MAKE IT FAST

Using Modern Brower APIs to Monitor and Improve the Performance of your Web Applications

nic jansma | nicj.net | @nicj

WHO AM I?

Nic Jansma

nic@nicj.net

@nicj

http://nicj.net

SOASTA

- SOASTA (current)
- Microsoft (2005-2011)
- Founding member of W3C WebPerf Working Group

STATE OF PERFORMANCE MEASUREMENT

How do we measure performance?

SERVER

- HTTP logs (apache, nginx, haproxy)
- Server monitoring (top, iostat, vmstat, cacti, mrtg, nagios, new relic)
- Profiling (timestamps, xdebug, xhprof)
- Load testing (ab, jmeter, soasta, blazemeter, loadrunner)

DEVELOPER

- Browser developer tools (ie, chrome, ff, opera, safari)
- Network monitoring (fiddler, wireshark, tcpdump)

BUT...

- Measuring performance from the server and developer perspective is not the full story
- The only thing that really matters is what your end-user sees
- Measuring real-world performance of your end-users is tough

USER

(circa 2010)

- `var elapsedTime = Date.now() - startTime;`
- Boomerang: github.com/lognormal/boomerang

W3C WEBPERF WORKING GROUP

www.w3.org/2010/webperf

Founded 2010 to give developers the ability to assess and understand performance characteristics of their web apps

The mission of the Web Performance Working Group is to provide methods to measure aspects of application performance of user agent features and APIs

Microsoft, Google, Mozilla, Opera, Facebook, Netflix, etc

WORKING GROUP GOALS

- Expose information that was not previously available
- Give developers the tools they need to make their applications more efficient
- Little to no overhead
- Easy to understand APIs

PUBLISHED SPECS

- **Navigation Timing** (NT): Page load timings
- **Resource Timing** (RT): Resource load timings
- **User Timing** (UT): Custom site events and measurements
- **Performance Timeline**: Access NT/RT/UT and future timings from one API
- **High Resolution Time**: Better `Date.now()`

PUBLISHED SPECS (PT 2)

- **Page Visibility**: Visibility state of document
- **Timing control for script-based animations**: `requestAnimationFrame()`
- **Efficient Script Yielding**: More efficient than `setTimeout(...,0)`: `setImmediate()`

UPCOMING SPECS

- Beacon: Async send data (even after page is closed)
- Resource Hints: `rel="preconnect" rel="preload"`
- Resource Priorities: `lazyload`
- Frame Timing: Animation timings
- Navigation Error Logging: For failed navigations

PARTICIPATE!

www.w3.org/2010/webperf

public-web-perf@w3.org

github.com/w3c/web-performance

NAVIGATIONTIMING

www.w3.org/TR/navigation-timing

Goal: Expose accurate performance metrics describing your visitor's page load experience

Current status: Recommendation

Upcoming: NavigationTiming2

HOW IT WAS DONE BEFORE

(this isn't accurate)

```
<html><head><script>
var start = new Date().getTime();
function onLoad {
    var pageLoadTime = (new Date().getTime()) - start;
}
body.addEventListener(“load”, onLoad, false);
</script>...</html>
```

WHAT'S WRONG WITH THIS?

- It only measures the time from when the HTML gets parsed to when the last sub-resource is downloaded
- It misses the initial DNS lookup, TCP connection and HTTP request wait time
- `Date().getTime()` is not reliable

INTERLUDE

`DOMHighResTimeStamp`

	Date	**`DOMHighResTimeStamp`**
Accessed Via	`Date().getTime()`	`performance.now()`
Resolution	millisecond	sub-millisecond
Start	Unix epoch	`navigationStart`
Monotonically Non-decreasing	No	Yes
Affected by user's clock	Yes	No
Example	`1420147524606`	`3392.275999998674`

NAVIGATIONTIMING

window.performance.navigation

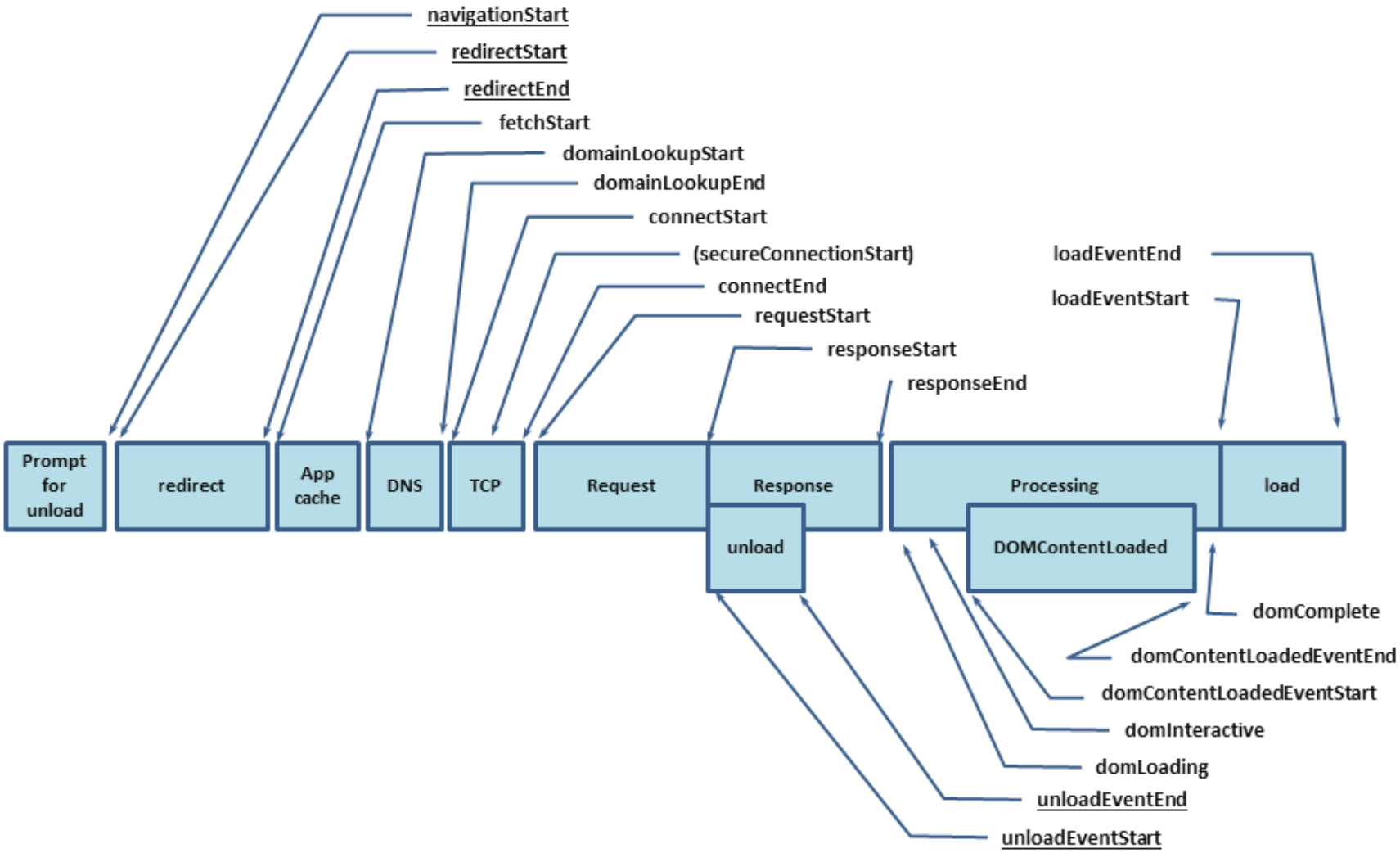
```
interface PerformanceNavigation {
    const unsigned short TYPE_NAVIGATE = 0;
    const unsigned short TYPE_RELOAD = 1;
    const unsigned short TYPE_BACK_FORWARD = 2;
    const unsigned short TYPE_RESERVED = 255;
    readonly attribute unsigned short type;
    readonly attribute unsigned short redirectCount;
};
```

NAVIGATIONTIMING

window.performance.timing

```
interface PerformanceTiming {
    readonly attribute unsigned long long navigationStart;
    readonly attribute unsigned long long unloadEventStart;
    readonly attribute unsigned long long unloadEventEnd;
    readonly attribute unsigned long long redirectStart;
    readonly attribute unsigned long long redirectEnd;
    readonly attribute unsigned long long fetchStart;
    readonly attribute unsigned long long domainLookupStart;
    readonly attribute unsigned long long domainLookupEnd;
    readonly attribute unsigned long long connectStart;
    readonly attribute unsigned long long connectEnd;
    readonly attribute unsigned long long secureConnectionStart;
    readonly attribute unsigned long long requestStart;
    readonly attribute unsigned long long responseStart;
    readonly attribute unsigned long long responseEnd;
    readonly attribute unsigned long long domLoading;
    readonly attribute unsigned long long domInteractive;
    readonly attribute unsigned long long domContentLoadedEventStart;
    readonly attribute unsigned long long domContentLoadedEventEnd;
    readonly attribute unsigned long long domComplete;
    readonly attribute unsigned long long loadEventStart;
    readonly attribute unsigned long long loadEventEnd;
};
```

NAVIGATIONTIMING



HOW TO USE

```
function onLoad() {
    if ('performance' in window && 'timing' in window.performance) {
        setTimeout(function() {
            var t = window.performance.timing;
            var ntData = {
                redirect: t.redirectEnd - t.redirectStart,
                dns: t.domainLookupEnd - t.domainLookupStart,
                connect: t.connectEnd - t.connectStart,
                ssl: t.secureConnectionStart ? (t.connectEnd - secureConnectio
                request: t.responseStart - t.requestStart,
                response: t.responseEnd - t.responseStart,
                dom: t.loadEventStart - t.responseEnd,
                total: t.loadEventEnd - t.navigationStart
            };
        }, 0);
    }
}
```

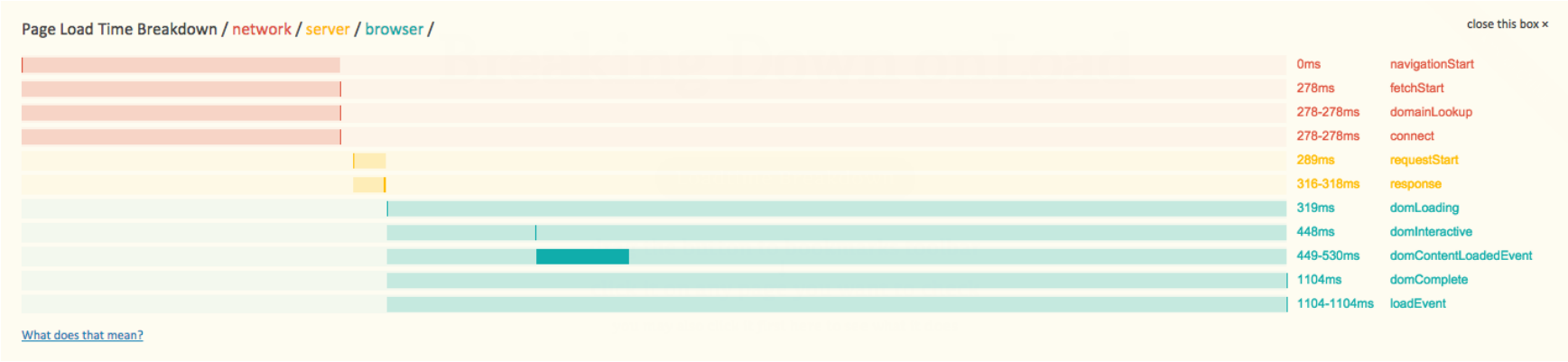
THEN WHAT?

DIY / Open Source

- Send this data to your backend for logging
- Show any page's timings via a bookmarklet: kaaes.github.io/timing
- Boomerang: github.com/lognormal/boomerang
- Boomcatch: cruft.io/posts/introducing-boomcatch
- BoomerangExpress: github.com/andreas-marschke/boomerang-express
- SiteSpeed.io: www.sitespeed.io
- Piwik: github.com/piwik/piwik

KAAES TIMING

kaaes.github.io/timing



BOOMERANG

github.com/lognormal/boomerang

日本語

this, is boomerang

boomerang always comes back, except when it hits something.

what?

boomerang is a piece of javascript that you add to your web pages, where it measures the performance of your website from your end user's point of view. It has the ability to send this data back to your server for further analysis. With boomerang, you find out exactly how fast your users think your site is.

boomerang is opensource and released under the BSD license, and we have a whole bunch of documentation about it.

how?

- Use cases — Just some of the uses of boomerang that we can think of
- How it works — A short description of how boomerang works internally
- Bugs, hugs and code — This is where the community comes in
- TODO — There's a lot that we still need to do. Wanna help?
- Howto docs — Short recipes on how to do a bunch of things with boomerang
- API — For all you hackers out there
- Elsewhere — A list of articles about boomerang

who?

boomerang comes to you from the Exceptional Performance team at Yahoo!, aided by the Yahoo! Developer Network.

where?

- Get a zip or tarball from github.com/lognormal/boomerang/archives/master
- Get the working source code from github.com/lognormal/boomerang

BOOMCATCH

Collects beacons + maps (statsd) + forwards (extensible)

cruft.io/posts/introducing-boomcatch

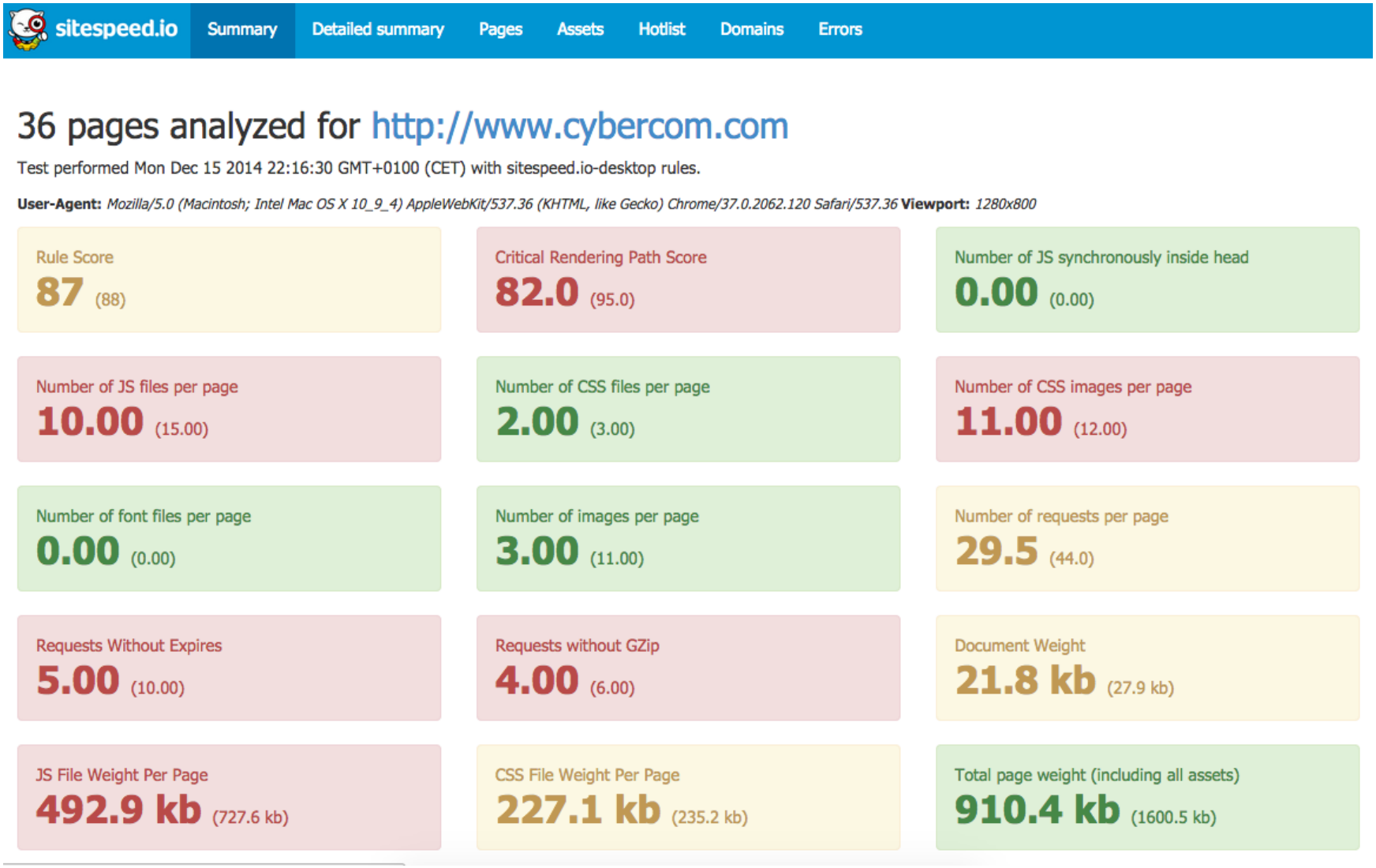
BOOMERANGEXPRESS

Collects beacons

github.com/andreas-marschke/boomerang-express

SITESPEED.IO

www.sitespeed.io



PIWIK

"generation time" = `responseEnd - requestStart`

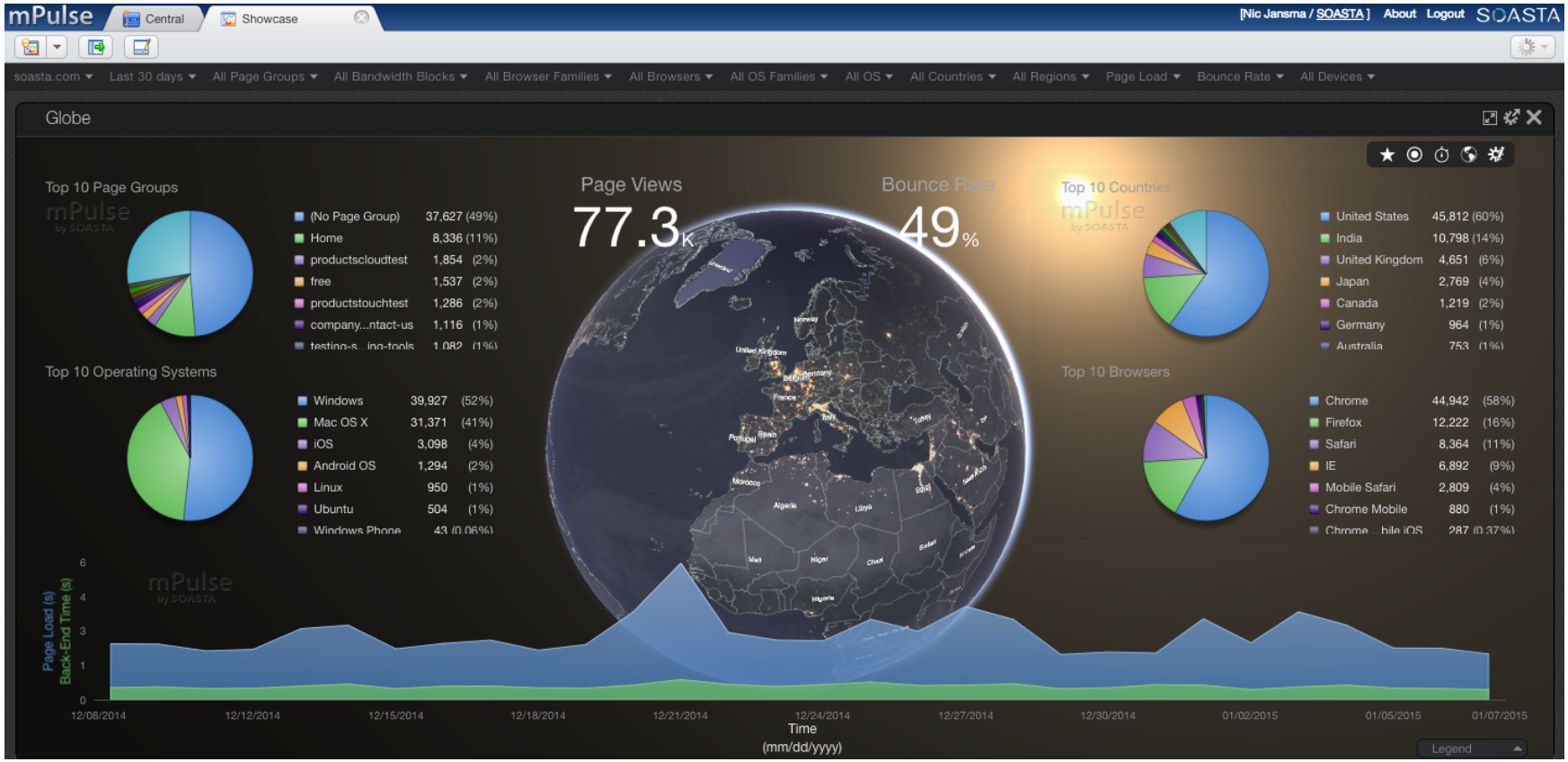
github.com/piwik/piwik

COMMERCIAL

- SOASTA mPulse: soasta.com
- Google Analytics Site Speed: google.com/analytics
- New Relic Browser: newrelic.com/browser-monitoring
- NeuStar WPM: neustar.biz
- SpeedCurve: speedcurve.com

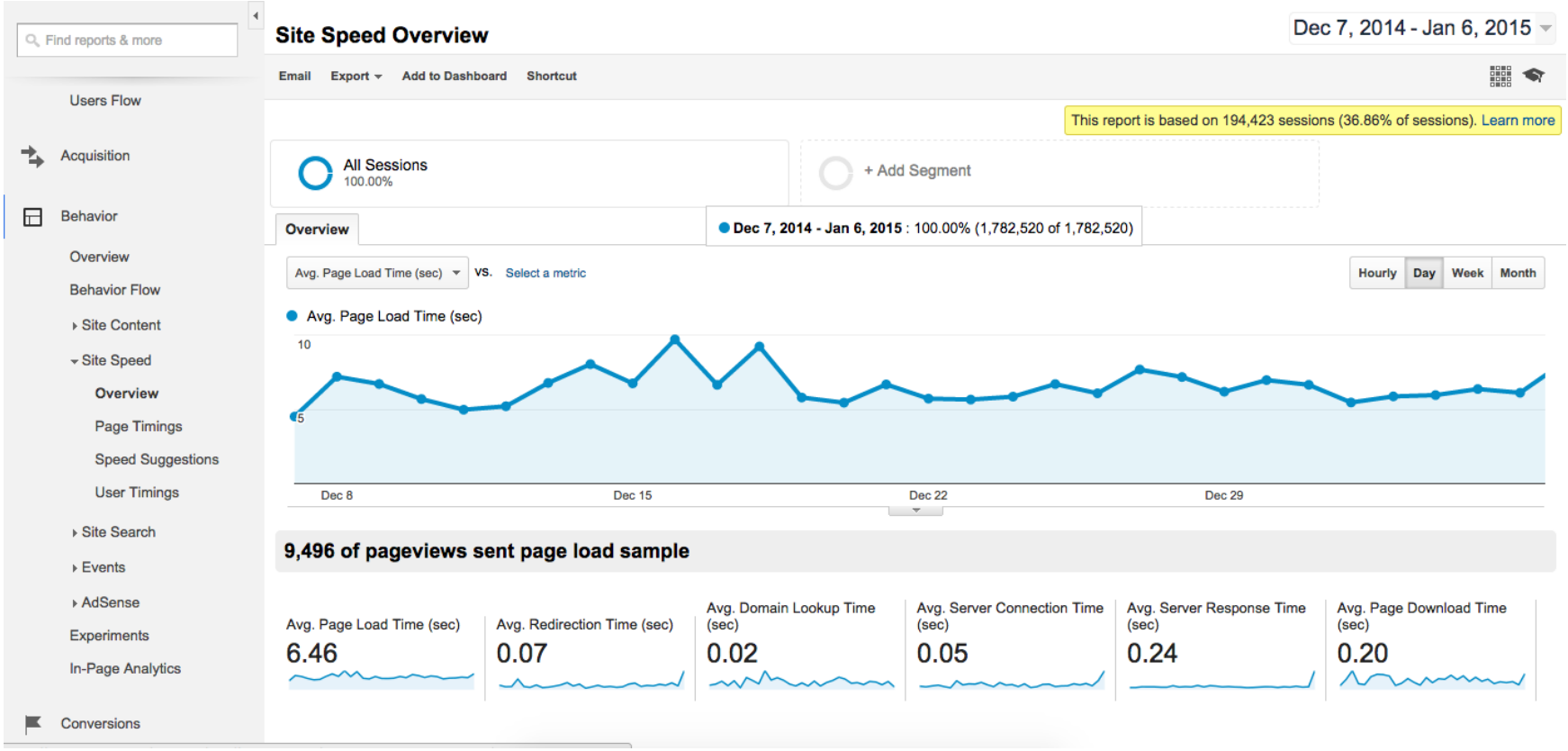
SOASTA MPULSE

soasta.com



GOOGLE ANALYTICS SITE SPEED

google.com/analytics

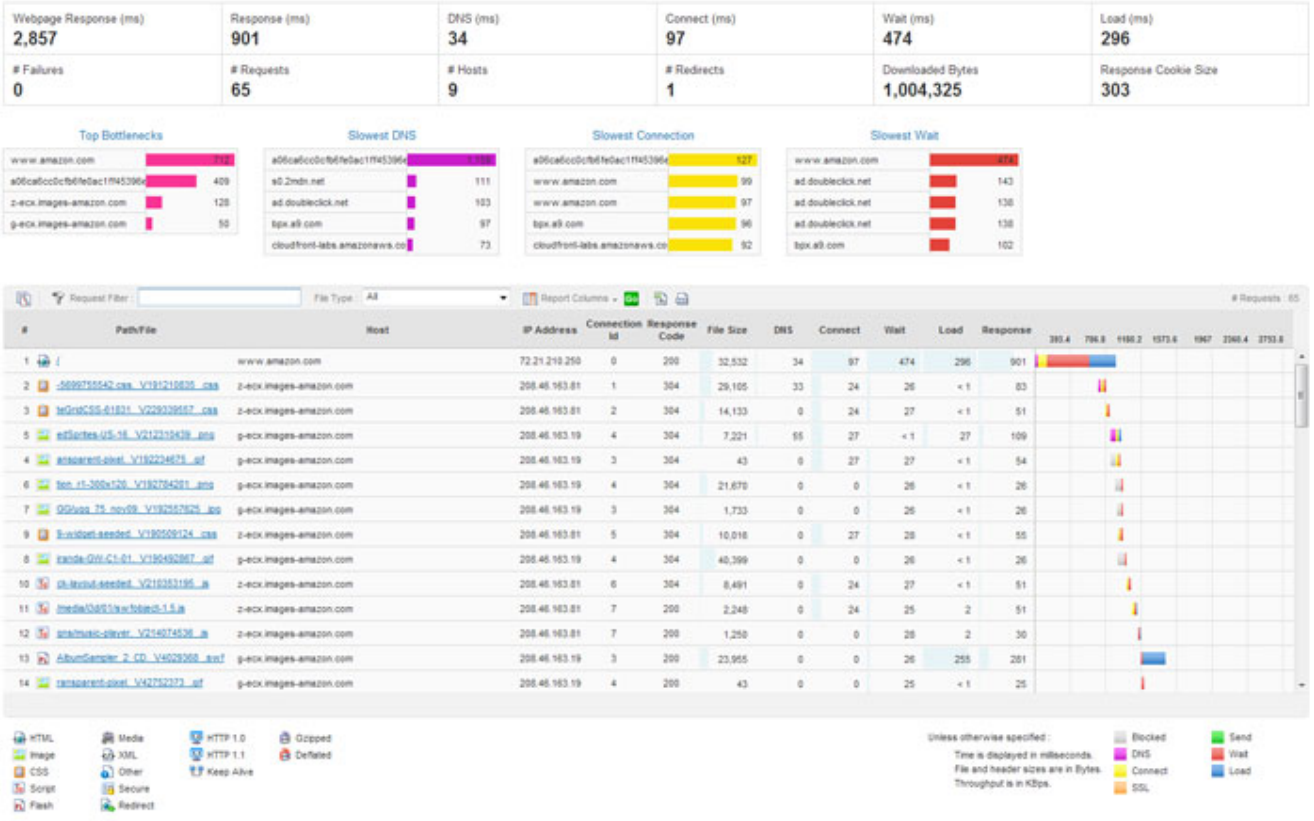


NEW RELIC BROWSER

newrelic.com/browser-monitoring

NEUSTAR WPM

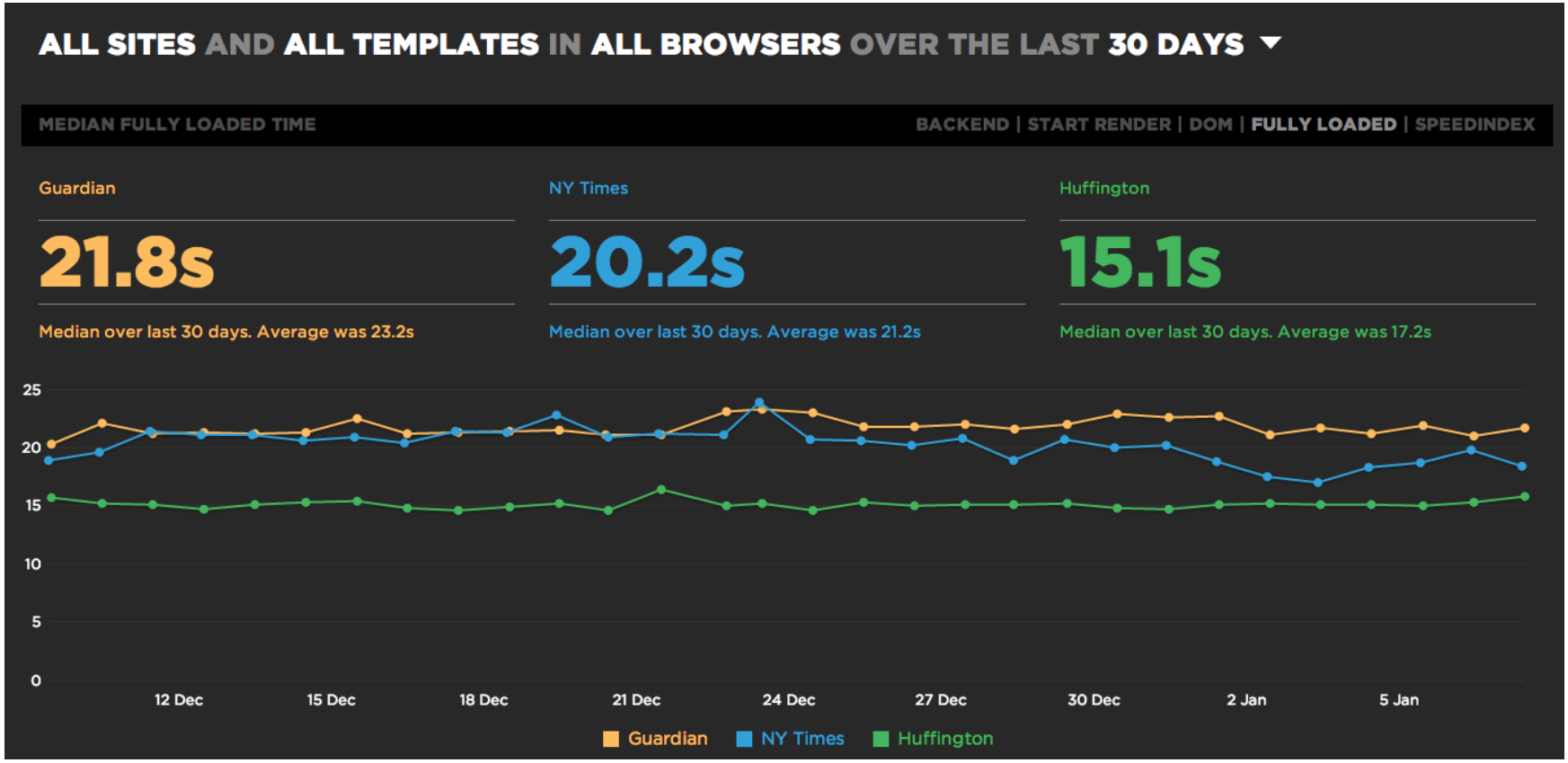
neustar.biz



SPEEDCURVE

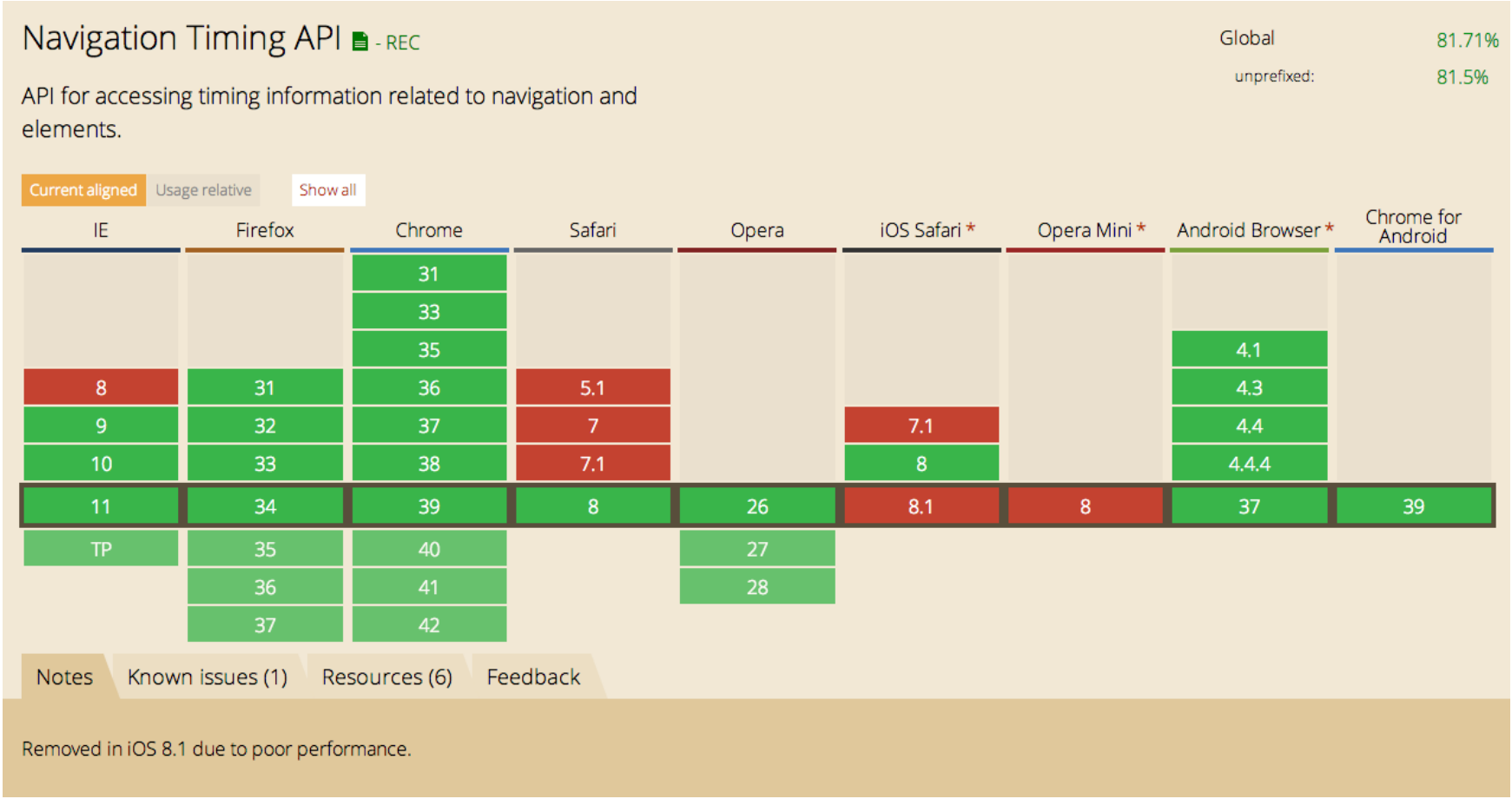
Runs on top of WebPageTest

speedcurve.com



NAVIGATIONTIMING

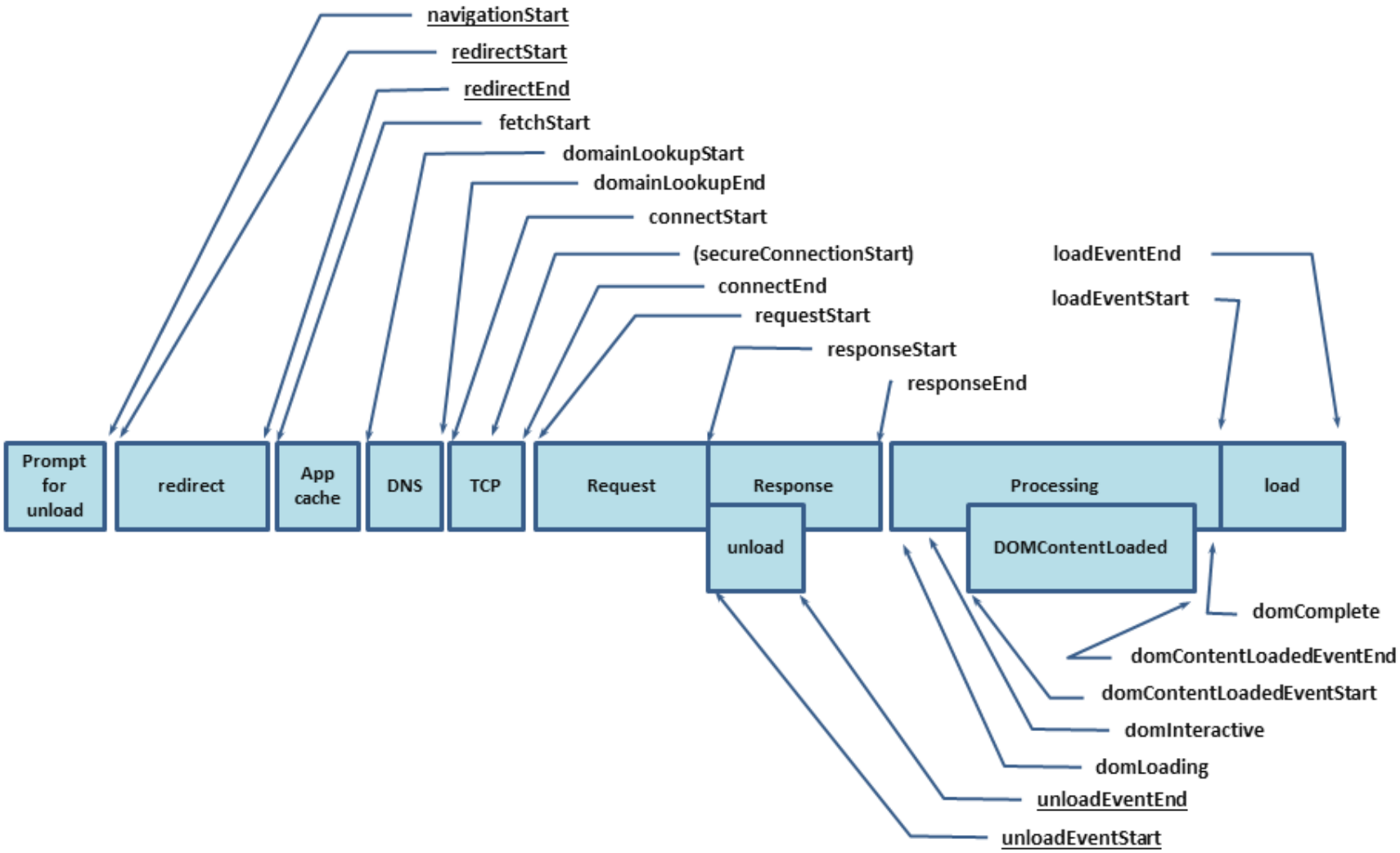
caniuse.com/#feat=nav-timing



TIPS

- Use `fetchStart` instead of `navigationStart` unless you're interested in redirects, tab init time, etc
- `loadEventEnd` will be 0 until *after* the body's `load` event has finished (so you can't measure it in the `load` event)
- We don't have an accurate way to measure the "request time", as "`requestEnd`" is invisible to us (the server sees it)
- `secureConnectionStart` isn't available in IE

TIPS


navigationStart
redirectStart
redirectEnd
fetchStart
domainLookupStart
domainLookupEnd
connectStart
(secureConnectionStart)
connectEnd
requestStart
responseStart
responseEnd
loadEventEnd
loadEventStart
Prompt for unload
redirect
App cache
DNS
TCP
Request
Response
Processing
load
unload
DOMContentLoaded
domComplete
domContentLoadedEventEnd
domContentLoadedEventStart
domInteractive
domLoading
unloadEventEnd
unloadEventStart

TIPS (PT 2)

- iOS still doesn't have support
- Home page scenarios: Timestamps up through `responseEnd` event may be 0 duration because some browsers speculatively pre-fetch home pages (and don't report the correct timings)
- If possbile, do any beaconing of the data as soon as possible. Browser `onbeforeunload` isn't 100% reliable for sending data
- Single-Page Apps: You'll need a different solution for "navigations" (Boomerang + plugin coming soon)

NAVIGATIONTIMING2

www.w3.org/TR/navigation-timing-2

DRAFT

Builds on NavigationTiming:

- Support for Performance Timeline
- Support for High Resolution Time
- timing information for link negotiation
- timing information for prerender

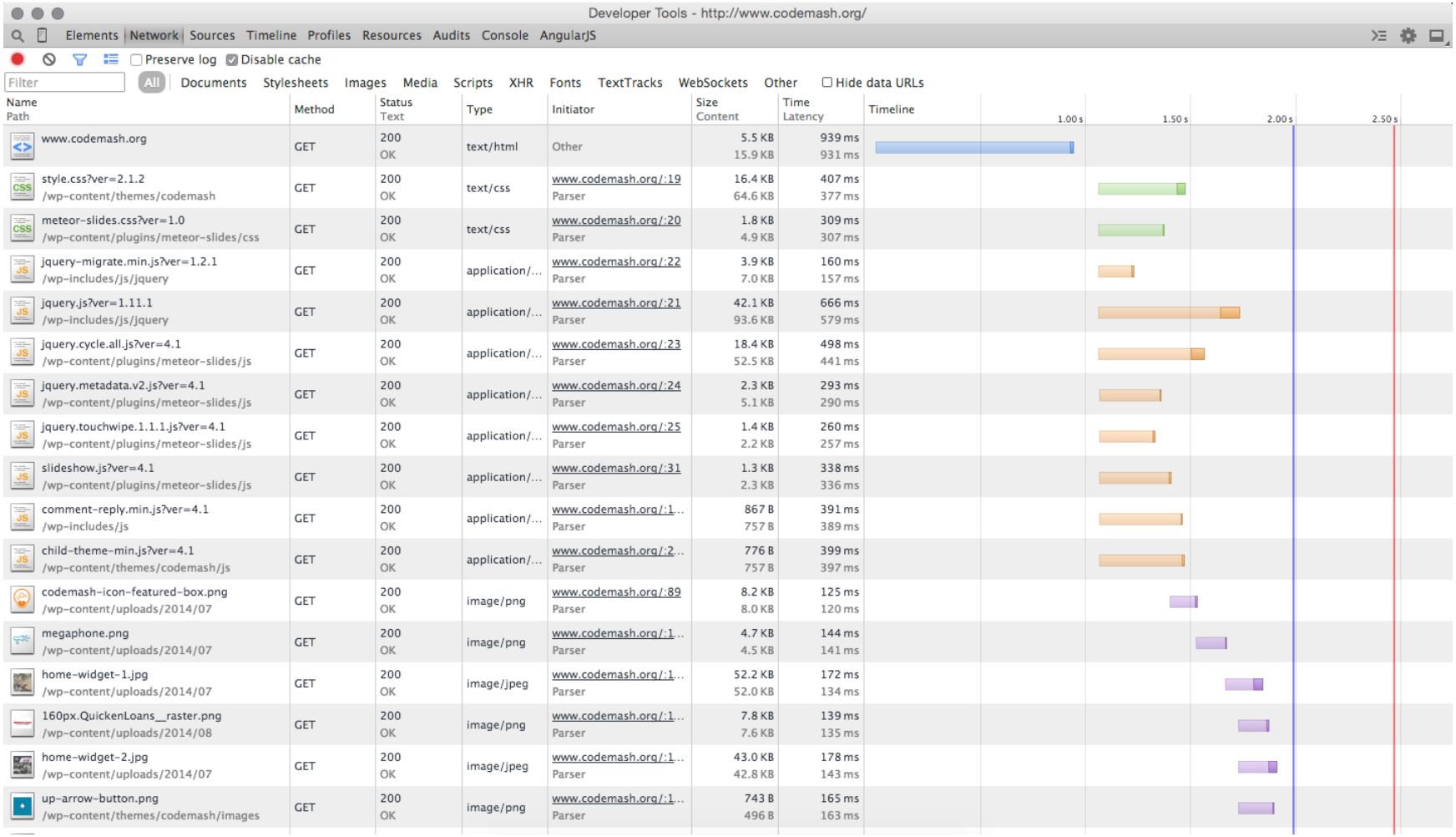
RESOURCETIMING

www.w3.org/TR/resource-timing

Goal: Expose sub-resource performance metrics

Current status: Working Draft

INSPIRATION



HOW IT WAS DONE BEFORE

For dynamically inserted content, you could time how long it took from DOM insertion to the element's onLoad event

HOW IT WAS DONE BEFORE

(this isn't practical for all content)

```
var start = new Date().getTime();
var image1 = new Image();
var resourceTiming = function() {
    var now = new Date().getTime();
    var latency = now - start;
    alert("End to end resource fetch: " + latency);
};

image1.onload = resourceTiming;
image1.src = 'http://www.w3.org/Icons/w3c_main.png';
```

WHAT'S WRONG WITH THIS?

- It measures end-to-end download time plus rendering time
- Not practical if you want to measure every resource on the page (IMG, SCRIPT, LINK rel="css", etc)
- `Date().getTime()` is not reliable

RESOURCETIMING

window.performance.getEntries()

```
interface PerformanceEntry {
    readonly attribute DOMString name;
    readonly attribute DOMString entryType;
    readonly attribute DOMHighResTimeStamp startTime;
    readonly attribute DOMHighResTimeStamp duration;
};

interface PerformanceResourceTiming : PerformanceEntry {
    readonly attribute DOMString initiatorType;

    readonly attribute DOMHighResTimeStamp redirectStart;
    readonly attribute DOMHighResTimeStamp redirectEnd;
    readonly attribute DOMHighResTimeStamp fetchStart;
    readonly attribute DOMHighResTimeStamp domainLookupStart;
    readonly attribute DOMHighResTimeStamp domainLookupEnd;
    readonly attribute DOMHighResTimeStamp connectStart;
    readonly attribute DOMHighResTimeStamp connectEnd;
    readonly attribute DOMHighResTimeStamp secureConnectionStart;
    readonly attribute DOMHighResTimeStamp requestStart;
    readonly attribute DOMHighResTimeStamp responseStart;
    readonly attribute DOMHighResTimeStamp responseEnd;
};
```

INTERLUDE: PERFORMANCETIMELINE

www.w3.org/TR/performance-timeline

Goal: Unifying interface to access and retrieve performance metrics

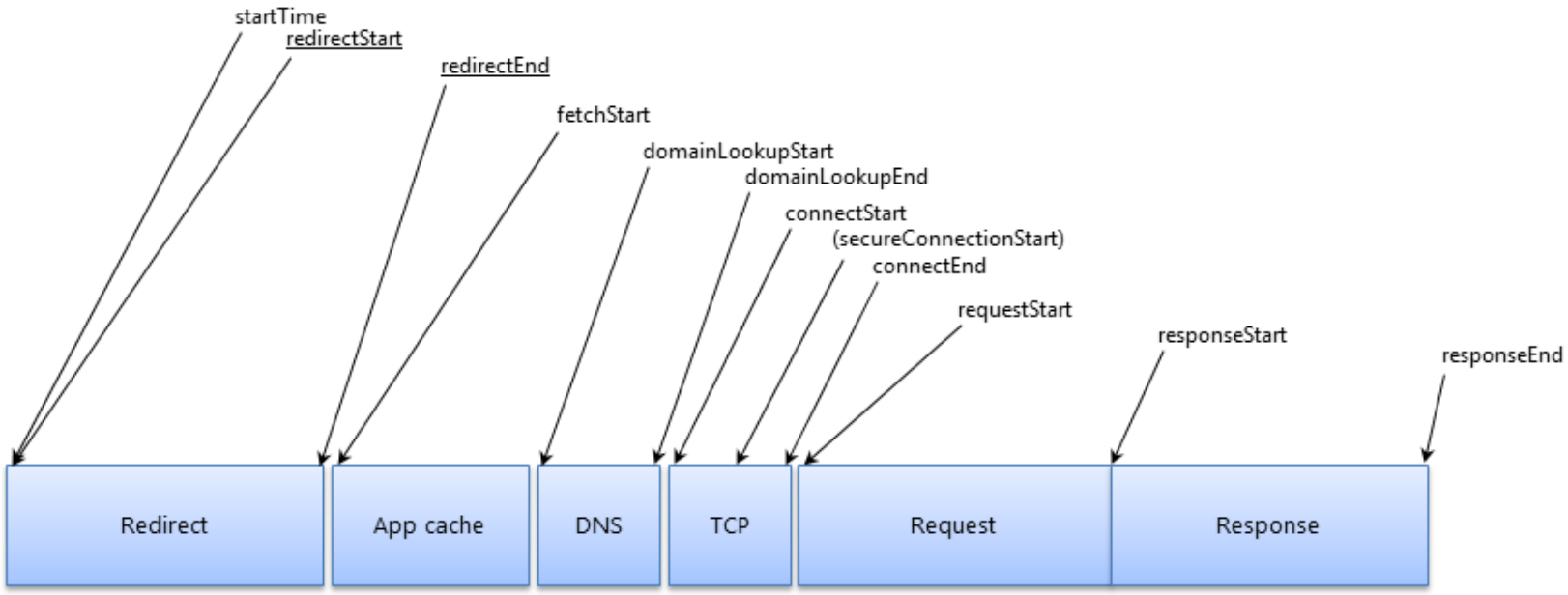
Current status: Recommendation

PERFORMANCETIMELINE

`window.performance`

- `getEntries()`: Gets all entries in the timeline
- `getEntriesByType(type)`: Gets all entries of the specified type (eg `resource`, `mark`, `measure`)
- `getEntriesByName(name)`: Gets all entries with the specified name (eg URL or mark name)

RESOURCETIMING



HOW TO USE

```
window.performance.getEntriesByType("resource")
[0]
```


```
{
    connectEnd: 566.357000003336,
    connectStart: 566.357000003336,
    domainLookupEnd: 566.357000003336,
    domainLookupStart: 566.357000003336,
    duration: 4.275999992387369,
    entryType: "resource",
    fetchStart: 566.357000003336,
    initiatorType: "img",
    name: "https://www.foo.com/foo.png",
    redirectEnd: 0,
    redirectStart: 0,
    requestStart: 568.4959999925923,
    responseEnd: 570.6329999957234,
    responseStart: 569.4220000004862,
    secureConnectionStart: 0,
    startTime: 566.357000003336
}
```

INITIATORTYPE

`localName` of that element:

- `img`
- `link`
- `script`
- `css: url(), @import`
- `xmlhttprequest`

USE CASES

- Send all resource timings to your backend analytics
- Raise an analytics event if any resource takes over X seconds to download (and trend this data)
- Watch specific resources (eg third-party ads or analytics) and complain if they are slow

BUFFER

- There is a ResourceTiming buffer (per IFRAME) that stops filling after its size limit is reached (default: 150 entries)
- Listen for the `onresourcetimingbufferfull` event
- `setResourceTimingBufferSize(n)` and `clearResourceTimings()` can be used to modify it
- Don't just: `setResourceTimingBufferSize(99999999)` as this can lead to browser memory growing unbound

COMPRESSING

- Each resource is ~ 500 bytes `JSON.stringify()`'d
- HTTP Archive tells us there's 99 HTTP resources on average, per page, with an average URL length of 85 bytes
- That means you could expect around 45 KB of ResourceTiming data per page load
- Compress it: nicj.net/compressing-resourcetiming

COMPRESSING

Converts:


```
{
    "responseEnd":323.1100000002698,
    "responseStart":300.5000000000000,
    "requestStart":252.68599999981234,
    "secureConnectionStart":0,
    "connectEnd":0,
    "connectStart":0,
    "domainLookupEnd":0,
    "domainLookupStart":0,
    "fetchStart":252.68599999981234,
    "redirectEnd":0,
    "redirectStart":0,
    "duration":71.42400000045745,
    "startTime":252.68599999981234,
    "entryType":"resource",
    "initiatorType":"script",
    "name":"http://foo.com/js/foo.js"
}
```

COMPRESSING

To:

```
{
    "http://": {
        "foo.com/": {
            "js/foo.js": "370,1z,1c",
            "css/foo.css": "48c,5k,14"
        },
        "moo.com/moo.gif": "312,34,56"
    }
}
```

Overall, compresses ResourceTiming data down to 15% of its original size

github.com/nicjansma/resourcetiming-compression.js

TIMING-ALLOW-ORIGIN

- By default, cross-origin resources expose timestamps for only the `fetchStart` and `responseEnd` attributes
- This is to protect your privacy (attacker can't load random URLs to see where you've been)
- Override by setting `Timing-Allow-Origin` header
- `Timing-Allow-Origin = "Timing-Allow-Origin" ":" origin-list-or-null | "*"`
- If you have a CDN, **use this**
- **Note**: Third-party libraries (ads, analytics, etc) must set this on their servers. 5% do according to HTTP Archive. Google, Facebook, Disqus, mPulse, etc.

BLOCKING TIME

- Browsers will open a limited number of connections to each unique origin (protocol/server name/port)
- If there are more resources than the # of connections, the later resources will be "blocking", waiting for their turn to download
- `duration` includes Blocking time!
- So in general, don't use `duration`, but this is all you get with cross-origin resources.

BLOCKING TIME

Calculate:

```
var waitTime = 0;
if (res.connectEnd && res.connectEnd === res.fetchStart)
{
    waitTime = res.requestStart - res.connectEnd;
}
else if (res.domainLookupStart)
{
    waitTime = res.domainLookupStart - res.fetchStart;
}
```

DIY / OPEN SOURCE

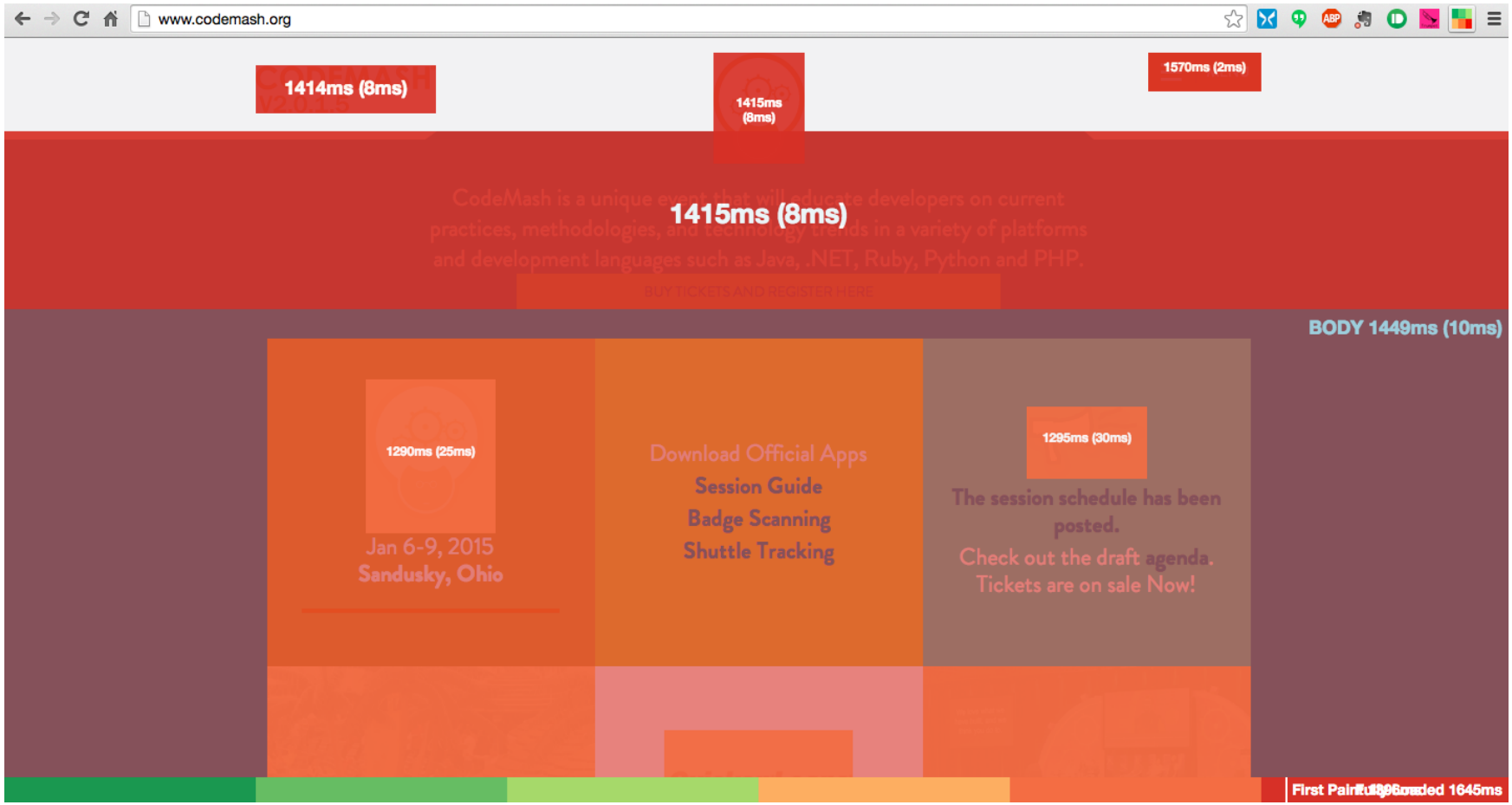
- Compress + send this data to your backend for logging
- Show any page's resources via a bookmarklet: github.com/andydavies/waterfall
- Heatmap bookmarklet / Chrome extension: github.com/zeman/perfmap
- Nurun's Performance Bookmarklet: github.com/nurun/performance-bookmarklet
- Boomerang supports ResourceTiming: github.com/lognormal/boomerang

ANDY DAVIES' WATERFALL.JS

github.com/andydavies/waterfall

MARK ZEMAN'S PERFMAP

github.com/zeman/perfmap



NURUN'S PERFORMANCE BOOKMARKLET

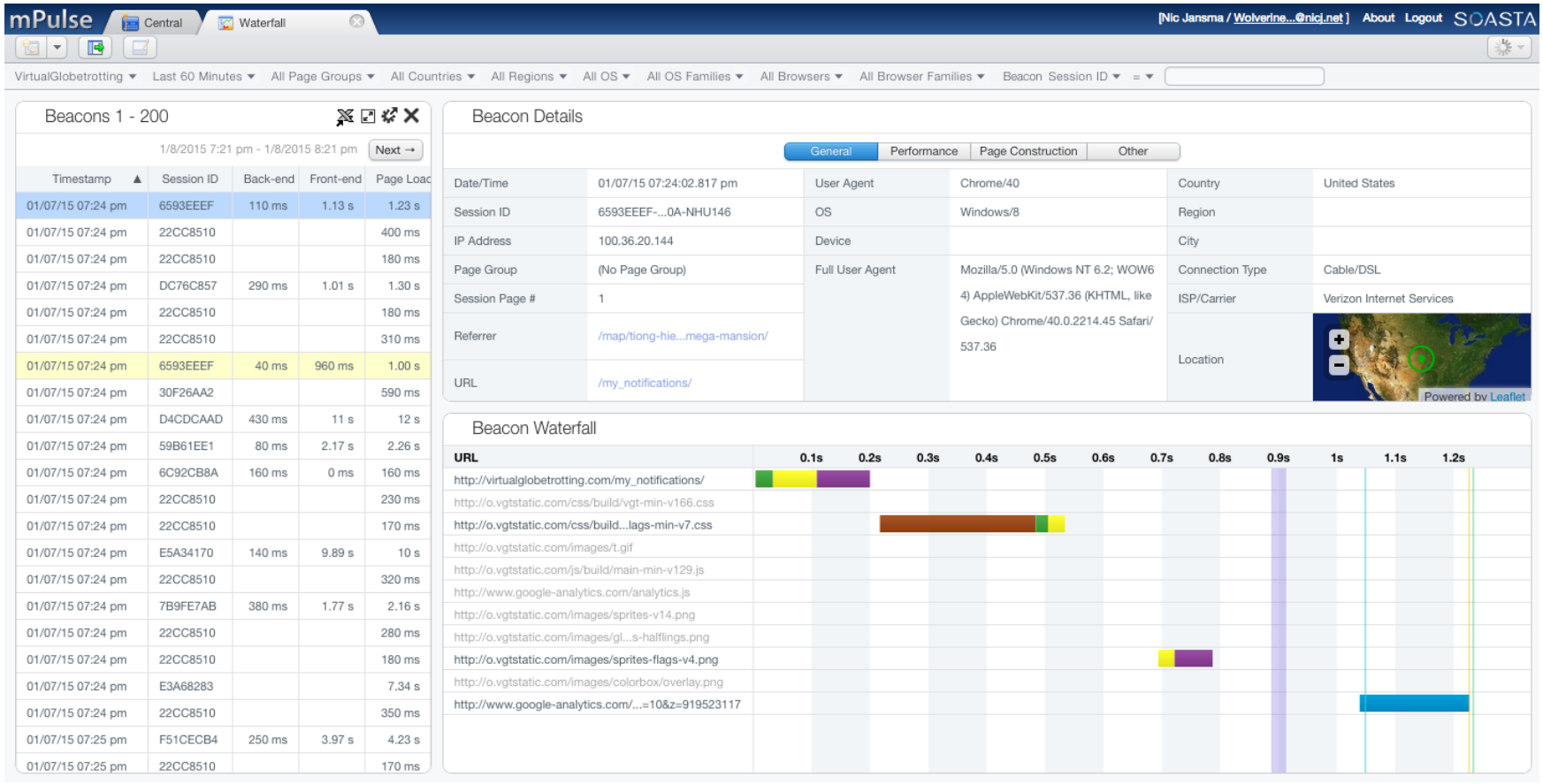
github.com/nurun/performance-bookmarklet

COMMERCIAL

- SOASTA mPulse: soasta.com
- New Relic Browser: newrelic.com
- App Dynamics Web EUEM: appdynamics.com

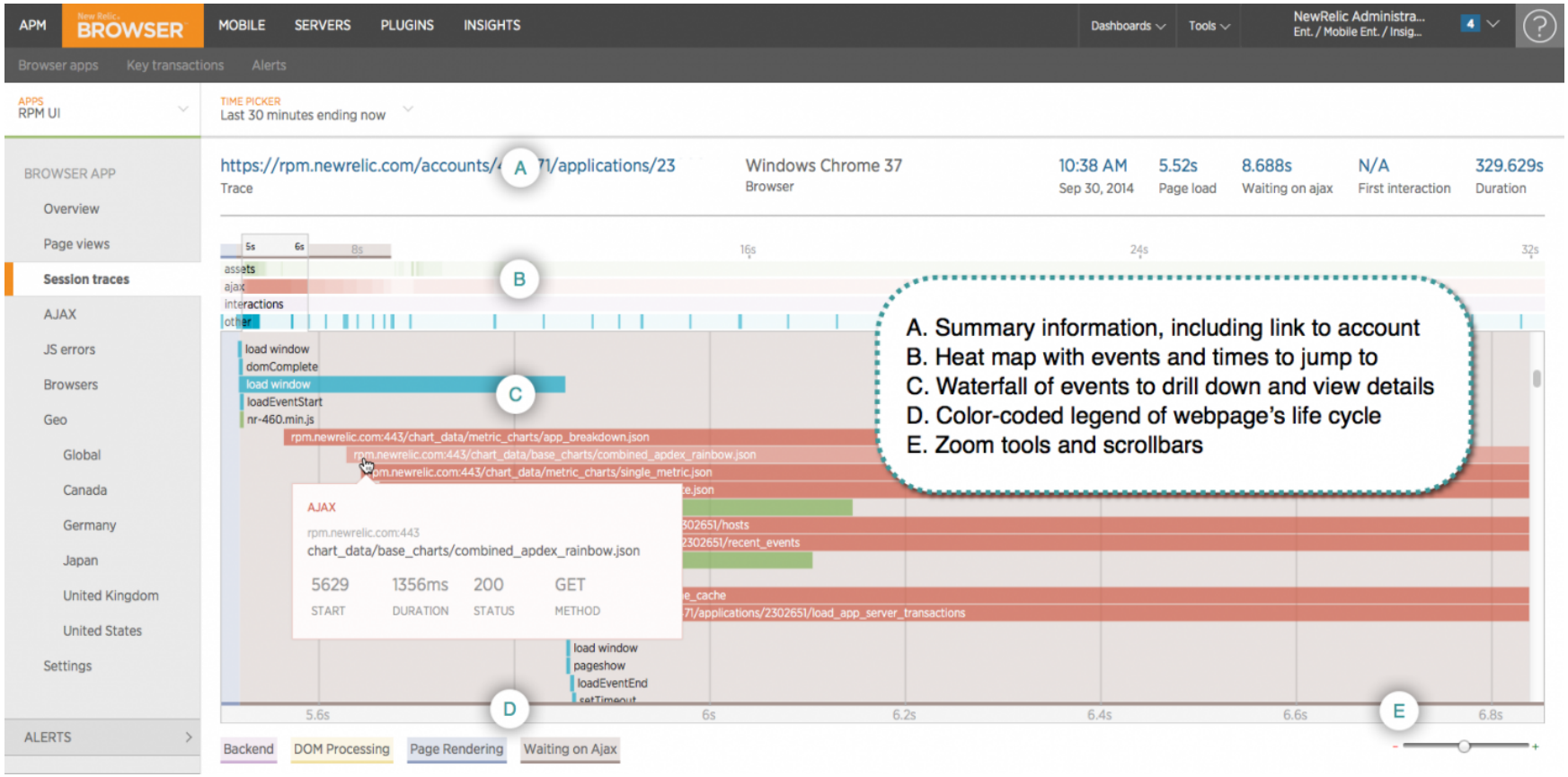
SOASTA MPULSE

soasta.com



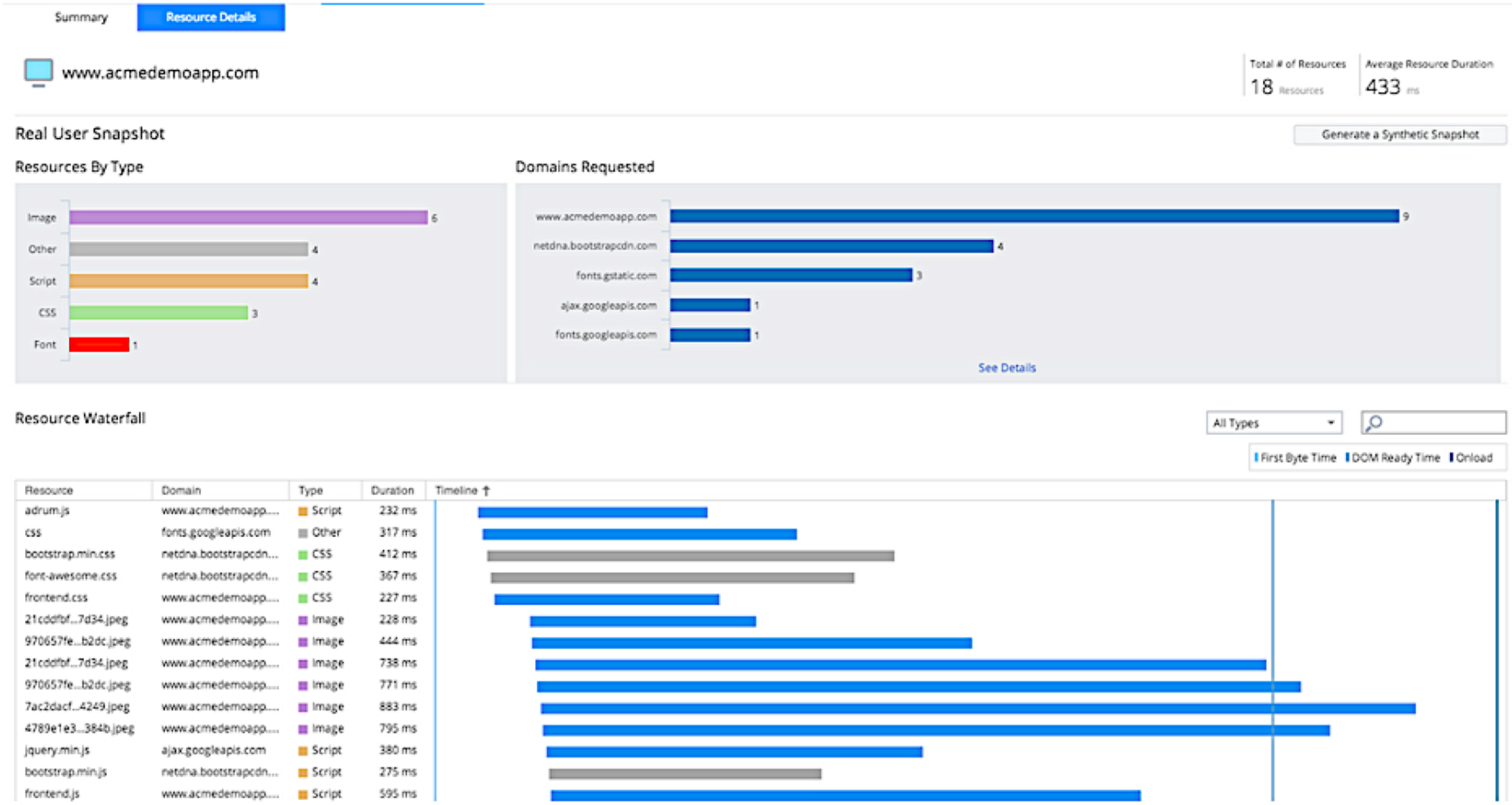
NEW RELIC BROWSER

newrelic.com/browser-monitoring



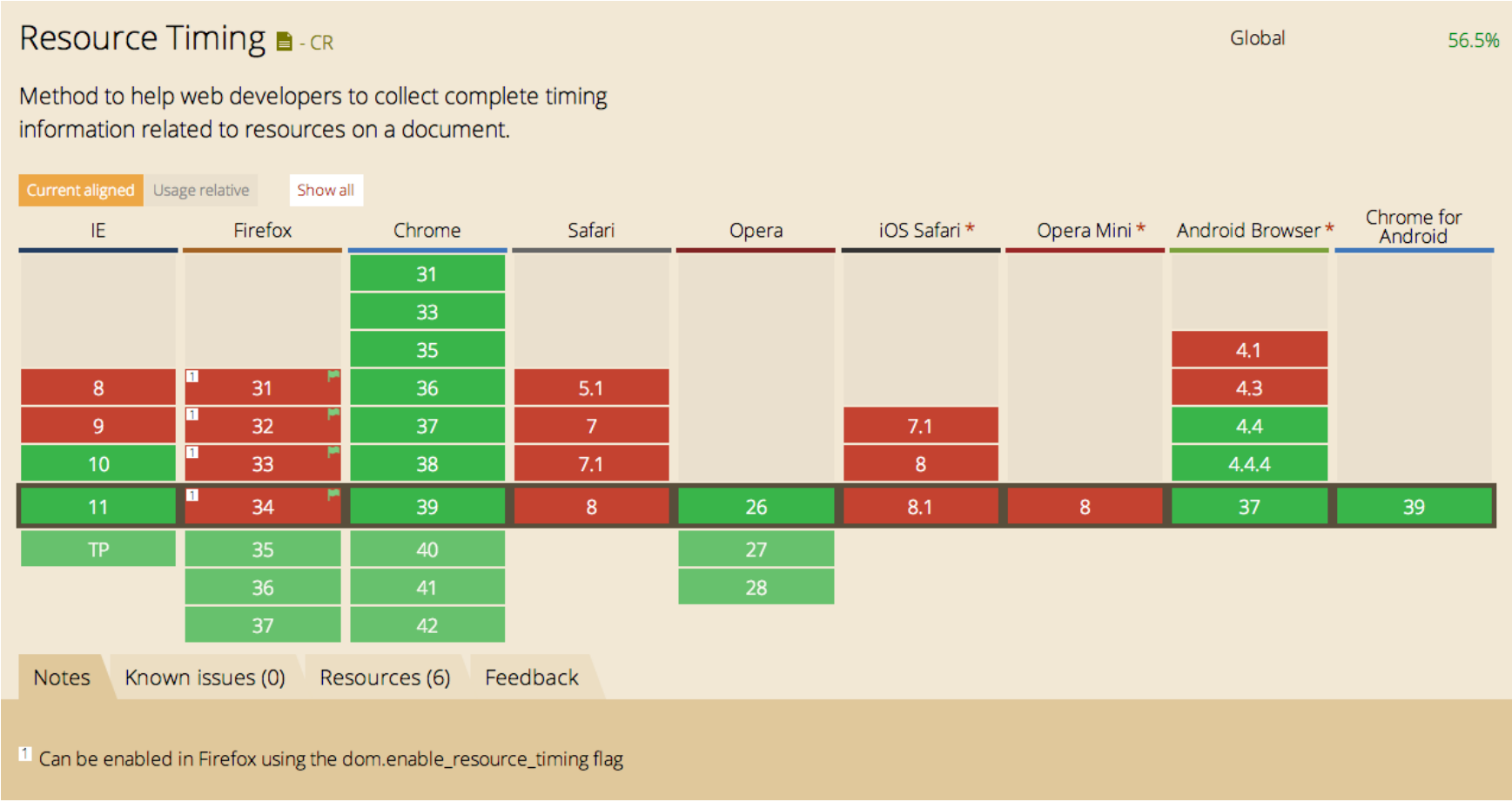
APP DYNAMICS WEB EUEM

appdynamics.com



RESOURCETIMING

caniuse.com/#feat=resource-timing



TIPS

- For many sites, most of your content will not be same-origin, so ensure all of your CDNs and third-party libraries send `Timing-Allow-Origin`
- What isn't included in ResourceTiming:
  - The root HTML page (get this from `window.performance.timing`)
  - Transfer size or content size (privacy concerns)
  - HTTP code (privacy concerns)
  - Content that loaded with errors (eg 404s)

TIPS (PT 2)

- If you're going to be managing the ResourceTiming buffer, make sure no other scripts are managing it as well
- The `duration` attribute includes Blocking time (when a resource is behind other resources on the same socket)
- Each `IFRAME` will have its own ResourceTiming data, and those resources won't be included in the parent FRAME/document. So you'll need to traverse the document frames to get all resources. See github.com/nicjansma/resourcetiming-compression.js for an example
- `about:blank`, `javascript:` URLs will be seen in RT data

USERTIMING

www.w3.org/TR/user-timing

Goal: Standardized interface to note timestamps ("marks") and durations ("measures")

Current status: Recommendation

HOW IT WAS DONE BEFORE

```
var start = new Date().getTime();
// do stuff
var now = new Date().getTime();
var duration = now - start;
```

WHAT'S WRONG WITH THIS?

- *Nothing really, but...*
- `Date().getTime()` is not reliable
- We can do better!

USERTIMING

window.performance

```
partial interface Performance {
    void mark(DOMString markName);

    void clearMarks(optional  DOMString markName);

    void measure(DOMString measureName, optional DOMString startMark,
        optional DOMString endMark);

    void clearMeasures(optional DOMString measureName);
};
```

HOW TO USE - MARK

```
// mark
performance.mark("start");
performance.mark("end");

performance.mark("another");
performance.mark("another");
performance.mark("another");
```

HOW TO USE - MARK

```
// retrieve
performance.getEntriesByType("mark");

[
    {
        "duration":0,
        "startTime":150384.48100000096,
        "entryType":"mark",
        "name":"start"
    },
    {
        "duration":0,
        "startTime":150600.5250000013,
        "entryType":"mark",
        "name":"end"
    },
    ...
]
```

HOW TO USE - MEASURE

```
// measure
performance.mark("start");
// do work
performance.mark("start2");

// measure from "now" to the "start" mark
performance.measure("time to do stuff", "start");

// measure from "start2" to the "start" mark
performance.measure("time from start to start2", "start", "start2");
```

HOW TO USE - MEASURE

```
// retrieval - specific
performance.getEntriesByName("time from start to start2", "measure");

[
    {
        "duration":4809.890999997151,
        "startTime":145287.66500000347,
        "entryType":"measure",
        "name":"time from start to start2"
    }
]
```

BENEFITS

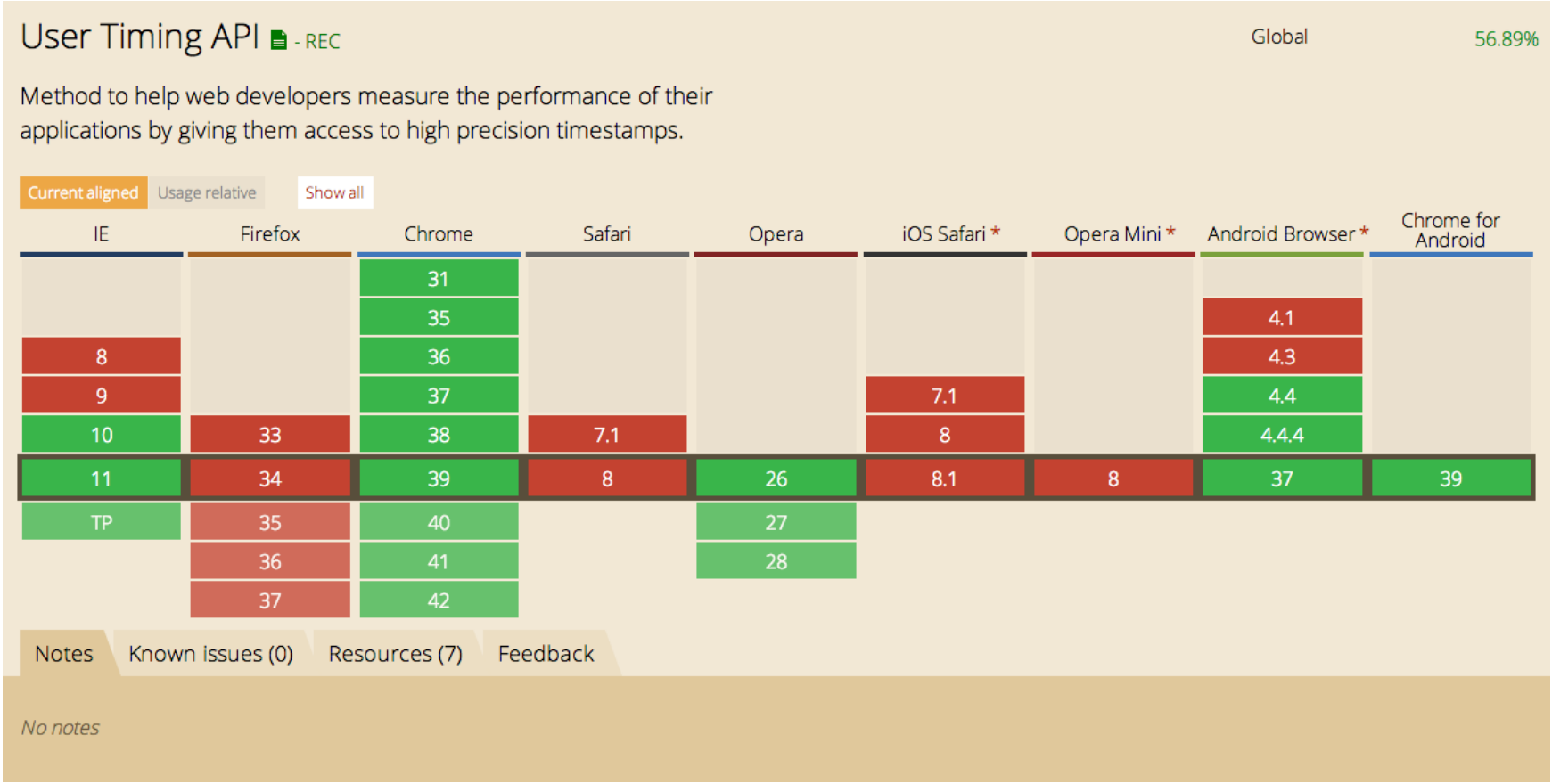
- Uses the `PerformanceTimeline`, so `marks` and `measures` are in the PerformanceTimeline along with other events
- Uses `DOMHighResTimestamp` instead of `Date` so sub-millisecond, monotonically non-decreasing, etc
- More efficient, as the native browser runtime can do math quicker and store things more performantly than your JavaScript runtime can

USE CASES

- Easy way to add profiling events to your application
- Note important scenario durations in your Performance Timeline
- Measure important durations for analytics
- Browser tools are starting to add support for showing these

USERTIMING

caniuse.com/#feat=user-timing



USERTIMING.JS

- Polyfill that adds UserTiming support to browsers that do not natively support it.
- UserTiming is accessed via the PerformanceTimeline, and requires window.performance.now() support, so UserTiming.js adds a limited version of these interfaces if the browser does not support them
- github.com/nicjansma/usertiming.js

DIY / OPEN SOURCE

- Compress + send this data to your backend for logging
- WebPageTest sends UserTiming to Google Analytics, Boomerang and SOASTA mPulse

COMMERCIAL

- SOASTA mPulse: soasta.com
- WebPageTest: webpagetest.org

TIPS

- Not the same as Google Analytic's "User Timings" API (`_trackTiming(...)`)

YOUR JOB
MAKE IT FAST!

LINKS

- Presentation: slideshare.net/nicjansma
- Code: github.com/nicjansma/talks

Thanks - Nic Jansma - nicj.net - @NicJ