

**Zen and the Art of Telemetry:**

An Inquiry into Performance



ANTHONY ZHANG

Zen and the Art of Telemetry: An Inquiry into  
Performance

## What?

I work with the Desktop Performance team.  
We find slow things and turn them into fast things.



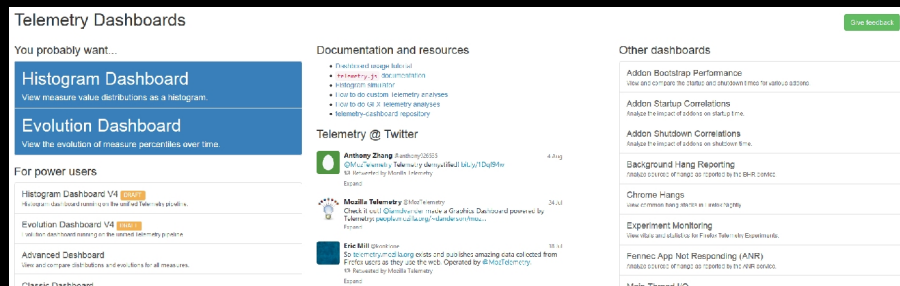
Performance <3 data.  
That's where Telemetry comes in.

Hi, I'm Anthony from the Desktop Performance team at Mozilla. Today I'll be talking about Telemetry and how you can use it to answer questions.

The desktop performance team has two main goals: to identify slow things, and to fix them. The focus is mainly on desktop Firefox and the Gecko platform.

## Enter Telemetry

Firefox obtains (anonymous) metrics while browsing.  
We receive several hundred gigabytes of these per day.  
All that data is put together and made publicly available.

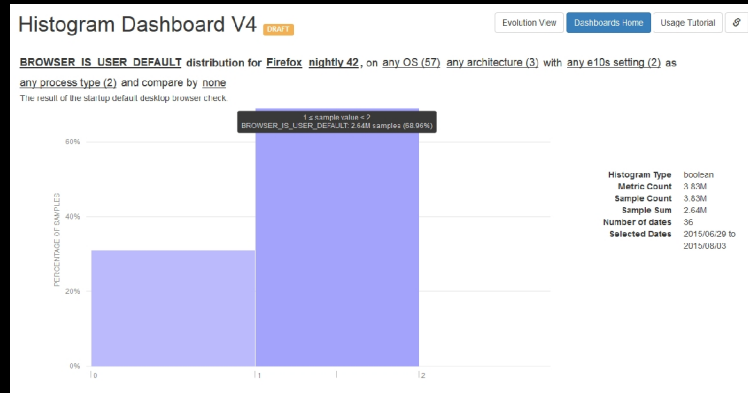


Check out [telemetry.mozilla.org](https://telemetry.mozilla.org)!

Telemetry is our measurements and analytics platform. It includes everything from the code that ships out in your browser, to our data processing pipeline, all the way to the tools that squirt out numbers onto a webpage. It's actually gotten a bit of an overhaul lately, so make sure to go check out all the new things we've put in!

## Q. How often is Firefox the default browser?

The purpose of Telemetry is to answer questions.



A. 68.96% of our nightly users have Firefox as their default browser.

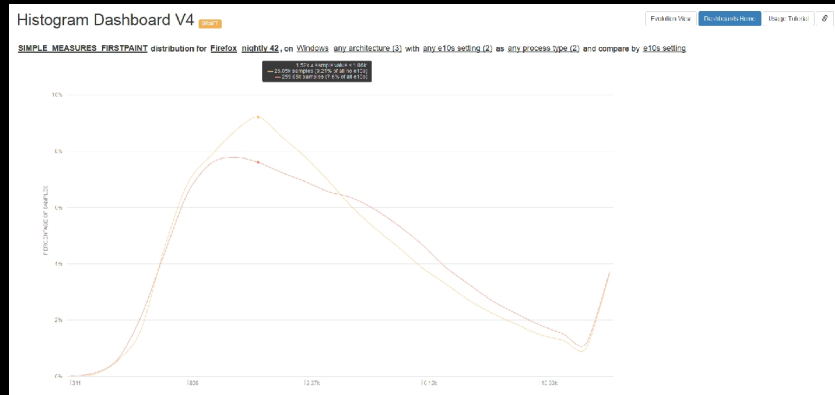
Many of us benefit from real, hard numbers to back up our intuition. Unfortunately, these were always a bit hard to come by – our tools were hard to use, and they couldn't always answer our questions.

The new and improved Telemetry tools now bring the data closer to the people who need it. For example, here's the new histogram dashboard, which lets you view distributions of Telemetry measures in several different ways.

BROWSER\_IS\_USER\_DEFAULT is a Telemetry measure that has value 1 if the browser is the user's default, and 0 otherwise. In a histogram, each bar in the graph represents the number of samples that are in the bar's range, in this case from 1 inclusive to 2 exclusive. So, we can just read off the fraction of users from the chart.

## Q. Does e10s make startup faster?

The dashboards on [telemetry.mozilla.org](https://telemetry.mozilla.org) cover many common use cases.



A. No, it's slightly slower.

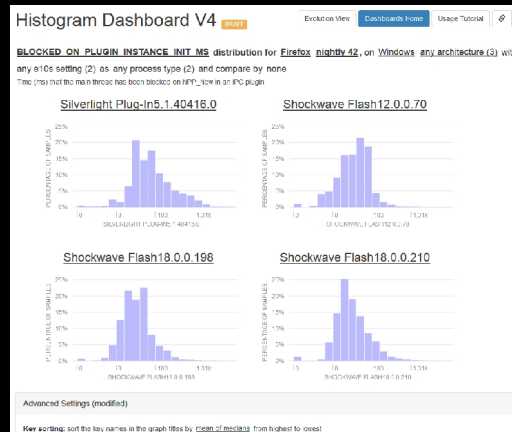
This one's a little more complicated. We want to see how e10s Firefox builds stack up against non-e10s Firefox builds in terms of how long they take to start up.

SIMPLE\_MEASURES\_FIRSTPAINT is a Telemetry measure that checks how long it takes for the first window paint to occur – the startup time, basically. Select “compare by e10s setting”, and you can easily see how they look when you put them next to each other.

From the histograms, you can tell that e10s builds on nightly 42 are slightly slower – more of the graph is to the right. The specific values here are around 2450 milliseconds for non-e10s builds, and around 2710 milliseconds for e10s builds.

## Q. Which plugins tend to freeze the browser on load?

Better dashboards make it easier to make data-driven decisions.



A. Silverlight and Flash (is anyone surprised?)

Let's try something more interesting. In our system there's the concept of a **keyed measure**, which is a measure that has sub-measures, each of which can have their own histograms.

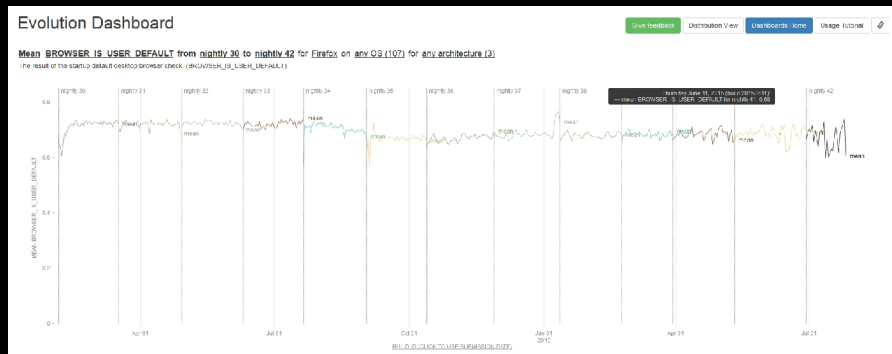
In our case, the keyed measure is `BLOCKED_ON_PLUGIN_INSTANCE_INIT_MS`, and each key in this measure is a plugin that blocks the main thread when instantiated. The histograms represent the number of milliseconds the main thread is blocked.

Here we've chosen to sort the keys by their median values. The top four keys are shown, and we can read off the names of those keys with the highest median blocking times.

Of course, it's Silverlight and Flash. Java makes an appearance a little lower in the list as well.

## Q. How has default-browser-ness changed over time?

There are also lots of other, more specialized views available on [telemetry.mozilla.org](https://telemetry.mozilla.org).



A. The fraction of nightly users with Firefox as their default browser is consistently around 70%.

There's also the evolution dashboard, which is intended to answer questions about how measure values change over time. Together with the distribution dashboard, we can answer a lot of questions already.

In this case we're plotting the average value of the browser default flag, which we've seen earlier. The evolution dashboard can show values between different versions, so we can actually see long term trends in our products.

Here, we see that around 70% of our users have Firefox as their default browser. However, in addition to what the distribution dashboard told us, we now know that the trend is staying roughly the same over time. This is useful for pinpointing regressions and comparing different versions with each other.

## Telemetry.js

Can't answer your question with the dashboards?  
Make your own with Telemetry.js!

```
console.log("Available dates:\n" + evolutionMap[""].dates().join("\n"));
});

Getting the overall median value of GC_MS on nightly 42 on Windows from July 19, 2015 to August 1, 2015:

Telemetry.init(function() {
  Telemetry.getEvolution("nightly", "42", "GC_MS", {os: "Windows_NT"}, true, function(evolutionMap) {
    var evolution = evolutionMap[""].dateRange(new Date("2015-07-19"), new Date("2015-08-01"));
    var histogram = evolution.histogram();
    console.log("Median GC_MS: " + histogram.percentile(50));
  });
});
```

### API Reference

```
Telemetry.init(function callback() { ... })
```

Initializes Telemetry.js with various pieces of metadata that are used by a lot of the other functionality in the library. When

The library runs in the browser or in Node.js.  
We also have an API!

Unfortunately, we can't answer every question with just these tools. That's why we made Telemetry.js, the library that powers many of our dashboards. It's on NPM if you want to use it with Node, and in the browser, you can import it on your webpage and start manipulating the data right away.

Check out our documentation! You can get up and running with less than 5 lines of code.

One of the cool things we're doing with Telemetry.js is the automated regression detector. Basically, we obtain the latest data every day for every measure, apply statistical magic to it, and send out alerts when things change significantly – we can detect performance regressions completely automatically!



**I still have questions!**

We've got you covered.  
Let's say we have a more unusual request.

**To what extent is start-up time correlated  
to the current phase of the moon?**

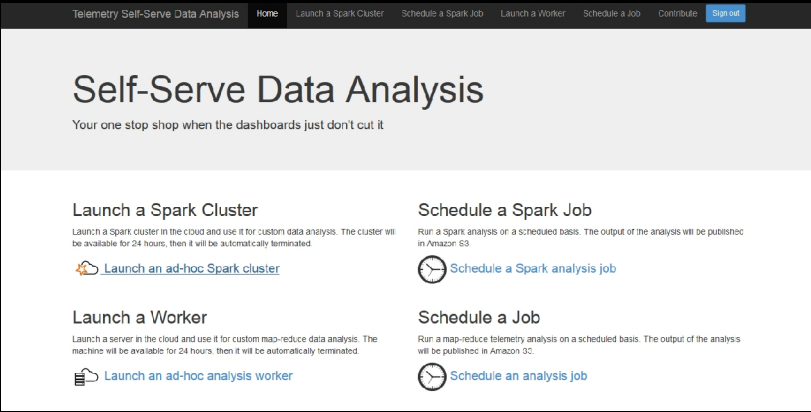
Follow along with the code on [git.io/vOhTI](https://git.io/vOhTI).

But even with the raw, aggregated data – that's still sometimes not enough. Sometimes, we need to see the individual submissions from each user.

For example, let's say we were wondering if the moon was causing the browser to start slower. This is a toy problem, but it does make for an interesting demonstration.

**Start your browsers...**

**For everything else, there's custom Telemetry analyses.**



Head on over to [telemetry-dash.mozilla.org](https://telemetry-dash.mozilla.org)!

We've got a super slick custom analysis platform set up. It uses Spark and IPython, all running on Amazon Web Services. This is available to anyone with a Mozilla email.

That's the site – [telemetry-dash.mozilla.org](https://telemetry-dash.mozilla.org). Log in, request an ad-hoc Spark cluster, and wait for it to start. We'll provision the servers, set up the analysis environment, and make it accessible over SSH, and you can start writing code within minutes.

Through the magic of slide shows, we're going to skip ahead to where the cluster is all set up for us.

## Titles are hard

Let's write code instead:

### Moon Phase Correlation Analysis

```
In [ ]: from moztelemetry import get_pings, get_pings_properties, get_one_ping_per_client
```

This [Wikipedia article](#) has a nice description of how to calculate the current phase of the moon. In code, that looks like this:

```
In [2]: def approximate_moon_visibility(current_date):  
    days_per_synodic_month = 29.530588853 # change this if the moon gets towed away  
    days_since_known_new_moon = (current_date - dt.date(2015, 7, 16)).days  
    phase_fraction = (days_since_known_new_moon % days_per_synodic_month) / days_per_synodic_month  
    return (1 - phase_fraction if phase_fraction > 0.5 else phase_fraction) * 2  
  
def date_string_to_date(date_string):  
    return dt.datetime.strptime(date_string, "%Y%m%d").date()
```

So far, so good.

There's already a few example notebooks set up in our environment, but we're going to start from a blank one.

First, we'll need all the libraries we'll be using. Most importantly, there's Matplotlib, Numpy, and our Telemetry bindings for Spark. The bindings let us easily access and manipulate the raw pings we'll be working with.

Calculating the moon phase is as simple as implementing the algorithm described on Wikipedia.

## Let's get messy

### Time to get ourselves some data!

Let's randomly sample 10% of pings for nightly submissions made from 2015-07-05 to 2015-08-05:

```
In [4]: pings = get_pings(sc, app="Firefox", channel="nightly", submission_date=("20150705", "20150805"), fraction=0.1, schema="v4")
```

Extract the startup time metrics with their submission date and make sure we only consider one submission per user:

```
In [5]: subset = get_pings_properties(pings, ["clientId", "meta/submissionDate", "payload/simpleMeasurements/firstPaint"])
subset = get_one_ping_per_client(subset)
cached = subset.cache()
```

Obtain an array of pairs, each containing the moon visibility and the startup time:

```
In [16]: pairs = cached.map(lambda p: (approximate_moon_visibility(date_string_to_date(p["meta/submissionDate"])), p["payload/simpleMeasurements/firstPaint"]))
pairs = np.asarray(pairs.filter(lambda p: p[1] != None and p[1] < 100000000).collect())
```

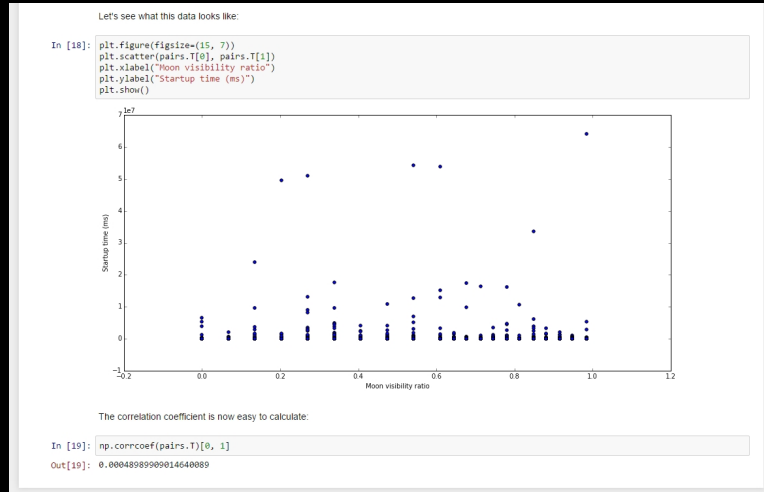
This part takes about 10 minutes to run.

There's about 900k total unique users submitting start-up time metrics in the specified period.

Now we're going to actually obtain the raw ping data. We'll be taking 10% of all submissions made from July 5 to August 5, then extracting the client ID, submission date, and start-up time from each one. Then, we're going to take only one random submission from each unique client ID – we don't want any unusual clients to be skewing our results. It's pretty easy to calculate the moon visibility given a submission date. The code over here will result in a whole bunch of pairs, each containing the moon visibility and start-up time.

## Show me the numbers!

Plots are a great way to check your answers:



Now we have everything we need to calculate correlation. Numpy has a super convenient way to calculate correlation matrices, so we'll just use that. We can also tell from the scatter plot that moving along the horizontal axis, we tend to have the same distribution of points, no matter what our position is. That's evidence toward the variables being independent.

## The verdict

We see that the correlation is roughly 0.0005.

The moon does not have a significant  
effect on Firefox start-up time.

(in the last month)

(on nightly)

(for now)

So, the moon is probably not slowing down Firefox.  
Good to know.

The tools that make this analysis possible are powerful  
enough to do a whole lot more.

## Links and stuff

### **Telemetry Dashboards/Documentation**

[telemetry.mozilla.org](https://telemetry.mozilla.org)

### **Telemetry Demystified**

[anthony-zhang.me/blog/telemetry-demystified](https://anthony-zhang.me/blog/telemetry-demystified)

### **Custom Telemetry Analyses**

[telemetry-dash.mozilla.org](https://telemetry-dash.mozilla.org)

### **Performance Team @ Mozilla**

[wiki.mozilla.org/Performance](https://wiki.mozilla.org/Performance)

### **Example Analysis: Moon Phase Correlation**

[git.io/vOhTI](https://git.io/vOhTI)

We're on Twitter too! @MozTelemetry

Also, #perf @ irc.mozilla.org.

That was a quick overview of what's possible with Telemetry.

There's a lot more to the details, and if you want to do some reading, there are plenty of links here to start off with.

Thanks for listening! I will now be taking questions.