Loading performance

Lighthouse is a great tool but you could do more than just get results from only one computer and one location (as good as the data you get is).

PageSpeedTest is even better. It allows you to test from different locations and devices and it presents more data about your site's performance...

If you're willing to pay for it.

Ever since the acquisition you can run WebPageTest on individual sites for free but the license for using their API is a paid service.

While not as flashy or geographically diersified as PageSpeedTest, there are some web native APIs that provide similar features.

Navigation Timing : measures the speed of requests for HTML documents

Resource Timing : measures the speed of requests for document-dependent resources, everything that is not HTML

So, with both of these APIs we can get a pretty accurate picture of how your site is performing for individual users.

For each page that you visit the browser will record a series of activities into a buffer. You can access the buffer with Javascript.

```
performance.getEntriesByType('navigation');
```

```
performance.getEntriesByType('resource');
```

DNS lookup time

```
// Measuring DNS lookup time
const [pageNav] = performance.getEntriesByType('navigation');
const totalLookupTime = pageNav.domainLookupEnd - pageNav.domainLookupSta
```

Warning

You can't always rely on timings to be populated. Timings provided in both APIs will have a value of 0 in some cases. For example, a DNS lookup may be served by a local cache.

Additionally, any timings for cross-origin requests may be unavailable if those origins don't set a `Timing-Allow-Origin` response header.

Connection negotiation

Connecting to a web server may also add to the latency of your web content. The connection latency includes TLS negotiation if we're using HTTPS.

The connection phase consists of three timings:

- `connectStart` indicates when the browser starts to open a connection to a web server
- `secureConnectionStart` marks when the client begins TLS negotiation
- `connectEnd` indicates when the connection to the web server has been established

Measuring total connection time is similar to measuring total DNS lookup time:

```
connectEnd - connectStart
```

if it's a regular HTTP connection or

```
connectionEnd - secureConnectionStart
```

If you're using HTTPS.

Be careful when using `secureConnectionStart`, the value may be 0 if HTTPS isn't used or if the connection is persistent.

The following script shows how to measure connection time:

1. Capture all the `navigation` performance timings into an array
2. Measure normal connection time by subtracting `connectEnd` from `connectStart`
3. Set a variable to hold our secure connection time and initialize it to 0
4. Check if the value of `secureConnectionStart` is greater than 0
    1. If it is then we measure connection speed as `connectEnd` minus `secureConnectionStart`

```
// 1
const [pageNav] = performance.getEntriesByType('nav'navigation'// 2
const connectionTime = pageNav.connectEnd - pageNav.connectStart;

//3
let tlsTime = 0;

// 4
if (pageNav.secureConnectionStart > 0) {
  // i
  tlsTime = pageNav.connectEnd - pageNav.secureConnectionStart;
}
```

Once DNS lookup and connection negotiation ends, we can worry about the timings for documents and associated resources.

Getting data from our users

In this section we'll look at intrisic factors of our application's performance, things like server and client-side architectures, as well as resource size and aspects of our application that we can optimize

Both Navigation Timing and Resource Timing describe loading performance with the following metrics:

- **fetchStart** marks when the browser begins to fetch a resource or a document. This happens before the actual request, and is where the browser checks HTTP, Cache, and Service Worker instances)
- **workerStart** marks when a request starts being handled within a service worker's fetch event handler. This will be 0 when no service worker is controlling the current page

- **requestStart** is when the browser makes the request. responseStart is when the first byte of the response arrives
- **responseEnd** is when the last byte of the response arrives.

These timings allow you to measure multiple aspects of loading performance:

1. Collect all navigation timings into an array
2. The `fetchTime` is the result of subsctacting `fetchStart` from the `responseEnd`
3. We first initalize a variable for the service worker timing
4. We check if the value of 'workerStart' is greater than 0 and if it is we calculate the `workerTime` as `workerStart` minus `fetchStart`

```
//1
const [pageNav] = performance.getEntriesByType('nav'navigation'//2
const fetchTime = pageNav.responseEnd - pageNav.fetchStart;

// 3
let workerTime = 0;

// 4
if (pageNav.workerStart > 0) {
  workerTime = pageNav.responseEnd - pageNav.workerStart;
}
```

You can also measure other aspects of request/response latency.

For example, we can measure how long the request took to complete. This will measure the request duration only and not include any other aspect that may increase the latency of your page load.

You can also measure the time it took for the response to complete. This measures the time the browser spent downloading the response object.

```
const [pageNav] = performance.getEntriesByType('navigation');

const requestTime = pageNav.responseStart - pageNav.requestStart;
```

```
const responseTime = pageNav.responseEnd - pageNav.responseStart;
```

Other things you can measure

Navigation Timing and Resource Timing is useful for more than what we've covered so far. Here are some other relevant timings you can measure:

Measuring latency of page redirects :Redirects are an overlooked source of added latency, especially redirect chains : Latency gets added in a number of ways, such as HTTP-to-HTTPs hops, as well as 302/uncached 301 redirects. The `redirectStart`, `redirectEnd`, and `redirectCount` timings are helpful in assessing redirect latency

How long does it take to run the code in unload event handlers? :In pages that run code in an unload event handler, the browser must execute that code before it can navigate to the next page :`unloadEventStart` and `unloadEventEnd` measure document unloading timings

Measuring timings for large document processing :Document processing is particularly useful if your website sends very large HTML payloads : If this describes your situation, the `domInteractive`, `domContentLoadedEventStart`, `domContentLoadedEventEnd`, and `domComplete` timings may be of interest.

What to do with the data we collect?

Once we have collected the data we want, we want to do something with it.

We can collect the data using [performance observer] to process the collected data.

```
// 1
const observer = new PerformanceObserver(function(list) {
  for (const entry of list.getEntries()) {
    console.log(entry.name);
  }
});
```

```
// 2
performanceObserver.observe({
  entryTypes: [
    'navigation',
    'resource'
  ]
});
'navigation'
```

Then, assuming you've setup your Google Analytics account on your site, you can send performance data to your analytics account for further analysis and reporting.

You could also post your data to an endpoint configured to store and handle the data to present at a later time.

Further reading

- User-centric performance metrics
- UserTiming in Practice
- Resource Timing in Practice