Ajax Performance Analysis

Ryan Breen

gomez
Ensuring quality web experiences℠

Ajax Performance Analysis

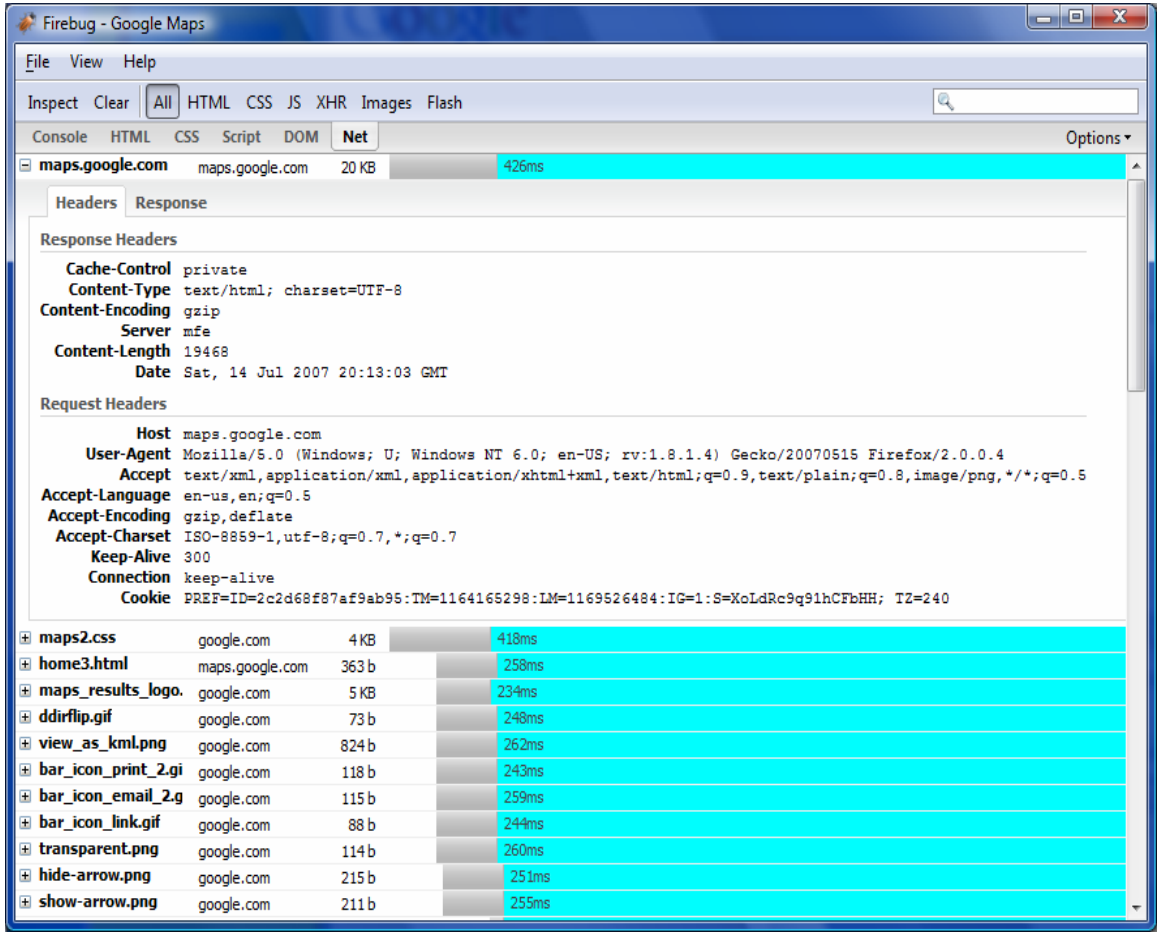
- **Who**
  - Ryan Breen: VP Technology at Gomez and blogger at ajaxperformance.com
- **Goals**
  - Survey tools available to developers
  - Understand how to approach performance optimization
  - Real world examples of how to improve the end user experience (with exciting data!)

Understand the tools available

- **Network visualization**
  - Firebug
  - WebKit Web Inspector
  - IBM Page Detailer
- **Client side profiling**
  - Firebug and Firebug Lite
  - Dojo.Profile

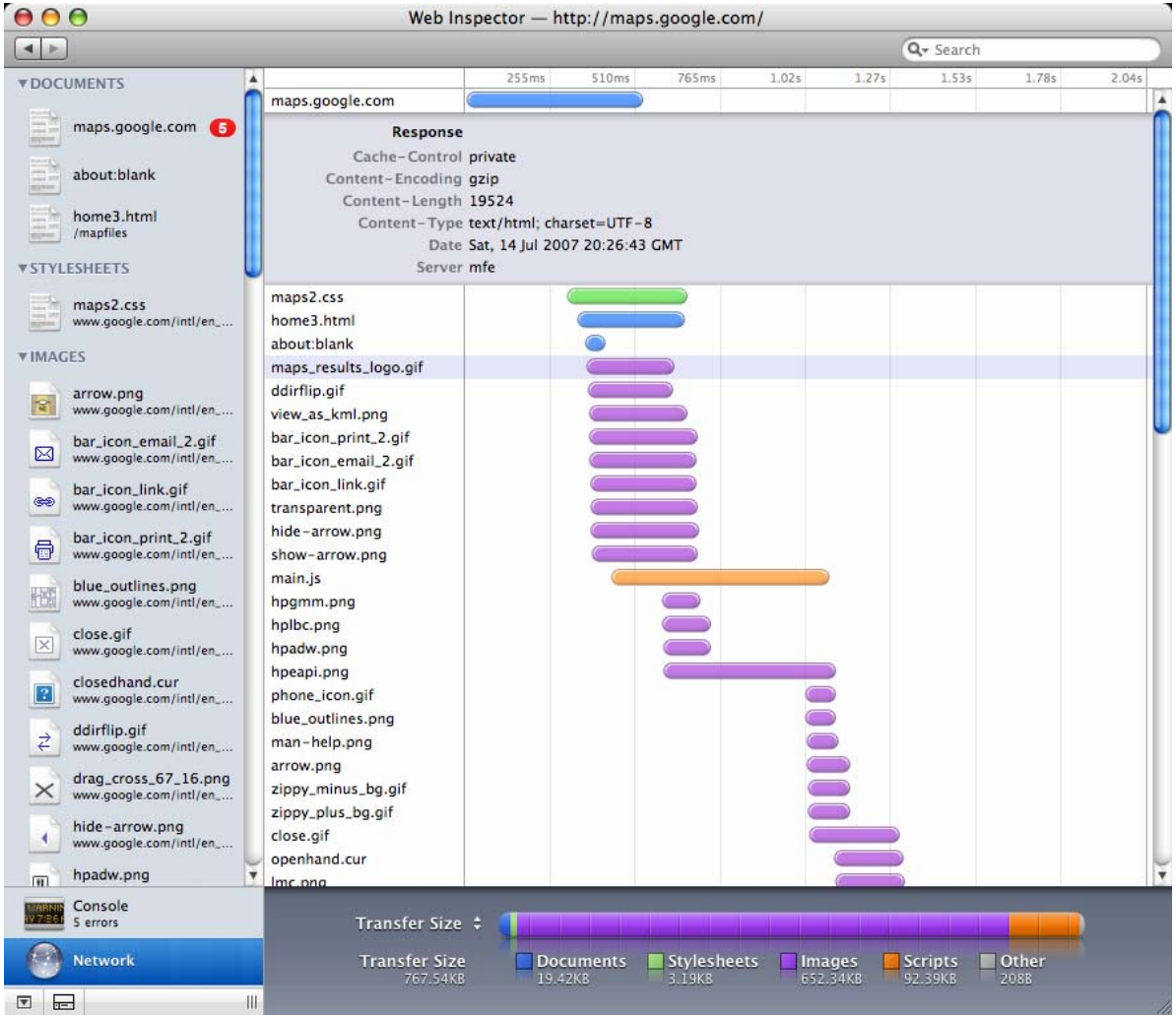
Network Visualization

- **Firebug**
  - Network visualization since version 1.0
  - Useful information, but not overwhelming in detail
  - A great addition to a tool you are already (or should be) using



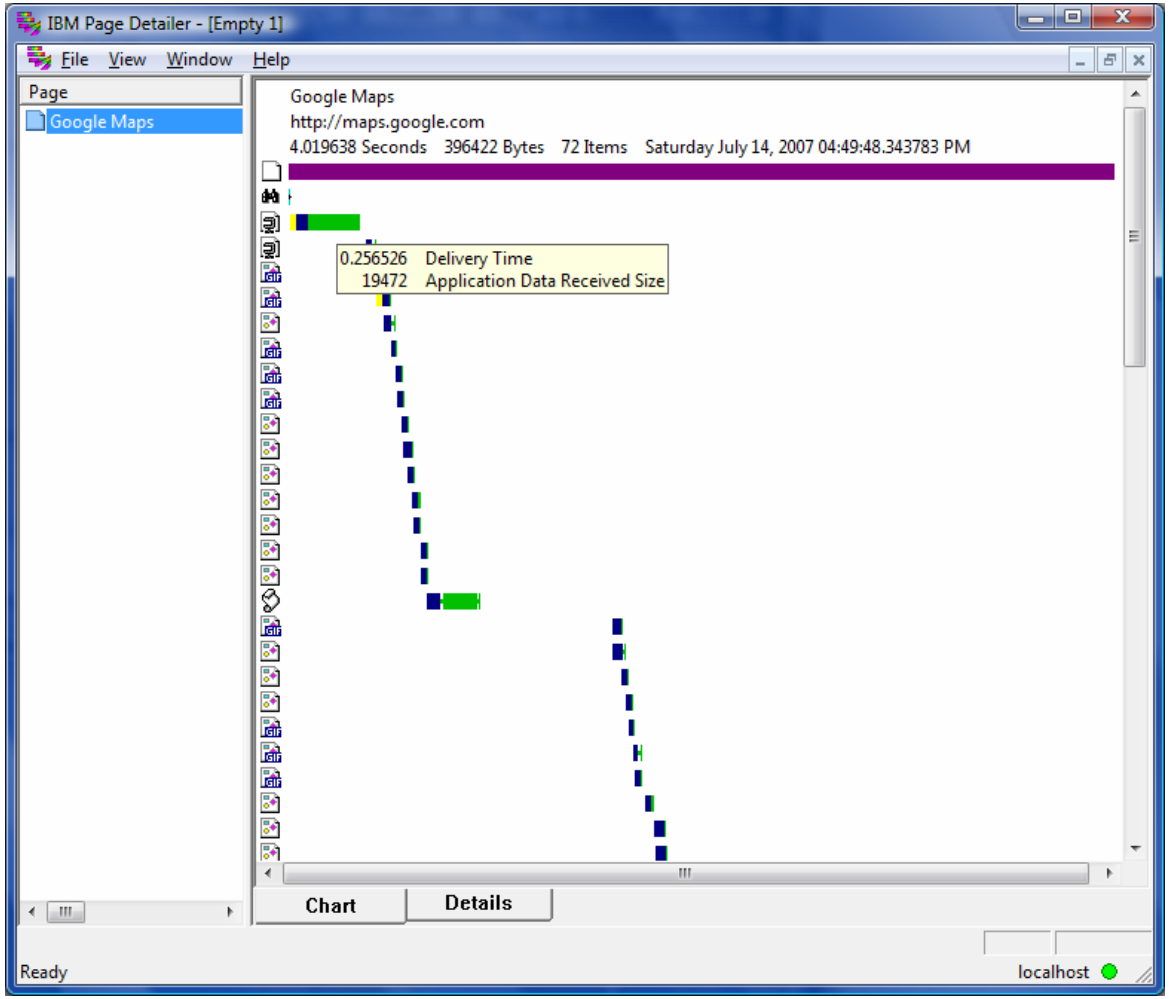
Network Visualization

- **Web Inspector**
  - Currently available in WebKit nightlies
  - Beautiful presentation
  - Part of a comprehensive toolkit for Safari, a worthy companion to Firebug



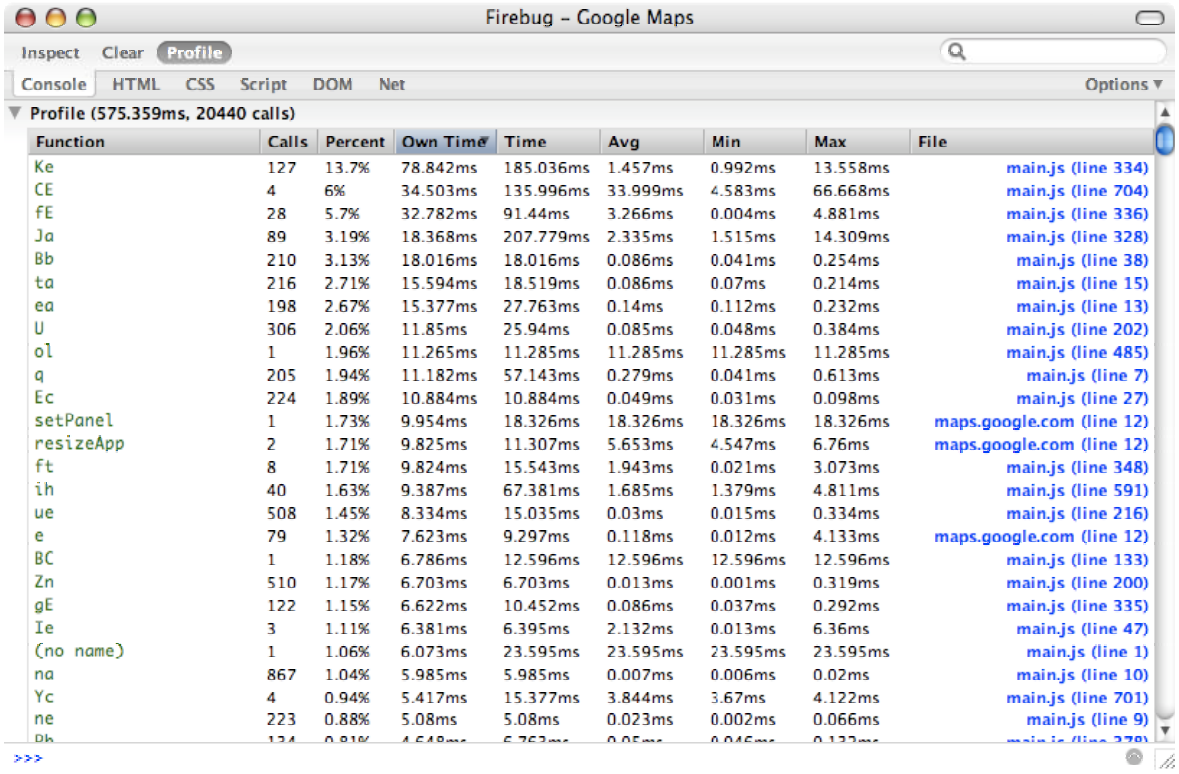
Network Visualization

- **IBM Page Detailer**
  - Basic version is free
  - OS-level network profiling, so supports any browser on Windows
  - Very focused, unitasker
  - Much more granular network analysis
  - Really, really nerdy



Client side profiling

- **Firebug**
  - JS function call profiler
  - Firefox only



Function	Calls	Percent	Own Time	Time	Avg	Min	Max	File
Ke	127	13.7%	78.842ms	185.036ms	1.457ms	0.992ms	13.558ms	main.js (line 334)
CE	4	6%	34.503ms	135.996ms	33.999ms	4.583ms	66.668ms	main.js (line 704)
fE	28	5.7%	32.782ms	91.44ms	3.266ms	0.004ms	4.881ms	main.js (line 336)
Ja	89	3.19%	18.368ms	207.779ms	2.335ms	1.515ms	14.309ms	main.js (line 328)
Bb	210	3.13%	18.016ms	18.016ms	0.086ms	0.041ms	0.254ms	main.js (line 38)
ta	216	2.71%	15.594ms	18.519ms	0.086ms	0.07ms	0.214ms	main.js (line 15)
ea	198	2.67%	15.377ms	27.763ms	0.14ms	0.112ms	0.232ms	main.js (line 13)
U	306	2.06%	11.85ms	25.94ms	0.085ms	0.048ms	0.384ms	main.js (line 202)
ol	1	1.96%	11.265ms	11.285ms	11.285ms	11.285ms	11.285ms	main.js (line 485)
q	205	1.94%	11.182ms	57.143ms	0.279ms	0.041ms	0.613ms	main.js (line 7)
Ec	224	1.89%	10.884ms	10.884ms	0.049ms	0.031ms	0.098ms	main.js (line 27)
setPanel	1	1.73%	9.954ms	18.326ms	18.326ms	18.326ms	18.326ms	maps.google.com (line 12)
resizeApp	2	1.71%	9.825ms	11.307ms	5.653ms	4.547ms	6.76ms	maps.google.com (line 12)
ft	8	1.71%	9.824ms	15.543ms	1.943ms	0.021ms	3.073ms	main.js (line 348)
ih	40	1.63%	9.387ms	67.381ms	1.685ms	1.379ms	4.811ms	main.js (line 591)
ue	508	1.45%	8.334ms	15.035ms	0.03ms	0.015ms	0.334ms	main.js (line 216)
e	79	1.32%	7.623ms	9.297ms	0.118ms	0.012ms	4.133ms	maps.google.com (line 12)
BC	1	1.18%	6.786ms	12.596ms	12.596ms	12.596ms	12.596ms	main.js (line 133)
Zn	510	1.17%	6.703ms	6.703ms	0.013ms	0.001ms	0.319ms	main.js (line 200)
gE	122	1.15%	6.622ms	10.452ms	0.086ms	0.037ms	0.292ms	main.js (line 335)
Ie	3	1.11%	6.381ms	6.395ms	2.132ms	0.013ms	6.36ms	main.js (line 47)
(no name)	1	1.06%	6.073ms	23.595ms	23.595ms	23.595ms	23.595ms	main.js (line 1)
na	867	1.04%	5.985ms	5.985ms	0.007ms	0.006ms	0.02ms	main.js (line 10)
Yc	4	0.94%	5.417ms	15.377ms	3.844ms	3.67ms	4.122ms	main.js (line 701)
ne	223	0.88%	5.08ms	5.08ms	0.023ms	0.002ms	0.066ms	main.js (line 9)

Client side profiling

- **Pure JavaScript solutions**
  - Dojo.Profile
  - Firebug Lite
- **More limited functionality, but available cross browser**

How to approach performance

- **Avoid Premature Optimization**
- **Understand your users**
  - Connection profile
  - Usage model: how do they interact with your application?
- **Measure, then optimize along two axes**
  1. Reduce overall latency
  2. Hide the remaining latency from user perception
- **Define performance targets**

Latency

- **The goal of performance optimization is to hide latency from the end user**
- **In high speed networks, latency dominates**
  - The speed of establishing connections and issuing requests has not scaled in pace with the speed of transmitting data
  - Round trip times are slow, transfer completion times are fast

Optimizing against Latency

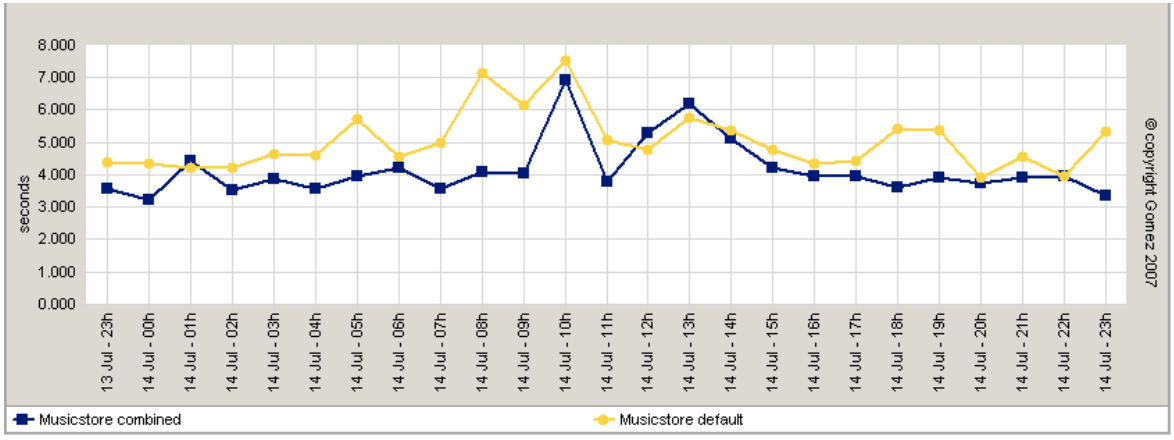
- **How do we reduce latency?**
  - Fewer requests means better performance
    - No matter the connection profile, there is no better way to improve performance
    - Every other trick is a way to hide the cost of downloading objects
  - So, pack more information into each request
  - Reuse connections
  - Increase connection parallelism
  - Minimize bandwidth used where possible
  - Respect caching, but don't put too much faith in it

Packing more into a request

- **Image concatenation**
  - Replace multiple requests for small images with one request
  - Use CSS background-position to select
  - Further optimization: use CSS transparency to represent disabled images

Packing more into a request

- **JS and CSS bundling**
  - JavaScript and CSS are frequently in HEAD
  - JavaScript is downloaded and executed serially, exacerbating the latency experienced by the end user
  - Combine scripts and CSS into one file to avoid round trip costs
  - Many frameworks or web toolkits now do this on the fly



Data Summary

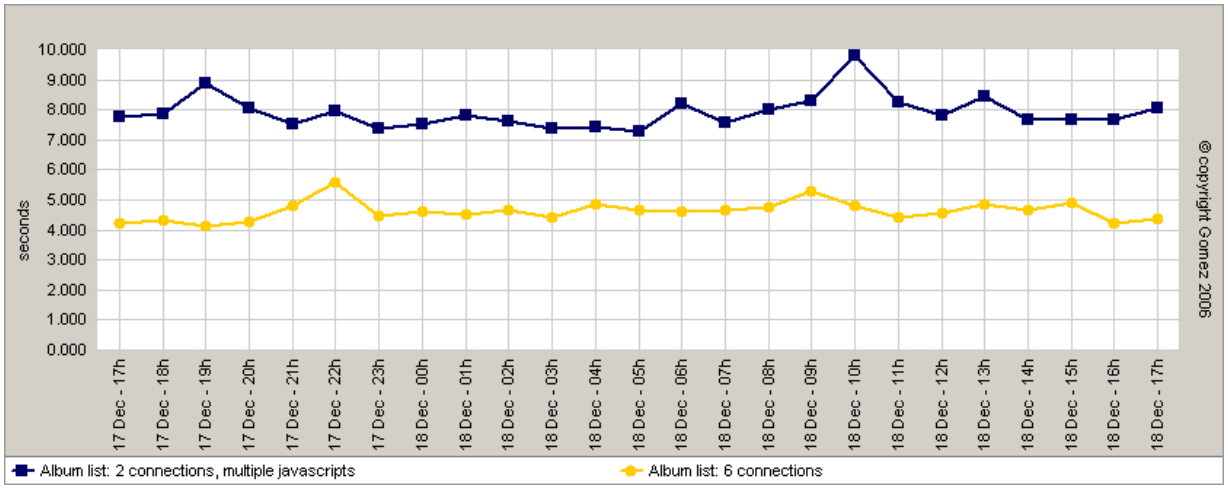
Test Name	Avg Response Time (sec)	Availability (%)	Details
Musicstore combined	4.174	100.00	
Musicstore default	4.986	100.00	

Reuse connections

- **Connection persistence is a huge win**
  - Creating each connection adds latency
  - Without connection persistence, that latency hit is incurred for each object on the page
  - Caution: Persistence is frequently broken due to firewall configuration issues or other operational mishaps

Increase connection parallelism

- **Most browsers allow 2 connections per host**
  - But that host check is a simple string match against hostname
  - We can trick the browser into send 6 requests at a time with a DNS wildcard
  - Potential savings of 40%
  - This trick is used by every RIA map application



Optimizing against Latency

- **Latency is a focus, but also reduce bandwidth**
  - HTTP Compression for text artifacts
  - JS obfuscation – Potential side benefit of reducing readability
- **Don’t overlook the simple things**
  - When redirecting to a directory, remember the trailing slash

Caching

- **Caching effectively reduces requests on subsequent visits**
  - So avoid breaking it with performance tweaks
- **But you can't trust caching to hide performance crimes**
  - YUI Blog numbers: 40-60% of users and 20% of page hits had empty cache
  - All new users have an empty cache, so avoid making a bad first impression

Caching

- **Where possible, avoid freshness checks**
  - Getting that 304 is still a round trip
  - The latency for the request is often indistinguishable from a 200
  - Freshness typically involves a tradeoff between useless refreshes and stale data

Caching

- **An elegant solution from Google GWT**
  - The file name for ImageBundles includes an MD5 sum of the contents, so cache headers can be set for infinite cacheability
  - Users are guaranteed to have the newest copy
- **Solution could easily be extended to JS / CSS bundles**

Perceived Performance

- **Despite our best efforts, some latency remains**
- **How can we hide it?**
  - Understand the role of the user's perception
  - Defer anything that isn't critical to that perception

Postponing non-critical latency

- **Avoid the defer attribute**
  - Only supported in IE
  - Only defers execution, not network overhead
- **Alternative mechanisms**
  - DOM scripting
  - Dojo's XHR approach

Postponing events

- **onload vs. ondomready**
  - ondomready lets us schedule events sooner for more timely behavioral or presentational modifications
  - But ondomready may occur before the document is interactive
- **Is there room for a 3rd?**
  - How about onperceivedrender
  - How would that be calculated

Questions?

- **Contact Information**

  rbreen@gomez.com

  http://ajaxperformance.com