Harnessing the Power of HTML5 WebSocket to Create Scalable Real-time Applications

Brian Albers & Peter Lubbers, Kaazing
About Peter Lubbers

- Director of Documentation and Training, Kaazing
- Co-Founder San Francisco HTML5 User Group
  http://www.sfhtml5.org/
- Ultra Distance Runner
- Twitter: @peterlubbers
About Brian Albers

• Vice President of Research and Development, Kaazing
• Ultra Couch Potato
• www.kaazing.com (yes, we’re hiring)
Pro HTML5 Programming

• Co-authors of the Apress book *Pro HTML5 Programming*
• Alpha Release now available: [www.prohtml5.com](http://www.prohtml5.com)
• Special 50% Off Coupon Code: APRESSPROHTML5WR
  Expires May 31st
The Perils of Polling
Vote **NO** on Tin Cans and String for Communication!
Agenda

• Tin-can communication
• Meet HTML5 WebSocket
• HTML5 WebSocket in the enterprise
• Scaling through simplicity
Today’s Requirements

• Today’s Web applications demand reliable, real-time communications with near-zero latency
• Not just broadcast, but bi-directional communication
• Examples:
  • Financial applications
  • Social networking applications
  • Online games
  • Smart power grid
HTTP Is Not Full Duplex
About HTTP

• HTTP was originally designed for document transfer
• Until now, it has been cumbersome to achieve real-time, bi-directional web communication due to the limitations of HTTP
• HTTP is half-duplex (traffic flows in only one direction at a time)
• Header information is sent with each HTTP request and response, which can be an unnecessary overhead
About Ajax and Comet

• Great toilet cleaners…

• Ajax (Asynchronous JavaScript and XML) can be used to build interactive Web apps
  o Content can change without refreshing the entire page
  o User-perceived low latency

• “Real-time” often achieved through polling and long-polling (Comet or Reverse Ajax)
  o Comet lack of a standard implementation
  o Comet adds lots of complexity
Half-Duplex Architecture

- HTTP (half-duplex)
  - Custom web client
  - Custom web app
  - Long-polling

- Custom web client
  - Custom web app
  - User-initiated refresh

- Custom web client
  - Custom web app
  - Nailed-up request

- Custom web client
  - Custom web app
  - Timed refresh
  - Multiple inefficient, non-standard connection strategies

Server
- Java EE Container
- Server-side logic
  - Transformation logic
  - Connection management
  - Custom management for each web app

- RSS Client
  - Client-specific authentication & recovery
  - Custom-TCP (News Feed)

- EJB Client
  - Client-specific authentication & recovery
  - RMI-TCP (Database)

- JMS Client
  - Client-specific authentication & recovery
  - JMS-TCP (Messaging Broker)

- XMPP Client
  - Client-specific authentication & recovery
  - XMPP-TCP (Chat Server)
Polling

- Polling is “nearly real-time”
- Used in Ajax applications to simulate real-time communication
- Browser sends HTTP requests at regular intervals and immediately receives a response
- In low-message-rate situations, many connections are opened and closed needlessly
Polling Architecture

![Diagram of Polling Architecture]

Server

Browser

Time: Requests every n seconds

Copyright © 2010 - Kaazing Corporation. All rights reserved.
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 most of the network traffic
- In high-message rate situations, long-polling results in a continuous loop of immediate polls
Long Polling Architecture

Time: Requests every $n$ seconds
Streaming

- More efficient, but sometimes problematic
- 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

Request

Response

Browser

Time: Requests every $n$ seconds
Comet Poker Demo
HTTP Request Headers

POST /gwt/EventService HTTP/1.1
Host: gpokr.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US)
   AppleWebKit/532.5 (KHTML, like Gecko) Chrome/4.1.249.1064 Safari/532.5
Referer: http://gpokr.com/gwt/7F5E66657B938E2FDE9CD39095A0E9E6.cache.html
Content-Length: 134
Origin: http://gpokr.com
Content-Type: text/plain; charset=utf-8
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US, en; q=0.8
Accept-Charset: ISO-8859-1, utf-8; q=0.7, *; q=0.3
Cookie:
   __utmxz=247824721.1273102477.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd
   =(none); JSESSIONID=E7AAE0E60B01FB88D1E3799FAD5C62B3;
   __utma=247824721.1247485893.1273102477.1273104838.1273107686.3;
   __utmc=247824721; __utmb=247824721.4.10.127
HTTP Response Headers

200 OK
Server: Apache-Coyote/1.1
Expires: Thu, 06 May 2010 01:06:51 GMT
Content-Type: text/plain; charset=UTF-8
Content-Length: 303
Date: Thu, 06 May 2010 01:06:50 GMT

• Total (unnecessary) HTTP request and response header information overhead: 871 bytes (example)
• Overhead can be as much as 2000 bytes
HTTP Header Traffic Analysis

- Example network throughput for HTTP request and response headers associated with polling
  - **Use case A:** 1,000 clients polling every second:
    - Network throughput is \((871 \times 1,000) = 871,000\) bytes = \(6,968,000\) bits per second (~6.6 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 second:
    - Network throughput is \((871 \times 100,000) = 87,100,000\) bytes = \(696,800,000\) bits per second (~665 Mbps)
Comet: 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
Enter HTML5 WebSocket!
HTML5 WebSocket

- W3C API and IETF Protocol
- Full-duplex text-based socket
- Enables web pages to communicate with a remote host
- Traverses firewalls, proxies, and routers seamlessly
- Leverages Cross-Origin Resource Sharing (CORS)
- Share port with existing HTTP content (80/443)
Possible WebSocket Architecture
HTML5 WebSocket Schemes

• WebSocket
  o `ws://www.websocket.org/text`

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

• Connection established by upgrading from the HTTP protocol to the WebSocket protocol using the same TCP connection

• Once upgraded, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode
**HTML5 WebSocket Handshake**

**Client**

GET /text HTTP/1.1  
Upgrade: WebSocket  
Connection: Upgrade  
Host: www.example.com  
Origin: http://example.com  
WebSocket-Protocol: sample  
...

**Server**

HTTP/1.1 101 WebSocket Protocol Handshake  
Upgrade: WebSocket  
Connection: Upgrade  
WebSocket-Origin: http://example.com  
WebSocket-Location: ws://example.com/demo  
WebSocket-Protocol: sample  
...

Copyright © 2010 - Kaazing Corporation. All rights reserved.
HTML5 WebSocket Frames

- Frames can be sent full-duplex, in either direction at any the same time

- Each frame of data:
  - Starts with a 0x00 byte and End with a 0xFF byte
  - Contains UTF-8 data in between

- Example:
  - `\x00Hello, WebSocket\0xff`

- 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
Using the WebSocket API
// Checking for browser support
if (window.WebSocket) {
    document.getElementById("support").innerHTML = "HTML5 WebSocket is supported";
} else {
    document.getElementById("support").innerHTML = "HTML5 WebSocket is not supported";
}
Using the WebSocket API

```javascript
// Create new WebSocket
var mySocket = new WebSocket("ws://www.websocket.org");

// Associate listeners
mySocket.onopen = function(evt) {
    alert("Connection open…");
};

mySocket.onmessage = function(evt) {
    alert("Received message: " + evt.data);
};

mySocket.onclose = function(evt) {
    alert("Connection closed…");
};
```
Using the WebSocket API

```javascript
// Sending data
mySocket.send("HTML5 WebSocket Rocks!");

// Close WebSocket
mySocket.close();
```
Browser Support for WebSocket

- Chrome 4.0+
- WebKit Nightly builds
- Support planned for Firefox: “We really really want to support WebSockets in the next version of Firefox.” –Christopher Blizzard, Mozilla
Example: Native WebSocket in Chrome

```html
<!DOCTYPE html>

<head>
  
</head>

<body>

  window.WebSocket

  function WebSocket() { [native code] }

</body>
```
WebSocket Emulation

• Kaazing WebSocket Gateway
  • http://www.kaazing.com/download
  • Makes WebSocket work in all browsers today (including I.E. 6)

• Flash WebSocket implementation
  • http://github.com/gimite/web-socket-js
  • Requires opening port on the server’s firewall
Beyond WebSocket

• Once you have WebSocket, you can communicate with WebSocket Servers and back-end servers and directly with message brokers.

• You can extend client-server protocols to the web:
  • XMPP, Jabber
  • Pub/Sub (Stomp/AMQP)
  • Gaming protocols
  • Any TCP-based protocol

• Browser becomes a first-class network communication citizen.
Financial Applications
Example: WebSocket-Based Quake II Game Port

http://code.google.com/p/quake2-gwt-port
Twitter Feed

WSJ Market News Asian Shares Boosted By Optimism On Greece http://on.wsj.com/8YWHui
WSJ Economy News Cost of Federal Bailouts Starts to Ease http://on.wsj.com/dijk8H
WSJ Tech News #tech Exclusive Video: Bill Gross Talks About TweetUp http://on.wsj.com/aLOAru
WSJ Business Republicans Keep Filibuster on Tap http://on.wsj.com/cPFOP8

http://kaazing.me
WebSocket Demo
WebSocket in the Enterprise
Securing WebSocket Traffic

- WebSocket defines the ws:// and wss:// schemes
- WSS is WS over TLS (Transport Layer Security), formerly known as SSL (Secure Socket Layer) support (Similar to HTTPS)
- An HTTPS connection is established after a successful TLS handshake (using public and private key certificates)
- HTTPS is not a separate protocol, but it is HTTP running on top of a TLS connection (default ports is 443)
WebSocket and Web Intermediaries

- Web Socket protocol itself is unaware of proxy servers and firewalls
- WebSocket features an HTTP-compatible handshake so that HTTP servers can share their default HTTP and HTTPS ports (80 and 443) with a WebSocket server
Proxy Servers

- A proxy server is a server that acts as an intermediary between a client and another server (for example, a web server on the Internet)
- Commonly used for content caching, Internet connectivity, security, and enterprise content filtering
- Typically set up between a private network and the Internet
- Proxy servers can monitor traffic and close a connection if it has been open for too long
Proxy Server Problems

- Problems for web applications that have a long-lived connection (for example, Comet HTTP streaming or HTML5 Web Sockets)
- HTTP proxy servers
- May also buffer unencrypted HTTP responses, thereby introducing unpredictable latency during HTTP response streaming
- Without any intermediary servers, a WebSocket connection can be established smoothly, as long as both the server and the client understand the Web Socket protocol
Types of Proxy Servers
WebSocket Proxy Traversal

```
Web Sockets or Web Sockets Secure?
  /         \
 /           \
Web Sockets   Web Sockets Secure
  |                      |
  |                      |
  v                      v
Explicit Proxy Configuration?
  /         \
 /           \
Yes         No
  /         \
 /           \
Send HTTP CONNECT
  / \
 /   \
Yes   No
  / \
 /   \
Transparent Proxy Server?  Transparent Proxy Server?
  /         \
 /           \
Yes         No
  /         \
 /           \
Send HTTP CONNECT
  / \
 /   \
Yes   No
  / \
 /   \
Send TLS Handshake
```

* Depending on explicit and transparent proxy server configuration and behavior

http://www.infoq.com/articles/Web-Sockets-Proxy-Servers
Load Balancing Routers

- **TCP** (layer-4) load-balancing routers should work well with HTML5 Web Sockets, because they have the same connection profile: connect once up front and stay connected, rather than the HTTP document transfer request-response profile.

- **HTTP** (Layer 7) load-balancing routers expect HTTP traffic and can easily get confused by WebSocket upgrade traffic. For that reason, Layer 7 load balancing routers may need to be configured to be explicitly aware of WebSocket traffic.
Firewalls

• Since firewalls normally just enforce the rules for inbound traffic rejection and outbound traffic routing (for example, through the proxy server), there are usually no specific WebSocket traffic-related firewall concerns

• For regular socket connections (for example, for a desktop client) you must open a port on the firewall
WebSocket Traffic Analysis
WebSocket Poker Demo
Dramatic Reduction in Network Traffic

- With WebSocket, each frame has only 2 bytes of packaging (a 500:1 or even 1000:1 reduction)
- No latency involved in establishing new TCP connections for each HTTP message
- Dramatic reduction in unnecessary network traffic and latency
- Remember the Polling HTTP header traffic? 665 Mbps network throughput for just headers
WebSocket Framing Analysis

• Example network throughput overhead associated with WebSocket framing
  
  • **Use case A**: 1,000 clients receive 1 message per second: Network throughput is \((2 \times 1,000) = 2,000\) bytes = 16,000 bits per second (~0.015 Mbps)
  
  • **Use case B**: 10,000 clients receive 1 message per second: Network throughput is \((2 \times 10,000) = 20,000\) bytes = 160,000 bits per second (~0.153 Mbps)
  
  • **Use case C**: 100,000 clients receive 1 message per second: Network throughput is \((2 \times 100,000) = 200,000\) bytes = 1,600,000 bits per second (~1.526 Mbps)
Polling vs. WebSocket

The diagram compares the number of bits per second for Polling and Web Sockets in different use cases:

- **Use Case A**: Polling 6,968,000, Web Sockets 16,000
- **Use Case B**: Polling 69,680,000, Web Sockets 160,000
- **Use Case C**: Polling 696,800,000, Web Sockets 1,600,000
Latency Reduction
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)
Q&A
Web2.0 Expo Special!

10% Off Any HTML5 Training
With Coupon Code WEB2EXPO
http://tech.kaazing.com/training/index.html