KAAZING

HTML5 Realtime and WebSocket Code Lab

HTML 5 SAN FRANCISCO

GTUG San Francisco

@peterlubbers
@franksalim
Kaazing

WebSocket Demo

http://demo.kaazing.com/racer/

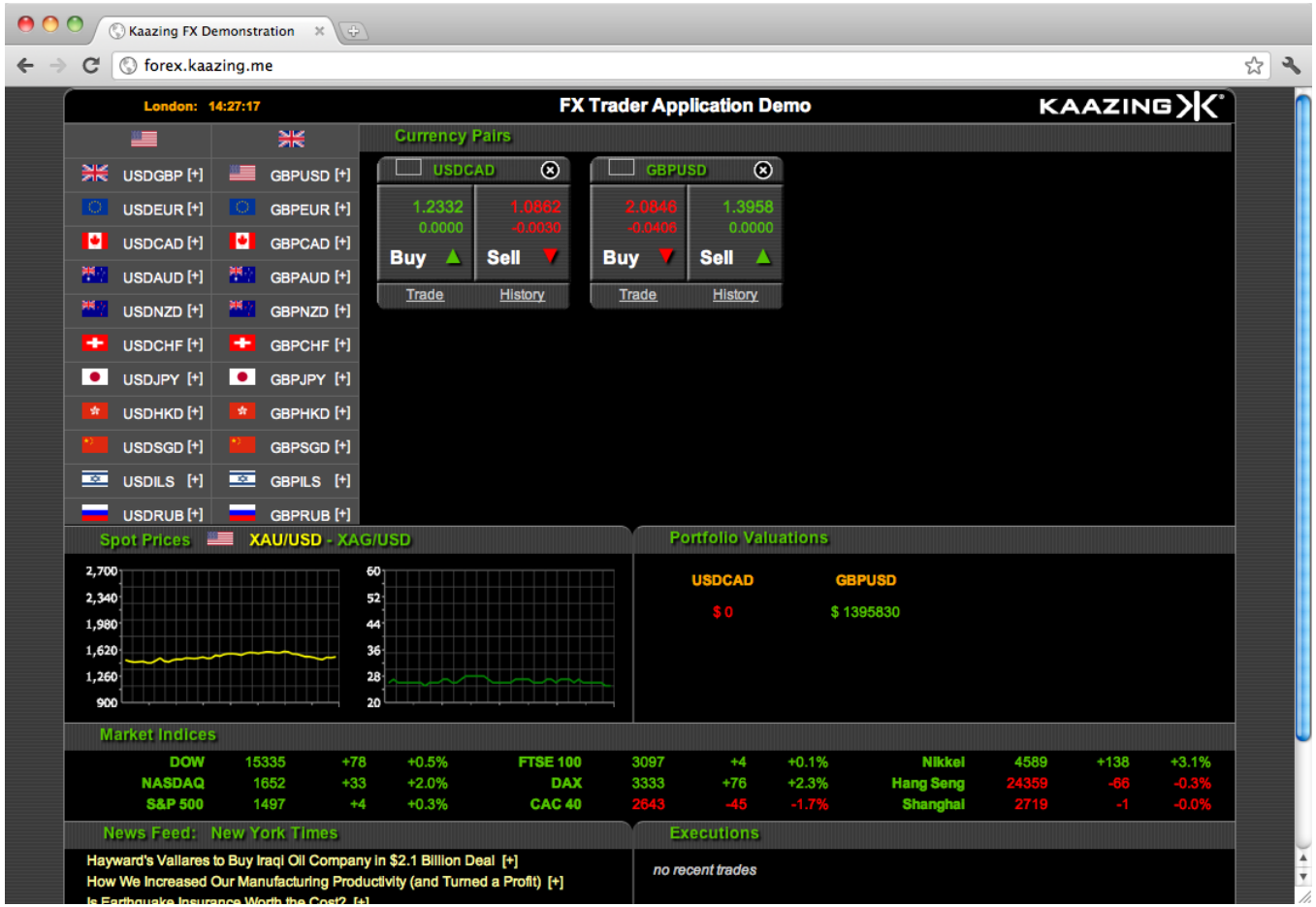
WebSocket Demo

- Plink: http://labs.dinahmoe.com/plink/#

WebSocket Demos

FX Trader Application Demo

http://demo.kaazing.me/forex



Example: *CollectdViewer*

Server Monitor System

http://bergmans.com/WebSocket/collectdViewer.html

Agenda

- Demos
- Introduction to WebSocket
- WebSocket API
- WebSocket Protocol
- Real-World WebSocket

Introduction to WebSocket

Networked Applications

- On the LAN: Reliable, real-time communication
- On the web: ?
  - Mostly idle
  - Mostly broadcast
  - *Nearly* real-time
- Web + WebSockets = reliable *and* real-time
- Approaches on the web:
  - WebSockets
  - Http-based

HTTP

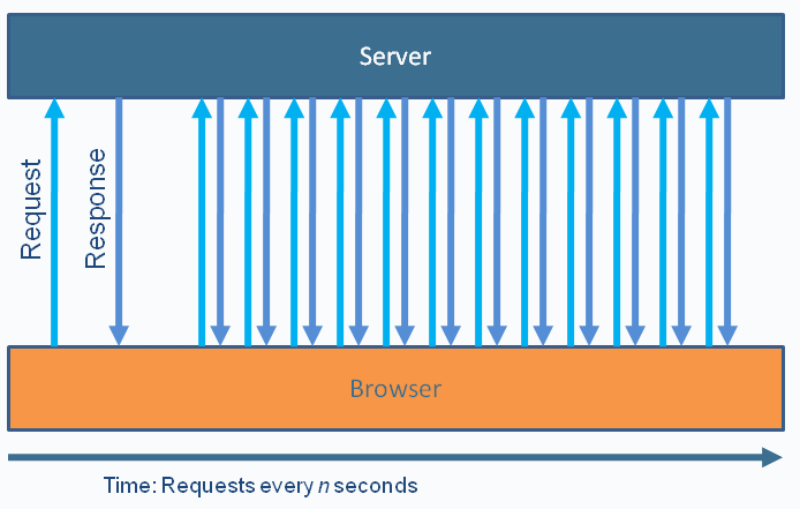
- HTTP is half-duplex
  - Traffic flows in only one direction at a time
  - Bidirectional communications are complicated to manage
- HTTP is stateless
  - Redundant information sent with each HTTP request and response

Emulating full-duplex HTTP

- AJAX (Asynchronous JavaScript + XML)
  - Content can change without loading the entire page
  - User-perceived low latency
- Comet
  - Technique for server push
  - Lack of a standard implementation
  - Comet adds lots of complexity

Polling

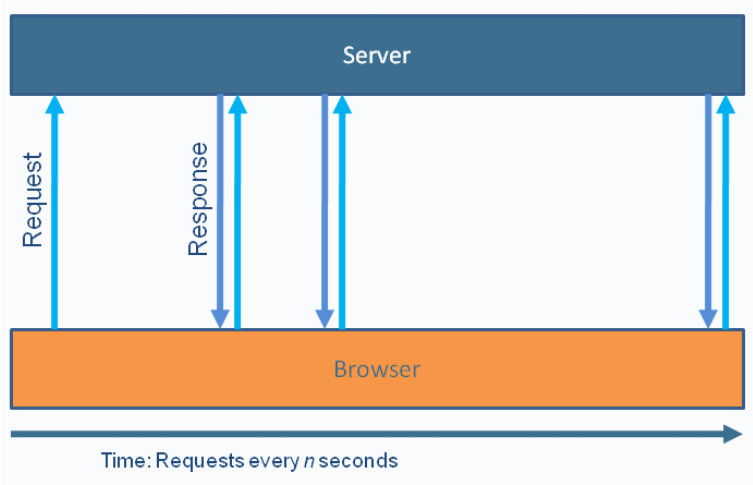
- Polling is "nearly real-time"
- Used in Ajax applications to simulate real-time communication
- Browser sends HTTP requests at regular intervals and immediately receives a response



Long Polling

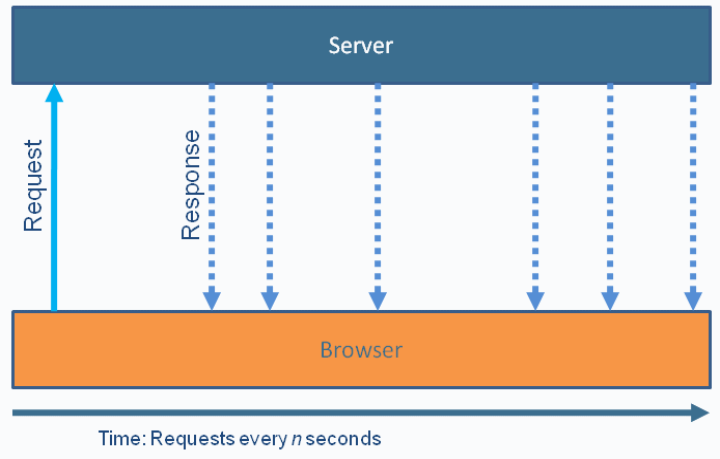
a/k/a Asynchronous polling

- Browser sends a request to the server, server keeps the request open for a set period
- Speed limited by response-request-response
- Request/response headers add overhead on the wire

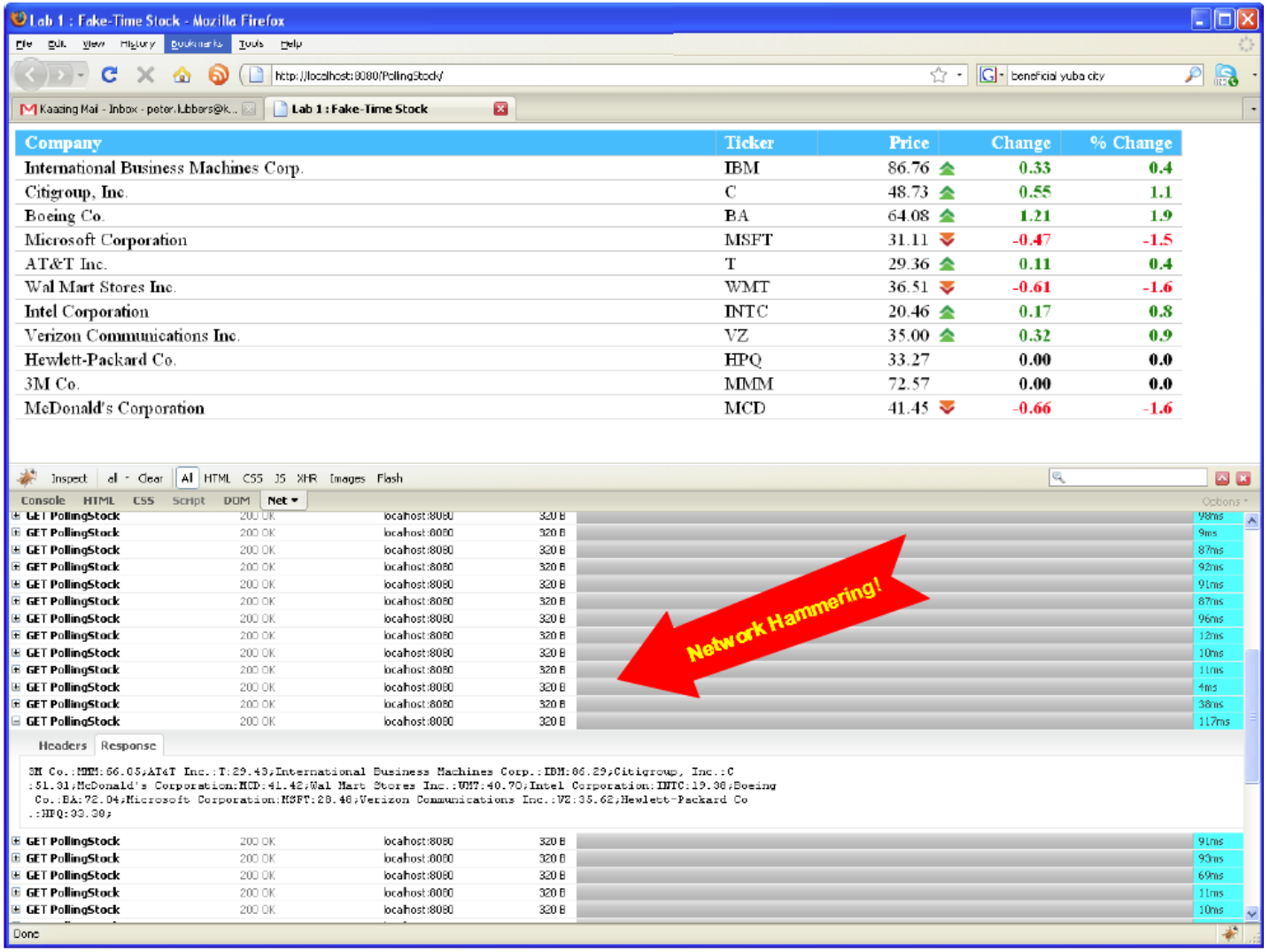


Streaming

- 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



Comet Polling Example



HTTP Request Headers

KAAZING

Client

```
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;
```

HTTP Response Headers

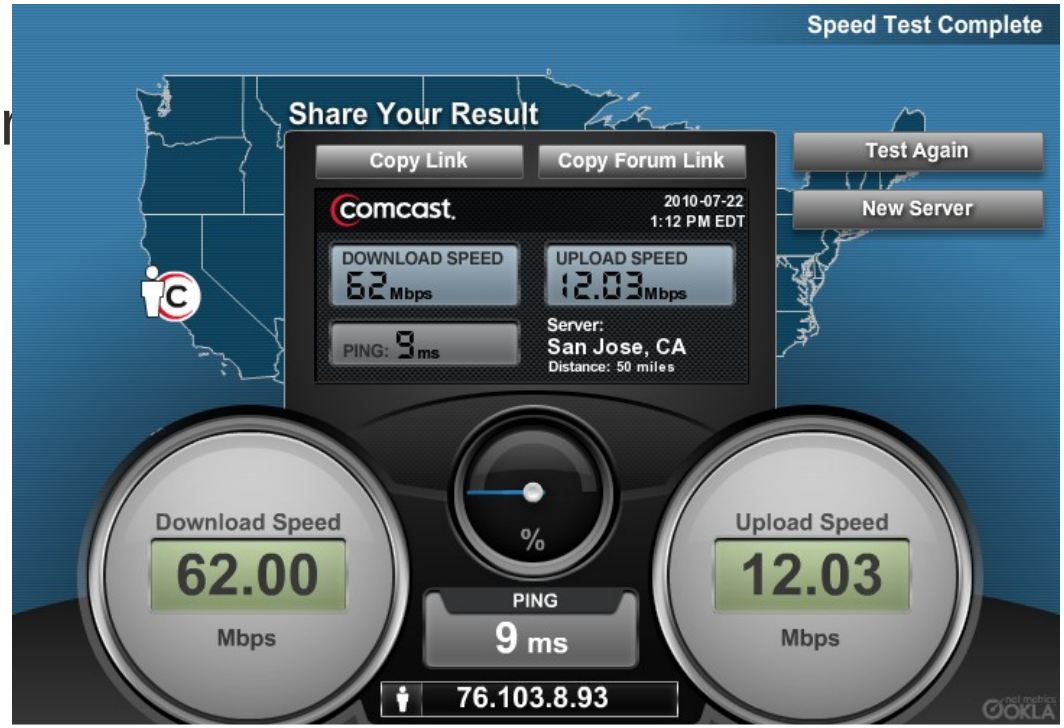
Server

```
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
```

- Total overhead: 871 bytes (example)
- Often 2K+ bytes
  - e.g. cookies

Upload/Download Ratios

- Most users have Internet connections where upload to download ratios are between 1:4 and 1:20
  - Result: 500 byte HTTP request header request could take as long to upload as 10 kB of HTTP response data takes to download



HTTP Header Traffic Analysis

Client	Overhead Bytes	Overhead Mbps
1,000	871,000	~6,6*
10,000	8,710,000	~66
100,000	87,100,000	~665

```
* 871,000 bytes = 6,968,000 bits = ~6.6 Mbps
```

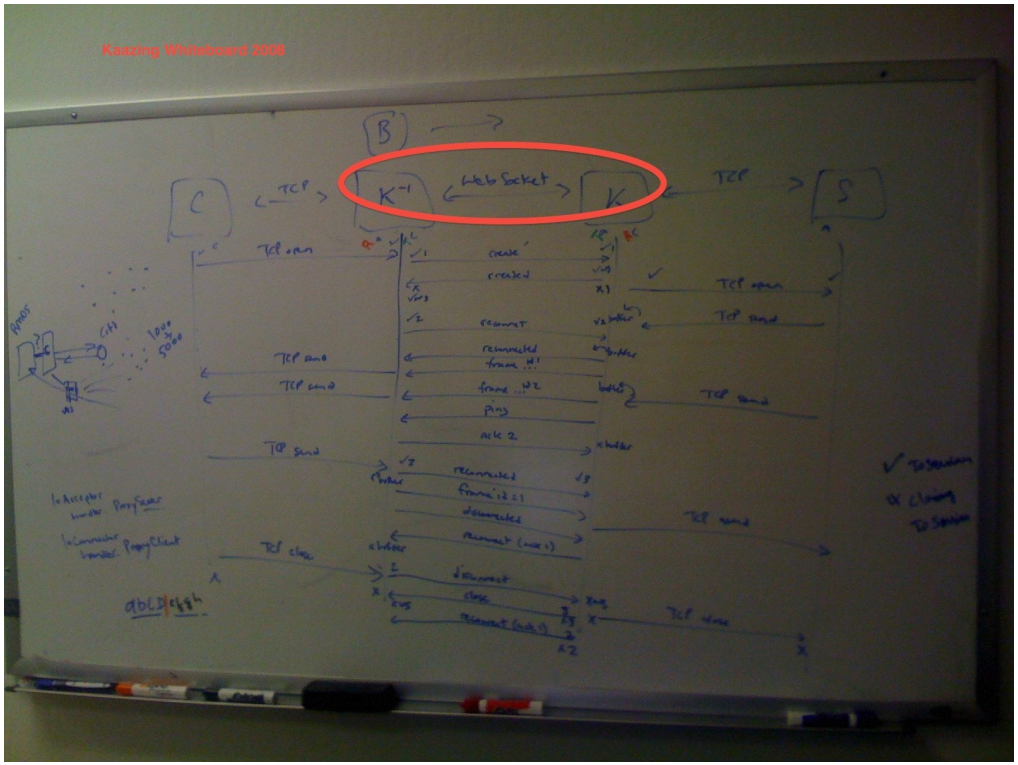
Overheard…

"R*educing 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)

WebSocket History

- Originally added to HTML5 Spec as TCPConnection
- Moved to its own specification



HTML5 WebSocket

- Cross-web communications w/ remote host
  - Full-duplex (bi-directional), single socket
  - Shares port with existing HTTP content
  - Traverses firewalls and proxies
  - ws:// and wss://
- W3C API (Javascript)
- IETF Protocol

USING THE WEBSOCKET API

Checking for support

JavaScript

```
var status =  document.getElementById("support");
if (window.WebSocket) { // or Modernizr.websocket
  status.innerHTML = "HTML5 WebSocket is supported";
} else {
  status.innerHTML = "HTML5 WebSocket is not supported";
}
```

Using the WebSocket API

KAAZING

JavaScript

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

// Associate listeners
mySocket.onopen = function(evt) {
};
mySocket.onclose = function(evt) {
  alert("closed w/ status: " + evt.code};
};
mySocket.onmessage = function(evt) {
  alert("Received message: " + evt.data);
};
mySocket.onerror = function(evt) {
  alert("Error);
};
```

Using the WebSocket API

KAAZING

JavaScript

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

// Close WebSocket
mySocket.close();
```

WebSocket API

Available ?	`window.WebSocket` or `Modernizr.websocket`
Events	`onopen`, `onerror`, `onmessage`
Functions	`send`, `close`
Attributes	`url`, `readyState`, `bufferedAmount`, *`extensions`*, *`protocol`*

http://dev.w3.org/html5/websockets/

Italics: -08 and later

Browser Support

Native:

- Chrome 4+
- Safari 5+
- Firefox 4+
- Opera 10.7+
- Internet Explorer 10+

Emulation:

- Kaazing WebSocket Gateway
- socket.io
- SockJS
- web-socket.js (Flash)

http://caniuse.com/#search=WebSocket





WebSocket Servers Libraries

- Kaazing
- Socket.io (node.js)
- Apache-websocket
- Cramp
- Nowjs
- SockJS
- SuperWebSocket
- Jetty
- Atmosphere
- APE Project
- Xsockets
- Orbited
- Atmosphere
- Autobahn
- CouchDB
- Netty
- Misultin
- Cowboy
- YAWS
- Juggernaut
- PHP Websocket
- websockify
- ActiveMQ
- HornetMQ
- phpwebsocket
- Protocol::WebSocket
- em-websocket
- Jwebsocket
- WaterSprout Server
- Pywebsocket
- And more…

- **Client Libraries**
- Web-socket-js (JavaScript)
- AS3 WebSocket (ActionScript)
- .NET WebSocket client (.NET)
- Anaida (.NET)
- WebSocket Sharp (.NET)
- Silverlight WebSocket client
- Java WebSocket Client
- Arduino C++ WebSocket client
- Ruby-web-socket
- ZTWebSocket (Objective-C)
- Libwebsockets (C)

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
- Socket.io
  - http://socket.io
  - Alternate API
  - Adds heartbeat, timeouts and disconnection
  - Uses Flash when available

THE WEBSOCKET PROTOCOL

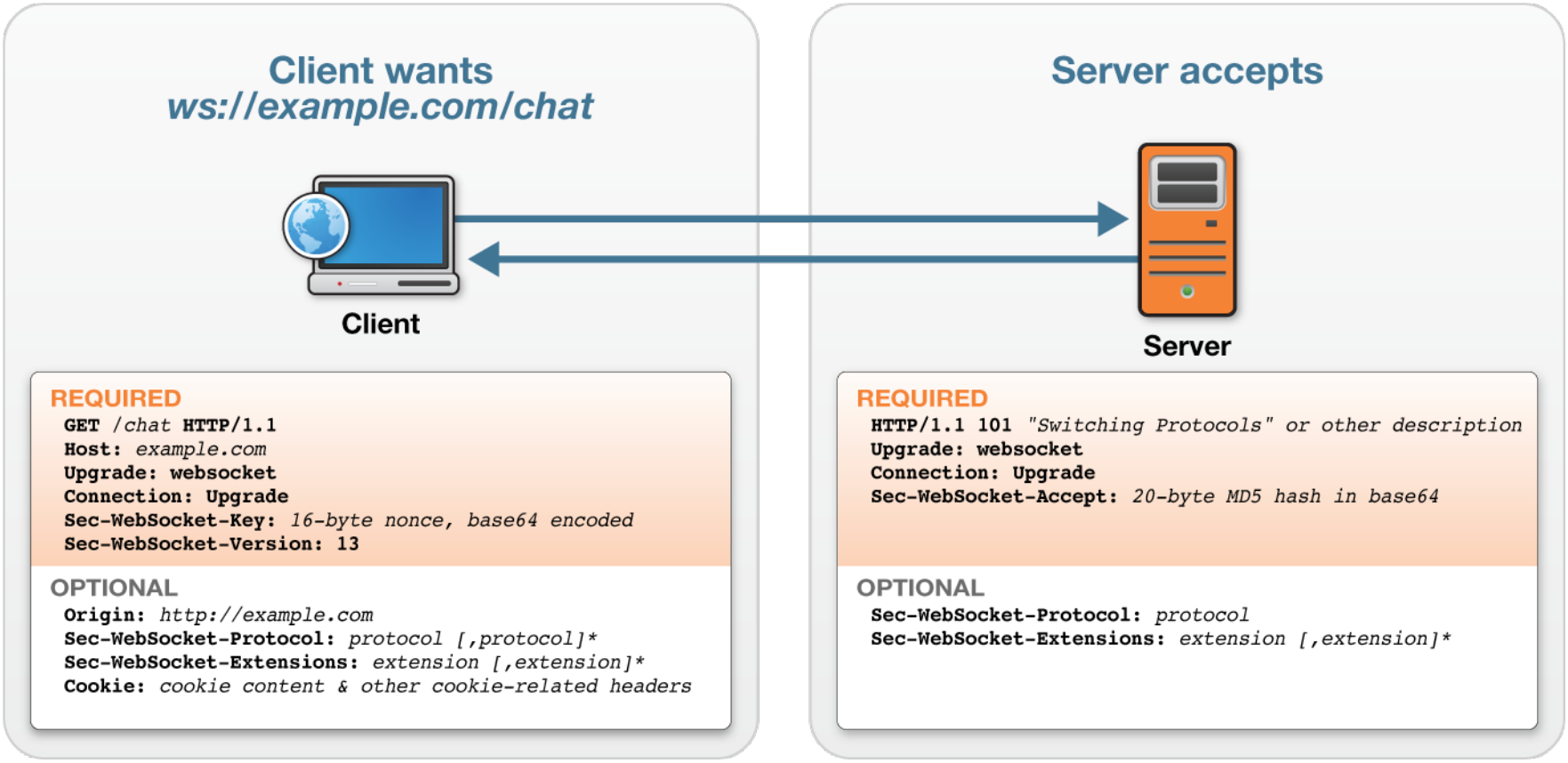
"draft-hixie-thewebsocketprotocol-*xx*" IETF Network Working Group

Version	Date	Details
-00	Jan. 9 2009	• Initial version
-52	Oct. 23 2009	• Subprotocol concept introduced
-76	May 6 2010	• Used in older browsers (FF4, etc.)

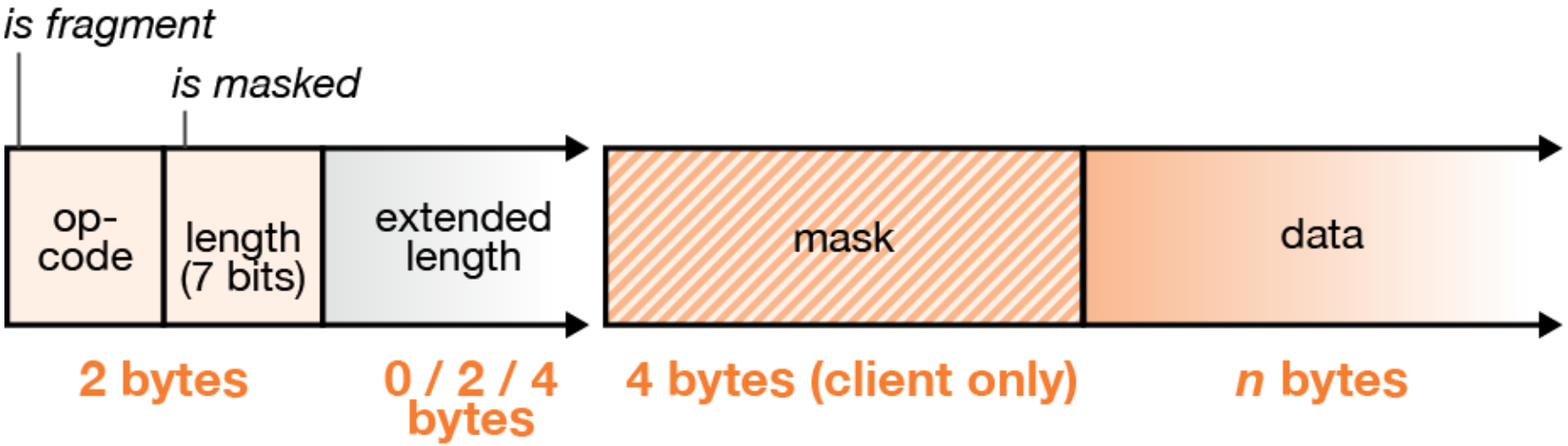
"draft-ietf-hybi-thewebsocketprotocol-*xx*" (IETF HyBi Working Group)

Version	Date	Details
-01	Aug. 31 2010	• Added binary format
-04	Jan. 11 2011	• Introduced data masking to address proxy server security issue • Introduced including protocol version number in handshake
-14	Sep. 8 2011	• Guidance on version number negotiation
RFC 6455	Dec. 2011	• Final version http://tools.ietf.org/html/rfc6455

WebSocket Protocol History

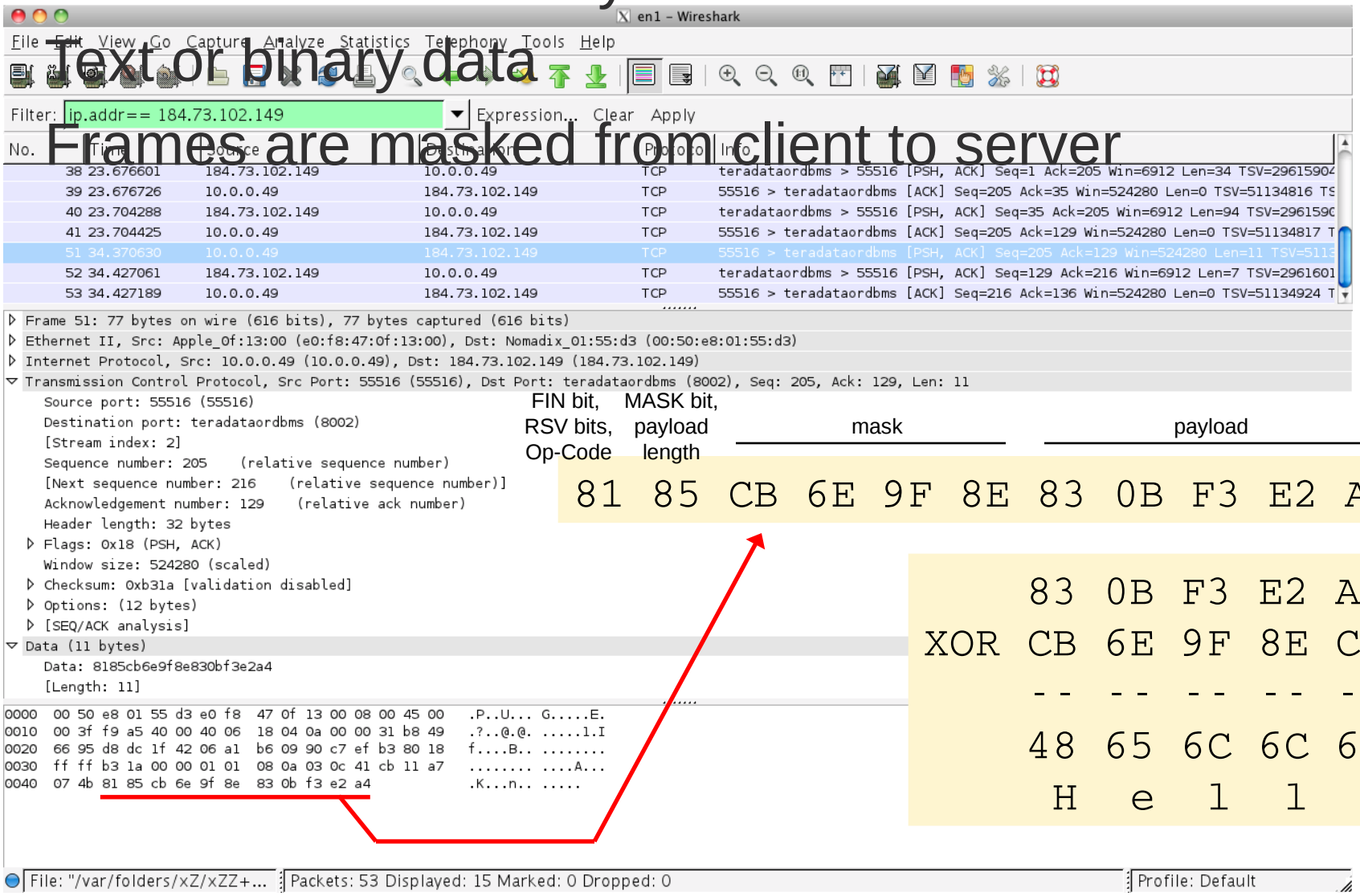


WebSocket Handshake



WebSocket Frames

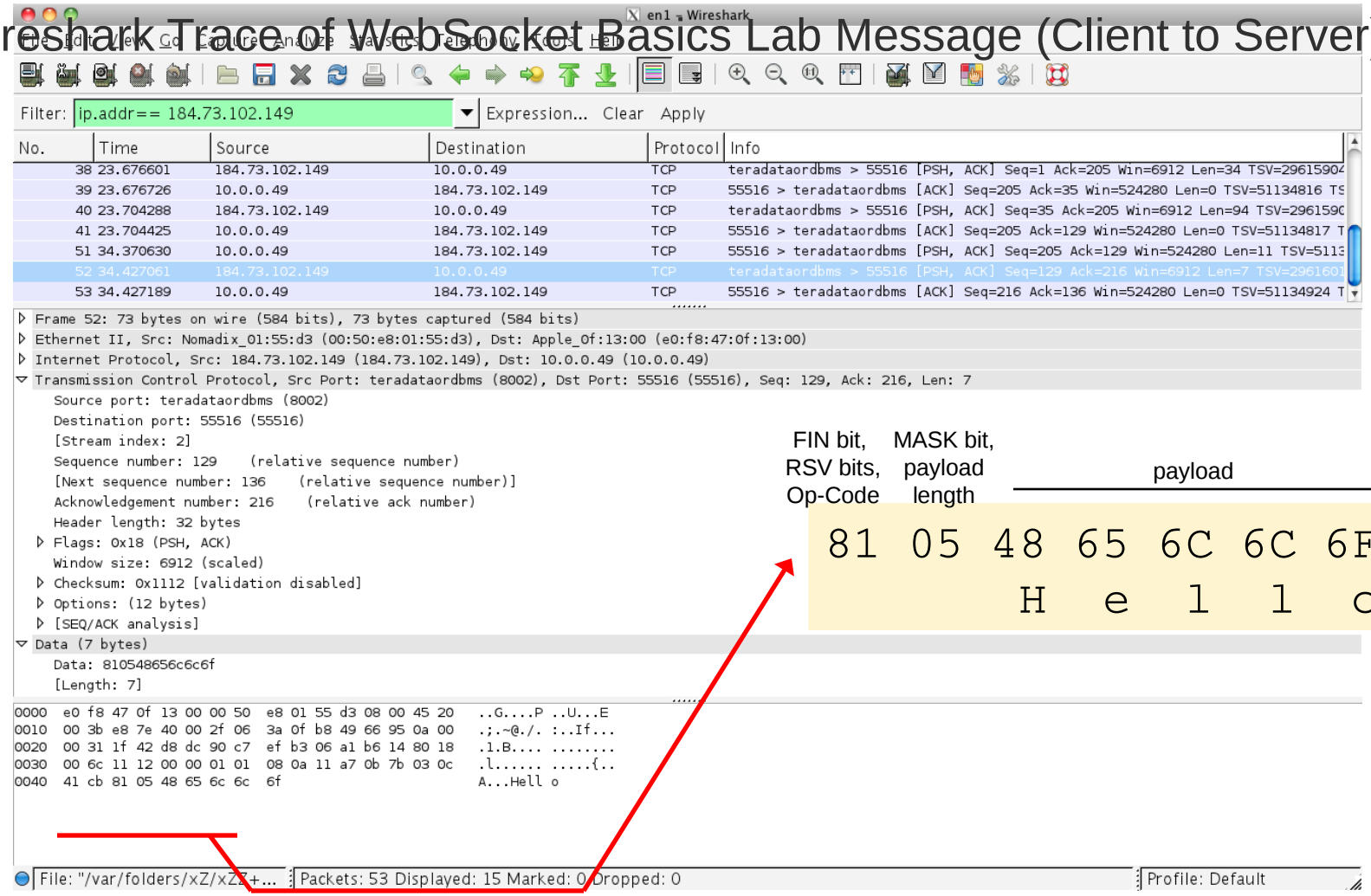
- Have a few header bytes
- Text or binary data
- Frames are masked from client to server



WebSocket Frames

Wireshark Trace of WebSocket Basics Lab Message (Server to Client)

Wireshark Trace of WebSocket Basics Lab Message (Client to Server)



WebSocket Efficiency

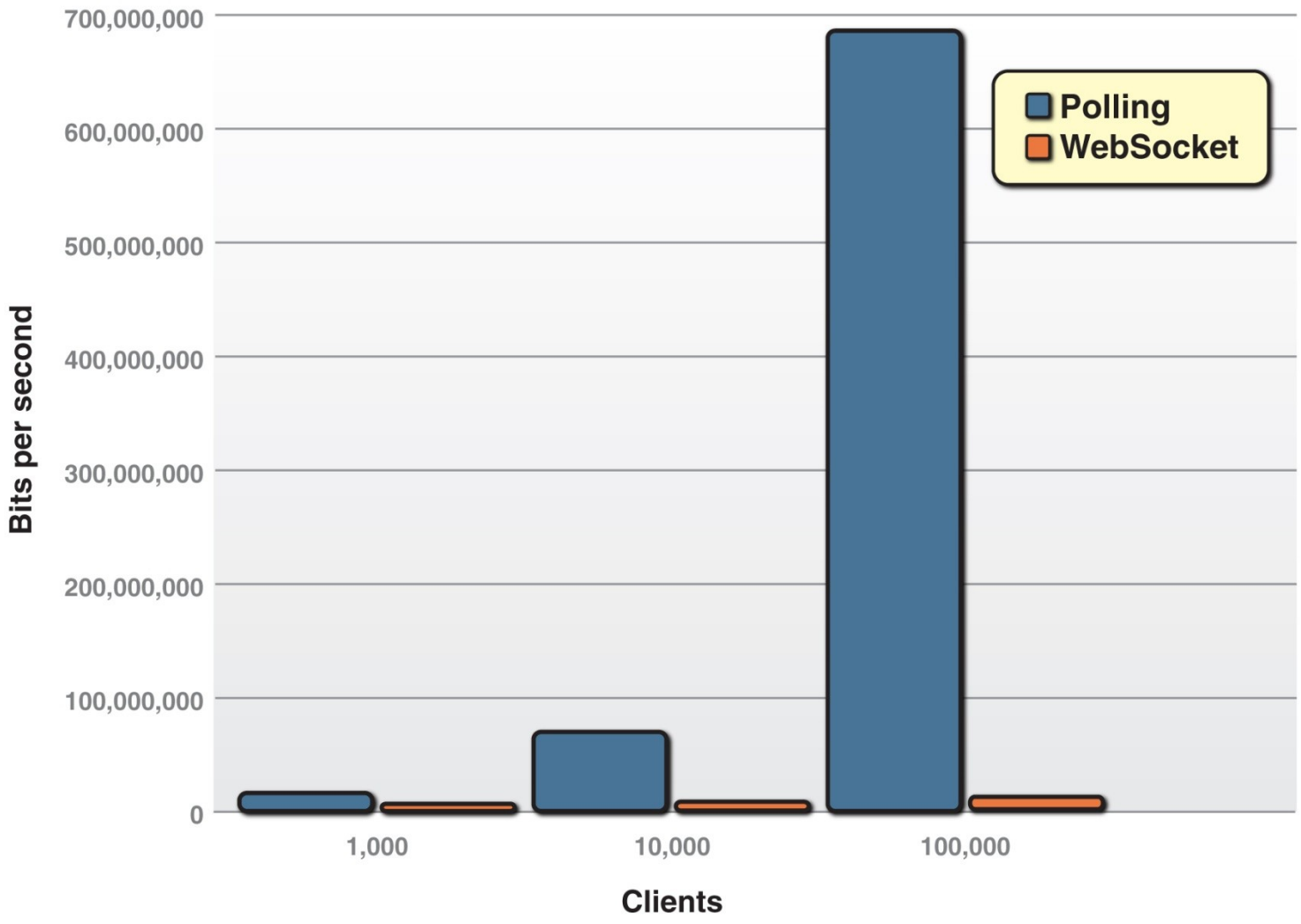
	HTTP	WebSocket
Overhead	100s of bytes	2-6 bytes (typical)
Latency	New connection each time	None: Use existing connection
Latency (polling)	Wait for next interval	No waiting
Latency (long polling)	None, if request sent earlier + time to set up next request	No waiting

WebSocket Framing Analysis

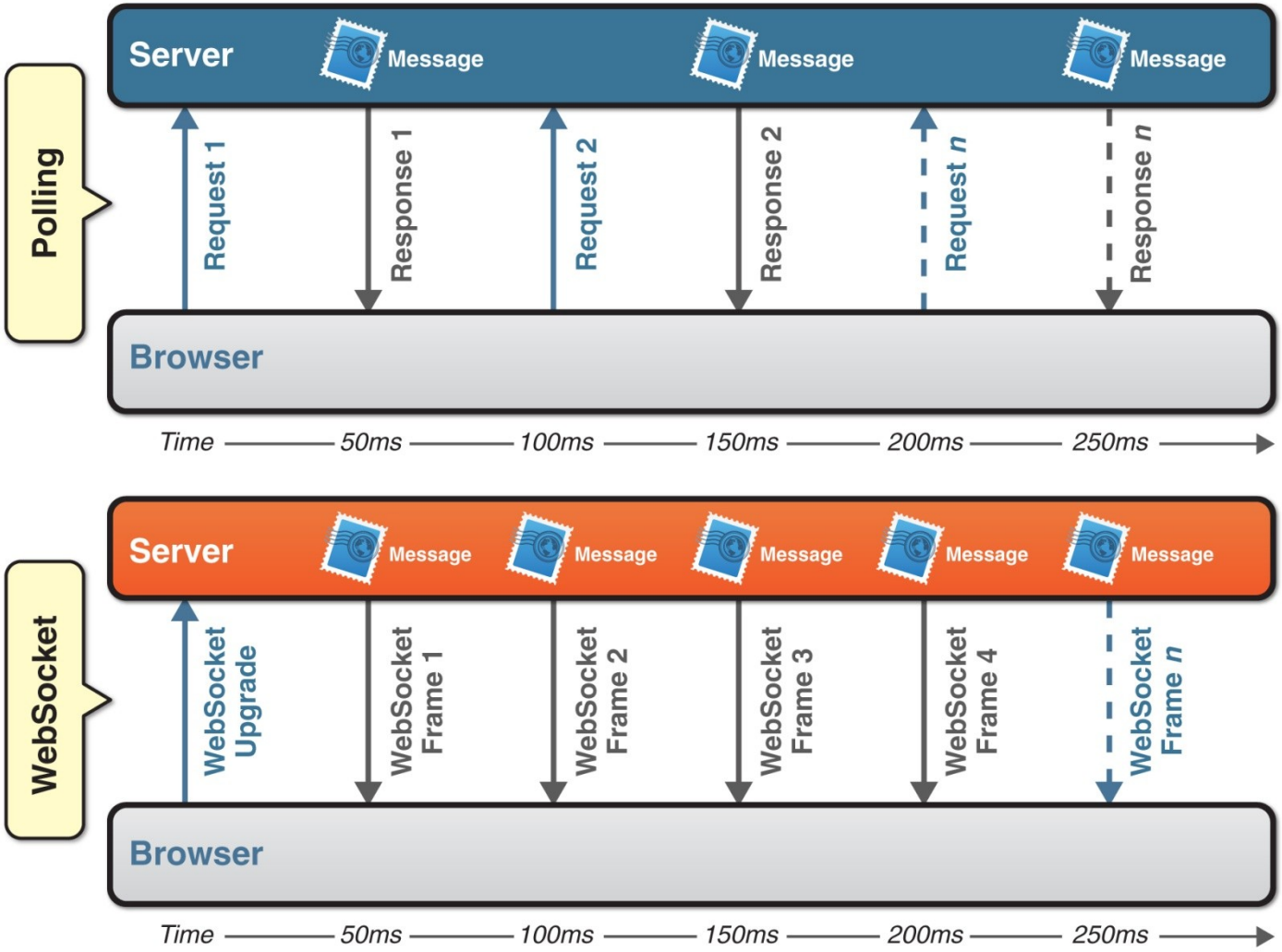
Client	Overhead Bytes	Overhead Mbps
1,000	2,000	~0.015*
10,000	20,000	~0.153
100,000	200,000	~1.526

```
* 2,000 bytes = 16,000 bits (~0.015 Mbps)
```

Polling vs. WebSocket



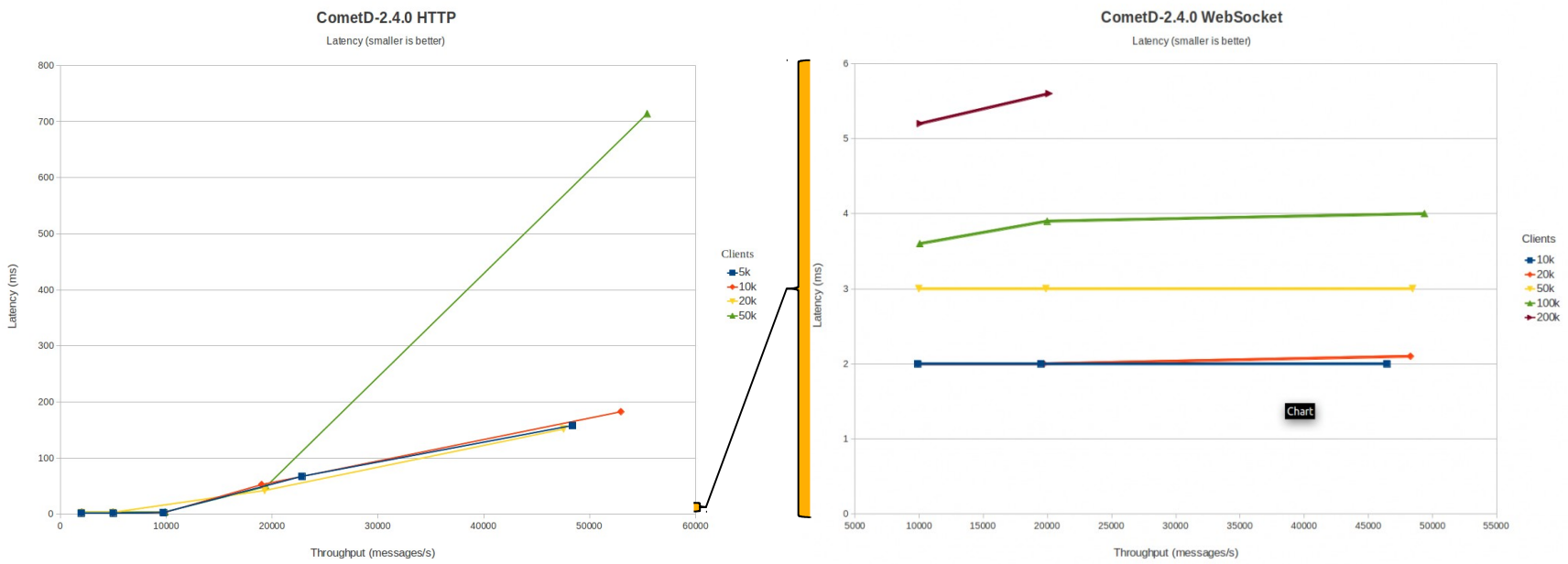
Latency Reduction



WebSocket Benchmark

Using Comet

Using WebSocket



http://webtide.intalio.com/2011/09/cometd-2-4-0-websocket-benchmarks/

WebSocket Summary

- Extends network applications across the web
  - Desktop apps
  - Browser-based apps
  - Mobile apps
- *Far* more efficient than HTTP
- Part of the HTML5 Standard
- Older browsers can play too

Using WebSocket…

- Extends legacy systems to the web
  - Message brokers, databases, etc.
- Extends client-server protocols to the web:
  - XMPP, Jabber
  - Pub/Sub (Stomp/AMQP)
  - Gaming protocols
  - Any TCP-based protocol
  - RFB/VNC
- Your browser becomes a first-class network citizen

“Why?”

- Better responsiveness
- Better scalability
  - Less traffic on the wire
  - Less work for the server
- Easier back-end development
  - Custom commands = better match to your needs
- Easier migration of existing systems
  - Just a new UI

WEBSOCKETS IN THE REAL WORLD

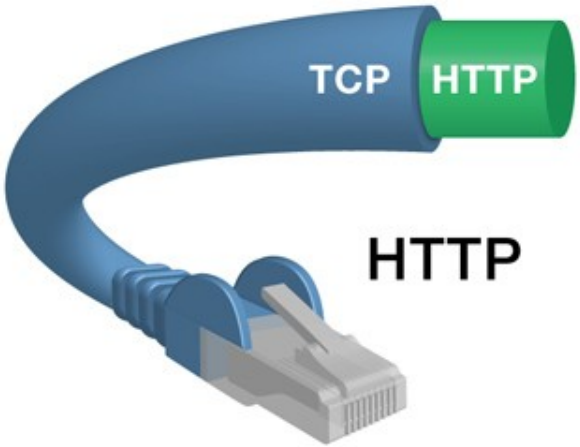
Security

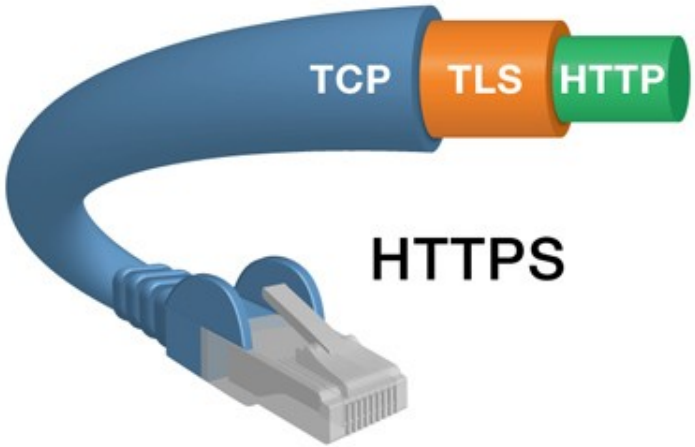
- Components of secure communication :
  1. Transport Layer
  2. Authentication
  3. Authorization
  4. Origin-Based Security and CORS

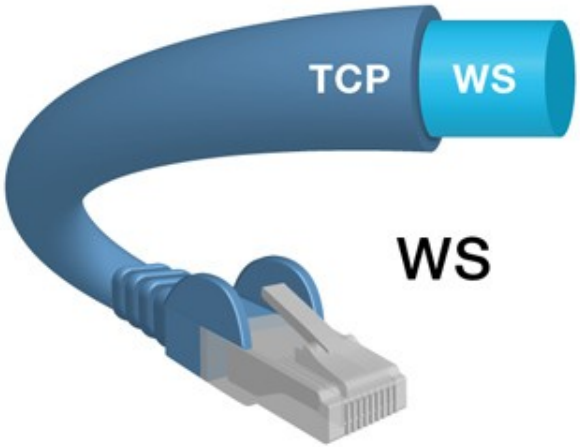
Transport Layer Security (TLS)

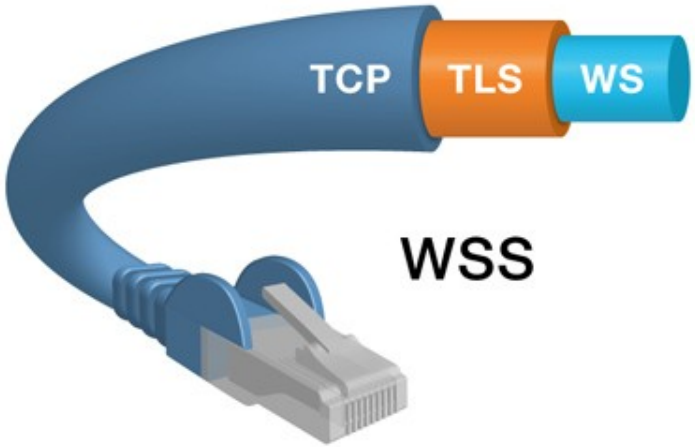
- Also known as SSL (Secure Socket Layer) support
- HTTP over TLS is called HTTPS
  - Default port is 443
  - HTTPS is not a separate protocol
  - An HTTPS connection is established after a successful TLS handshake (using public and private key certificates)

ws:// and wss:// schemes









Authentication

- Mechanism by which systems identify users and check whether users really are who they represent themselves to be

- Authentication process

  Step 1) Server issues a challenge using the `HTTP 401 Authorization Required` code

  Step 2) Client responds by providing the requested authentication information if it can

Authorization

- Mechanism by which a system determines users' level of access
  - For example, a web page can have viewer, moderator, and administrator privileges
- Access rights are typically stored in the policy store that is associated with the application

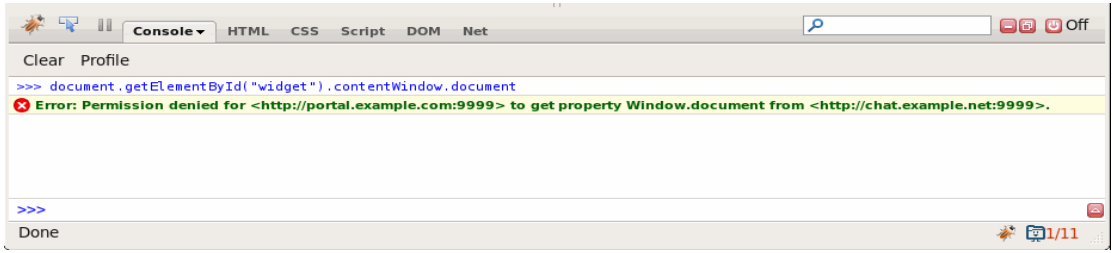
Web Origin Concept

- Web Origin Concept RFC 6454: http://www.ietf.org/rfc/rfc6454.txt
- An Origin is a subset of an address used for modeling trust relationships on the web
- Origins consist of a scheme, a host, and a port:
  - **Scheme**: `http:, https:, ws:, wss:`
  - **Host**: `www.example.com, img.example.com, 192.0.2.10`
  - **Port**: `80, 443`

Same-Origin Policy

"Generally speaking, documents retrieved from distinct origins are isolated from each other" – W3C
http://www.w3.org/Security/wiki/Same_Origin_Policy

- Browsers prevent a script or document loaded from one origin from communicating with a document loaded from another origin
- Original security model for HTML
  - Introduced in Netscape Navigator 2.0
  - Too limiting in this age of client-side web apps



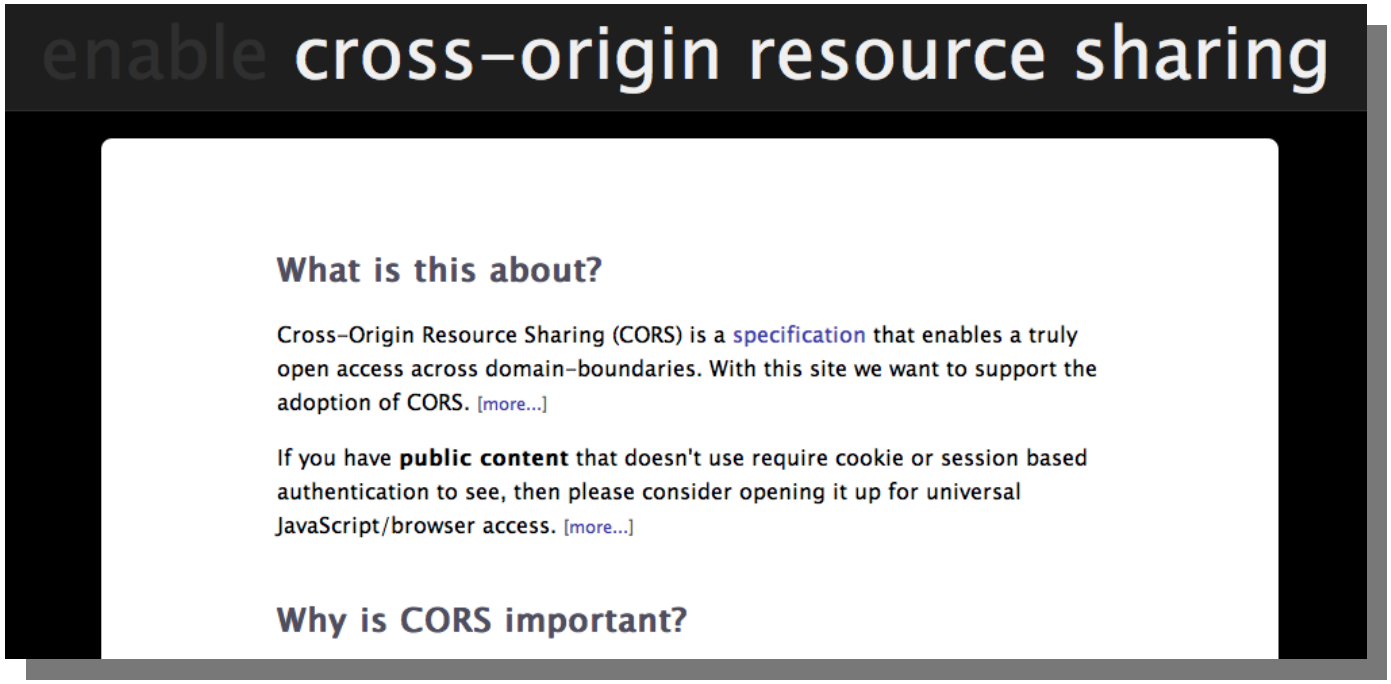
Origin Quiz

Which URLs have the same Origin?

1. `http://www.example.com/index.html`
2. `https://www.example.com/index.html`
3. `http://img.example.com/html5.png`
4. `http://www.example.com:8080/page2.html`
5. `http://192.0.2.10:80/index.html*`
6. `http://www.example.com/about.html`

* Where 192.0.2.10 resolves to www.example.com

CORS

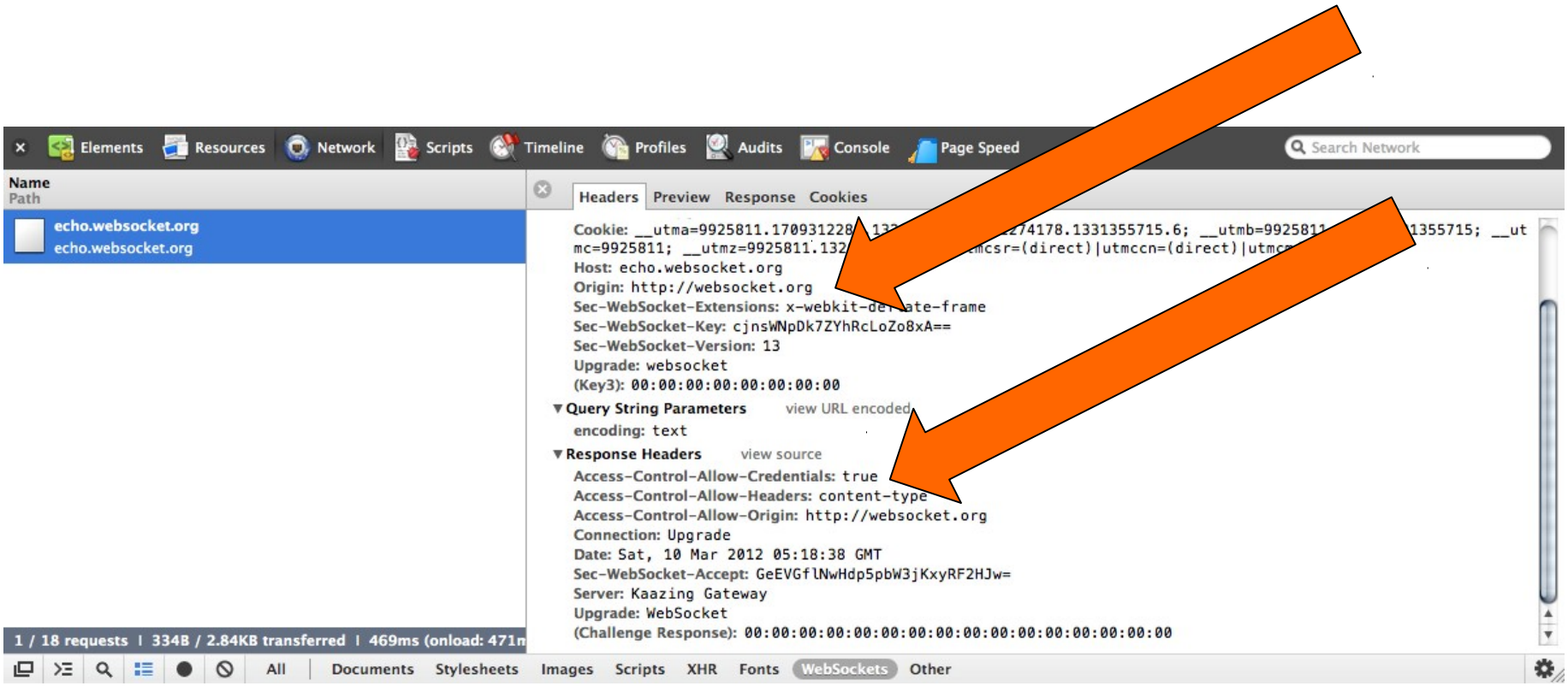


enable-cors.org

Cross-Origin Requests

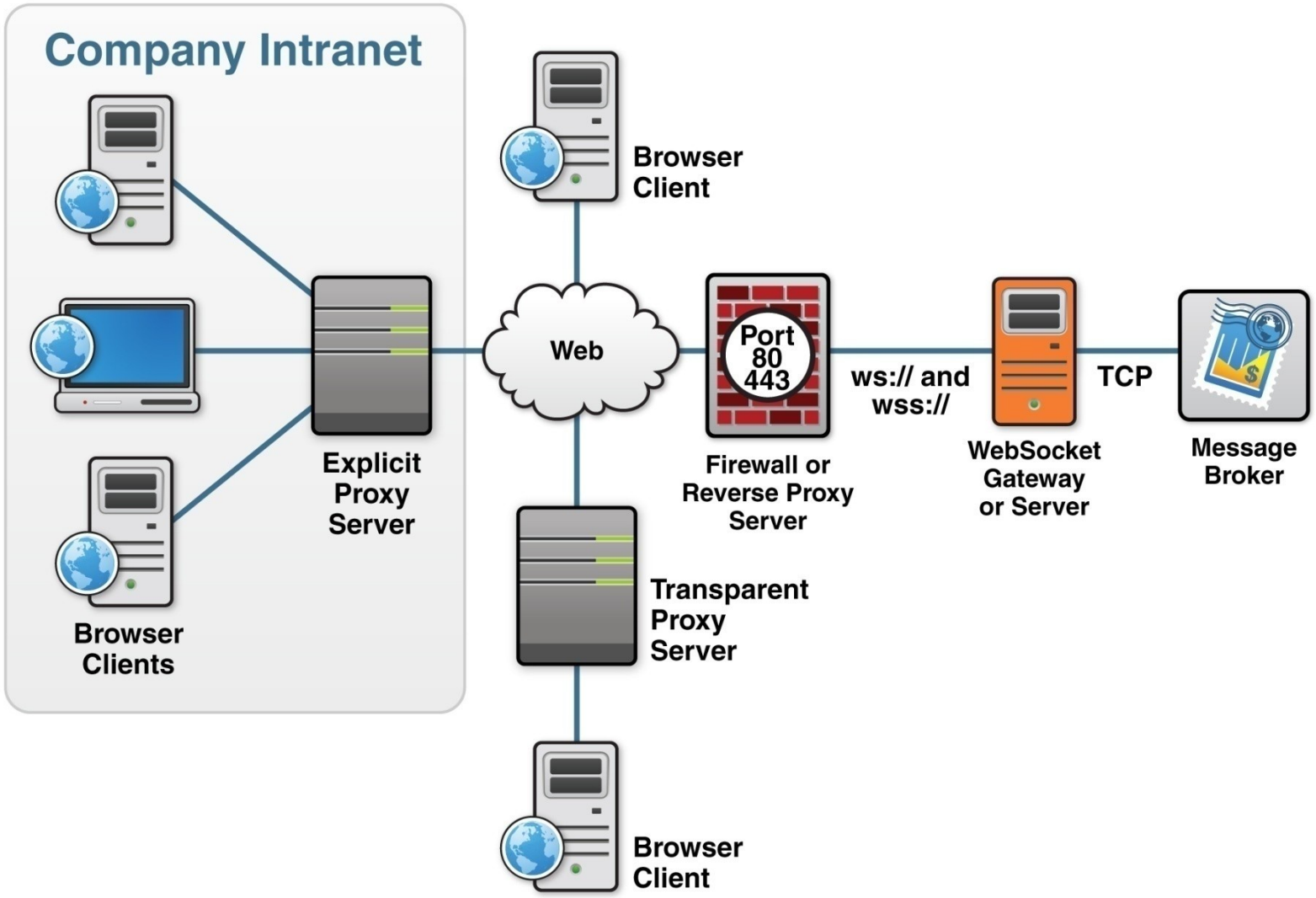
- Have an **Origin** header
  - Contains the request's origin
  - Produced by the browser
  - Cannot be changed by application code
  - Differs from **referer** [*sic*]: **referer** is a complete URL (can include full path)
- Originating page's server must approve (**Access-Control-Allow-* headers**)

CORS



Intermediaries

Types of Proxy Servers



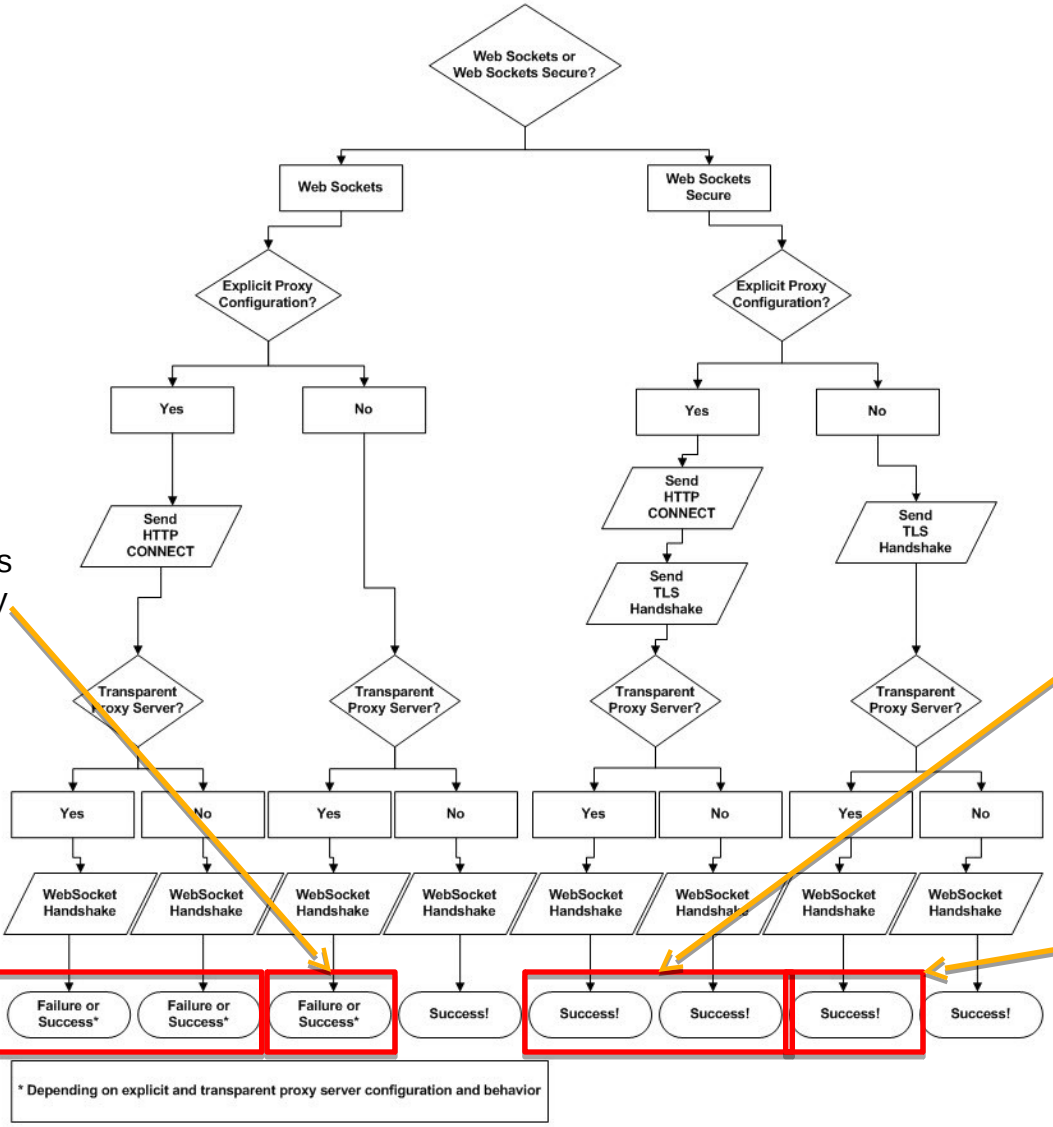
http://www.infoq.com/articles/Web-Sockets-Proxy-Servers

Proxy Servers

- WebSocket protocol is unaware of proxy servers and firewalls
  - HTTP-based handshake

Proxy type	Connection	Outcome
Explicit	Unencrypted	1. HTTP CONNECT 2. WebSocket connection flows to destination
Explicit	Encrypted	
Transparent	Unencrypted	Proxy strips extra headers, connection fails
Transparent	Encrypted	Connection tunnels past proxy

Proxy Traversal Tree



Case 2 Unencrypted WebSocket Connections and **Transparent** Proxy Servers

Case 1 Unencrypted WebSocket Connections and **Explicit** Proxy Servers

Case 3 Encrypted WebSocket Connections and **Explicit** Proxy Servers

Case 4 Encrypted WebSocket Connections and **Transparent** Proxy Servers

Load Balancing Routers

1. **TCP** (layer-4) work well with WebSockets
2. **HTTP** (Layer 7) expect HTTP traffic, can get confused by WebSocket upgrade traffic. May need to be configured to be explicitly aware of WebSocket traffic.

Firewalls

- Usually no specific WebSocket concerns
- Stateful packet inspection may need to know about WS protocol (or use WSS)