HTML5 Communications

The New Network Framework for the Web

Frank Greco

Kaazing Corporation

Goal

Describe the components of the W3C standard for Web Communications and how they can be used in applications that communicate over the Web.

Agenda

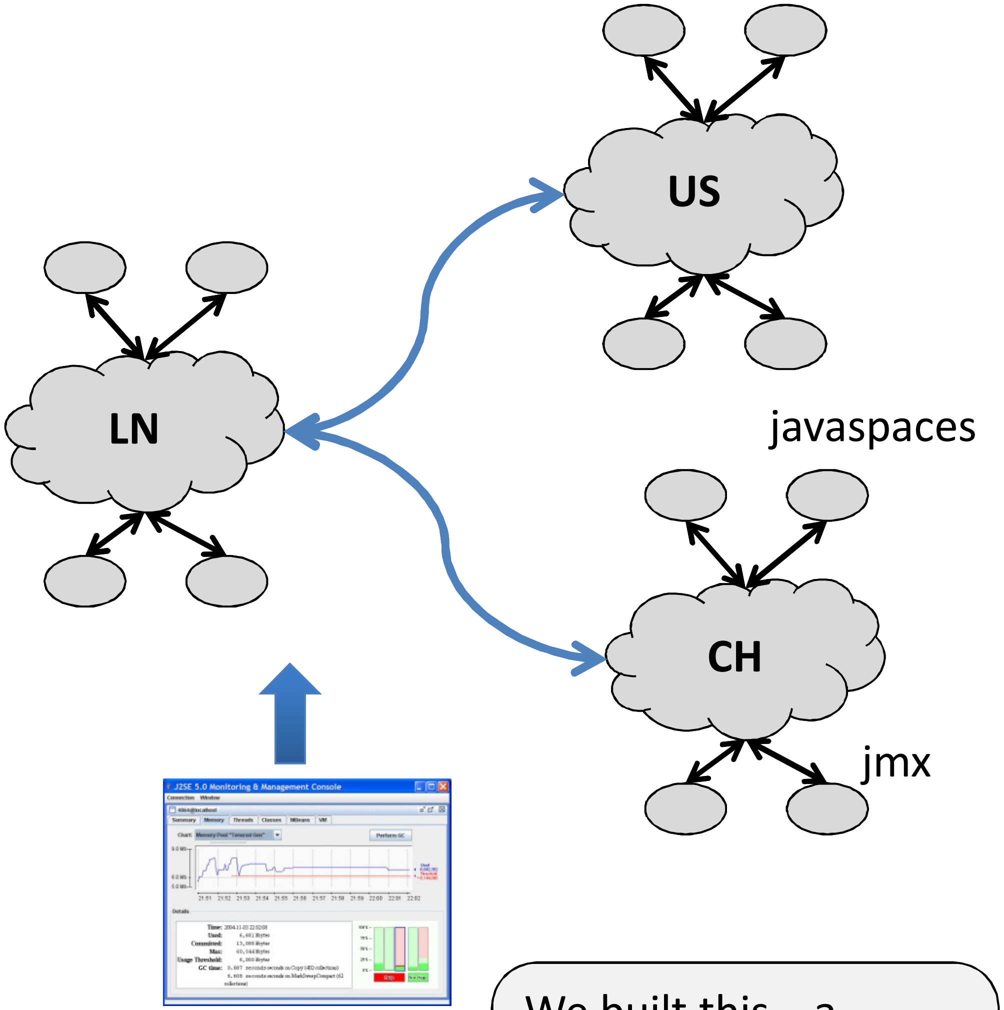
- Why I Needed New Plumbing...
- The Web, Sockets and “Real-Time” *(event-driven)*
- The W3C/HTML5/IETF Standards
- Websockets, Server-Sent Events and Cross-Domain Communications
- Comparison to Current Techniques
- Implementation
- Deployment Architectures
- What’s Next for the Web

Just Who is Frank Greco?

- Director of Technology, Americas - Kaazing
- Java Champion
- Chair of NYJavaSIG, First and Largest Java User Group in North America (6k+ members)
- Senior Architect - Large Distributed Applications in Financial Services
- Frequent Presenter on Cloud Computing / HPC

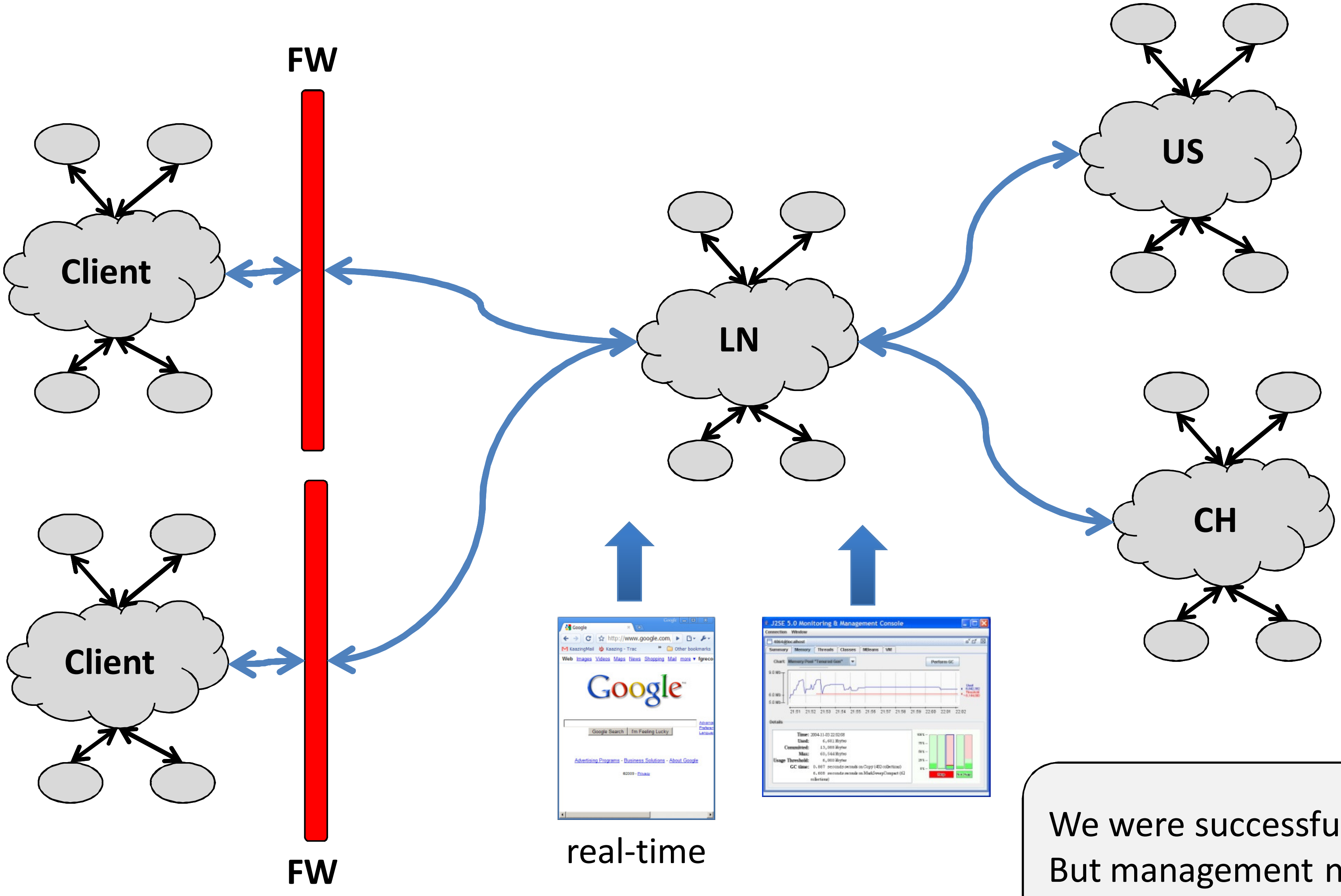
Extending an Architecture over a Web Infrastructure

Real-life use case...

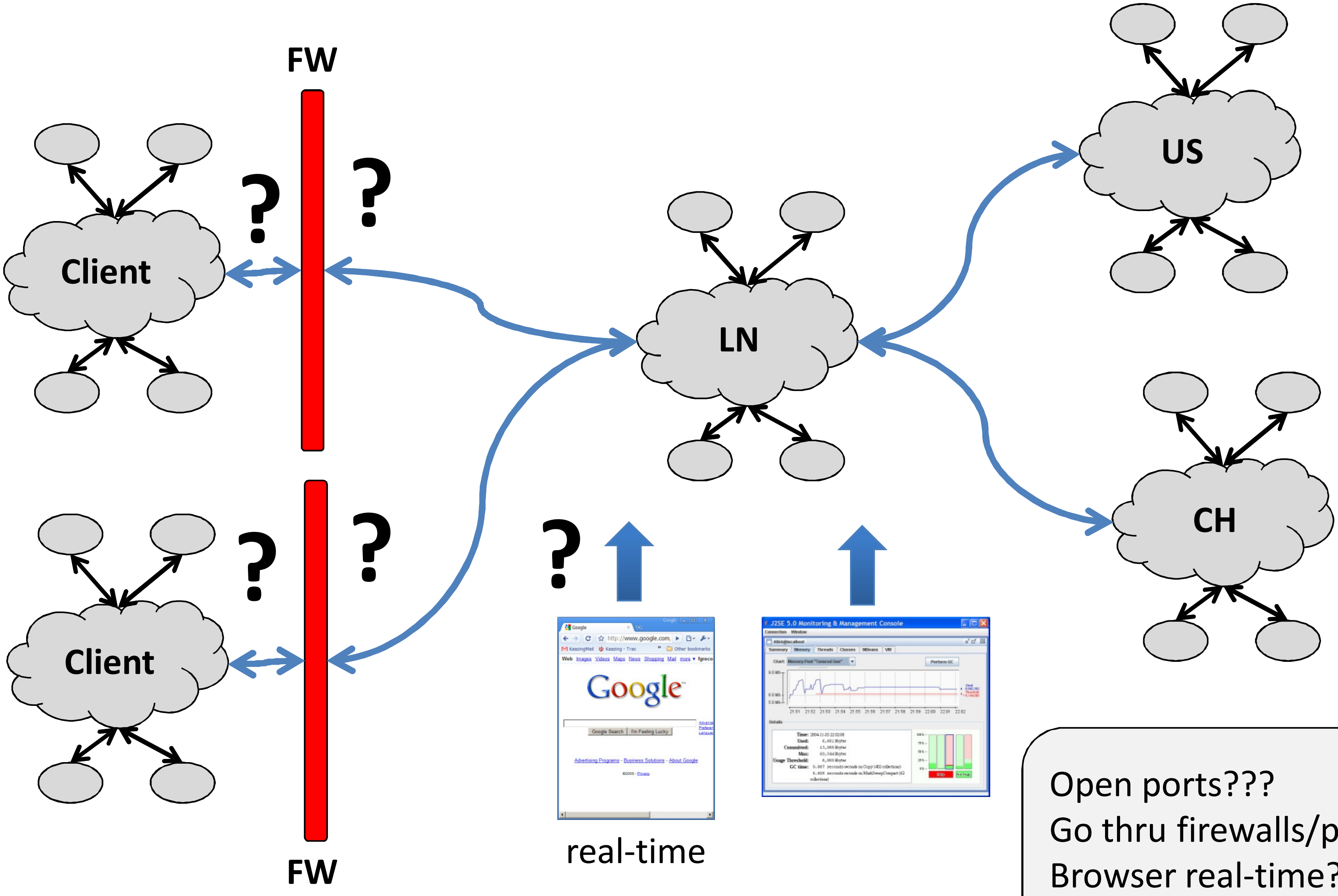


Time: 2004-11-03 22:02:08
Used: 6,681 kbytes
Committed: 13,088 kbytes
Max: 60,544 kbytes
Usage Threshold: 6,000 kbytes
GC time: 0.887 seconds seconds on Copy (402 collections)
6.608 seconds seconds on MarkSweepCompact (62 collections)

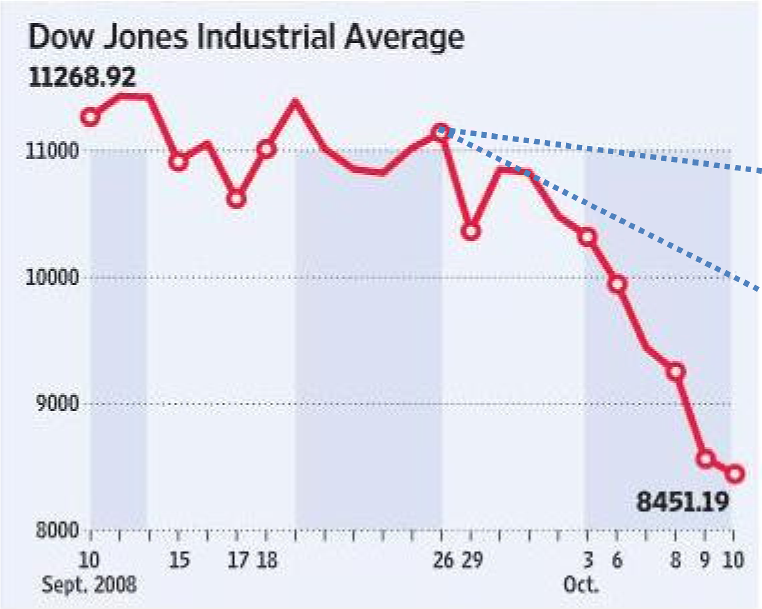
We built this… a super, duper, global services management system.



We were successful! But management now wanted this…



Open ports???
Go thru firewalls/px??
Browser real-time??
Full protocol??

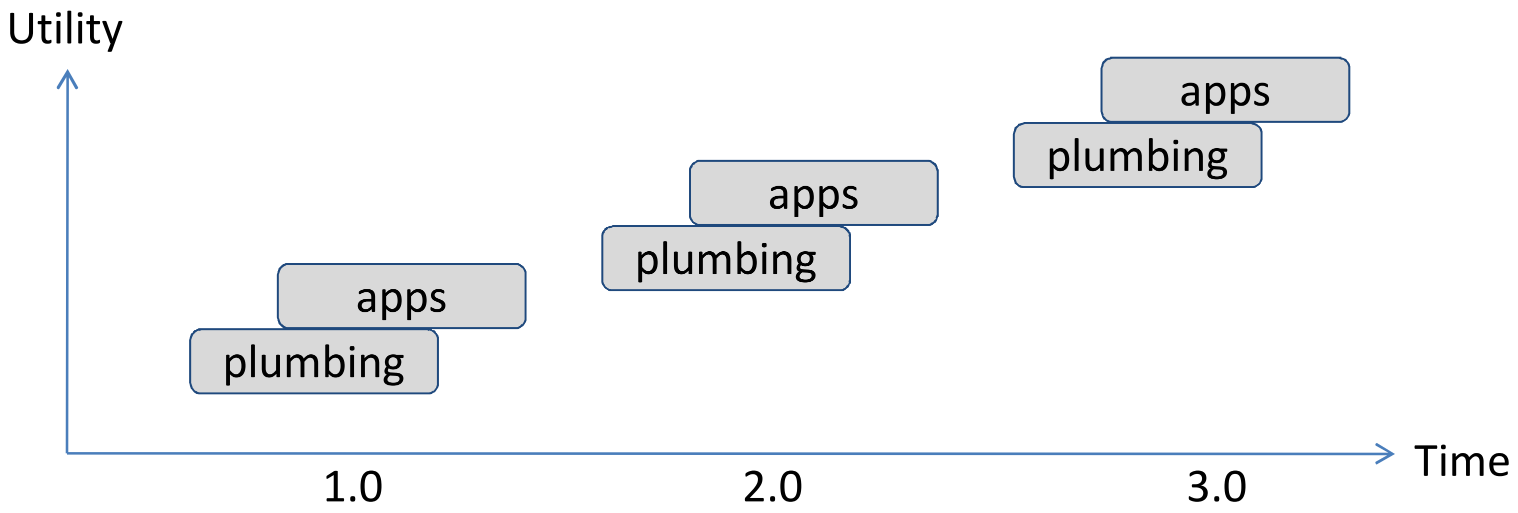


Frank invested in stock market

Therefore not necessary for Frank to complete project…

Web Waves

- Web 1.0 – Static Pages
- Web 2.0 – Dynamic Generation of pages
- Web 3.0 – Read/Write Web, Event-Driven
  - – Reactivate TCP protocols over the Web

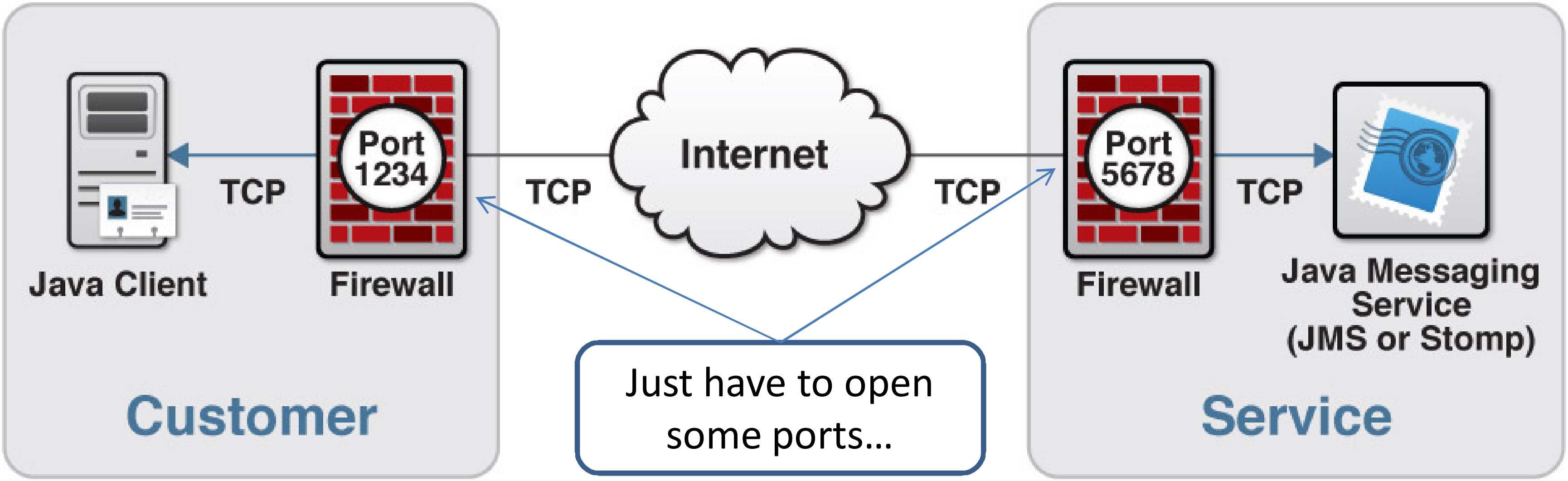


Today’s Requirements

- Web apps demand reliable, “real-time” communications with no additional latency
  - Financial applications
  - Social networking applications
  - Online games
  - Collaborative environments
  - Embedded computing (HVAC, security, electricity)
  - Mobile
  - Et al…

You’re asked to create an app that needs real-time publish/subscribe data, perhaps in a browser…

Client-Server Architecture

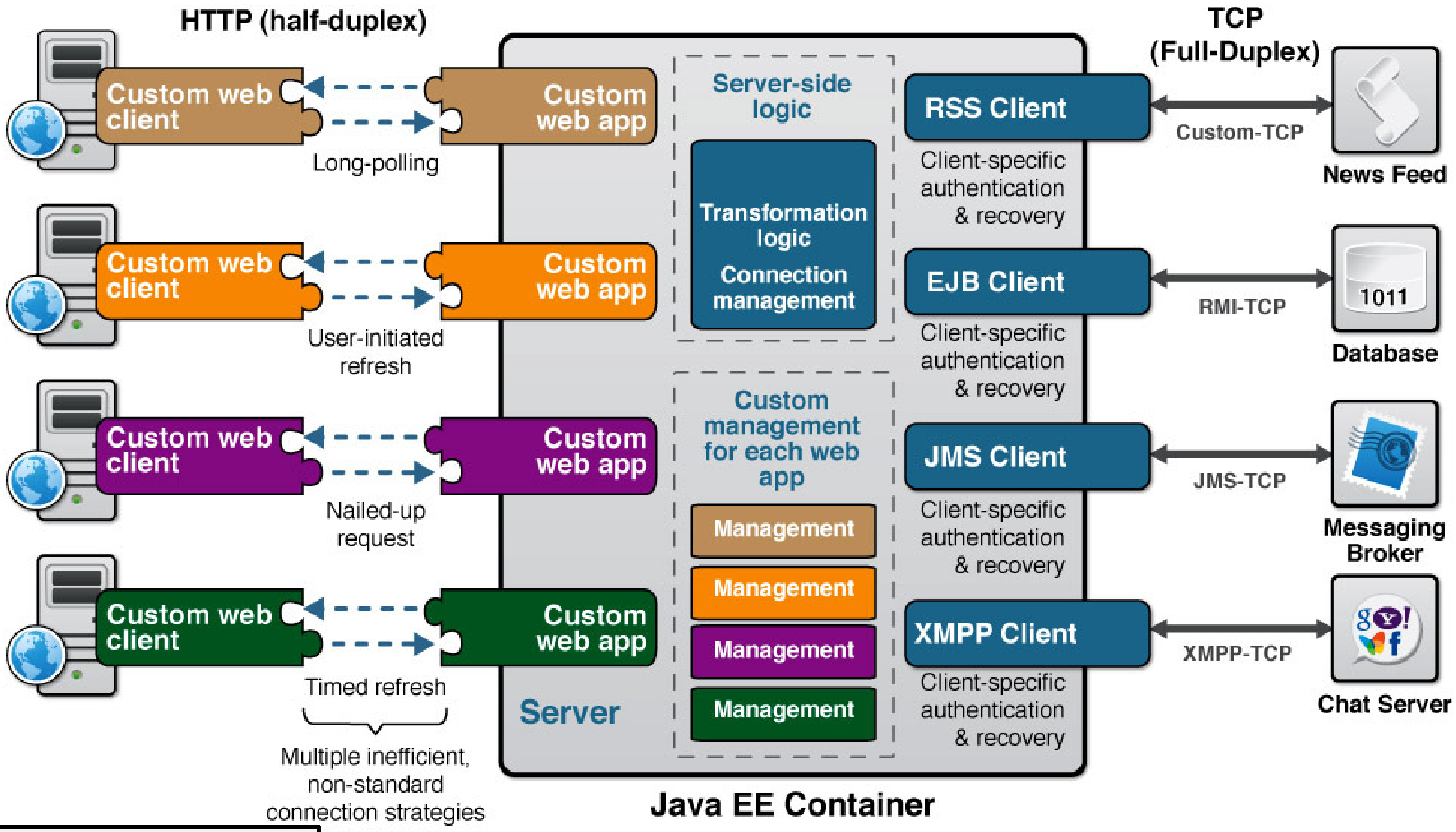


Just have to open some ports…

Use Java on client…

Need to traverse several proxies between the endpoints…

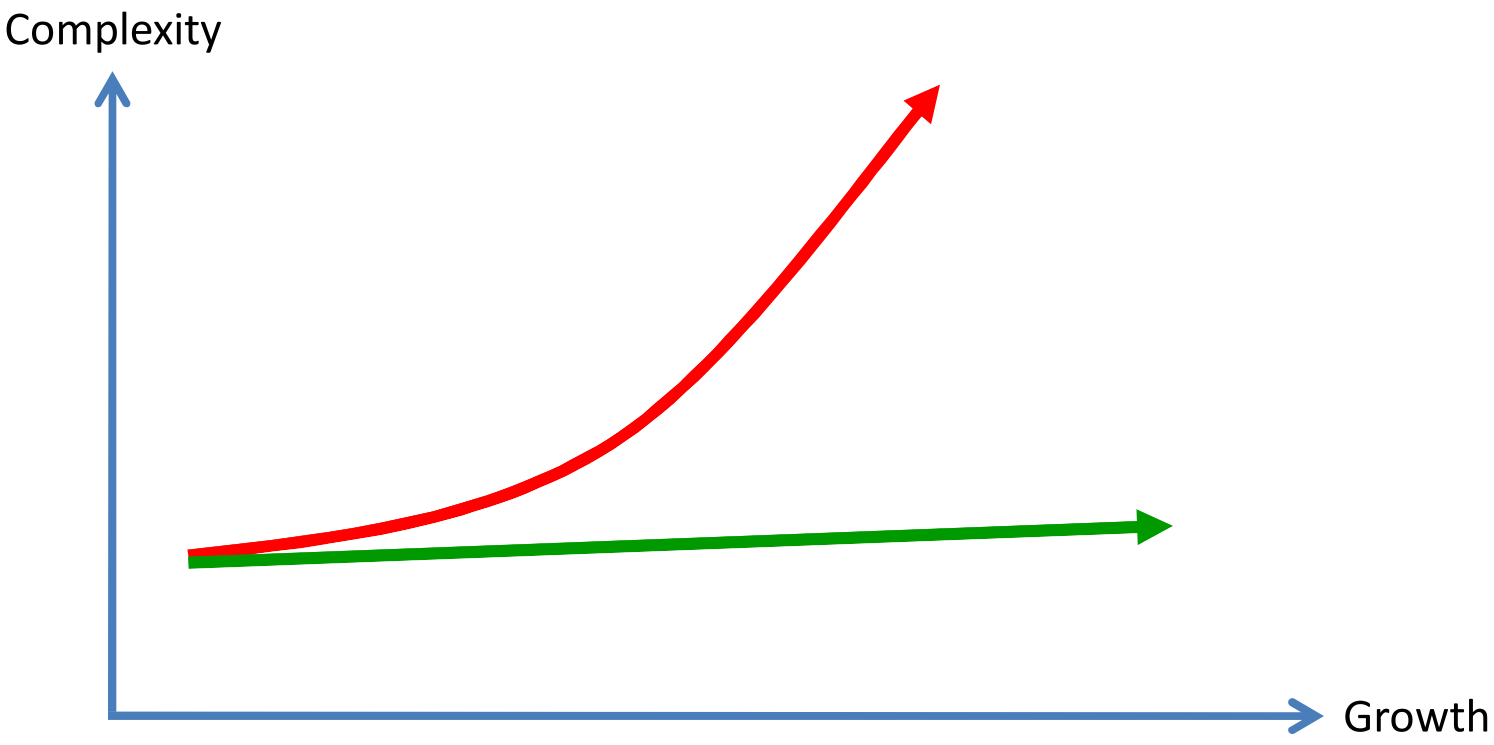
Half-duplex Architecture



Protocol mismatches…
Inefficient…

Scalability Issues

Scalability



Scalability = Growth / Complexity

Simple things scale…

Problems with HTTP

- Real-time cumbersome to achieve, primarily due to the limitations of HTTP
- HTTP is half-duplex (traffic flows in only one direction at a time)
- Rich Internet Applications (with Ajax, Comet, JavaFX, Silverlight and Flash) are becoming richer, but still limited by HTTP
- HTTP adds latency.   Latency sucks...

Additional Latency… Bad…

- Financial Applications
- News
- Government Alerts
- Health Care
- Sports
- Gaming

- System Monitoring
- Auctions
- Corporate Collaboration
- Dynamic Cloud Management
- Security Systems
- etc…

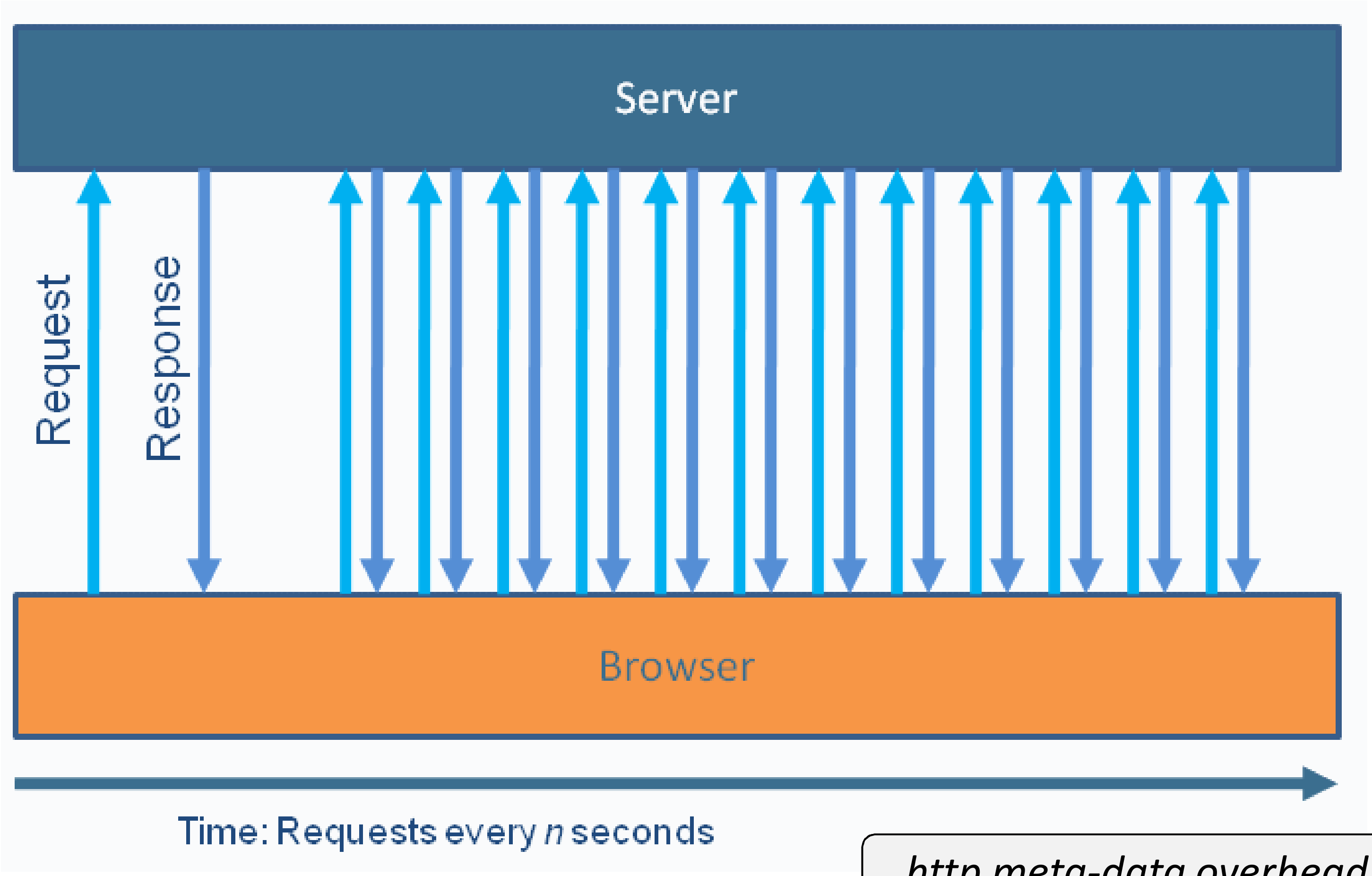
Polling and Long-Polling

Real-time solutions??

- AJAX – ***poll*** server for updates, wait at client
- Comet – ***poll*** server for updates, wait at the server; uses two connections and requires unnecessary complexity
- Provide user-perceived low latency (“nearly” real-time)
- Used in Ajax applications to *simulate* real-time communication

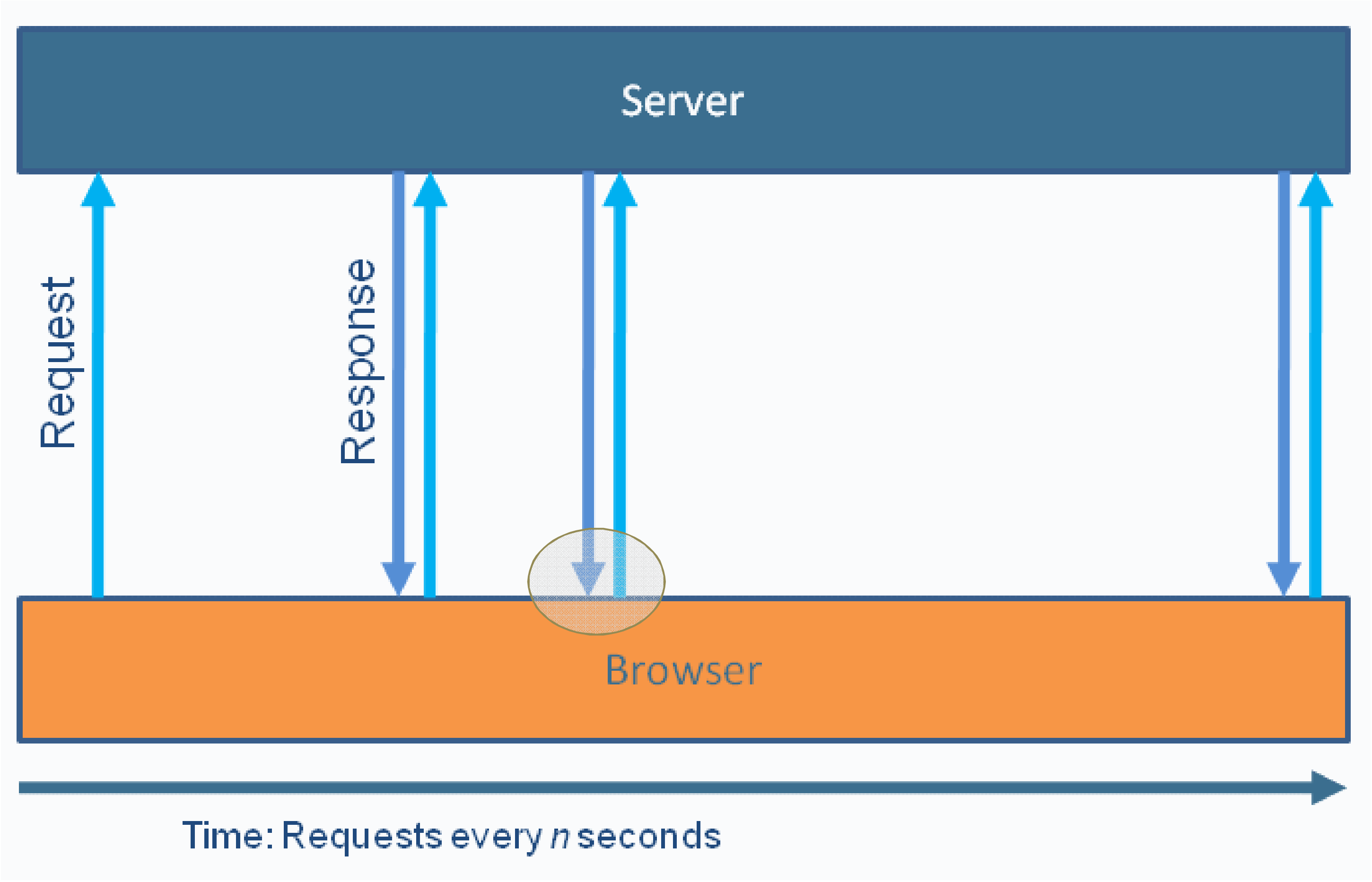
Polling leads to poor resource utilization

Polling Architecture



http meta-data overhead!

Long-Polling Architecture



Example Simple 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
```

691 chars

Example Simple 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
```

180 chars

Total 871 chars.
Potential even greater!

HTTP Header Traffic Analysis

Example network throughput for Polling HTTP req/resp headers

Use case A: 10,000 clients polling every 60 seconds

- Network throughput is (871 x 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 x 10,000)/1 = 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 x 100,000)/10 = 8,710,000 bytes = 69,680,000 bits per second (66 Mbps)

Streaming

- More efficient, sometimes problematic
- Possible complications
  - Firewalls and proxies (esp buffering proxies)
  - Response builds up and must be flushed periodically
  - Cross-domain issues to do with browser connection limits

HTML5

W3C/IETF *Standard*

HTML5 – Industry Standard

- Backed by Google, Apple, Mozilla, et al
- W3C HTML standards – Web direction

➡ ♦ **Communication** (sockets, cross-site)

♦ Drag ‘n’ drop

♦ Offline

♦ New HTML tags

♦ Graphics (2D)

♦ Storage (transient, persistent)

♦ Compatibility

♦ Video/Audio

- Kaazing major contributor to W3C HTML5 Web Socket specification
- IETF - Web Socket Protocol draft specification

last call Oct 2009

HTML5 Communication

- Proxy/Firewall-friendly TCP Socket for Browsers
  - ♦ Web Socket (ws/wss), bidirectional
- Standardized HTTP Streaming
  - ♦ Server-Sent Events (downstream)
- Secure Cross-Site Remote Connectivity
  - ♦ Cross-Origin Resource Sharing – XMLHttpRequest Level 2
- Secure Browser Cross-Document Messaging
  - ♦ postMessage to inline frames (iframe)

Website and Stock Portfolio

WEBSOCKETS DEMO

HTML5 WebSockets

- Full-Duplex Text Socket - Single connection
- Traverses Firewalls/Proxies/Routers
- Leverages Cross-Site Access Control
- Integrates with Cookie-based Authentication
- Works with HTTP Load Balancers
- WebSockets and Secure WebSockets
  - `ws://www.kaazing.com/clear-text-stuff`
  - `wss://www.kaazing.com/encrypted-stuff`

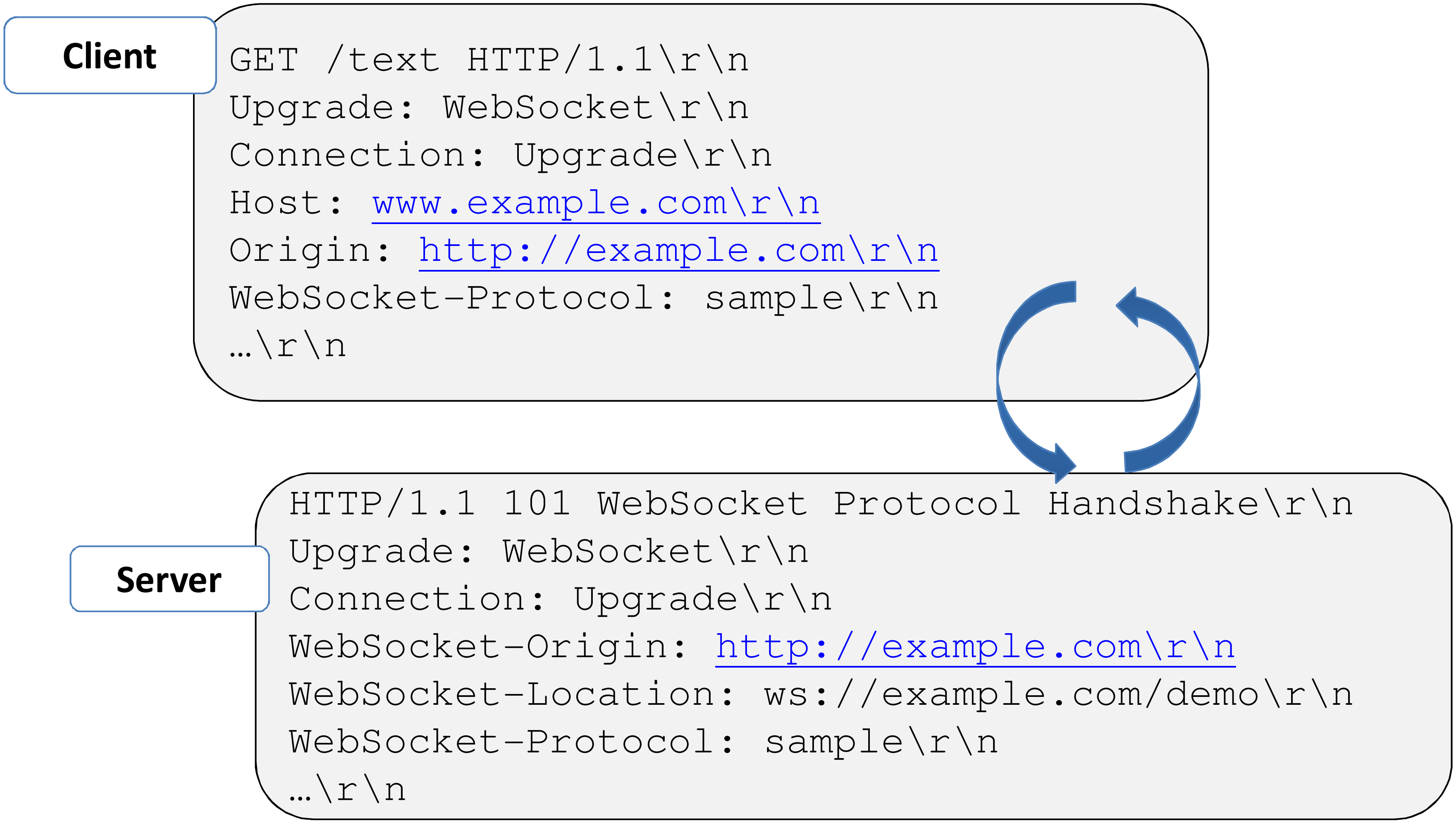
HTML5 Server-Sent Events

- Essentially a Standardization of Comet
- Streaming from server to web client (e.g., browser)
- “Web Push” (*downstream*)
- Optional Guaranteed Event/Message Delivery
- Wire Protocol (name/value text- based)

HTTP -> Web Socket

- Connection established by upgrading from the HTTP protocol to the WebSocket protocol
- 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\r\n
Upgrade: WebSocket\r\n
Connection: Upgrade\r\n
Host: www.example.com\r\n
Origin: http://example.com\r\n
WebSocket-Protocol: sample\r\n
…\r\n
```

Server

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

Web Socket Frames

- Frames can be sent full-duplex
- Either direction at any time
- Text Frames use terminator
  - `\x00Hello, WebSocket\0xff`
- Binary Frames use length prefix
  - `\x00\0x10Hello, WebSocket`

Drastic Reduction in Network Traffic

- With WebSocket, each frame has **only** 2 bytes of packaging
- No latency involved in establishing new TCP connections for each HTTP message
- Remember the Polling HTTP header traffic?

  66 Mbps network throughput for just headers

HTTP Header Traffic Analysis

Example network throughput for WebSocket req/resp headers

Use case A: 10,000 frames every 60 seconds

- Network throughput is (2 x 10,000)/60 = 333 bytes = 2,664 bits per second (**2.6 Kbps**) [was 1.1 Mbps]

Use case B: 10,000 frames every second

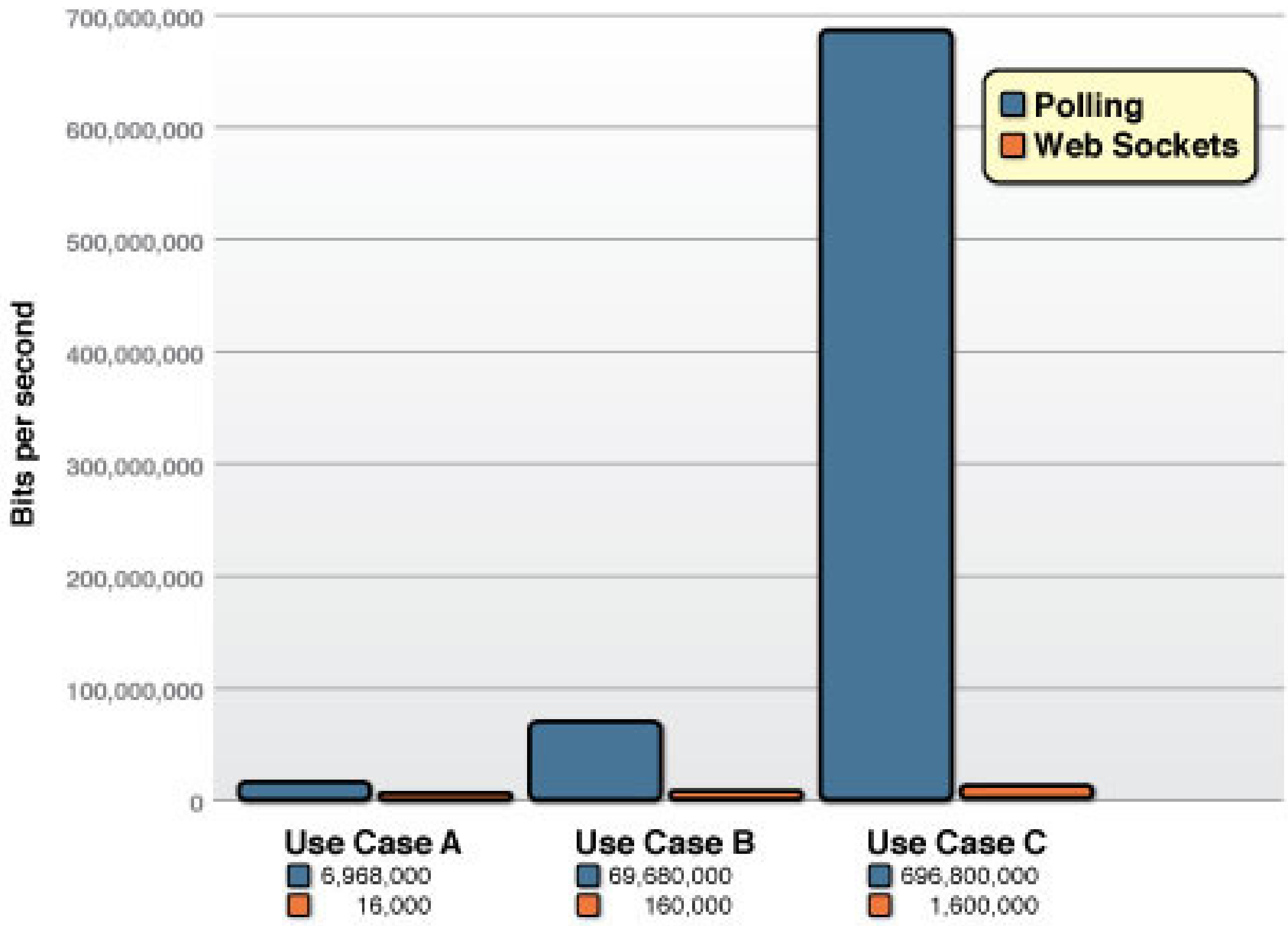
- Network throughput is (2 x 100,000)/1 = 20,000 bytes = 160,000 bits per second (**156 Kbps**) [was 66 Mbps]

Use case C: 100,000 frames every 10 seconds:

- Network throughput is (2 x 100,000)/10 = 20,000 bytes = 160,000 bits per second (**156 Kbps**) [was 66 Mbps]

1/500 !

Polling vs. Web Sockets



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)

WebSocket APIs

JavaScript

```
var mysock = new WebSocket(“ws://www.websocket.org”);
.
.
.
// Associating listeners

mysock.onopen = function(evt) {
        alert(“Connection open…”);
};

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

mysock.onclose = function(evt) {
        alert(“Connection closed…”);
};
```

WebSocket APIs

JavaScript

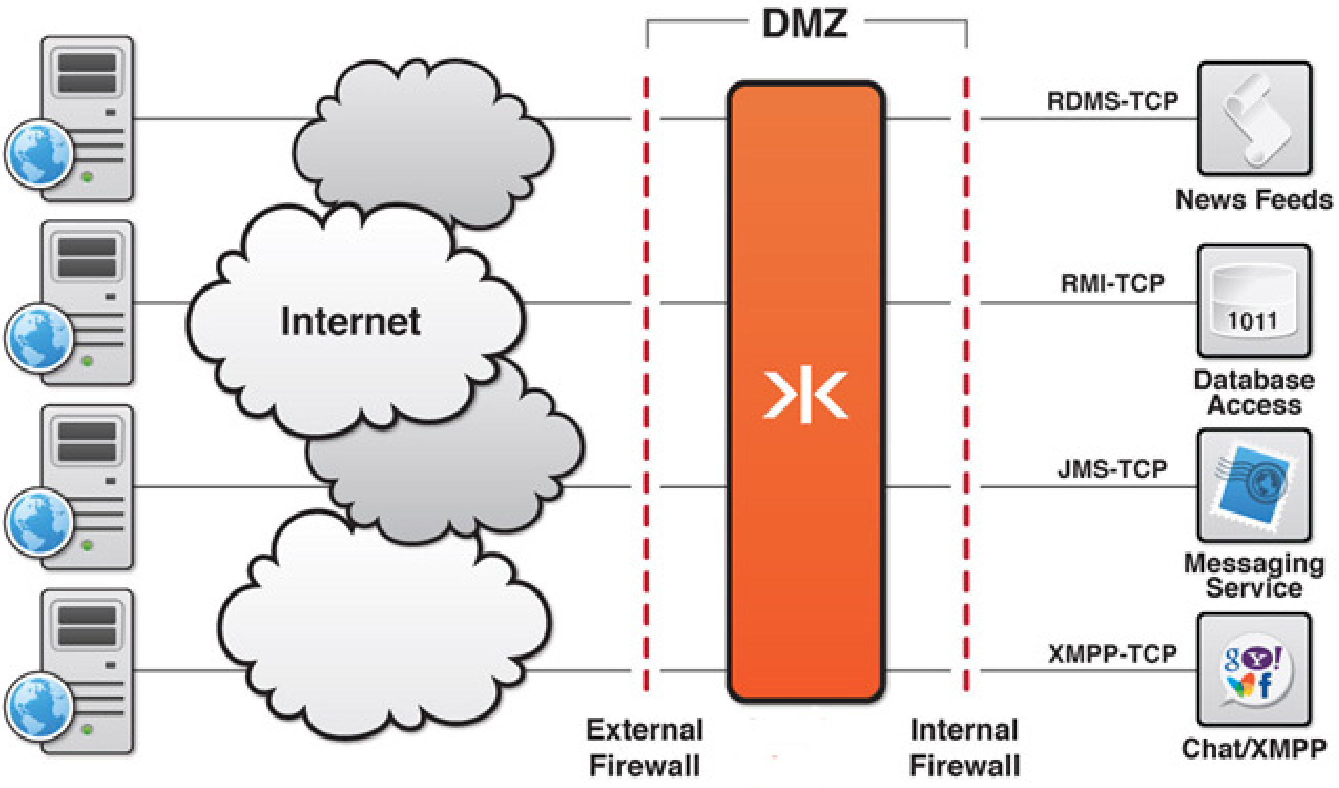
```
var mysock = new WebSocket(“ws://www.websocket.org”);
.
.
.
// Sending data

mysock.send(“Hello WebSocket!”);
mysock.send(“Kaazing Rocks!”);
.
.
.
mysock.disconnect();
```

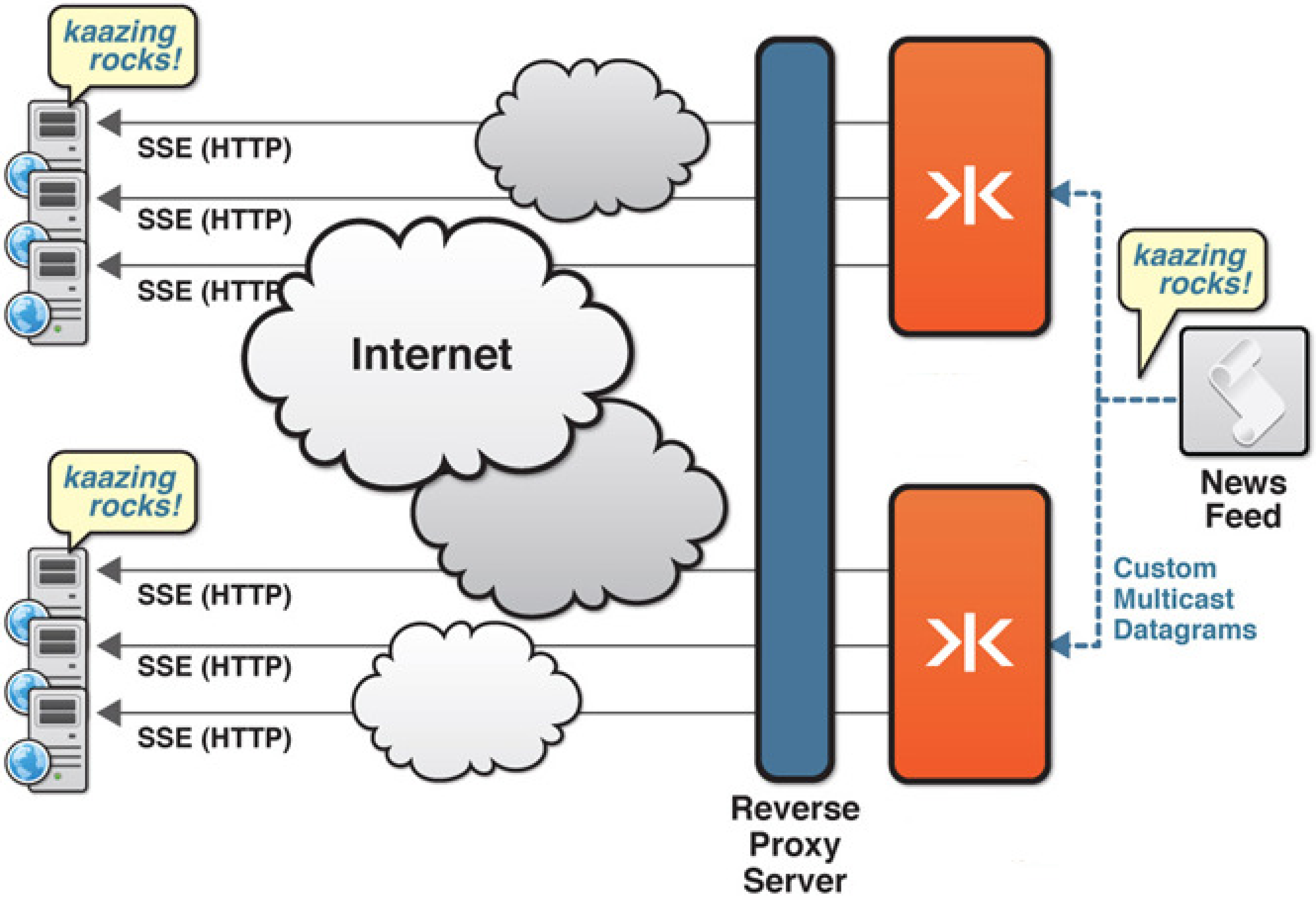
Deployment Architectures

EXAMPLES

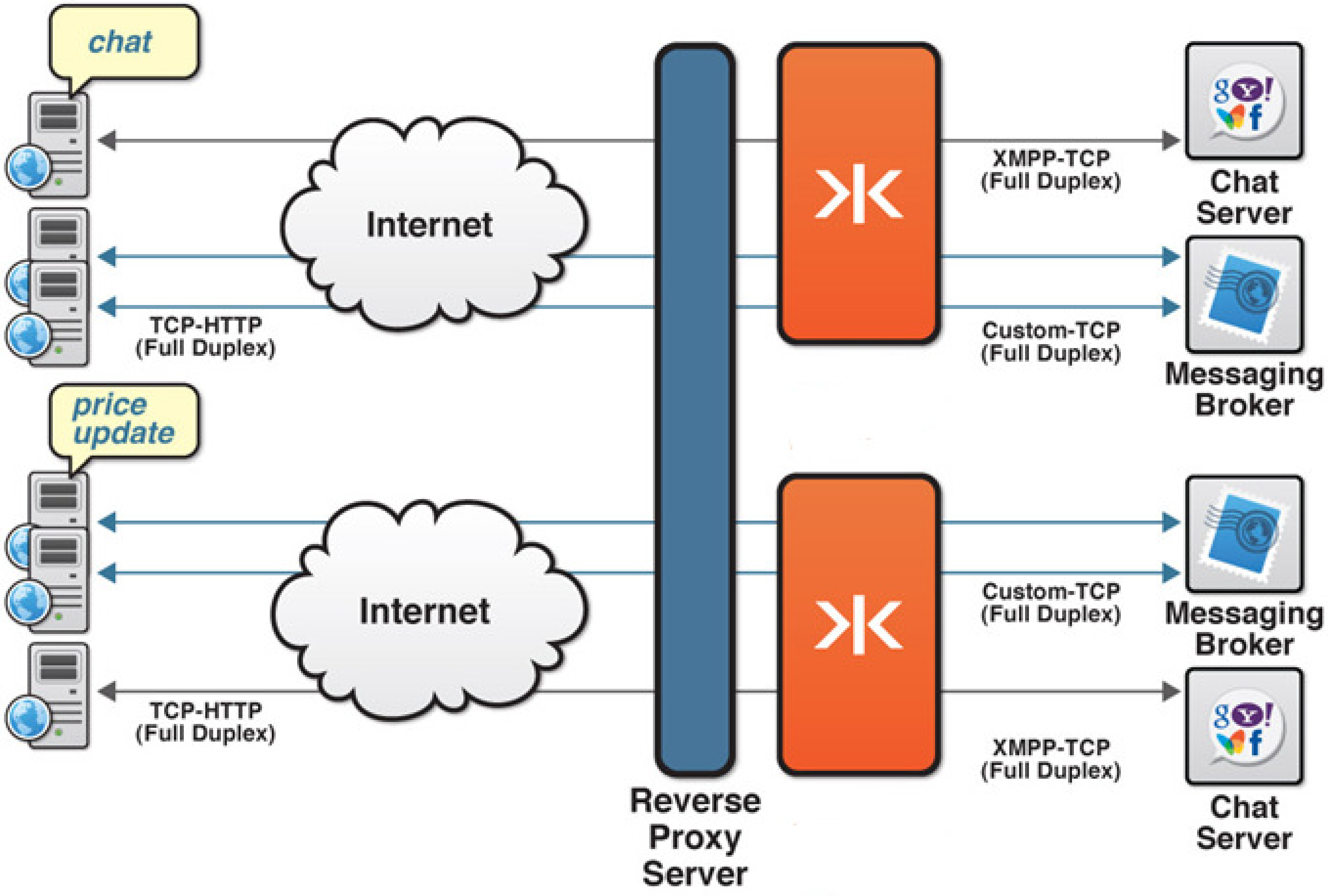
Enterprise Architecture



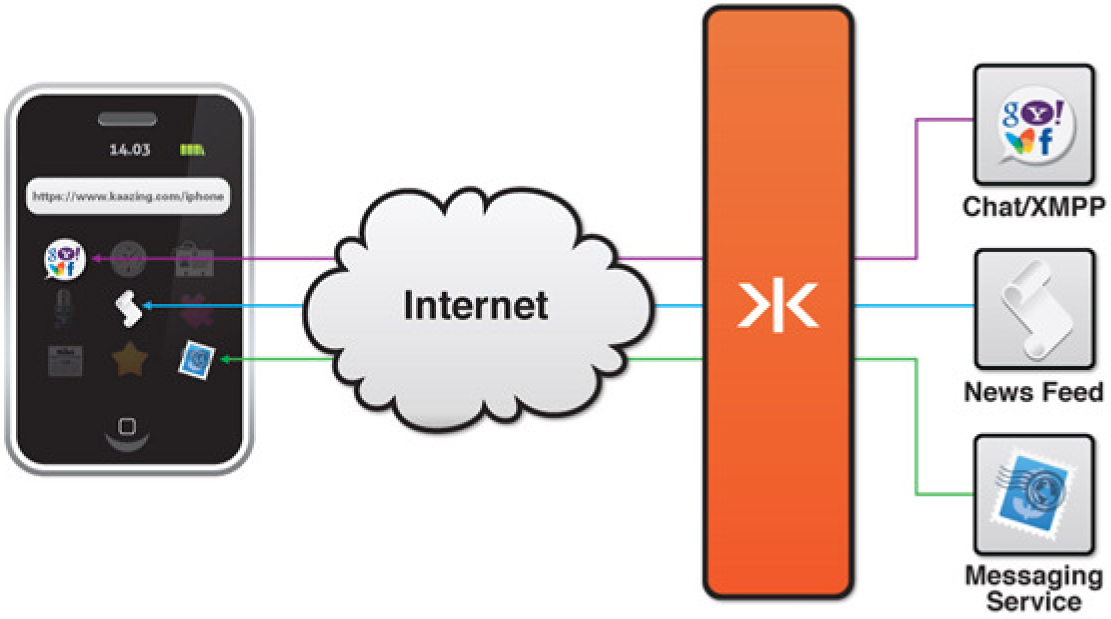
Server-Sent Events Architecture



WebSocket Architecture

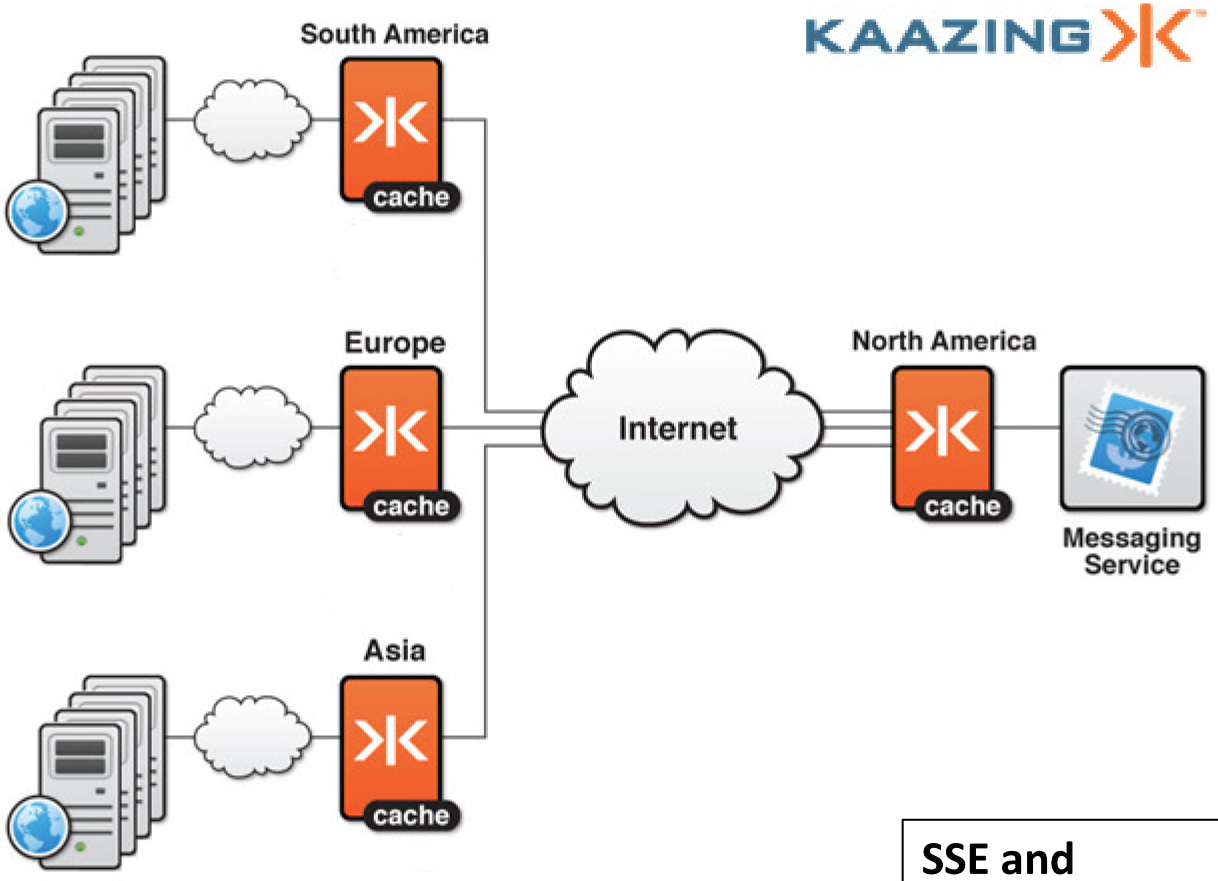


Aggregating at the Browser



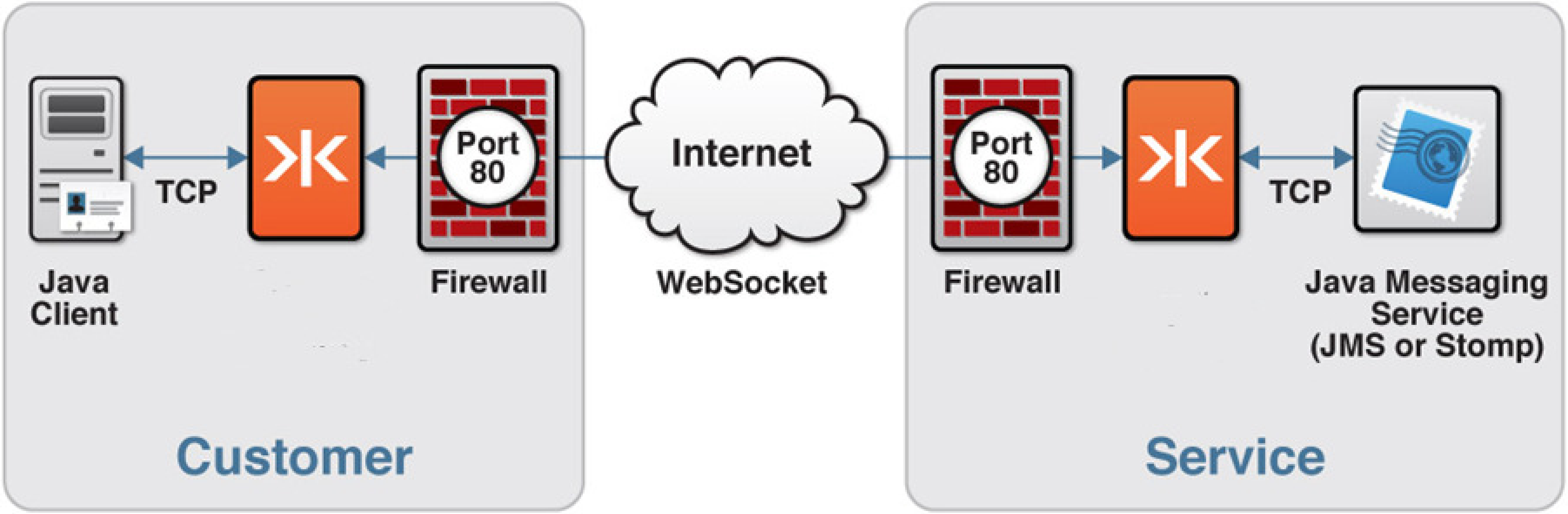
Reduce dependencies on portal servers and portal farms.

Leveraging Fan-Out Topology



SSE and Publish/Subscribe…

Virtual Private Connection



If you had HTML 5 Communications, what could you build?

For Business Applications

Financial Services

- Sending real-time data (prices, news) to browser
- Providing web services and analytics directly to clients' spreadsheets
- Collaborating in real-time with brokerage clients
- Providing equity research reports immediately when available
- Monitoring bond pricing batch runs in real-time
- Broadcasting high-priority messages that affect customers' portfolios globally
- Developing zero-install trading platform that's compatible with all browsers

For Business Applications

Media, Telecommunications, Utilities, Ecommerce, etc…

- Airport/Train Schedules
- Supply Chain updates
- Webmail
- Real-time Auction services
- Traffic, Weather, Emergency Warnings
- Sports Scores/News
- Press Releases
- Social Network Collaboration Tools
- Gaming
- Telemetry, Power Grid Monitoring, Water Usage, etc.

For Developers

- Build high performance, scalable messaging for web apps
- Use JMS messaging directly to/from the browser
- Create real-time web apps for mobile devices
- Traverse proxies and firewalls
- Web-based notification system for internal/external users
- Move to event-based model and improve server scalability
- Extend any TCP protocol over the web

For Corporate Web

- Allow real-time collaboration with employees and partners
- Create real-time workflow web apps for document collaboration
- Perform real-time system management/monitoring (JMX, Tibco, WebLogic, databases, et al) over corporate web
- Aggregate information from various sources without complicated portals

Appendix

HTML5 WebSocket APIs

JavaScript
Flash/Flex
Silverlight
Java/JavaFX

KAAZING

WebSocket APIs

JavaScript

```
var mysock = new WebSocket(“ws://www.websocket.org”);
.
.
.
// Associating listeners

mysock.onopen = function(evt) {
        alert(“Connection open…”);
};

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

mysock.onclose = function(evt) {
        alert(“Connection closed…”);
};
```

WebSocket APIs

JavaScript

```
var mysock = new WebSocket(“ws://www.websocket.org”);
.
.
.
// Sending data

mysock.send(“Hello WebSocket!”);
mysock.send(“Kaazing Rocks!”);
.
.
.
mysock.disconnect();
```

Server-Sent Event – DOM/APIs

JavaScript

```
<!--  New HTML DOM Element -->
<eventsource src=“http://www.kaazing.com/sse”
             onmessage=“alert(event.data)”>
…
// (alternatively)
// JavaScript HTML DOM API for dynamic creation

var myes = document.createElement(“eventsource”);
myes.addEventListener(“message”, function(evt) {
       alert(evt.data);
       },
   false);

myes.addEventSource(“http://www.kaazing.com/sse”);
```

Inter-Document Messaging – APIs
Sending Messages

JavaScript

```
// Send msg string to targetWindow
document.getElementById(“iframe”).
    contentWindow.postMessage(msg, targetOrigin);
```

Inter-Document Messaging – APIs
Receiving 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
  }
}
```

Questions?

Thanks!

- Kaazing Web site: www.kaazing.com
- Kaazing Tech Network: http://tech.kaazing.com/
- Download Kaazing Enterprise Gateway: http://www.kaazing.com/download

TM

KAAZING