Server push and web sockets
Responsive web apps, v1.0

• **Problem:** Client page needs info from server

• **Solution:** AJAX allows client to *pull* info
  - XMLHttpRequest makes asynchronous requests
    • Hacks to get around cross-domain restrictions
  - Uses standard HTTP request/response protocol
    • Small payload messages have high overhead
    • Latency introduced by HTTP processing
Responsive web apps, v2.0

• **Problem:** Server needs to *push* info to client
  – e.g. update stock price, movement of players, etc.

• **Possible solutions:**
  – **Polling:** Client makes periodic AJAX requests
    • Works well if you know the correct polling interval
    • Otherwise wastes network/server resources

Responsive web apps, v2.0

• **Problem:** Server needs to *push* info to client
  – e.g. update stock price, movement of players, etc.

• **Possible solutions:**
  – **Long-polling:** Client sends HTTP request, server waits until it has data to send in response
  – Hanging request may have high resource costs

[Diagram showing the long-polling process]

Responsive web apps, v2.0

• Problem: Server needs to *push* info to client
  – e.g. update stock price, movement of players, etc.

• Possible solutions:
  – Streaming: Server maintains open response
    continuously updated with push events
    • Subject to buffering by agents in network

Streaming: HTTP response

- **Response from server**
  - **Status line:**
    - Protocol version, status code, status phrase
  - **Response headers:** extra info
  - **Body:** optional data

HTTP/1.1 200 OK
Date: Thu, 17 Nov 2011 15:54:10 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Wed, 14 Sep 2011 17:04:27 GMT
Content-Length: 285

<html> ...

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Information</td>
<td>100 = server agrees to handle client’s request</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 = request succeeded; 204 = no content present</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>301 = page moved; 304 = cached page still valid</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>403 = forbidden page; 404 = page not found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>500 = internal server error; 503 = try again later</td>
</tr>
</tbody>
</table>
Streaming: HTTP response

- **Chunked response**
  - Each chunk specifies size in hex, last chunk = 0

```
HTTP/1.1 200 OK
Date: Thu, 17 Nov 2011 15:54:10 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Wed, 14 Sep 2011 17:04:27 GMT
Transfer-Encoding: chunked
29
<html><body><p>The file you requested is 5,340 bytes long and was last modified: 1d Sat, 20 Mar 2004 21:12:00 GMT 13 .</p></body></html> 0
```
Comet

• Comet (via polling or streaming)
  – Simulate bi-directional communication
    • Using HTTP request/response protocol
    • Often 2 connections: 1 downstream, 1 upstream
    • Resource expensive and error prone to write

http://www.websocket.org/quantum.html
HTML5

Taxonomy & Status on January 20, 2013

- W3C Recommendation
- Proposed Recommendation
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated

by Sergey Mavrody (Creative Commons Attribution-ShareAlike)
Web Sockets - CR

Bidirectional communication technology for web apps

<table>
<thead>
<tr>
<th></th>
<th>Chrome</th>
<th>Safari</th>
<th>Opera</th>
<th>iOS Safari</th>
<th>Opera Mini</th>
<th>Android Browser</th>
<th>Chrome for Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>8</td>
<td>31</td>
<td>37</td>
<td>39</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>36</td>
<td>37</td>
<td>39</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>35</td>
<td>40</td>
<td>7.1</td>
<td>7.1</td>
<td>4.4</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>36</td>
<td>40</td>
<td>8</td>
<td>27</td>
<td>8.1</td>
<td>37</td>
</tr>
<tr>
<td>TP</td>
<td>37</td>
<td>40</td>
<td>40</td>
<td>37</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

Global: 85.22% + 1.3% = 86.51%
unprefixed: 85.22% + 1.18% = 86.39%
HTML5 Web Sockets

• **Web sockets:**
  – JavaScript interface for client-side
  – **Full-duplex** communication
    • Using a single object, send string or binary data
    • Low latency, low header overhead (strings = 2 bytes)
  – Initial **handshake** over HTTP
    • Upgraded to web socket protocol
      – Some proxies may not like and drop the connection
    • Runs on **port 80** allowing it to traverse NATs

"Reducing kilobytes of data to 2 bytes...and reducing latency from 150ms to 50ms is far more than marginal. In fact, these two factors alone are enough to make Web Sockets seriously interesting to Google."

- Ian Hickson
Web socket protocol

- **URL prefix:**
  - `ws://` for normal connections, `wss://` for secure

- **HTTP-compatible handshake:**

  ```
  GET ws://echo.websocket.org/?encoding=text HTTP/1.1
  Origin: http://websocket.org
  Cookie: __utma=99as Connection: Upgrade
  Host: echo.websocket.org
  Sec-WebSocket-Key: uRovscZjNo1/umbTt5uKmw==
  Upgrade: websocket
  Sec-WebSocket-Version: 13
  
  HTTP/1.1 101 WebSocket Protocol Handshake
  Date: Fri, 10 Feb 2012 17:38:18 GMT
  Connection: Upgrade
  Server: Kaazing Gateway
  Upgrade: WebSocket
  Access-Control-Allow-Origin: http://websocket.org
  Access-Control-Allow-Credentials: true
  Sec-WebSocket-Accept: rLHCKw/5KsO9GAH/ZSFhBATDKrU=
  Access-Control-Allow-Headers: content-type
  ```
Web socket protocol

• After handshake:
  – HTTP connection broken down
  – Replaced by WebSocket connection
    • Over the same TCP/IP connection
    • Upgrade is one way, can't go back to HTTP

• Framing:

<table>
<thead>
<tr>
<th>Bit</th>
<th>+0..7</th>
<th>+8..15</th>
<th>+16..23</th>
<th>+24..31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FIN</td>
<td>Opcode</td>
<td>Mask</td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://chimera.labs.oreilly.com/books/1230000000545/ch17.html#websocket_protocol
Example messages

• A single-frame unmasked text message
  – 0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f (contains "Hello")

• A fragmented unmasked text message
  – 0x01 0x03 0x48 0x65 0x6c (contains "Hel")
  – 0x80 0x02 0x6c 0x6f (contains "lo")

• Unmasked Ping request and masked Ping response
  – 0x89 0x05 0x48 0x65 0x6c 0x6c 0x6f (contains a body of "Hello")
  – 0x8a 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58
    (contains a body of "Hello", matching the body of the ping)

• 256 bytes binary message in a single unmasked frame
  – 0x82 0x7E 0x0100 [256 bytes of binary data]

• 64KiB binary message in a single unmasked frame
  – 0x82 0x7F 0x000000000000010000 [65536 bytes of binary data]
Web socket examples

Echo Test
The first section of this page will let you do an HTML5 WebSocket test against the echo server. The second section walks you through creating a WebSocket application yourself.

You can also inspect WebSocket messages using your browser.

Try it out

- Location:
  - ws://echo.websocket.org
  - wss://echo.websocket.org

- Log:
  - CONNECTED
  - SENT: Rock it with HTML5 WebSocket
  - RESPONSE: Rock it with HTML5 WebSocket

- Message:
  - Rock it with HTML5 WebSocket
  - Connect
  - Disconnect

- http://www.websocket.org/echo.html
- http://demo.kaazing.com/livefeed/
- http://rumpetroll.com/
- http://labs.dinahmoe.com/plink/
- http://www.youtube.com/watch?v=64TcBiqmVko
WebSocket interface

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);
};
Simple text echo client

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script>
function init()
{
    websocket = new WebSocket("wss://echo.websocket.org/");
    websocket.onopen = function(e) { onOpen(e) };
    websocket.onclose = function(e) { onClose(e) };
    websocket.onmessage = function(e) { onMessage(e) };
    websocket.onerror = function(e) { onError(e) };
}
function onOpen(e)
{
    writeToScreen("CONNECTED");
    message = "Hello world!";
    writeToScreen("SENT: " + message);
    websocket.send(message);
}
function onClose(e)
{
    writeToScreen("DISCONNECTED");
}
function onMessage(e)
{
    writeToScreen('RESPONSE: ' + e.data);
    websocket.close();
}
function onError(e)
{
    writeToScreen('ERROR: ' + e.data);
}
function writeToScreen(message)
{
    document.getElementById("output").innerHTML += message + "<br />";
}
window.addEventListener("load", init, false);
</script>
</head>

<body>
<h2>WebSocket Test</h2>
<div id="output"></div>
</body>
</html>
```
Supported data types

• Send data as:
  – Text
  – ArrayBuffer
  – Blob

```javascript
// Sending String
connection.send('your message');

// Sending canvas ImageData as ArrayBuffer
var img = canvas_context.getImageData(0, 0, 400, 320);
var binary = new Uint8Array(img.data.length);
for (var i = 0; i < img.data.length; i++)
{
    binary[i] = img.data[i];
}
connection.send(binary.buffer);

// Sending file as Blob
var file = document.querySelector('input[type="file"]').files[0];
connection.send(file);

// Setting binaryType to accept received binary as either 'blob' or 'arraybuffer'
connection.binaryType = 'arraybuffer';
connection.onmessage = function(e)
{
    console.log(e.data.byteLength); // ArrayBuffer object if binary
};
```

http://www.html5rocks.com/en/tutorials/websockets/basics/
Client programming in practice

• WebSockets relatively new
  – Browser support is now widespread
  – W3C candidate recommendation
  – Network proxies may mess things up

• Other techniques more mature
  – XMLHttpRequest

• Option: use a 3rd party client library
  – Library provides semantics of bi-directional communication, but encapsulates details
  – e.g. socketio, cometd, Hookbox, orbited
Web socket server

• The server side
  – You need server-side support!
    • Support a large # of open WebSocket Connections
    • Traditional stacks (e.g. LAMP) do not deal well with this

• Commercial: Kaazing, ...

• Some free options:
  – apache-websocket: apache module
    • Develop your own module (in C) for app-specific details
  – pywebsocket: apache module / standalone server
    • Requires mod_python
  – Jetty WebSocketServlet
Other server options...

• C/C++
  • libwebsockets
  • Mongoose
  • POCO C++ Libraries
  • Tufão
  • Wslay
  • QtWebsocket

• Erlang
  • Yaws

• Go
  • go.net/websocket
  • webrocket

• Haskell
  • websockets

• Java
  • Apache Tomcat 7
  • Play Framework
  • Atmosphere
  • Bristleback
  • GlassFish 3.1, Grizzly
  • HLL WebSockets
  • JBoss 7
  • Jetty 7
  • jWebsocket
  • Netty 3.3
  • MigratoryData WebSocket Server

• .NET Framework
  • Internet Information Services (IIS) 8, ASP.NET 4.5
  • Windows Communication Foundation 4.5 through NetHttpBinding
  • Fleck
  • SuperWebSocket
  • XSockets.NET

• Clojure
  • http-kit
  • aleph

• Nginx
  • Proxy (since version 1.3.13)
  • Push Stream (3-rd party module)

• Node.js
  • Socket.IO
  • WebSocket-Node

• Objective-C
  • SocketRocket
  • BLWebSocketsServer

• Perl
  • Mojolicious
  • PocketIO

• PHP
  • php-websocket
  • Ratchet

• Python
  • WebSocket-for-Python
  • txWS
  • AutobahnPython

• Ruby
  • EM-WebSocket

• Other
  • apache-websocket
  • mod_websocket for lighttpd
  • nginx supports websocket since version 1.3
Summary

• **Responsive interactive web apps**
  – Often requires low latency bi-directional communication
  – Existing solutions:
    • Ajax polling, long polling, HTTP streaming
    • Really hacks working with an ill-suited HTTP request/response protocol
  – HTML5 web sockets:
    • Simple client-side API
    • Requires server supporting web sockets
    • You have to develop app-specific logic in some way
      – e.g. Apache module, Java servlet, ...