Web Performance Pitfalls

Theresa Enghardt[1], Thomas Zinner[1], and Anja Feldmann[2]

[1] TU Berlin
theresa@inet.tu-berlin.de, zinner@inet.tu-berlin.de
[2] Max-Planck Institute for Informatics
anja@mpi-inf.mpg.de

Abstract. Web performance is widely studied in terms of load times, numbers of objects, object sizes, and total page sizes. However, for all these metrics, there are various definitions, data sources, and measurement tools. These often lead to different results and almost all studies do *not* provide sufficient details about the definition of metrics and the data sources they use. This hinders reproducibility as well as comparability of the results. This paper revisits the various definitions and quantifies their impact on performance results. To do so we assess Web metrics across a large variety of Web pages.
Amazingly, even for such "obvious" metrics as load times, differences can be huge. For example, for more than 50% of the pages, the load times vary by more than 19.1% and for 10% by more than 47% depending on the exact definition of load time. Among the main culprits for such difference are the in-/exclusion of initial redirects and the choice of data source, e.g., Resource Timings API or HTTP Archive (HAR) files. Even "simpler" metrics such as the number of objects per page have a huge variance. For the Alexa 1000, we observed a difference of more than 67 objects for 10% of the pages with a median of 7 objects. This highlights the importance of precisely specifying all metrics including how and from which data source they are computed.

Keywords: Web performance · Measurement.

1 Introduction

Web browsing is one of the most prevalent applications in today's Internet. Thus, understanding its performance is critical. Hereby, both metrics as well as experiments have to realistically reflect possible performance improvements for actual users. Moreover, they need to be reproducible. However, quantifying Web performance is challenging due to Web page diversity, heterogeneous devices types and browsers, choice of metrics, including network-centric, browser-centric, and user-centric metrics, and the lack of well-established standards. Given this diversity, it is critical that studies provide sufficient details regarding their choice of metrics, data sources, and tools, to (a) understand and interpret the results, (b) to compare results across studies, and (c) to reproduce them independently.

For instance, Page Load Time (PLT) is a common metric used to estimate user-perceived quality (QoE) and to evaluate mechanisms for improving Web

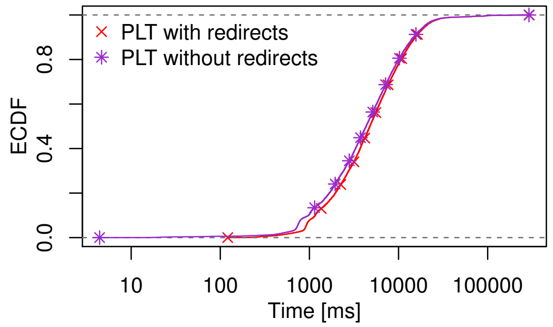


Fig. 1: Page Load Time (PLT) with and without initial redirects.

browsing. Thus, inaccuracies can lead to skewed results which may even lead to wrong conclusions. PLT is often defined as "time until onLoad[3] event". A less considered aspect is the *start point* of the measurement. PLT may include initial redirects, e.g., when a browser starts loading `http://example.com` and is redirected to `https://www.example.com`—the actual landing page. Such redirects increase PLT. To highlight that the discrepancies are non-negligible Figure 1 depicts PLTs with and without initial redirects[4]. According to the most recent W3C Navigation Timings specification [4] initial redirects should be included in all browser timings. But, whether redirects actually occur in a page load depends on the web workload, i.e., whether one starts with `http://example.com` or `https://www.example.com`. Moreover, even the end point of the measurement is not always well specified (see the Survey section), nor is it obvious how to precisely measure it. We are not aware of any prior work that quantifies the impact of the exact choice of metric on the measurement results.

The main contributions of this paper are as follows. 1.) We survey Web performance studies and summarize which measurement tools, methods, and metrics are used. Amazingly, we find that a third of these studies do not provide precise definitions of their metrics and/or data sources. However, it allows us to identify tools which are typically used for evaluating Web performance. 2.) We realize a test environment that allows us to compare different tools against a baseline to assess their accuracy. Among our results are that in-/exclusion of initial redirects skews the page load times by up to 47% for 10% of the pages. Moreover, object sizes differ from the packet trace for more than 60% of objects. This is critical as metrics derived from object sizes, e.g., Byte Index of loaded objects over time, differ by more than 50%. 3.) We discuss lessons learned regarding Web performance measurements and provide guidance on how to increase the accuracy of measured load times and object sizes[5]. Most importantly: 1.) HAR files are the most reliable data source for object counts and sizes. Resource timings underestimate these metrics, as they do not include objects in embedded frames, and they often do not provide object sizes for cross-origin objects. 2.) As redirects may highly influence load times, make a conscious choice whether to include them.

[3] See Figure 2 for an overview and Appendix A for more explanation.
[4] For details regarding the methodology and the corresponding dataset see Section 4.
[5] Our tools are publicly available at https://github.com/theri/web-measurement-tools

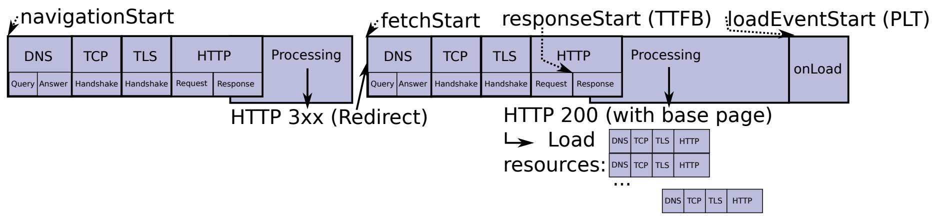


Fig. 2: Browser events and timings. See Appendix A for more details.

2 Web Metrics and Tools

Typical Web metrics include load times, object sizes, number of objects, and page sizes. Each of these metrics has various definitions and data sources. Moreover, there are different tools to measure them which we outline in the following.

2.1 Load Times

The time for loading Web pages strongly correlates with user experience [30]. To load a Web page, a browser usually loads the base document, parses it, constructs a Document Object Model (DOM), loads the referenced objects, processes them, and displays the results. Figure 2 shows a detailed view of this process including the browser events which are the basis of several commonly used load times metrics.

Definitions for load times: Typically, *Page Load Time (PLT)* is defined as the time until the *onLoad event*. However, in the eye of the user, the actual Web page display is often finished earlier, e.g., when the content is first displayed on the screen. Thus, other timings include *domContentLoaded*, when all objects referenced in the base document have been loaded, *Time To First Paint (TTFP)*, when the first content is rendered, or *Above The Fold Time (AFT)*, when the part of the page visible on the user's screen has been fully rendered. Start times can be the *navigationStart*, the *fetchStart*, or when the first DNS request or TCP connection is opened.

Data sources for load times: Load times based on browser navigation events are available through the standardized Navigation Timings API [3, 4]. Moreover, Time To First Paint (TTFP) is currently being standardized [7]. Being standardized implies that these metrics are available for different browsers based on a "similar" definition. HTTP Archive (HAR) files [8] also include onLoad and domContentLoaded times. However, AFT is not standardized, and estimating it requires not only load time data but also object positions within the Web page [12]. Load times are available through the Resource Timings API from version 1 [5] onward or from HAR files. Object positions are available by querying the DOM, e.g., using JavaScript.

Tools: Most popular browsers[6] implement Navigation Timings and Resource Timings. The standardized version of TTFP is not yet supported by all browsers[7]

[6] See http://gs.statcounter.com/browser-market-share/desktop/worldwide
[7] Chrome and Opera support it, Firefox is still validating their implementation.

as of September 2018. AFT is realized via a browser plugin available for Chrome [12]. HAR files can be exported using built-in developer tools.

To automate page loads, both Chrome and Firefox provide remote debugging interfaces, i.e., the Chrome DevTools Protocol[8], and Firefox Marionette[9]. For both interfaces, there is a variety of clients to navigate to a page and interact with it, e.g., to inject JavaScript code to export a timing.

Browser automation frameworks such as Selenium [9] allow more complex Web page interactions using a standardized webdriver interface, which controls Firefox using the Marionette protocol. The authors of Selenium advise against using it for Web performance testing, as its complex setup may incur significant performance overhead [10]. Furthermore, WebPagetest [11] integrates different browser automation frameworks into a single platform. It provides a Web-based User Interface for Navigation Timings, HAR files, load times, and Speed Index.

2.2 Number and Size of Objects

Number and sizes of objects are used to estimate the complexity of Web pages and are needed to compute metrics such as Object Index or Byte Index.

Possible definition of object count, object size, and derived metrics: Nowadays, Web pages often fetch objects continuously even after the initial page load has completed. Therefore, *object counts* should only include those objects loaded until the onLoad event. This can be done by either observing HTTP request-response pairs or by using the objects in the DOM. With regards to *object size*, networking-related studies usually use the *encoded size*, i.e., the number of bytes transferred over the network. One alternative is the *decoded size*, namely the number of bytes after decompression. However, as objects are transferred over HTTP there is overhead, namely the HTTP headers. Unfortunately, it is often unclear if the object size includes the header or not. The *total page size* is the sum of all object sizes. *Byte Index* is the integral of sizes of objects loaded over time, see [2].

Data sources for object sizes: One way to derive the number of objects is to count the number of HTTP request-response pairs using the list of entries in a HAR file. The number of objects involved in constructing a page is available via the Resource Timings API. HAR files [8] as well as Resource Timings version 2 [6] provide encoded and decoded body size of each object. In addition, HAR files include HTTP headers, possibly including a Content-Length header, and header size[10], while Resource Timings also includes the transfer sizes of header and body. An alternative is to extract the number of objects from a packet capture trace if it is possible to successfully decrypt all elements. However, exact object sizes can be off due to TLS padding.

[8] https://chromedevtools.github.io/devtools-protocol/

[9] https://firefox-source-docs.mozilla.org/testing/marionette/marionette

[10] Note that for HTTP/2, logged header sizes do not correspond to bytes on the wire anymore due to HTTP/2 header compression.

Table 1: Survey of Web performance studies: Metrics and data sources.

Metrics	Definition	Data source	Used in papers
PLT	Time of onLoad	Navigation Timings	6
		HAR file	1
		unknown	2
	Time to load all objects	HAR file	1
	unknown	unknown	3
DOM Time	Time of domContentLoaded	Navigation Timings	1
AFT	Visible content rendered	Resource Timings	2
Object load times	Time until object responseEnd	Resource Timings	1
		HAR file	1
Object size	number of bytes transferred	HAR file	2
		unknown	2
Number of objects	HTTP request-responses before onLoad	Resource Timings	1
	number of DOM resources	HAR file	4

3 Survey of Web Studies

Given the variety of metrics definitions, data sources, and tools, we survey Web performance studies published at SIGCOMM, IMC, PAM, NSDI, and CoNEXT during the last 8 years. In total, we include 15 papers [12–26], two of which include links to their code repositories in their papers.

Table 1 summarizes the metrics and data sources of the surveyed papers. Many of them use PLT, as it is well-known and widely used across academia and industry, standardized by W3C, and readily available from various data sources. However, the surveyed papers use diverse definitions and data sources which surprisingly are often not even specified in the paper. We note that only one of the surveyed papers even mentions initial redirects. Several papers compare PLT with other metrics such as AFT, which is more user-centric, but not standardized and, thus, harder to measure. Finally, several papers in the survey (also) measure the number, size, and load times of individual objects to compute integral metrics to quantify the page load process. Such metrics are readily available from the data sources. But many papers fail to precisely specify how they measure or compute these metrics.

Tools used to fetch pages: Table 2 summarizes which browsers and automation tools are used in the surveyed papers. Chrome is most popular, with Firefox in second place. Most studies use the DevTools interface but some use Selenium. To highlight the need for more information we point out that one paper uses a dataset and testbed without stating either the browser or the tools used. Overall, we conclude that a variety of different tools are used, with yet unclear effects on the results.

4 Methodology

So far we have pointed out that many different Web performance studies used different metrics. In this section, we explain our setup to understand the impact of different metrics. To compare the impact of different frameworks[11] and dif-

[11] Our scripts instrument browser automation frameworks directly to give us more control and avoid the overhead of an integrated framework such as WebPagetest.

Table 2: Survey of Web performance studies: Browsers and automation tools.

Browser	Automation tool	Used in papers
Chrome (desktop)	DevTools	6
	Selenium	1
	unknown	1
Chrome (mobile)	adb shell	1
Firefox (desktop)	Selenium	1
	unknown	2
phantomJS	-	1

ferent Web pages we use the following tools[12]: 1.) Firefox 61.0.2 with Selenium 3.14.0 and geckodriver 0.21.0, 2.) Firefox 61.0.2 with Marionette, and 3.) Chrome 69 with Chrome DevTools.

We load pages from a Thinkpad L450 with Debian Stretch. To avoid bandwidth issues, our vantage point is directly connected to a university network. To minimize the effects of DNS caching and delay to the resolver, we use a recursive resolver close[13] to our vantage point instead of popular open resolvers. Since the most commonly used workload are the Alexa Top Lists despite their limitations [28] we also use a snapshot of the global Alexa Top 1000[14], and the Alexa 10001 to 11000 for Marionette and ChromeDevTools. We then repeatedly accessed each page 10 times with the different frameworks. This ensures that all experiments for a single page are done within a reasonable time window. Overall, the experiments were executed between 18. September and 11. October 2018.

For each page, we first initialize a new browser profile with a cold browser cache. We then fetch the page and wait for it to load[15]. As data sources, we export Navigation Timings, Resource Timings, TTFP, and the HAR file using the native HAR export of the browser via har-export-trigger 0.6.1. In parallel, we also run a packet capture to derive our baseline. If one of the data sources does not yield any data, we log an error and exclude the page load attempt from the data set.

5 Results

In this section, we point out various pitfalls with Web performance metrics.

5.1 Pitfall: Redirects

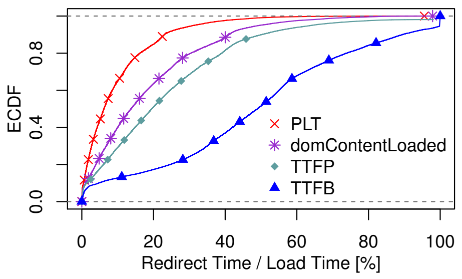
As already pointed out, see Figure 1, initial redirects can increase PLT substantially, especially for short page loads. Timings excluding redirects may be more representative of page loads by actual users due to browser optimizations, e.g., the user types the first few letters and then clicks on a URL suggested by the

[12] For realistic browser behavior, which includes the rendering engine, we open Web browsers including the graphical user interface rather than using them in headless mode.

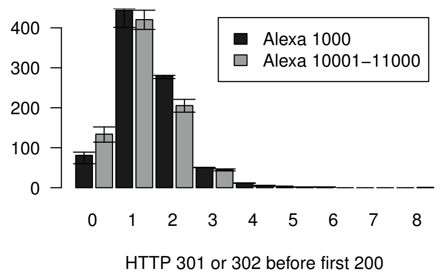
[13] Close in terms of network distance.

[14] 18. September 2018 for Alexa 1000 and 30. September 2018 for Alexa 10001-11000.

[15] We instruct the browser automation tool to wait for the onLoad event.



(a) Nav. Timings: Redirect share of load time



(b) HAR: Number of redirects

Fig. 3: Effects of initial redirects.

browser, or the browser automatically uses HTTPS due to HSTS or adds "www" to domain names the user types[16]. In contrast, load times including redirects are representative of page loads if a user types in the full URL and presses Enter. However, a conscious choice should be made and the web workload adjusted accordingly.

To assess the impact of redirects we first count the number of server-side redirects [17] for both the Alexa 1000 and 10000-11000, see Figure 3b. The most common cause for a redirect is that a page is no longer available via HTTP and the browser is redirected to the HTTPS version. Given that many pages have migrated to HTTPS, e.g., 75% of Web pages loaded by Firefox users in September 2018 [29], this is not surprising. Other reasons for redirects include pointers to subdomains, e.g., for localized versions of the content based on the geolocation. Often both occur and lead to two redirects.

Next, we revisit page load times[18].To quantify their contribution to the load time, we show, in Figure 3a, the relative percentage of load times of redirects for all Web pages. Redirects account for 6.1% of PLT for 50% of the pages and for 23% of PLT for 10% of pages. This implies that the PLT with or without redirects differs by this amount. The difference is even larger for user-centric load time metrics as these are usually shorter. For instance, Time To First Paint (TTFP) differs by 19.1% for 50% of pages and by 47% for 10% of pages. Indeed, the time for the redirects is about the same as the Time To First Byte after the redirect for about 50% of pages. The reason is that most redirects, typically involve an additional name resolution, TCP connection establishment, TLS handshake[19], and HTTP request.

In summary, we make the following observations: 1.) Redirects account for a significant share of PLT and a substantial share of user-centric load time metrics such as TTFP. 2.) Studies should make a conscious choice on in-/exclude redirects, see Section 6.

[16] See, e.g., https://support.mozilla.org/en-US/kb/search-web-address-bar.

[17] Server-side redirects use HTTP status 301 or 302. Client-side redirects use status 200 and contain the redirection URL in the response content, which we do not log.

[18] For Navigation Timings, redirects are the time between navigationStart and fetchStart. For HAR files, we use the time before the first HTTP 200 response.

[19] In September and early October 2018, TLS 1.3 was still not deployed.

Table 3: Object sizes: Accuracies for unencrypted objects.

Comparison	**Browser**	**Match**	**Counted too many bytes**			**Counted too few bytes**		
		Cases [%]	Cases [%]	99%q [KB]	Max [KB]	Cases [%]	99%q [KB]	Max [KB]
Content-Length	Firefox	100	0	0	0	0	0	6.8
	Chrome	100	0	0	0	0	0	0
HAR body size	Firefox [21]	72.6	13.4	66.28	2170	14	0.13	852.4
	Chrome	91.9	0.5	0	303.4	7.6	0.3	2925
Res body size	Firefox	39.6	0.8	0	2910	59.6	196.6	5092
	Chrome	46	0.5	0	276.5	53.5	181.5	5092

5.2 Pitfall: Object Sizes

Next, we take a closer look at object sizes. In particular, we explore if different data sources are consistent with the baseline from the packet capture trace and if they yield similar results.

Comparison with the baseline for unencrypted objects: To validate the object sizes recorded by the different data sources, see Section 2.2, we compare them against the baseline which we get via the packet capture trace. This, unfortunately, is only possible for objects loaded over unencrypted HTTP/1.0 or HTTP/1.1. If TLS is used object sizes may be incorrect due to padding. For computing the baseline, we extract HTTP request and response pairs from the packet capture trace and exclude objects with missing bytes. For the remaining object, we separate the TCP payload into the HTTP header and body, and count bytes[20].Finally, we match the object to the corresponding HAR and Resource Timing (Res) data based on timestamp. Hereby, we exclude ambiguous cases, i.e., where multiple HAR entries match an object from the trace.

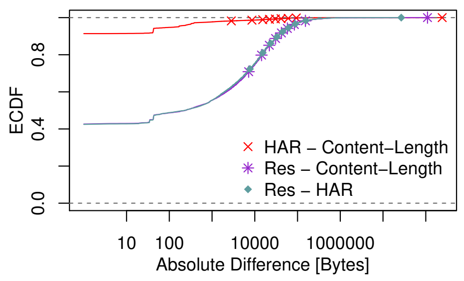
The resulting comparison is summarized in Table 3 If the Content-Length header is present its information is mostly consistent with the traces. None of the other data sources is that good. Rather, we find that the accuracy varies widely across data sources and browsers. When manually investigating the most significant mismatches, we find that Resource Timings set object size to 0 for most cross-origin objects[22].In HAR files, body size is often set to -1 if the browser did not succeed in loading a resource. In several cases, Firefox counted too many bytes if redirects happened. Apparently, it is returning the size of the redirect destination instead of the actual object size.

Comparison of data sources for all objects Next, we explore the consistency of the results for *all* objects including those that are transferred over an encrypted connection. Figure 4, shows the object size differences for the same object and various data source combinations, i.e., HAR file body size (HAR), Content-Length header taken from HAR file, and Resource Timings encoded body size (Res). Since Content-Length is a close approximation to the baseline

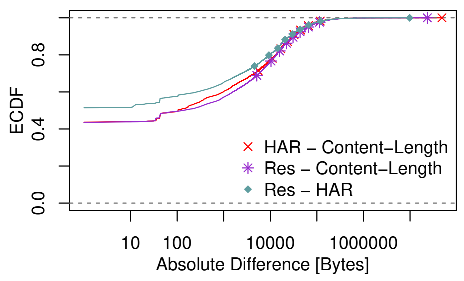
[20] See analysis script eval/validate_object_sizes.py in our repository.

[21] In HAR files, Firefox logs body size including headers, contradicting [8], see https://dxr.mozilla.org/mozilla-central/source/devtools/server/actors/network-monitor/network-response-listener.js#428, accessed 28.09.2018. Thus, we subtract header size from all object sizes.

[22] Unless the 'Timing-Allow-Origin' header is set, see [6].



(a) Firefox



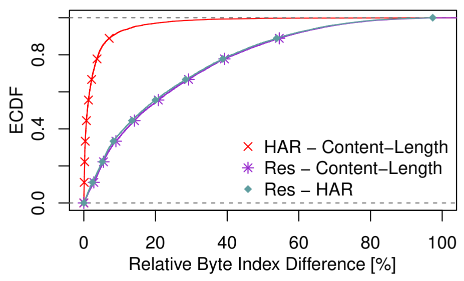
(b) Chrome

Fig. 4: Object sizes: Differences due to metric for all objects.

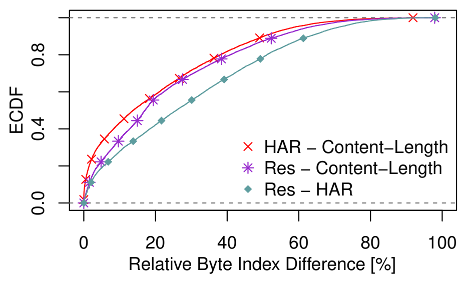
for unencrypted objects we use it as a baseline. Res provides the exact same object size as Content-Length in only 42.5% of cases for Firefox and in 43.4% of cases for Chrome. This is consistent with the results for unencrypted objects, see Table 3. For HAR, Firefox provides an object size which matches the Content-Length for 91.3% of cases, see Figure 4a. Thus, we conclude that HAR's accuracy is better than for unencrypted objects. In contrast, Chrome provides an object size which matches the Content-Length in only 39.4% of cases for all objects. When investigating the difference, we find that Chrome sets HAR body and header size to -1 for all HTTP/2 objects[23].

From this, we conclude: 1.) Content-Length provides the most accurate object size but is not always available. 2.) Resource Timings are an unreliable data source for object sizes, as they do not provide sizes for cross-origin objects, except when explicitly allowed. 3.) HAR body size is inaccurate for a significant number of objects, due to bugs in both Firefox and Chrome (whereby Firefox is more accurate than Chrome).

5.3 Pitfall: Object Count and ByteIndex



(a) Firefox



(b) Chrome

Fig. 5: Byte Index: Difference due to data source.

Amazingly, we find that not only the object sizes differ by data sources but also the object counts (for the same page download)! For the Alexa 1000 dataset,

[23] In Firefox, only HTTP/2 Server Push objects lack body size and timings.

Table 4: Missing data source: Successful page loads vs. errors for Alexa 1000 run

Tools	**Success**		**before onLoad**	**No data**		**No Res**	**No HAR**	**No Res and HAR**
	Min	Max	Median	Min	Max	Median	Median	Median
Firefox with Selenium	880	898	0	60	76	35	4	3
Firefox with Marionette	915	922	0	37	43	38	3	2
Chrome with DevTools	740	801	100	32	81	14.5	31.5	21.5

object counts from HAR and Res always differ by at least one object and by 7 or more objects for 50% of cases. For 10% of the cases, they are off by more than 67 objects. Numbers for Alexa 10000 are similar. Among the main contributor to this difference is that Resource Timings do not include objects loaded within commonly embedded HTML Inline Frames (iframes)[24]. Rather, these objects are recorded in the Resource Timeline for the iframe.

Next, we quantify the impact of object size and count differences on the Byte Index [2], which captures page load progress, i.e., loaded bytes over time. In Figure 5, we plot the relative difference between Byte Index for the same page load, calculated from HAR body sizes, Resource Timings body sizes, and Content-Length header (using the HAR body size if the Content-Length header is missing). For Firefox, see Figure 5a, the Byte Index is almost identical for Content-Length and HAR body size, but differs by 17.1% for Res in 50% of the pages loads, and by 56.4% for 10%. For Chrome, see Figure 5b, the Byte Index derived from both Res and HAR differs substantially from the Byte Index derived from Content-Length.

Thus, we conclude: 1.) Resource Timings do not include all objects of a Web page download. 2.) Byte Indexes from Resource Timings vs. HAR files differ by 13.8%/17% in median and by more than 50% for 10% of the pages.

5.4 Pitfall: Data Source Availability

Besides being inaccurate some data sources do not even provide us with any data for some Web page access. More precisely, Table 4 shows the number of successful page loads as well as the errors for the Alexa 1000 for different browsers and automation tools. Firefox with Marionette yields the best results in terms of successful runs that include all data elements. Using Chrome often yields invalid timings, in particular, for the onLoad event. The main culprit is a too early export of the data—the page load has not yet been completed even though Chrome was instructed to wait for the onLoad event[25]. For a non-negligible number of Web pages, we did not get any results for *all* or for *some* of the 10 repeated page loads per browser framework. For most of these page downloads, the browser never invoked the onLoad event, see also Section 2.1, and, thus, timed out without exporting any data. Using a different tool would not fix the problem in some cases: Investigating both the error messages logged by the browser automation tool as well as the captured traces we find that common reasons are no DNS response, not being able to establish a TCP connection,

[24] See the examples in Section 4.2 of [6].
[25] Using the Page.frameStoppedLoading event instead did not resolve this problem.

or certificate errors. We limit intermittent connectivity issues by spreading out page loads over time. But, we cannot rule out filtering, e.g., due to our vantage point. Still, manual tests for some of the page loads showed that these also fail for different vantage points. These even involve domains of large application service providers and content distribution networks, which are hosting resources only under subdomains[26].

In summary, our conclusions are: 1.) Not all domains in the Alexa Top Lists point to actual Web pages. 2.) Firefox with Marionette is more likely to provide complete data than Firefox with Selenium or Chrome.

6 Guidelines for Web Performance Measurement

Next, we derive some guidelines for designing and conducting experiments.

Use HAR files and Navigation Timings, not Resource Timings: As shown in Section 5.2, Resource timings are an unreliable data source, as they do not include resources of embedded frames and often do not provide sizes for cross-origin objects.

Choose whether to exclude redirects: Redirects significantly contribute to page load times. Yet, they may not be representative for typical end-user Web browsing, recall Section 5.1. It is possible to exclude redirects upfront, e.g., by adjusting the hit list to post-redirect URLs. However, post-redirect URLs may change, e.g., due to geolocation or HTTPS migration. Such changes may lead to more page load failures, compared to starting from the "base" URLs of `http://` and the top-level domain name. Alternatively, redirects can be excluded in retrospect by computing the timings relative to fetchStart instead of navigationStart for Navigation Timings resp. relative to the start time of the first HTTP 200 object for HAR files.

Choice of tools: Make a conscious choice whether to use a framework that integrates browser automation tools, such as WebPagetest [11], or write your own scripts. The first has the advantage that it enables comparing multiple browsers out of the box, while the latter gives more explicit control over details. Note, WebPagetest provides Navigation Timings and HAR files. So pitfalls related to Resource Timings do not apply. Moreover, it provides additional metrics such as SpeedIndex. WebPagetest always includes redirects—in line with the W3C definition of load times.

Use up-to-date software: Major Web browsers are updated rather often, typically every 1-2 months. While research projects typically last longer one has to address the trade-off of updating to a newer version during the study: On the one hand, software updates may fix bugs and provide performance optimizations so that the results are more representative of state-of-the-art setups and actual user experience. On the other hand, updating may cause compatibility issues, e.g., with measurement tools that are updated less often and hinder backward compatibility. We recommend to consciously address this trade-off and to include the version numbers of the used tools. See Appendix B for more details.

[26] Examples include `microsoftonline.com` and `googleusercontent.com`.

Disable features for a quiet browser: Modern browsers do not just load the requested Web page. Rather they often automatically load additional data, e.g., software updates or blocklists, or transmit performance statistics to the browser vendor. This can cause significant performance overhead. We, thus, recommend turning off such features. See Appendix B for more details.

Record and compare different data sources: Whenever possible multiple data sources should be recorded to enable cross-checks. Data sources include but are not limited to Navigation Timings, Resource Timings, and HAR files. Combining them helps improve accuracy. When choosing metrics it is essential to understand their status with regards to standardization, e.g., published as W3C Recommendation, and to which extent the implementation conforms to the standard.

Mind new protocols: Deployment of new protocols always has the chance of invalidating existing assumptions about traffic both in general as well as for Web traffic. Moreover, new protocols may require updates to the measurement and evaluation setup or trigger so far unknown bugs in the evaluation. Recent examples include the increased deployment of HTTP/2 and QUIC which use features such as header compression and HTTP/2 Server Push.

7 Conclusion

We show that Web metrics highly depend on which specific metrics, data sources, and/or measurement tools are used. For example, initial redirects can cause Page Load Times (PLTs) to vary by 6.1% in median and by more than 23% for 10% of pages. The impact is even larger for user-centric metrics such as Time To First Paint (TTFP), with 19.1% and 47%, respectively. Furthermore, HAR files and Resource Timings provide widely differing object sizes and numbers of objects which in turn bias derived metrics, e.g., Byte Index varies by 17.1% for 50% of pages and by 54.2% for 10% of pages. However, in almost all Web measurement studies none of the metrics or the data sources are described in sufficient detail. Moreover, they often ignore the bias of the above differences.

Thus, our study clearly highlights the need to (a) improve documentation, (b) choose metrics consciously and with all caveats in mind, (c) double check the results against alternative metrics, and (d) enable qualitative comparisons. To enable this we strongly follow the recommendations of a recent Dagstuhl seminar on reproducibility and suggest that conferences and journals should not count the pages needed to document the precise measurement/simulation setup and the used metrics against the available page limit.

Acknowledgements

Thanks to Dominik Strohmeier for the discussion and the pointers to resources, to our shepherd Jelena Mirkovic, as well as our anonymous reviewers.

References

1. Paxson, V.: "Strategies for sound internet measurement. In: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, pp. 263–271. ACM, New York (2004)
2. Bocchi, E., De Cicco, L., and Rossi, D.: Measuring the Quality of Experience of Web users. In: ACM SIGCOMM Computer Communication Review, 46(4), 8-13. ACM, New York (2016)
3. W3C Recommendation: Navigation Timing, https://www.w3.org/TR/navigation-timing/. Version 17 December 2012. Last accessed 29 August 2018
4. W3C Working Draft: Navigation Timing Level 2, https://www.w3.org/TR/2018/WD-navigation-timing-2-20181130/. Version 30 November 2018. Last accessed 17 December 2018
5. W3C Candidate Recommendation: Resource Timing Level 1, https://www.w3.org/TR/resource-timing-1/. Version 30 March 2017. Last accessed 29 August 2018
6. W3C Working Draft: Resource Timing Level 2, https://www.w3.org/TR/resource-timing-2/. Version 11 October 2018. Last accessed 13 October 2018
7. W3C First Public Working Draft: Paint Timing 1, https://www.w3.org/TR/paint-timing/. Version 07 September 2017. Last accessed 10 October 2018
8. W3C Editor's Draft: HTTP Archive (HAR) format, https://w3c.github.io/web-performance/specs/HAR/Overview.html. Version 14 August 2012. Last accessed 29 August 2018
9. Bruns, A., Kornstadt, A., and Wichmann, D: "Web application tests with selenium." In: IEEE software 26.5 (2009)
10. Selenium Documentation: Worst Practices, https://seleniumhq.github.io/docs/worst.html. Last accessed 29 August 2018
11. Meenan, P.: Webpagetest. https://www.webpagetest.org. Last accessed 17 December 2018
12. Alemnew, A., Christophides, V., Teixeira, R. and Rossi, D.: Narrowing the gap between QoS metrics and Web QoE using Above-the-fold metrics. In: PAM 2018-International Conference on Passive and Active Measurement (pp. 1-12). Springer (2018)
13. Ludin, S.: Measuring What is Not Ours: A Tale of 3rd Party Performance. In: PAM 2017-International Conference on Passive and Active Measurement (p. 142). Springer (2017)
14. Erman, J., Gopalakrishnan, V., Jana, R., and Ramakrishnan, K. K.: Towards a spdyier mobile web?. In: IEEE/ACM Transactions on Networking, 23(6). ACM, New York (2015)
15. Qian, F., Gopalakrishnan, V., Halepovic, E., Sen, S., and Spatscheck, O.: TM 3: flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking. In: Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (p. 3). ACM, New York (2015)
16. Wang, X. S., Krishnamurthy, A., and Wetherall, D.: Speeding up Web Page Loads with Shandian. In: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16) (pp. 109-122). USENIX association (2016)
17. Wang, X. S., Balasubramanian, A., Krishnamurthy, A., and Wetherall, D.: Demystifying Page Load Performance with WProf. In NSDI 2013 (pp. 473-485).

18. Butkiewicz, M., Madhyastha, H. V., and Sekar, V.: Understanding website complexity: measurements, metrics, and implications. In: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (pp. 313-328). ACM, New York (2011)
19. Kelton, C., Ryoo, J., Balasubramanian, A., and Das, S. R.: Improving User Perceived Page Load Times Using Gaze. In: Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17) (pp. 545-559) USENIX Association. (2017)
20. Varvello, M., Schomp, K., Naylor, D., Blackburn, J., Finamore, A., and Papagiannaki, K.: Is the web http/2 yet?. In: International Conference on Passive and Active Network Measurement (pp. 218-232). Springer, Cham.
21. Netravali, R., and Mickens, J.: Prophecy: Accelerating mobile page loads using final-state write logs. In: Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association. (2018)
22. Netravali, R., Nathan, V., Mickens, J., and Balakrishnan, H.: Vesper: Measuring Time-to-Interactivity for Web Pages. In: Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association. (2018)
23. Netravali, R., Goyal, A., Mickens, J., and Balakrishnan, H.: Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), USENIX Association. (2016)
24. Zaki, Y., Chen, J., Ptsch, T., Ahmad, T., and Subramanian, L.: Dissecting web latency in ghana. In: Proceedings of the 2014 Conference on Internet Measurement Conference (pp. 241-248). ACM, New York (2014)
25. Han, B., Qian, F., Hao, S., and Ji, L.: An anatomy of mobile web performance over multipath TCP. In: Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (p. 5). ACM, New York (2015)
26. Naylor, D., Finamore, A., Leontiadis, I., Grunenberger, Y., Mellia, M., Munaf, M., ... and Steenkiste, P: The cost of the S in HTTPS. In: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies (pp. 133-140). ACM, New York (2014)
27. Google Web Fundamentals: User-centric Performance Metrics https://developers.google.com/web/fundamentals/performance/user-centric-performance-metrics. Last accessed 05 September 2018
28. Scheitle, Q., Hohlfeld, O., Gamba, J., Jelten, J., Zimmermann, T., D. Strowes, S., and Vallina-Rodriguez, N.: A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In: Internet Measurement Conference 2018. ACM, New York (2018)
29. Letsencrypt: Percentage of Web Pages Loaded by Firefox Using HTTPS https://letsencrypt.org/stats/#percent-pageloads. Last accessed 30 September 2018.
30. Egger, S., Hossfeld, T., Schatz, R., and Fiedler, M.: Waiting times in quality of experience for web based services. In Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on (pp. 86-96). IEEE. (2012)
31. Barth, A.: The web origin concept. RFC 6454 (2011)

A A Web Page Load Explained

In this section, we explain a Web page load in more detail. See also Figure 2 and the processing models in the Navigation Timings specifications [3, 4].

The starting point for a new page load, also called navigation, of a particular URL, is called *navigationStart* in [3]. Initially, *fetchStart* is set to the same value, but if a redirect occurs, *fetchStart* is overwritten before the new URL is loaded.

If another page has been previously loaded by the browser, e.g., in the same browser tab, this document has to be first unloaded. Then, the browser checks its cache to see whether the page is already there. If the page is not in the cache, the browser usually resolves the hostname (resulting in a DNS query and usually answer), establishes a TCP connection, and performs a TLS handshake if the scheme of the URL is `https`. Then, the browser issues an HTTP GET request for the URL. As soon as it receives an HTTP reply, which always contains a status line, headers, and body, the browser processes the reply.

If the reply contains an HTTP status code of 3xx, such as "301 Moved Permanently" or "302 Found", this means that the server redirects the browser to a different URL, which is given in the "Location" header in the HTTP response. This redirect may be a same-origin redirect, which roughly means that both the old and the new URL have the same scheme (`http` or `https`), hostname, and port (see RFC 6454 [31] for details), or it may be a cross-origin redirect. For same-origin redirects, the start and end time of the redirect are recorded as Navigation Timings *redirectStart* and *redirectEnd* [3], while for cross-origin redirects they are not. Unfortunately, nearly all redirects we observed are cross-origin, as the purpose of the redirect is to use a different scheme (HTTPS instead of HTTP) or hostname (`www.example.com` instead of `example.com`). The same-origin policy is an important security and privacy feature in the Web, so information access is often restricted to, e.g., the same hostname.

Given the new URL to be fetched, the browser records the current time as *fetchStart*, potentially overwriting the old value[27]. It then checks its application cache again, resolves the host name if needed, establishes a new TCP connection, performs a new TLS handshake, and sends an HTTP request for the new URL. If it gets an HTTP reply, this may be another redirect, an error code such as "404 Not Found" or "503 Internal Server Error", or the request may succeed with a "200 OK". In the latter case, the body of the HTTP response usually contains the base document of the Web page in HyperText Markup Language (HTML). As soon as the browser starts receiving this document, it parses it and starts constructing the Document Object Model (DOM) of the page. For example, the document may reference additional resources, such as JavaScript, Cascading Stylesheets (CSS), or images. Typically, for each of these additional resources, the browser has to issue a new HTTP request, unless the resource

[27] After a redirect, the browser overwrites the old *fetchStart* value before it fetches the new URL using a GET request. This implies that once the page load is finished, *fetchStart* is the start time of the loading of the final base page, as all previous values related to redirects are overwritten.

is proactively sent by the server using HTTP/2 Server Push. Each new HTTP request may involve an additional name resolution, TCP handshake, and TLS handshake, because resources are often hosted on different servers than the base page. The browser now simultaneously fetches new resources, continues to parse the HTML base page, and processes the CSS and Javascripts, even though these processes may block each other. See Wang et. al [17] for a detailed explanation of this complex process.

At some point, the browser flushes the current state of the DOM to the rendering engine. The time at which this happens corresponds to *Time To First Paint (TTFP)*. The point at which all resources in the DOM have been loaded is called *DOMContentLoaded* and recorded in the Navigation Timings and HAR file. However, processing of the page usually continues, until, eventually, the browser fires the *onLoad* event for the page which is recorded in the Navigation Timings and HAR file. The onLoad Time is usually taken as *Page Load Time (PLT)*. At this point, the page load is considered finished. However, onLoad usually triggers the execution of one or more javascripts, which may result in loading more resources, sending data, e.g., to third parties, or other network traffic. In fact, most modern Web pages load resources continuously long after the onLoad event. Thus, Related Work usually stops counting objects after onLoad.

B Details of Lessons Learned

Next, we outline additional details regarding our lessons learned, which led to our guidelines for Web performance measurement, recall Section 6.

Software versions: The Debian Linux distribution includes a version of the Firefox browser which is usually quite dated. This can have a major impact on load times. For instance, in Firefox version 61 ("Firefox Quantum"), parts of the code have been rewritten and optimized, which makes the browser much faster than previous versions. Consequently, carrying out Web page loads using an older version results in unrealistically long load times. However, updating Firefox frequently to the newest version can result in incompatibilities with measurement tools. For instance, not every version of the HAR Export Trigger extension works with every version of Firefox, so it has to be updated along with the browser. However, the upside is that in newer versions of Firefox, HAR Export Trigger is supposed to work without having the developer panel open.

Browser traffic unrelated to page loads: Modern browsers usually issue a significant number of requests that are not directly related to the page load that a user has requested. For instance, Firefox by default loads blocklists for "safe browsing", to protect users from malware or phishing. It also automatically checks for updates and may even automatically download and install these updates for the entire browser or for individual browser extensions. These queries can involve substantial data transfers: For example, we observed the automatic download of a binary related to an H264 media component which we never activated or requested: 500 KB were downloaded in the background. Worse yet, the state of such updates is often stored in the browser profile. This may cause such downloads to be triggered for every fresh browser profile, i.e., each of our page loads. Additionally, the Chrome browser by default issues queries to various Google servers, e.g., it tries to connect each browsing session to a Google account. We provide configurations for Firefox and Chrome to turn off most features that generate such traffic, see our repository https://github.com/theri/web-measurement-tools.

Logging a trace and client-side SSL keys To be able to better debug and validate measurement setups and tools, we recommend capturing packet traces that include at least ports 53 (DNS), 80 (HTTP), and 443 (HTTPS). Encrypted traffic can be decrypted after logging the SSL session keys within the browser: Firefox and Chrome log keys into a specified SSLKEYLOGFILE. Note that this option must be compiled into Firefox. It, e.g., does not work with the Firefox binary in the Debian repositories.

C Artifacts Related to This Paper

The following artifacts are available:

Our tools, such as measurement and evaluation scripts: See https://github.com/theri/web-measurement-tools. This repository includes the scripts to automatically load Web pages using Firefox with Selenium and Marionette, and using Chrome with DevTools. Furthermore, it includes the analysis scripts we used to generate our plots.

Data set of Web page loads: See http://dx.doi.org/10.14279/depositonce-8100. This dataset includes data from all of our experiment runs, see Section 4. It can be used along with our evaluation scripts to reproduce the plots in this paper, see https://github.com/theri/web-measurement-tools for details.