May 2020
npm@chromium.org
Public document

LargestContentfulPaint is one of the new web vitals metrics in Chrome, so web developers and analytics providers are starting to gather RUM data by themselves via the web performance API. However, there are some problems with the current implementation and its behavior with respect to the values reported on the Chrome User Experience Report (CrUX). This document talks about some of these and the path to solve them.

The Problems

One common developer complaint about the current API is the fact that the API does not say when the algorithm has stopped. LargestContentfulPaint stops upon first user input (scrolling counts), or upon unloading the page. The former is hard to detect by a developer, so generally they'd have to keep an observer for LCP through the duration of the page, which seems unnecessary in some cases, since the developer would be able to stop observing after the first user interaction. A suggestion is to issue a 'final' LCP entry which would declare that the value of LCP is the last one and the browser is confident that user input did not cause the LCP value to be premature.

There are other problems with reporting LCP candidates in the API instead of the final value. The LCP API presents the developer with a series of candidates, but there is no way to select between them in a way that reliably matches CrUX.

Firstly, even though we may count removed elements in the future, currently a removed element is deemed invalid as an LCP candidate in the CrUX implementation, which means that another candidate needs to be surfaced. However this candidate may have timestamps that are smaller than those of the removed element, and entries in the list provided to PerformanceObserver are sorted by startTime. In LargestContentfulPaint, startTime is the renderTime (or loadTime for images for which the rendering time cannot be exposed). This means that if the candidate occurs and invalidation occurs shortly afterwards, it's likely that the observer will receive the LCP entry for the element plus the entry for the new candidate after the element was removed in the same callback. In this case, the observer has no way to know that the 'correct' candidate was in fact the entry with a smaller timestamp. This means developers would overestimate LCP in this case.

Another problem related to relying on startTime to determine the ordering of LCP entries within a single observer callback is the inconsistency for cross-origin images without Timing-Allow-Origin. In entries corresponding to these images, startTime is the loadTime of the image, which may be much earlier than the renderTime. If there is some other text or same-origin image that is an LCP candidate and whose renderTime is between the loadTime of the cross-origin image and its renderTime, then the developer will process the cross-origin image first even if it's a later LCP candidate, perhaps even the final LCP. This means that the developer could deduce the wrong LCP candidate in some cases.

And to top it off, there's a problem with images that are loading: the CrUX implementation currently does not report the LCP when the algorithm ends and the largest candidate corresponds to an image that has not yet finished loading, but the web API may report LCP candidates before such image begins loading, and developers have no way to know that there's no more candidates afterwards because of this pending image load. This is more worrisome than the previous problem because we believe that not reporting is indeed correct here (and then counting non-reported loads as 'slow' for LCP). In these cases, the developer is underestimating LCP, potentially by a lot because new candidates are not dispatched once the largest slow image begins loading. We're currently in the process of gathering some data to understand how commonly this occurs, but it seems that it's not extremely uncommon for this problem to occur, and it certainly may disproportionately affect some websites, to the point that the LCP values from the web API will be completely different from those reported on CrUX.

The Very Large Image example illustrates this: most page loads for such a website should have 'no-LCP' value and hence the page should be considered slow by both the web API and by CrUX. It's worth noting that this is also incorrect behavior from CrUX: a no-LCP value is a bad page load for the purposes of LCP, whereas the current logic seems to be outright ignoring these page loads.

Proposed Solution

I propose implementing a solution similar to the one suggested: a final LCP entry. We may still surface regular LCP entries, but for a developer that is only looking to gather data that matches the one gathered internally by Chrome, they'd be able to just observe the final LCP. The only caveat is that we would not apply the heuristic about confidence on the value: the goal is to enable the developer to query the information being reported to CrUX about their website as easily as possible, and this heuristic is not currently applied to that data. If/when CrUX changes to use such a heuristic in the future, we'd similarly work on changing the final LCP specification and implementation to match such heuristic.

This solves the problem about not knowing when the algorithm has stopped. A developer interested in all the LCP values would still be able to know this: once it receives a final LCP entry, it would know that no new regular LCP entries would be received, since the final LCP entry can only occur once the algorithm has stopped.

This also solves the problem of complexity surrounding removed elements and relying on startTime. A developer would know that the final LCP value corresponds to an element that was not removed, at least until we start considering removed elements as valid LCP candidates. In addition, the final LCP also corresponds to one of the regular LCP entries, and any regular LCP entry with a size larger than the final LCP entry size must have been invalidated (removed) at some point. This also does not add complexity to the change to now count removed elements, since the final LCP will simply change to reflect that change as well. Attribution would be more limited because the |element| attribute value will be null, but there will still be some attribution options available, such as |url| and |id|.

Finally, it also solves the problem with LCP not being reported due to large images that are loading. It would make sense to report a ‘no-LCP’ value as the final LCP, since this would be useful when LCP stops upon user input but there is no LCP candidate (due to extremely early user input, or due to large image loading).

The tricky part is implementing the final LCP API in such a way that it reliably dispatches the entry when such entry is available. This is especially important given the recommendation to consider no-LCP as ‘slow’. The main requirement would be to ensure that the entry is available to the observer or the performance timeline by the time the page visibility hidden callback begins running. Since the entry may be generated upon page becoming hidden, it’s not possible to guarantee that the observer callback is run by then, but the developer could call takeRecords() and obtain the entry as long as it’s created by that time.

One consideration about having a ‘final entry’ for LCP is that we may want to consider defining LCP for SPA navigations in the future. I believe this would not limit the SPA use-case because we could surface one ‘final entry’ for every SPA navigation, hence enabling tracking multiple final LCPs in a single page load.