer b);`
- `Book m(int i, Session s, @PathParam("isbn")String is`

# Chat server

```
@ServerEndpoint("/chat")
public class ChatBean {
    static Set<Session> peers = Collections.synchronizedSet("...");

    @OnOpen ①
    public void onOpen(Session peer) {
        peers.add(peer);
    }

    @OnClose ②
    public void onClose(Session peer) {
        peers.remove(peer);
    }

    . . .
}
```

- ① Called when connection is opened
- ② Called when connection is closed

## Chat server (contd)

```
    . . .
@OnMessage
public void message(String message) {
    for (Session peer : peers) { ①
        peer.getBasicRemote().sendObject(message);
    }
}
```

- ① Iterate over all the connected clients

# Chat server (simplified)

```
@ServerEndpoint("/chat")
public class ChatBean {
    @OnMessage
    public void message(String message, Session endpoint) {
        for (Session peer : endpoint.getOpenSessions()) {
            peer.getBasicRemote().sendObject(message);
        }
    }
}
```

# URI-template matching

```
@ServerEndpoint("/orders/{order-id}") ①
public class MyEndpoint {
    @OnMessage public void processOrder(@PathParam("order-id")String orderId) { ②
        . . .
    }
}
```

- ① Template specified as {order-id}
- ② Value specified using @PathParam

# Client endpoint

```
@ClientEndpoint  
public class HelloClient {  
    @OnMessage public void message(String message, Session session) {  
        ①  
    }  
}
```

```
WebSocketContainer c = ContainerProvider.getWebSocketContainer();  
c.connectToServer(HelloClient.class, "hello"); ②
```

- ① process message from server
- ② connect to server

# Programmatic endpoint

```
public class MyEndpoint extends Endpoint { ①
    @Override
    public void onOpen(Session session) {
        session.addMessageHandler(new MessageHandler.Text() { ②
            public void onMessage(String name) { ③
                try {
                    session.getBasicRemote().sendText("Hello " + name); ④
                } catch (IOException ex) { }
            }
        });
    }
}
```

① Extend an abstract class [Endpoint](#)

② Add handler for each message

③ Handle the text message

④ Send the message to client

# Programmatic endpoint packaging

```
ServerEndpointConfiguration config =  
    ServerEndpointConfigurationBuilder.create(MyEndpoint.class, "/foo")  
    .build();
```

# **Server and client configuration**

---

## Server and client configuration

- **Server**

- **URI matching algorithm**
  - **Subprotocol and extension negotiation**
  - **Message encoders and decoders**
  - **Origin check**
  - **Handshake response**
-

# Server and client configuration

- **Server**

- **URI matching algorithm**
- **Subprotocol and extension negotiation**
- **Message encoders and decoders**
- **Origin check**
- **Handshake response**

- **Client**

- **Requested subprotocols and extensions**
  - **Message encoders and decoders**
  - **Request URI**
-

## Relationship with Dependency Injection

- Full Dependency Injection support required in endpoints
  - Field, method, constructor injection
- Interceptors permitted too



## Relationship with Servlet 3.1

- Allows a portable way to upgrade HTTP request
- New API
  - `HttpServletRequest.upgrade(ProtocolHandler handler)`

- **Authenticates using Servlet security mechanism during opening handshake**
  - Endpoint mapped by **ws://** is protected using security model defined using the corresponding http:// URI
- **Authorization defined using**  
**<security-constraint>**
- **Transport Confidentiality using **wss://****
  - Access allowed over encrypted connection only

# Debugging WebSocket messages (Wireshark)

Wireshark interface showing network traffic capture. The filter bar at the top has "http" selected. The main table lists 11 captured frames.

No.	Time	Source	Destination	Protocol	Length	Info
11	9.489449000	::1	::1	HTTP	648	GET /HelloWebSocket/ HTTP/1.1
13	9.491601000	::1	::1	HTTP	2134	HTTP/1.1 200 OK (text/html)
18	9.669322000	::1	::1	HTTP	501	GET /HelloWebSocket/echo HTTP/1.1
20	9.669489000	::1	::1	HTTP	543	GET /favicon.ico HTTP/1.1
22	9.670298000	::1	::1	HTTP	205	HTTP/1.1 101 Switching Protocols
24	9.671010000	::1	::1	HTTP	1624	HTTP/1.1 404 Not Found (text/html)
26	12.411987000	::1	::1	WebSocket	98	WebSocket Text [FIN] [MASKED]
28	12.413161000	::1	::1	WebSocket	108	WebSocket Text [FIN]
30	13.011122000	::1	::1	WebSocket	98	WebSocket Text [FIN] [MASKED]
32	13.013172000	::1	::1	WebSocket	108	WebSocket Text [FIN]

## References

- Java EE 7 samples - <https://github.com/javaee-samples/javaee7-samples>
- WildFly 8 - <http://wildfly.org>, [@WildFlyAS](http://github.com/wildfly)
- Undertow - <http://undertow.io>
- Slides generated with Asciidoctor and DZSlides backend
- Original slide template - Dan Allen & Sarah White

# Arun Gupta

@arungupta