**What is TBT?** [**#**](https://web.dev/tbt/#what-is-tbt)

The Total Blocking Time (TBT) metric measures the total amount of time between [First Contentful Paint (FCP)](https://web.dev/fcp/) and [Time to Interactive (TTI)](https://web.dev/tti/) where the main thread was blocked for long enough to prevent input responsiveness.

The main thread is considered "blocked" any time there's a [Long Task](https://web.dev/custom-metrics/#long-tasks-api)—a task that runs on the main thread for more than 50 milliseconds (ms). We say the main thread is "blocked" because the browser cannot interrupt a task that's in progress. So in the event that a user *does* interact with the page in the middle of a long task, the browser must wait for the task to finish before it can respond.

If the task is long enough (e.g. anything above 50 ms), it's likely that the user will notice the delay and perceive the page as sluggish or janky.

The *blocking time* of a given long task is its duration in excess of 50 ms. And the *total blocking time* for a page is the sum of the *blocking time* for each long task that occurs between FCP and TTI.

For example, consider the following diagram of the browser's main thread during page load:

The above timeline has five tasks, three of which are Long Tasks because their duration exceeds 50 ms. The next diagram shows the blocking time for each of the long tasks:

So while the total time spent running tasks on the main thread is 560 ms, only 345 ms of that time is considered blocking time.

|  |  |  |
| --- | --- | --- |
|  | **Task duration** | **Task blocking time** |
| Task one | 250 ms | 200 ms |
| Task two | 90 ms | 40 ms |
| Task three | 35 ms | 0 ms |
| Task four | 30 ms | 0 ms |
| Task five | 155 ms | 105 ms |
| **Total Blocking Time** | | **345 ms** |

**How does TBT relate to TTI?** [**#**](https://web.dev/tbt/#how-does-tbt-relate-to-tti)

TBT is a great companion metric for TTI because it helps quantify the severity of how non-interactive a page is prior it to becoming reliably interactive.

TTI considers a page "reliably interactive" if the main thread has been free of long tasks for at least five seconds. This means that three, 51 ms tasks spread out over 10 seconds can push back TTI just as far as a single 10-second long task—but those two scenarios would feel very different to a user trying to interact with the page.

In the first case, three, 51 ms tasks would have a TBT of **3 ms**. Whereas a single, 10-second long tasks would have a TBT of **9950 ms**. The larger TBT value in the second case quantifies the worse experience.

**How to measure TBT** [**#**](https://web.dev/tbt/#how-to-measure-tbt)

TBT is a metric that should be measured [in the lab](https://web.dev/user-centric-performance-metrics/#in-the-lab). The best way to measure TBT is to run a Lighthouse performance audit on your site. See the [Lighthouse documentation on TBT](https://web.dev/lighthouse-total-blocking-time) for usage details.

# Lazy load third-party resources with facades

Dec 1, 2020

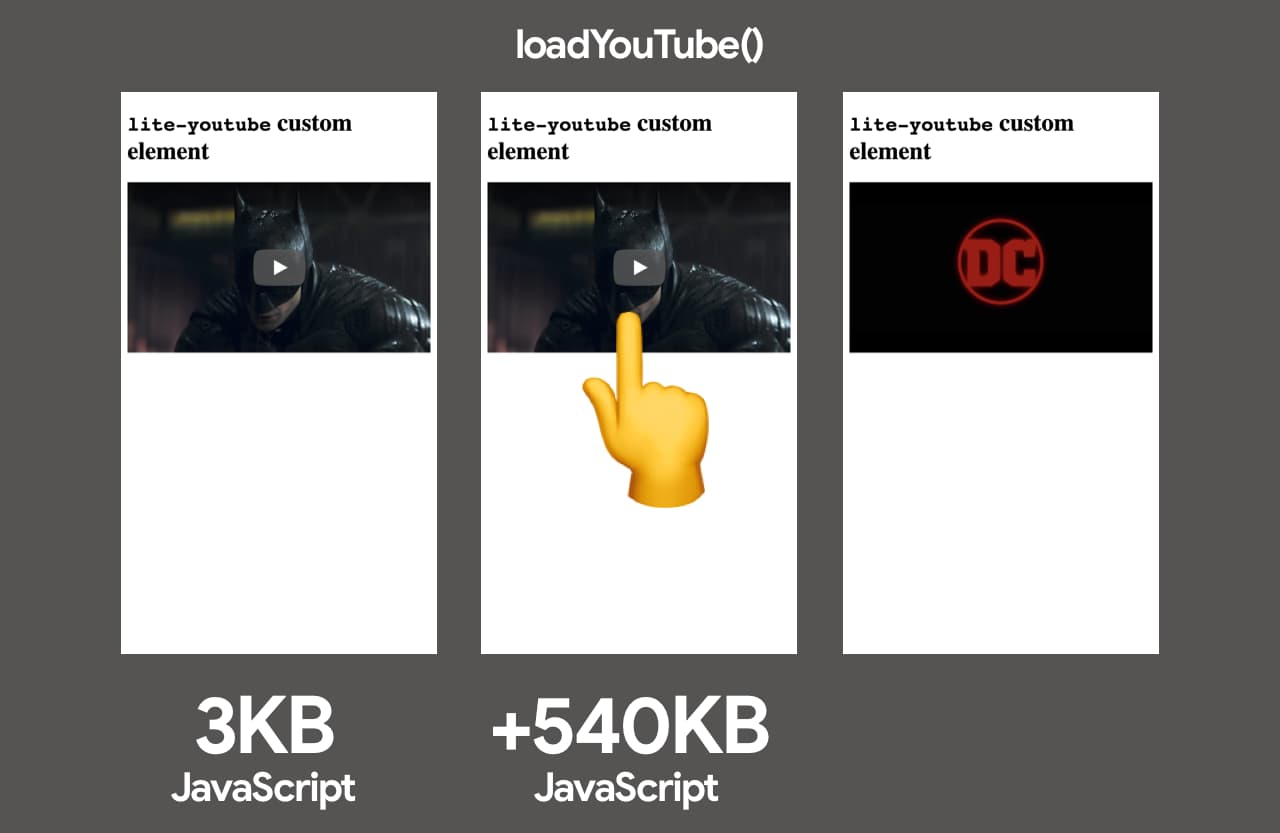
Appears in: [Performance audits](https://web.dev/lighthouse-performance)

[Third-party resources](https://web.dev/third-party-javascript/) are often used for displaying ads or videos and integrating with social media. The default approach is to load third-party resources as soon as the page loads, but this can unnecessarily slow the page load. If the third-party content is not critical, this performance cost can be reduced by [lazy loading](https://web.dev/fast/#lazy-load-images-and-video) it.

This audit highlights third-party embeds which can be lazily loaded on interaction. In that case, a facade is used in place of the third-party content until the user interacts with it.

**Key Term**:

A facade is a static element which looks similar to the actual embedded third-party, but is not functional and therefore much less taxing on the page load.

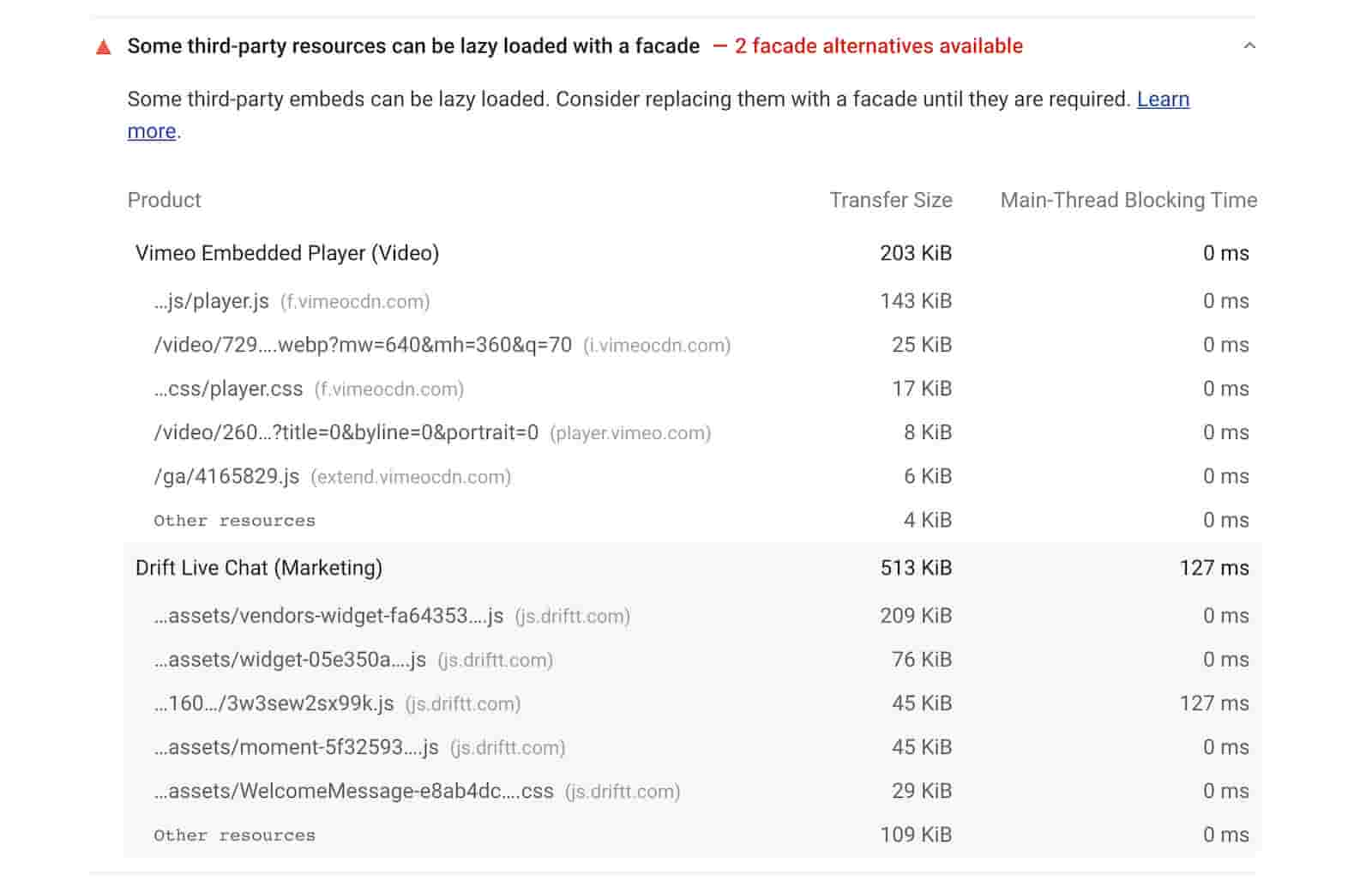
Loading YouTube embedded player with a facade.

## How Lighthouse detects deferrable third-party embeds [#](https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=devtools#how-lighthouse-detects-deferrable-third-party-embeds)

Lighthouse looks for third-party products which can be deferred, such as social button widgets or video embeds (for example, YouTube embedded player).

The data about deferrable products and available facades is [maintained in third-party-web](https://github.com/patrickhulce/third-party-web/).

The audit fails if the page loads resources belonging to one of these third-party embeds.

Lighthouse third-party facade audit.

## How to defer third-parties with a facade [#](https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=devtools#how-to-defer-third-parties-with-a-facade)

Instead of adding a third-party embed directly to your HTML, load the page with a static element that looks similar to the actual embedded third-party. The interaction pattern should look something like this:

1. On load: Add facade to the page.
2. On mouseover: The facade preconnects to third-party resources.
3. On click: The facade replaces itself with the third-party product.

## Recommended facades [#](https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=devtools#recommended-facades)

In general, video embeds, social button widgets, and chat widgets can all employ the facade pattern. The list below offers our recommendations of open-source facades. When choosing a facade, take into account the balance between the size and feature set. You can also use a lazy iframe loader such as [vb/lazyframe](https://github.com/vb/lazyframe).

### YouTube embedded player [#](https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=devtools#youtube-embedded-player)

* [paulirish/lite-youtube-embed](https://github.com/paulirish/lite-youtube-embed)
* [justinribeiro/lite-youtube](https://github.com/justinribeiro/lite-youtube)
* [Daugilas/lazyYT](https://github.com/Daugilas/lazyYT)

### Vimeo embedded player [#](https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=devtools#vimeo-embedded-player)

* [luwes/lite-vimeo-embed](https://github.com/luwes/lite-vimeo-embed)
* [slightlyoff/lite-vimeo](https://github.com/slightlyoff/lite-vimeo)

### Live chat (Intercom, Drift, Help Scout, Facebook Messenger) [#](https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=devtools#live-chat-(intercom-drift-help-scout-facebook-messenger))

* [calibreapp/react-live-chat-loader](https://github.com/calibreapp/react-live-chat-loader) ([blog post](https://calibreapp.com/blog/fast-live-chat))

**Caution**:

There are some tradeoffs when lazily loading third-parties with facades as they do not have the full range of functionality of the actual embeds. For example, the Drift Live Chat bubble has a badge indicating the number of new messages. If the live chat bubble is deferred with a facade, the bubble appears when the actual chat widget is loaded in after the browser fires requestIdleCallback. For video embeds, autoplay may not work consistently if it's loaded lazily.

## Writing your own facade [#](https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=devtools#writing-your-own-facade)

You may choose to [build a custom facade solution](https://wildbit.com/blog/2020/09/30/getting-postmark-lighthouse-performance-score-to-100#:~:text=What%20if%20we%20could%20replace%20the%20real%20widget) which employs the interaction pattern outlined above. The facade should be significantly smaller in comparison to the deferred third-party product and only include enough code to mimic the appearance of the product.

If you would like your solution to be included in the list above, check out the [submissions process](https://github.com/patrickhulce/third-party-web/blob/master/facades.md).

Best solution:

## Loading on delay is no longer a good idea in most circumstances

Adding a script after a delay is [not anything new](https://stackoverflow.com/a/59454541/2702894) and [I have expressed concerns / problems with a very similar solution of using a delay on the script before](https://stackoverflow.com/a/63371186/2702894).

Now since my first answer I listed there **things have changed a lot and I would no longer advise using setTimeout for delaying scripts due to massive variability in connection speeds in the real world**.

Google Page Speed Insights is much more relevant now due to the introduction of [Web Vitals](https://web.dev/vitals/), with a huge [update to Google Search algorithm being implemented next year](https://moz.com/blog/core-web-vitals) around these metrics.

These metrics are measured in the real world, so hacks are not going to cut it anymore.

## A better way - loading on intent

As such a much better approach is to load facebook messenger **on intent**.

By this I mean:

1. Add the facebook messenger script as the very last item to download, ensure it has the async or defer attribute. **do not initialise the script**
2. add a custom "facebook messenger" button to your page.
3. when a user clicks that button initialise the messenger.
4. replace your custom button with the facebook chat button (it should automatically appear so just hide your button).

A lot of the issues with messenger are due to the massive code bloat and high CPU execution time of that code.

If you implement it this way you are not delaying any rendering (due to async or defer on the script tag) and you are not having to worry about the overhead of initialising the messenger until the user presses your button.

## Advanced way to determine when to include bloated scripts

You need to know the network speed (and CPU speed if possible) and then you can include the facebook script early if the network speed is fast or late if it is slow.

### Network Speed

Luckily nowadays we can actually calculate network speed pretty reliably and make a decent guess for CPU speed.

Network speed can be calculated by using a [performanceObserver](https://developer.mozilla.org/en-US/docs/Web/API/PerformanceObserver).

You look at the first 10 requests on a page (using `entryType == "resource"``) and from that we can calculate network latency and download speed.

Then we set some threshold that we think is fast enough to include facebook messenger early or late and act accordingly.

You can also use performance observer to see when all resources are downloaded on the page **if you know how many requests the page has** and then trigger the facebook script download.

### CPU speed

With regards to CPU - if the browser supports it we can gather how many cores a processor has using window.navigator.hardwareConcurrency, 4 or less we assume it is slower, 5 or more we assume it is faster. It isn't super accurate but it can be a nice addition to the above method for making fine adjustments.

Alternatively for calculating CPU load you could use performanceObserver again to listen for longtask items. If there have been no long tasks for X seconds then include the facebook script.

This can be achieved by setting a timer for x seconds and then listening for entryType== "longtask" and then resetting the timer if a longtask entry occurs.

This final way of doing it ensures that on a very slow device facebook messenger is not being loaded until the rendering and bootstrapping of the essential page items is complete.

## Final Thoughts

**In the end it all depends on how important Facebook Messenger (or any third party library) is as to when you load it and how you handle it.**

Using the [web vitals library](https://github.com/GoogleChrome/web-vitals) to measure Real User Metrics (RUM) is far better for making decisions like this than arbitrarily adding a delay, you may find most of your users do not experience a problem.

Default solution:

My solution was like this I set a cookie for 30 min and if cookie is active chat is loaded for every page after first button click here is the code.

HTML/CSS

<div class="fake-button"><img src="messenger-icon.svg"></div>

.animate > img {

animation: pulse 1s linear 4;

}

@keyframes pulse {

0% { transform: scale(1); }

25% { transform: scale(1.1); }

50% { transform: scale(1); }

75% { transform: scale(0.9); }

100% { transform: scale(1); }

}

And JS code

$(window).load(function () {

if (getCookie('fb-chat')) {

FB.XFBML.parse();

}

});

$('.fake-button').on('click', function () {

$(this).addClass('animate');

FB.XFBML.parse();

setCookie('fb-chat', true, 0.0216);

}) .on("animationend", function(){

FB.CustomerChat.showDialog();

$(this).removeClass('animate');

});

window.fbAsyncInit = function() {

FB.init({

xfbml : false, #set value from true to false

version : 'v3.2'

});

};

(function(d, s, id) {

var js, fjs = d.getElementsByTagName(s)[0];

if (d.getElementById(id)) return;

js = d.createElement(s); js.id = id;

js.src = 'https://connect.facebook.net/en\_US/sdk/xfbml.customerchat.js';

fjs.parentNode.insertBefore(js, fjs);

}(document, 'script', 'facebook-jssdk'));

function getCookie(name) {

var v = document.cookie.match('(^|;) ?' + name + '=([^;]\*)(;|$)');

return v ? v[2] : null;

}

function setCookie(name, value, days) {

var d = new Date;

d.setTime(d.getTime() + 24\*60\*60\*1000\*days);

document.cookie = name + "=" + value + ";path=/;expires=" +

d.toGMTString();

}

function deleteCookie(name) {

setCookie(name, '', -1);

}