WebSocket and Java

Interactive web application HTML4

- HTTP is half-duplex
- Polling
- Long Polling
- Ajax
- Complex, Inefficient, Wasteful

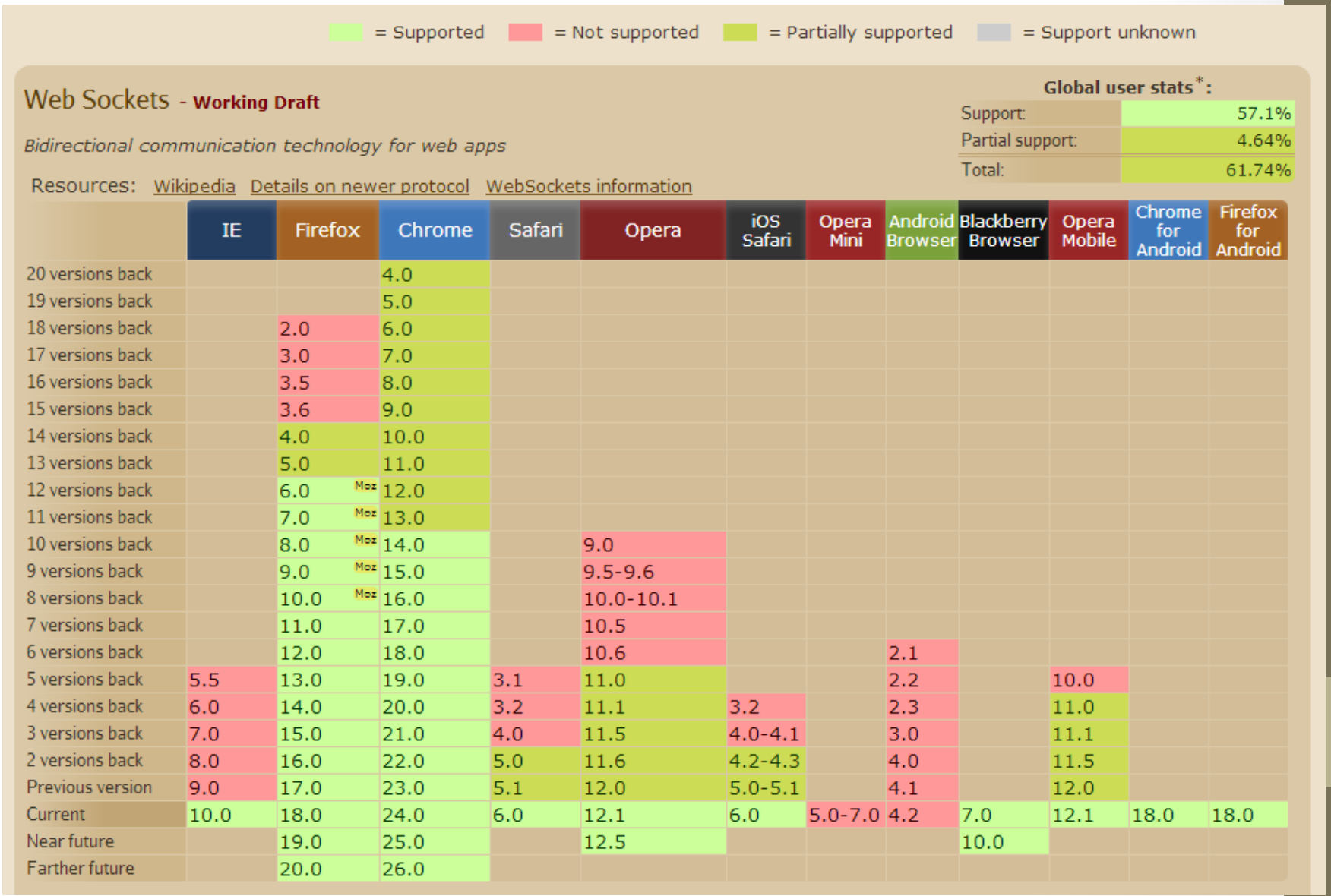
WebSocket Protocol

- TCP based, bi-directional, full-duplex messaging
- Part of HTML5
- IETF-defined Protocol: RFC 6455
- W3C defined JavaScript API
- Uses HTTP upgrade handshake
- Supports HTTP proxies, filtering, authentication and intermediaries

How does it work?

1. Establish connection
   - Single TCP connection
2. Send messages in both direction
   - Bi-directional
3. Send messages independent of each other
   - Full Duplex
4. End connection

Browser Support



	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
20 versions back			4.0									
19 versions back			5.0									
18 versions back		2.0	6.0									
17 versions back		3.0	7.0									
16 versions back		3.5	8.0									
15 versions back		3.6	9.0									
14 versions back		4.0	10.0									
13 versions back		5.0	11.0									
12 versions back		6.0 Moz	12.0									
11 versions back		7.0 Moz	13.0									
10 versions back		8.0 Moz	14.0		9.0							
9 versions back		9.0 Moz	15.0		9.5-9.6							
8 versions back		10.0 Moz	16.0		10.0-10.1							
7 versions back		11.0	17.0		10.5							
6 versions back		12.0	18.0		10.6			2.1				
5 versions back	5.5	13.0	19.0	3.1	11.0			2.2		10.0		
4 versions back	6.0	14.0	20.0	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	15.0	21.0	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	16.0	22.0	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	17.0	23.0	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	18.0	24.0	6.0	12.1	6.0	5.0-7.0	4.2	7.0	12.1	18.0	18.0
Near future		19.0	25.0		12.5				10.0			
Farther future		20.0	26.0									

WebSocket API (JavaScript)

```
var websocket = new WebSocket("ws://www.host.com/path");
websocket.onopen = function(evt) { onOpen(evt) };
websocket.onclose = function(evt) { onClose(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) }; }

function onMessage(evt) { alert( evt.data); }
function onError(evt) { alert( evt.data); }
..

websocket.send("client to server");
```

JSR 356
Java API for WebSocket

Java API for WebSocket

- **Endpoint**: Client or server
- **Connection**: Network connection between two endpoints
- **Peer**: Other endpoint of the connection
- **Session**: represents a sequence of websocket interactions between and end point and a peer

Annotations

- **@ServerEndpoint**
  - Class level annotation for websocket server endpoint
- **@ClientEndpoint**
  - Class level annotation for websocket client endpoint
- **@OnOpen**
  - Method level annotation signifies a method to be called whenever a new client connects to this endpoint
- **@OnClose**
  - Method level annotation signifies a method to be called whenever a new client is about to disconnects from this endpoint
- **@OnMessage**
  - Method level annotation signifies a method to be called whenever an incoming message is received

Code

```
@ServerEndpoint(value = "/hello-world")
public class HelloWorld {

    @OnMessage
    public String arrivedMessage(String content) {
        return ”Arrived: " + content;
    }

}
```

More Code!

```
@ServerEndpoint(value = "/hello-world")
public class HelloWorld {
    private Set<Session> peers = Collections.synchronizedSet(…)

    @OnOpen
    public void onOpen (final Session peer) {
        peers.add(peer); // add to the list
    }
    @OnClose
    public void onClose(final Session peer) {
        peers.remove(peer);
    }

    private void sendMessageToPeer(String msg) {
        for(Session s : peers) {
          if(s.isOpen()) s.getBasicRemote().sendObject(msg);
        }
    }

}
```

JSON and Java

Why JSON?

There was XML

... and the DOM... and SAX...

Then, there was JSON

```
{ "message" : "Hello World!" }
```

```
JsonGenerator generator = Json.createGenerator(System.out)
// or generator =
Json.createGenerator(servletResponse.getWriter())
generator
    .writeStartObject()
        .write("firstName", "John")
        .write("lastName", "Smith")
        .write("age", 25)
        .writeStartObject("address")
            .write("streetAddress", "21 2nd Street")
            .write("city", "New York")
            .write("state", "NY")
            .write("postalCode", "10021")
        .writeEnd()
        .writeStartArray("phoneNumber")
            .writeStartObject()
                .write("type", "home")
                .write("number", "212 555-1234")
            .writeEnd()
            .writeStartObject()
                .write("type", "fax")
                .write("number", "646 555-4567")
            .writeEnd()
        .writeEnd()
    .writeEnd();
generator.close();
```

Produces

```
{
   "firstName": "John", "lastName": "Smith", "age": 25,
   "address" : {
       "streetAddress": "21 2nd Street",
       "city": "New York",
       "state": "NY",
       "postalCode": "10021"
   },
   "phoneNumber": [
       {"type": "home", "number": "212 555-1234"},
       {"type": "fax", "number": "646 555-4567"}
    ]
 }
```

Or to build an object model...

```
JsonObject value = Json.createObjectBuilder()
     .add("firstName", "John")
     .add("lastName", "Smith")
     .add("age", 25)
     .add("address", Json.createObjectBuilder()
         .add("streetAddress", "21 2nd Street")
         .add("city", "New York")
         .add("state", "NY")
         .add("postalCode", "10021"))
     .add("phoneNumber", Json.createArrayBuilder()
         .add(Json.createObjectBuilder()
             .add("type", "home")
             .add("number", "212 555-1234"))
         .add(Json.createObjectBuilder()
             .add("type", "fax")
             .add("number", "646 555-4567")))
     .build();
// or from a stream..
JsonObject value2 = Json.createReader(inputStream).readObject();
```

And to read things from it...

```
JsonObject value2 = Json.createReader(inputStream).readObject()
int age = value2.getIntValue("age", 18);

JsonObject address = value2.getValue("address", JsonObject.class);
String city = "London";
if(address != null){
  city = address.getStringValue("city", "London");
}

JsonArray phoneNumbers = value2.getValue("phoneNumber",
JsonArray.class);
if(phoneNumbers != null){
  for(JsonValue val: value2){
    if(val instanceof JsonObject){
      JsonObject jo = (JsonObject)val;
      System.out.println(jo.getStringValue("number","Number Missing");
   }}}
```