In the brain of Peter Lubbers

Building Real-Time Applications with HTML5 Web Sockets

Club Corfair, Paris, 9 December 2009
Qui C’est Ce Mec?

Peter Lubbers

- Director of documentation and training at Kaazing
- Skills Matter and Zenika HTML5 training courses
- Co-author Apress Book *Pro HTML5 Programming* (Spring 2010)
In the brain of Peter Lubbers?

HTML5 Communication!

Training

Family

Ultra Running and blogging at http://runlake tahoe.blogspot.com

Lake Tahoe

Writing

Pro HTML5 Programming

Peter Lubbers, Dale Alber, Rama Sathya, Sue Smith
Agenda

• Introduction
• History of the real-time Web
• WebSocket and Server-Sent Events
• Cross-Document Messaging and XHR Level 2
• Beyond WebSocket
• Questions?
Let’s Get Started!
History of the Real-Time Web
Today’s Requirements

• Today’s Web applications demand reliable, real-time communications with near-zero latency

• Examples:
  • Financial applications
  • Social networking applications
  • Online games
  • Smart power grid
HTTP Limitations

- Until now, this has been cumbersome to achieve, primarily due to the limitations of HTTP
- HTTP is half-duplex (traffic flows in only one direction at a time)
- HTTP is a stateless, request-driven protocol
HTTP Limitations

• Header information is sent with each HTTP request and response, which can be an unnecessary overhead
• After a response is sent, the server may choose to close the socket
• Rich Internet Applications (with Ajax, Comet, Silverlight, and Flash) are becoming richer, but still limited by HTTP
Ajax and Comet

- Ajax (Asynchronous JavaScript and XML) is a technique for building highly interactive applications for the Web
- Content can change without loading the entire page
- Ajax Delivers:
  - User-perceived low latency
  - Single page
- “Real-time” often achieved through polling and long-polling (Comet)
Polling

- Polling is “nearly real-time”
- Sometimes used in Ajax applications to simulate real-time communication
- Browser sends HTTP requests at regular intervals and immediately receives a response
Polling Architecture

Server

Browser

Request
Response

Time: Requests every $n$ seconds
Long Polling

• Also known as asynchronous-polling
• Browser sends a request to the server and the server keeps the request open for a set period.
• HTTP headers, present in both long-polling and polling often account for more than half of the network traffic.
Long Polling Architecture

Time: Requests every n seconds
Streaming

• More Efficient, but still not perfect
• Possible complications:
  • Proxies and Firewalls
  • Response builds up and must be flushed periodically
  • Cross-domain issues to do with browser connection limits
Streaming Architecture

- Server
- Browser

Request → Response

Time: Requests every $n$ seconds
Current Approaches Are…

Fake Time!

Or: Not Really Time
Example of Polling in Action
HTTP Header Overhead
Example HTTP Request

GET /PollingStock//PollingStock HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5)
    Gecko/20091102 Firefox/3.5.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost:8080/PollingStock/
Cookie: showInheritedConstant=false; showInheritedProtectedConstant=false;
    showInheritedProperty=false; showInheritedProtectedProperty=false;
    showInheritedMethod=false; showInheritedProtectedMethod=false;
    showInheritedEvent=false; showInheritedStyle=false; showInheritedEffect=false
Example HTTP Response

HTTP/1.x 200 OK
X-Powered-By: Servlet/2.5
Server: Sun Java System Application Server 9.1_02
Content-Type: text/html;charset=UTF-8
Content-Length: 321
Date: Sat, 07 Nov 2009 00:32:46 GMT
HTTP Header Traffic Analysis

• Example network throughput for Polling HTTP request and response headers:
  
  • **Use case A**: 10,000 clients polling every 60 seconds:
    • Network throughput is \((871 \times 10,000)/60 = 145,166\) bytes = \(1,161,328\) bits per second (1.1 Mbps)
  
  • **Use case B**: 10,000 clients polling every second:
    • Network throughput is \((871 \times 10,000) = 8,710,000\) bytes = \(69,680,000\) bits per second (66 Mbps)
  
  • **Use case C**: 100,000 clients polling every 10 seconds:
    • Network throughput is \((871 \times 100,000)/10 = 8710000\) bytes = \(69,680,000\) bits per second (66 Mbps)
Network Throughput Comparison
Technology-Specific Problems

- **AJAX**
  - Enables clients to asynchronously poll for server-side events (at best)
  - Polling leads to poor resource utilization on both the client and server

- **Comet**
  - Lack of a standard implementation
  - Often requires complex techniques such as multiplexing or managing multiple domains
Headache 2.0

Your RIA client application
- Custom code to simulate a realtime 2-way connection
- Silverlight or Flash plug-in
- Browser

Lots to build

Costly server resources devoted to translating LAN protocol to HTTP

Can’t manage the actual client — end user and data source aren’t really connected

Messy, slow, error prone HTTP - Long polling, etc.)
Complexity does not scale
Desktop vs. Browser

- Desktop Networking
  - Full-duplex bidirectional TCP sockets
  - Access any server on the network

- Browser Networking
  - Half-duplex HTTP request-response
  - HTTP polling, long polling fraught with problems
Desktop Architecture
Enter HTML5!
About HTML5

• HTML5 is the next set of W3C HTML standards
• It offers new and enhanced features to address new HTML primitives, multimedia, offline use, communication, and so on
• Many of the browser vendors have already implemented several of these features
HTML5 Features

• HTML5 includes a wide range of new features, including:
  • Canvas
  • Workers
  • Geolocation
  • New form elements
  • Offline storage
  • Communication APIs
  • And more…
HTML5 Communication

- WebSocket
  - Proxy/Firewall-friendly text socket for browsers
- Server-Sent Events
  - Standardized HTTP streaming (downstream)
- XMLHttpRequest Level 2
  - Cross-origin XMLHttpRequest
- Cross Document Messaging
  - Secure inter-window (iframe) communication
Part of the HTML5 Spec?

• Web Sockets—like other pieces of the HTML5 effort such as Local Storage and Geolocation—was originally part of the HTML5 specification

• Moved to a separate standards document to keep the specification focused

• Web Sockets has been submitted to the Internet Engineering Task Force (IETF) by its creators, the WHATWG
HTML5
Web Sockets
HTML5 WebSockets

- HTTP-friendly TCP for the browser with minimal framing
- Full-duplex bidirectional communication
- Operates over a single socket
- Distributed client-server architecture
- No browser plug-ins
- Traverses proxies and firewalls seamlessly
- Allows authorized cross-origin communication
WebSocket Architecture
Powerful Simplicity

- Quick, easy application creation & deployment
- Reliable end-to-end communications
- Fast, standards-based 2-way communication

Kaazing libraries (JavaScript, Flex, Silverlight, Java, JavaFX)
Native WebSocket
Browser

Kaazing WebSocket Gateway
HTML5 WebSockets

• Connection established by upgrading from the HTTP protocol to the WebSocket protocol using the same TCP connection
• WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode
HTML5 WebSocket Schemes

- WebSocket
  
  `ws://www.websocket.org/text`

- WebSocket Secure
  
  `wss://www.websocket.org/encrypted-text`
HTML5 WebSocket Handshake

GET /text HTTP/1.1\r\nUpgrade: WebSocket\r\nConnection: Upgrade\r\nHost: www.websocket.org\r\n...

HTTP/1.1 101 WebSocket Protocol Handshake\r\nUpgrade: WebSocket\r\nConnection: Upgrade\r\n...

© 2009 – Kaazing Corporation
HTML5 WebSocket Frames

- Text and binary frames can be sent full-duplex, in either direction at any the same time
- Each frame of data:
  - Starts with a 0x00 byte
  - Ends with a 0xFF byte
  - Contains UTF-8 data in between
- Text Frames use terminator
  - \x00Hello, WebSocket\x0ff
- Binary Frames use length prefix:
  - \x00\0x10Hello, WebSocket
- There is no defined maximum size
  - If the user agent has content that is too large to be handled, it must fail the Web Socket connection
  - JavaScript does not allow >4GB of data, so that is a practical maximum
Drastic Reduction in Network Traffic

- With WebSocket, each frame has only 2 bytes of packaging (compare almost 500:1 or even 1000:1)
- No latency involved in establishing new TCP connections for each HTTP message
- Remember the Polling HTTP header traffic (66 Mbps network throughput for headers alone)?
HTTP Header Traffic Analysis

- Example network throughput for WebSocket HTTP request and response headers:
  - **Use case A**: 10,000 frames every 60 seconds:
    - Network throughput is \( \frac{2 \times 10,000}{60} = 333 \text{ bytes} = 2,664 \text{ bits per second (2.6 Kbps)} \)
  - **Use case B**: 10,000 frames every second:
    - Network throughput is \( 2 \times 10,000 = 20,000 \text{ bytes} = 160,000 \text{ bits per second (156 Kbps)} \)
  - **Use case C**: 100,000 frames every 10 seconds:
    - Network throughput is \( \frac{2 \times 100,000}{10} = 20,000 \text{ bytes} = 160,000 \text{ bits per second (156 Kbps)} \)
Network Throughput Comparison

[Bar chart showing Network Throughput comparison across Use Case A, Use Case B, and Use Case C. The chart compares Polling and WebSocket methods for each use case.]
Overheard…

"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 WebSocket seriously interesting to Google."

—Ian Hickson (Google, HTML5 spec lead)
HTML5 WebSockets API

• Creating a WebSocket instance:
  ```javascript
  var myWebSocket = new WebSocket
      ("ws://www.websocket.org");
  ```
HTML5 WebSockets API

• Associating listeners

```javascript
myWebSocket.onopen = function(evt) {
    alert("Connection open ...");
};
myWebSocket.onmessage = function(evt) {
    alert("Received Message: "+
    evt.data);
};
myWebSocket.onclose = function(evt) {
    alert("Connection closed.");
};
```
HTML5 WebSockets API

- Sending messages

```javascript
myWebSocket.send("Hello WebSocket!");
myWebSocket.close();
```
Browser Support

• Chromium support added just a few weeks ago (nightly builds)
• Webkit activity
Native WebSocket in Chromium
Demo
HTML5 Server-Sent Events
HTML5 Server-Sent Events

• Standardizes and formalizes how a continuous stream of data can be sent from a server to a browser
• Introduces EventSource—a new JavaScript API
SSE Architecture
EventSource API

• Connects to a server URL to receive an event stream:
  ```javascript
  var stream =
      new EventSource("http://news.kaazing.com");
  ```

• Set event handlers:
  ```javascript
  stream.onopen = function() { alert("open"); }  
  stream.onmessage = function(event) {
      alert("message: " + event.data); }
  stream.onerror = function() { alert("error"); }
  ```
Browser Support

• Partial support in Opera 9+
• Development in Firefox trunk
Demo
XMLHttpRequest Level 2
XMLHttpRequest Level 2

• XMLHttpRequest is the API that made Ajax possible
• XMLHttpRequest Level 2 significantly enhances XMLHttpRequest
• Improvements in the following areas:
  • Progress events
  • Cross-origin XMLHttpRequests
Progress Events

- XMLHttpRequest Level 2 supports named progress events:
  - `loadstart`
  - `progress`
  - `abort`
  - `error`
  - `load`
  - `loadend`

- `readyState` property and `readystatechange` events retained for backward compatibility
Progress Events

• Progress events have fields for:
  • The total amount of data to transfer
  • The amount that has already transferred
  • Whether the total is known (Boolean value)

• Example:
  ```javascript
  crossOriginRequest.upload.onprogress = function(e) {
    var total = e.total;
    var loaded = e.loaded;

    if (e.lengthComputable) {
      // do something with the progress information
    }
  }
  ```
Cross-Origin XMLHttpRequest

- XMLHttpRequest Level 2 allows for cross-origin XMLHttpRequests using Cross Origin Resource Sharing (CORS)
  
  [http://www.w3.org/TR/access-control/](http://www.w3.org/TR/access-control/)

- Example:

  ```javascript
  var crossOriginRequest = new XMLHttpRequest();
  crossOriginRequest.open("GET", "http://www.example.net/stockfeed", true);
  ```
Origin Security

• An origin is a subset of an address used for modeling trust relationships on the Web
• Origins are made up of a scheme, a host, and a port—different origin:
  • https://www.example.com
  • http://www.example.com
• The path is not considered in the origin value—same origin:
  • http://www.example.com/index.html
  • http://www.example.com/page2.html
Origin Security

- Cross-origin communication identifies the sender by origin
  - Allows receiver to ignore messages from origins it does not trust or does not expect to receive messages from (white list)
  - Applications must opt-in to receiving messages by adding an event listener for message events
  - No risk of messages interfering with an unsuspecting application
Preflight Requests

• For sensitive actions (for example, a request with credentials, or a request other than GET or POST), an OPTIONS preflight request is sent to the server to see if the action is supported and allowed

• Successful communication may require a CORS-capable server
Cross-Origin HTTP Headers

- A cross-origin exchange between a page hosted on www.example.com and a service hosted on www.example.net (Request):

```plaintext
POST /main HTTP/1.1
Host: www.example.net
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.3)
    Gecko/20090910 Ubuntu/9.04 (jaunty) Shiretoko/3.5.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/
Origin: http://www.example.com
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
```
Cross-Origin HTTP Headers

- Cross-origin exchange between a page hosted on www.example.com and a service hosted on www.example.net (Response):

  HTTP/1.1 201 Created
  Transfer-Encoding: chunked
  Server: Kaazing Gateway
  Date: Mon, 02 Nov 2009 06:55:08 GMT
  Content-Type: text/plain
  Access-Control-Allow-Origin: http://www.example.com
  Access-Control-Allow-Credentials: true
Cross-Origin XMLHttpRequest

• Allows you to build web applications that use services hosted on different origins
  • For example, a web application that used static content from one origin and Ajax services from another
• Allows client-side aggregation (the ultimate mashup)
Client-Side Aggregation
Browser Support

- Latest versions of
  - Firefox
  - Chrome
  - Safari
Cross Document Messaging

HTML5
Cross Document Messaging

- Enables secure cross-origin communication across iframes, tabs, and windows (using origin security)
- Defines the `PostMessage` API as a standard way to send messages
- Provides asynchronous message passing between JavaScript contexts
Cross Document Messaging

- Can be accomplished by direct scripting (a script running in one page tries to manipulate another document), but may not be allowed due to security restrictions.
PostMessage Overview
HTML5 PostMessage

- Allows you to communicate between documents served by different origins
- Also handy for same-origin messaging, because it provides a consistent, easy to use API (for example, HTML5 Web Workers)
HTML5 PostMessage

• Sending a message:

  myFrame.contentWindow.postMessage
  ('Hello, world',
  'http://www.example.com/');
HTML5 PostMessage

- Listening for messages:

```javascript
window.addEventListener("message", messageHandler, true);

function messageHandler(e) {
    switch(e.origin) {
        case "friend.example.com":
            // process message
            processMessage(e.data);
            break;
        default:
            // message origin not recognized
            // ignoring message
            break;
    }
}
```
Browser Support

• Firefox 3.5 and greater
• Safari 4.0 and greater
• Chrome 2.0 and greater
• Opera 9.6 and greater
• Internet Explorer 8 (supports cross-frame but not cross-window messaging)
Demo
What would YOU do with a WebSocket?
Extending HTML5 WebSockets

• Once you have a bi-directional, full-duplex socket connection, you can do all kinds of great things in a browser

• Any TCP-based protocol works over WebSocket
  • JMS, AMQP, STOMP, XMPP, IMAP, AMQP, IRC, and more
  • Custom Protocols

• Binary Protocols
  • Encode Binary as Text
Stomp

- Streaming Text Oriented Messaging Protocol
- Connect to a message broker to publish and subscribe to channels and topics
- Example message brokers that support Stomp:
  - RabbitMQ
  - ActiveMQ
  - More
Stomp Protocol

Stomp commands
- ABORT
- ACK
- BEGIN
- COMMIT
- CONNECT
- DISCONNECT
- SEND
- SUBSCRIBE
- UNSUBSCRIBE

Server frames
- ERROR
- MESSAGE
- RECEIPT

Example Stomp frame:
CONNECT
login: <username>
passcode:<passcode>
^@
Example Stomp Client

- Stomp Client

```javascript
var myStomp = new StompClient();
myStomp.onopen =
    function(headers) {
        myStomp.subscribe("/topic/destination");
    }
myStomp.onmessage =
    function(headers, body) { alert(body); }
myStomp.connect("ws://www.websocket.org/stomp");
myStomp.send("Hello STOMP!",
    "/topic/destination");
```
Kaazing WebSocket Gateway

• Enterprise Grade WebSocket server
• Seamlessly and reliably extends any TCP-based business messaging protocol to the Web with ultra high performance and minimal latency
• Requires no third-party browser plug-in or client-side installation
• Provides emulation for older browsers so you can code against the new standards today
Browser Certification

• Supports native HTML 5 Communications (when available) as well as emulation for current browsers

• Certified Browser Versions:
  • Apple Safari 3.0+
  • Google Chrome 1.0+
  • Microsoft Internet Explorer 5.5+
  • Mozilla Firefox 1.5+
  • Opera 9.5+
Client Libraries

- Stomp
  - Stomp-JMS Adapter
- AMQP
- XMPP (Jabber)
- IRC
- Darkstar
- More added all the time
Client Library Technologies

- JavaScript
- Adobe Flex (Flash)
- Microsoft Silverlight (Kaazing is a Microsoft Silverlight partner)
- Java and JavaFX (Kaazing is a Sun Microsystems partner)
Some Example Applications
# Financial Applications

![Stock Market Tracking](http://www.kaazing.com/demos/jsStock/stock-jquery.html)

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>SYMBOL</th>
<th>PRICE</th>
<th>CHANGE</th>
<th>SPARKLINE</th>
<th>OPEN</th>
<th>LOW</th>
<th>HIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE WALT DISNEY COMPANY</td>
<td>DIS</td>
<td>27.43</td>
<td>-0.76</td>
<td>ARROW DOWN</td>
<td>28.19</td>
<td>24.39</td>
<td>29.80</td>
</tr>
<tr>
<td>GARMIN LTD</td>
<td>GRMN</td>
<td>54.50</td>
<td>-3.49</td>
<td>ARROW DOWN</td>
<td>57.99</td>
<td>31.32</td>
<td>36.26</td>
</tr>
<tr>
<td>SANDISK CORPORATION</td>
<td>SNDK</td>
<td>19.02</td>
<td>-1.42</td>
<td>ARROW DOWN</td>
<td>20.44</td>
<td>16.22</td>
<td>22.26</td>
</tr>
<tr>
<td>GOODRICH CORPORATION</td>
<td>GR</td>
<td>51.90</td>
<td>-3.66</td>
<td>ARROW DOWN</td>
<td>55.56</td>
<td>47.11</td>
<td>57.57</td>
</tr>
<tr>
<td>NVIDIA CORPORATION</td>
<td>NVIDIA</td>
<td>13.46</td>
<td>-0.19</td>
<td></td>
<td>13.65</td>
<td>12.47</td>
<td>15.23</td>
</tr>
<tr>
<td>CHEVRON CORPORATION</td>
<td>CVX</td>
<td>57.89</td>
<td>-3.41</td>
<td>ARROW DOWN</td>
<td>61.30</td>
<td>61.48</td>
<td>75.12</td>
</tr>
<tr>
<td>THE ALLSTATE CORPORATION</td>
<td>ALL</td>
<td>32.63</td>
<td>1.61</td>
<td>ARROW UP</td>
<td>31.02</td>
<td>27.92</td>
<td>34.11</td>
</tr>
<tr>
<td>EXXON MOBIL CORPORATION</td>
<td>XOM</td>
<td>67.71</td>
<td>1.21</td>
<td>ARROW UP</td>
<td>66.52</td>
<td>65.07</td>
<td>71.17</td>
</tr>
<tr>
<td>METLIFE INC.</td>
<td>MET</td>
<td>25.64</td>
<td>-0.09</td>
<td>ARROW DOWN</td>
<td>25.73</td>
<td>22.16</td>
<td>39.20</td>
</tr>
<tr>
<td>J.C. PENNEY COMPANY INC.</td>
<td>JCP</td>
<td>32.66</td>
<td>-3.29</td>
<td>ARROW DOWN</td>
<td>32.95</td>
<td>29.86</td>
<td>36.24</td>
</tr>
<tr>
<td>OFFICEMAX INCORPORATED</td>
<td>DWX</td>
<td>12.22</td>
<td>-0.15</td>
<td>ARROW DOWN</td>
<td>12.41</td>
<td>11.17</td>
<td>13.65</td>
</tr>
<tr>
<td>AETNA INC.</td>
<td>AET</td>
<td>27.30</td>
<td>0.42</td>
<td>ARROW UP</td>
<td>26.87</td>
<td>24.19</td>
<td>29.56</td>
</tr>
<tr>
<td>CONOCOPHILLIPS</td>
<td>COP</td>
<td>42.39</td>
<td>-3.63</td>
<td>ARROW DOWN</td>
<td>46.02</td>
<td>41.96</td>
<td>51.27</td>
</tr>
<tr>
<td>UNITEDHEALTH GROUP INC.</td>
<td>UH</td>
<td>24.41</td>
<td>0.67</td>
<td>ARROW UP</td>
<td>24.36</td>
<td>23.93</td>
<td>26.79</td>
</tr>
</tbody>
</table>

Receiving 94 Updates Per Sec. and 1.69 KB Per Sec.
Financial Applications
Earth Control Game
http://apps.facebook.com/earthcontrol
Chess Vegas
http://www.chessvegas.com/
Degony Game
http://www.degony.com
Important URLs

• Kaazing Website:
  www.kaazing.com

• Kaazing Technology Network:
  http://tech.kaazing.com/

• Download Kaazing WebSocket Gateway:
  http://www.kaazing.com/download

• “When can I use” site:
  http://a.deveria.com/caniuse/

• Skills Matter:
  www.skillsmatter.com

• Zenika:
  http://www.zenika.com
THANKS!

Contact: peter.lubbers@kaazing.com (or training@kaazing.com)
Q&A