AMP: Hopes and Fears

This essay has evolved from my notes and comments from the AMP Roadshow in Sunnyvale on April 4 and feedback received since. Any factual errors are mine. Feedback is always welcome.

> The questions/issues I have are in bold throughout the rest of the article.

> at the #amproadshow learning about AMP, what's next and what cool things you can do with it.
>
> I'm ambivalent about AMP and am hoping to be convinced otherwise
>
> — Carlos Araya (@elrond25) April 4, 2018

What problem is AMP trying to solve?

Particularly since the introduction of AMP stories and the loosening JavaScript restrictions on adding JavaScript to AMP content it's starting to look just like another optimized library to create web content and, furthermore, the AMP cache is starting to look more and more like the old m-dot mobile sites of yesterday. When using an AMP cache link on desktop I've been redirected to the desktop version of the site and not given access to the AMP version; my expectation is that links to the cache will point me towards the AMP version of the content regardless of what form factor I'm using to access it.

> So it begs the question:
>
> **Beyond performance, what else are we trying to solve with it?**

User experience

One of the things that bothers me is that AMP looks like it wants to replace most other web technologies that try to improve the user experience.

> This is part of a more general question why Google competes with itself in so

many different categories? In the web space is AMP trying to provide a single solution that will work instead of existing web properties like Polymer and Angular?

So, it begs the first question in this category.

If this is for best users experiences why not support technologies that enhance what's already there instead of creating something new?

It may be true that this wouldn't solve the bloat problem of the web as it is today, but I don't think that being overly restrictive is a good solution either. AMP proposes a solution now and, to be honest, I can't think of a better.

The AMP team acts as gatekeepers for what is and isn't part of the AMP stack and platform. My problem, for example, is that I can't run the Prism syntax highlighter into AMP code (mine is generated by WordPress' wp-amp plugin compiled from source). It works under normal circumstances when used with non-AMP content so why should AMP content be any different? It may be true that I need to enable asynchronous highlighting on the Prism side of the equation but not because it's a requirement to use a library.

As an aide, Prism does support asynchronous highlighting via workers, just not by default. According to the Prism documentation:

> **Why is asynchronous highlighting disabled by default?**
>
> Web Workers are good for preventing syntax highlighting of really large code blocks from blocking the main UI thread. In most cases, you will want to highlight reasonably sized chunks of code, and this will not be needed. Furthermore, using Web Workers is actually **slower** than synchronously highlighting, due to the overhead of creating and terminating the Worker. It just appears faster in these cases because it doesn't block the main thread. In addition, since Web Workers operate on files instead of objects, plugins that hook on core parts of Prism (e.g. modify language definitions) will not work unless included in the same file (using the builder in the Download page will protect you from this pitfall). Lastly, Web Workers cannot interact with the DOM and most other APIs (e.g. the console), so they are notoriously hard to debug.

> From the Prism FAQ

Since we can't use JavaScript on AMP content, how can we leverage web APIs without being forced to use AMP supported wrappers? I understand that you're working on relaxing the restriction of using JavaScript in AMP content and experimenting with using workers to offload running scripts (`amp-script`) but, right now, I can't use any APIs that the AMP team doesn't allow. Some of my issues in this area:

- If we need AMP sanctioned wrappers for scripted and interactive content, How many wrappers will we need to address users' content cases?
- How many proposed wrappers are approved. My own experience with filling an issue on Syntax highlighting leaves me somewhat less than hopeful in that area (issue 14425 in the AMP repository, for completeness sake)

All JavaScript APIs require some sort of launch script or a link inside the page that is using it... neither of which is allowed in AMP by default. If we pick features like service workers we have the initialization script (replaced with AMP's amp-install-serviceworker, and most other examples should be covered by using amp-iframe and amp-bind but only if you choose to work with iframes in your pages and learn the AMP way of doing things.

If we move to talk about CSS features, they fall under another restriction in the AMP platform. They don't apply here.

Until the AMP team relaxes the restrictions for scripts on AMP content the AMP library itself will not accept scripts other than the ones they whitelist, so whenever you need a script that is not approved by AMP the page will still work but it will lose all benefits from AMP.

In my case with my code highlights I still get something that looks different enough from my content to be distinguishable from prose but not what I expected as syntax highlighted code.

I guess I could pre-process the content and insert the elements I need before the content is loaded but doesn't that defeat the purpose? If we move everything to the server and pre-process, server side render the content and preload, aren't we inviting the same types of bottlenecks and potential rendering delays because of user's network connectivity?

Where to go if there's a problem?

Until better debugging tooling is available it's very hard to do good AMP development. Given all the restrictions of what you can't do and how you should do what you can, it's hard to remember that your primary debugging tools for AMP are the browser's Dev Tools and the Google Search Console for your site, assuming that you're after SEO and Google SERP preloading, and while they are good tools they are not what most people use to debug HTML.

Most people won't bother to check the search console to see if there is a problem with search and AMP or know how to fix it so having AMP working is not guarantee that the tool is working properly, or at all...

I had a problem with linked data being incorrect, The AMP validator did not check for correct syntax in the JSON-LD content, so I got no validation errors and only found out that there was a problem (and that it had been fixed) when I went to the Google Search Console looking for something else.

AMP is different from most tools out there. If we misconfigure a script we expect it to fail. However, if we miss a tag or misquote an attribute in HTML the expectation, right or wrong, is that the page, up to the point where the error happened, will still render. With AMP I'm not certain this is the case, at least it wasn't my experience.

So whose responsibility is it? Mine? The plugin developer? The AMP team?

I also don't know if AMP Validation Errors will cause the page not to render or if these are warnings of things that must be fixed to be compliant. Furthermore, will any errors render the page not AMP-compliant and, therefore, unable to benefit from the AMP ecosystem?

Garbage in, Garbage out

I host my own installation of Wordpress so this is less of an issue for me. But for the people who host content in Wordppress.com this may come in as a surprise: AMP is enabled by default! (see AMP (Accelerated Mobile Pages) in the Wordpress.com support site).

Twitter is also sending users to AMP version of the content (when available) by default. Shouldn't this be an opt in experience rather than opt out or no choice at

all?

This is less of an issue for Twitter and self-hosted versions of Wordpress than it is for Wordpres.com hosted blogs because in the earlier cases, it's a conscious action to link to an AMP resource or install and activate the plugin. In Wordpress.com it's a setting enabled by default so it'll publish AMP until you, the content creator, disable it. This means that Wordpress.com authors may not be aware that they are publishing AMP for their blog.

This is not a problem for the AMP team but, in my opinion, will reflect poorly on AMP the platform because if we get low-quality content pushed into AMP nobody wins.

Duplicating effort?

When AMP first came out as an optimization library I had my reservations but saw it as an interim solution to the mobile performance issues that I see documented everywhere and that have gotten progressively worse.

When I first saw AMP, I saw it sold as a performance improvement for mobile load times. It may not be the purpose behind the project but that's how most people see it.

But even if the assumption that AMP is all about mobile performance (and, according to the AMP team, it is not) that is changing in ways that I wasn't expecting.

The concept of stories at least according to the news I've read out there, throws the concept of non-AMP sites out the window. If there is no equivalency check, what will search engines do when the content is different? Which version of the content will get priority?

According to Search Engine Land:

> Breaking with previous guidance, Google does not intend for this type of AMP content to match your non-mobile content. Instead, this mobile-only content should be unique. I followed up on this point with a Google rep, who characterized AMP Story content this way:

> > AMP stories is all about encouraging new forms of expression and storytelling, so we expect most publishers to not be already publishing this kind of content out on the web currently. AMP stories is meant to facilitate this and make it easier.
>
> The rep also added that "AMP Story content should be fulfilling and standalone."
>
> Because Google has historically argued specifically against creating different content for different users, this new "separate but equal" stance surprised me. Google further explained:
>
> > This does not run counter to the idea that the publisher may also have separate expressions [of] similar content published. Say a publisher has a long-form analysis article about college admissions data. Then, say they have a companion piece that's an interactive experience letting people play with the data through charts and other visualizations. It wouldn't be correct to call the long-form piece the canonical for the interactive. They each stand on their own, although being related.

This bothers me in so many levels but I'll start with the most basic concern and fear: Are we turning the mobile web into a single-vendor experience? Will this difference between stories and web content force publishers into AMP or be left behind by not providing good results in the search page? what happens if the search engine doesn't support cannonincal URLs like Google does?

AMP like many major projects like Angular, React, Bootstrap and Polymer are Open Source but the company that originally released the software maintains certain level of control either in the decision making process or in vetting changes to the codebase

How about leveraging other parts of the web platform to tell the same type of stories you do with AMP? Again, isn't improving the whole web experience the

name of the game? Having looked deeper at the stack AMP uses I can see a lot of good practices and things we should all be doing to improve users' experience with our web content. Creating a brand-new product that hides all the best practices under a set of restrictions and custom elements may not be the way to provide the interim solution that will work and teach the best practices that people, particularly beginners, need to learn.

Technical questions

There are technical questions that came up when researching AMP and writing this post a few more technical questions

Converting your content?

In talking to people at the Roadshow I now understand that AMP is a testbed to make things better and then bring it back to the wider web community; this is an important goal. I just have issues with the way AMP is doing this.

The example below is a basic sample HTML document conforming to the AMP HTML specification.

```
<!doctype html>
<html ⚡>
  <hea<html ⚡>meta charset="utf-8">
    <title>Sample document</title>
    <lin<title>canonical" href="./regular-html-version.html">
    <meta name="viewport" content="width=device-width,minimum-scale=1,init
    <style amp-custom>
    <meta name="viewport" content="width=device-width,minimum-scale=1,init
      h1 {color: red}
    org",
      "@type": "NewsArticle",
      "headline": "Article headline",
      "image": [
        "thumbnail1.jpg"
      ],
      "datePublished": "2015-02-05T08:00:00+08:00"
    }
```

```
    </script>
    <script async custom-element="amp-carousel" src="https://cdn.ampproje
    <style amp-boilerplate>body{-</script>
    {
      "@context": "http://schema.org",
      "@type": "NewsArticle",
      "headline": "Article headline",
      "image": [
        "thumbnail1.jpg"
      ],
      "datePublished": "2015-02-05T08:00:00+08:00"
    }
    (1,end) 0s 1 normal both;animation:-amp-start 8s steps(1,end) 0s 1 nor
  </body>
</html>
```

Some of the requirements and limitations of AMP are not evident. According to AMP HTML Specification an AMP document must:

- start with the doctype <!doctype html>
- contain a top-level <html ⚡> tag (<html amp> is accepted as well)
- contain <head> and <body> tags (They are optional in HTML)
- contain a <link rel="canonical" href="$SOME_URL"> tag inside their head that points to the regular HTML version of the AMP HTML document or to itself if no such HTML version exists
- contain a <meta charset="utf-8"> tag as the first child of their head tag
- contain a <meta name="viewport" content="width=device-width,minimum-scale=1"> tag inside their head tag. It's also recommended to include initial-scale=1
- contain a <script async src="https://cdn.ampproject.org/v0.js"></script> tag inside their head tag
- contain the AMP boilerplate code (head > style[amp-boilerplate] and noscript > style[amp-boilerplate]) in their head tag

If this was all that AMP required for compliance I'd be ok. I've always been a gun-ho fan of proper page structure and including the appropriate elements where they should be. Death to tagsoup markup!

> I love the idea of having metadata inlined in the document. This makes it

easier to enforce (it's part of the AMP html spec) but it may make it harder to support in the long run as any change to the doc would require.

I may even forgive the extra attribute (other than class) in the html element.

But this is just the beginning.

Amp not only requires a certain structure from your HTML page but it changes some elements for AMP-specific ones and completely disallows others. I'm mostly OK with the tag it forbids as they are mostly legacy elements that have been kept on the specifications for compatibility (don't break the web) reasons. I mean, seriously, frameset and frames? It's 2018 people... we have better ways of doing that!

The table below (also from the AMP HTML Specification) explains the limitations and changes that AMP makes to standard HTML elements. Some of these changes help explain the structure of the example page seen earlier.

Tag	Status in AMP HTML
script	Prohibited unless the type is `application/ld+json`. (Other non-executable values may be added as needed.) Exception is the mandatory script tag to load the AMP runtime and the script tags to load extended components.
noscript	Allowed. Can be used anywhere in the document. If specified, the content inside the `<noscript>` element displays if JavaScript is disabled by the user.
base	Prohibited.
img	Replaced with `amp-img`. Please note: `<img>` is a Void Element according to HTML5, so it does not have an end tag. However, `<amp-img>` does have an end tag `</amp-img>`.
video	Replaced with `amp-video`.
audio	Replaced with `amp-audio`.
iframe	Replaced with `amp-iframe`.
frame	Prohibited.
frameset	Prohibited.

object	Prohibited.
param	Prohibited.
applet	Prohibited.
embed	Prohibited.
form	Allowed. Require including amp-form extension.
input elements	Mostly allowed with exception of some input types, namely, `<input[type=image]>`, `<input[type=button]>`, `<input[type=password]>`, `<input[type=file]>` are invalid. Related tags are also allowed: `<fieldset>`, `<label>`
button	Allowed.
style	Required style tag for amp-boilerplate. One additional style tag is allowed in head tag for the purpose of custom styling. This style tag must have the attribute `amp-custom`.
link	`rel` values registered on microformats.org are allowed. If a `rel` value is missing from our whitelist, please submit an issue. `stylesheet` and other values like `preconnect`, `prerender` and `prefetch` that have side effects in the browser are disallowed. There is a special case for fetching stylesheets from whitelisted font providers.
meta	The `http-equiv` attribute may be used for specific allowable values; see the AMP validator specification for details.
a	The `href` attribute value must not begin with `javascript:`. If set, the `target` attribute value must be `_blank`. Otherwise allowed.
svg	Most SVG elements are allowed.

But it's the changes that they make to standard HTML elements that worry me. Do we really need an image custom element? video? audio?

The docs say it's ok to use `link rel=manfiest` but there are no references in the specification that `link rel="manifest" is allowed.

I don't think this solves the underlying problems of the web. It doesn't change the fact that loading speed and payload size numbers are getting slower and bigger, not faster and smaller for the mobile web.

See the HTTP Archive comparison data between January 1, 2017 and March 15, 2018 for an idea of what I'm talking about.

Granted, the HTTPArchive crawl may not catch AMP pages in its mobile results and that the pages in the crawl may not have a link to the AMP version of the page so the numbers may not reflect the actual improvements (if any) that AMP has made on the web. That said, I expect the numbers to change, one way or another, in future HTTP Archive cralws as they incorporate Wappalyzer as a tool in future crawls.

But AMP doesn't change the fact that we still have megabytes of images, even if they are compressed (if we get more space we'll just load more of them) huge bundles of JavaScript that will take tens of seconds to load and unblock page rendering on the web.

Even though people, companies and doc sites have been promoting best practices for years there has been no real improvement on how we work on the web and AMP, by hiding all the implementation details of lazy loading,

There are many more solutions outside of AMP that will give you performant web content. I think the biggest issue is that people may not know or may not care about these new technologies and smaller libraries that will take care of some performance bottlenecks.

Generating AMP?

Hand writing AMP is different than writing HTML but how do we generate AMP when working with CMS other than Wordpress? How much do we need to retool our systems based on the restrictions imposed by AMP?

Condé Nast has published information on how they generate the AMP content for each of their publications: Allure, Architectural Digest, Ars Technica, Backchannel, Bon Appétit, Brides, Condé Nast Traveler, Epicurious, Glamour, Golf Digest, GQ, Pitchfork, Self, Teen Vogue, The New Yorker, Vanity Fair, Vogue, W and Wired.

Their system is fairly intricate as the image below illustrates:

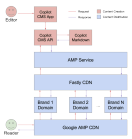

Figure 1: AMP Generation Process at

Condé Nast. From The Why and How of Google AMP at Condé Nast

And the AMP generation process itself looks like it's rather complex, running from Markdown to React that then converts the Markdown content into valid AMP HTML:

Figure 2: Condé Nast AMP Service Pipeline. From The Why and How of Google AMP at Condé Nast

So now we have to insert a way to identify when we want our CMS to serve AMP content and an AMP conversion process into our pipeline… but only for our AMP optimized content, right? I go back to the question I asked earlier about creating AMP only sites: Are they necessary?

Furthermore, I'm concerned about the amount of work needed to get AMP into a CMS. If you're not using a commercially available platform, how do you build AMP in addition to the standard content? As we saw in Converting your content? the restrictions of an AMP page are different than those of normal HTML (even if some of the restrictions are sensible and make sense). Sure we have WordPress and several other CMS platforms from AMP's Supported Platforms, Vendors and Partners page:

- AMPize.me
- Arc Publishing
- Canvas
- Drupal
- Fastcommerce
- Hatena
- Kentico
- Marfeel
- Rabbit
- Squarespace
- Textpattern
- TownNews
- Tumblr
- WordPress.com
- WordPress.org

Another thing that comes to mind when it comes to generating AMP programmatically is how much of the CSS we use in our normal site can we use with AMP?

The AMP specification requires developers to use no more than 50kb of CSS and, on its face, it's a sensible requirement. That is until we realize that CSS animations and any extra prefixing we need to acomodate browser idiosyncrasies would also fall under that limitation and we need to decide early on how will we make the CSS fit into the 50KB requirements or how much to take out to make the CSS fit the size restriction and still keep branding folks happy.

This is not to say we should dump hundred and hundred of lines of useless CSS into our pages but we should be the ones who decide how much CSS we use and how the CSS we use interacts with the existing content of the page. If we need to create one stylesheet for all our pages we can run them through UNCSS or similar tool to only serve the CSS the page actually uses and we also have the choice to inline it on the page.

Same thing with JavaScript.

How many AMP tags are too many?

There is a growing number of purpose-specific custom AMP elements. I can understand the need for `amp-img` and `amp-video` but I become more skeptic when I see `amp-vimeo`, `amp-youtube`,`amp-brid-player` or `amp-kaltura-`

`player`. There may be specialized features that require the special elements but I don't think that's the case for all of them.

Below is the list of custom media elements that are part of the AMP ecosystem (`amp-img` is also part of the list but I removed it because it's built into the AMP library):

- amp-3d-gltf
- amp-3q-player
- amp-anim
- amp-apester-media
- amp-audio
- amp-bodymovin-animation
- amp-brid-player
- amp-brightcove
- amp-dailymotion
- amp-google-vrview-image
- amp-hulu
- amp-ima-video
- amp-imgur
- amp-izlesene
- amp-jwplayer
- amp-kaltura-player
- amp-nexxtv-player
- amp-o2-player
- amp-ooyala-player
- amp-playbuzz
- amp-reach-player
- amp-soundcloud
- amp-springboard-player
- amp-video
- amp-vimeo
- amp-wistia-player
- amp-youtube

It's interesting that there is an AMP specific Vimeo and element when the project itself describes how you can use `amp-iframe` to create the same experience:

```
<amp-iframe width="500"
  title="Netflix House of Cards branding: The Stack"
```

```
  height="281"
  layout="responsive"
  sandbox="allow-scripts allow-same-origin allow-popups"
  allowfullscreen
  frameborder="0"
  src="https://player.vimeo.com/video/140261016">
</amp-iframe>
```

Same for the Youtube element (although it doesn't appear in the `amp-iframe` docs. All that we do is change the tag name to `amp-iframe` and add `layout="responsive"` and `sandbox= allow-scripts allow-same-origin allow-popups`.

```
<amp-iframe
  layout="responsive"
  sandbox="allow-scripts allow-same-origin allow-popups"
  width="560"
  height="315"
  src="https://www.youtube.com/embed/3vcGsrMdiUA?rel=0"
  frameborder="0"
  allow="autoplay; encrypted-media"
  allowfullscreen
  playsinline></amp-iframe>
```

There is nothing you can do in the `amp-youtube` that you can't do using an `amp-iframe` element. So why the difference? Are the optimizations needed for a given player enough reason to create another custom element, another script to load and

Specific Use Cases

My specific cases of what AMP doesn't allow are Codepen embeds and Prism.js code syntax highlighting.

The typical expected Codepen emebd looks like this:

```
<p data-height="438" data-theme-id="dark" data-slug-hash="MGWazx" data-de
```

```
<scri<a href="https://codepen.io/caraya/pen/MGWazx/">mbed/ei.js"></script>
</script>
```

As far as I understand AMP, this would not work in an AMP page. AMP will not let you use script tags in a page, and even if AMP allows data-* attributes on elements they are useless because there's nothing to process them with.

Same thing with Prism. It works by adding a script and a style sheet to the page and then expects properly coded blocks of preformated text for the content to be highlighted. To validate AMP content I am not allowed to have additional script tags on the head or before the end of my document. In theory, I should be able to merge Prism's CSS with the AMP required CSS and my site's CSS but would that work in 50KB of CSS that we're allowed to add?

And yes, I know that Codepen also provides `iframes` for embedding pens but it's not the recomended form to do so.

The problem with Prism is more involved and something that I'm still trying to figure out. It requires workers but I'm still trying to figure out how to make them talk to each other.

CORS for my own site?

One of the biggest issues is hosting the content in a Google-based and supported CDN. This causes the large, ugly URLs (like `https://www-gq-com.cdn.ampproject.org/c/s/www.gq.com/story/gq-eye-exclusive-fatherhood-style-series---todd-snyder` or `https://www-gq-com.cdn.ampproject.org/c/s/www.gq.com/`) but also has an interesting side effect that makes CORS necessary for your own content.

When you go to a site, for example: `https://www.gq.com/story/gq-eye-exclusive-fatherhood-style-series---todd-snyder` in a mobile device it'll redirect you to the version hosted in the AMP CDN which turns into `https://www-gq-com.cdn.ampproject.org/c/s/www.gq.com/story/gq-eye-exclusive-fatherhood-style-series---todd-snyder` and caches it to guarantee preloads and seemingly instant first load experiences.

But what happens if you need to load data for your application? Say you want to load product data for your shopping cart, or a tour date list for your favorite band? Those are stored on your server, not the cache and for them to work

properly you need to set CORS headers on them so they will work for origins other than yours.

On a side note, if you hit the AMP cache URL from a desktop browser it will redirect you to the desktop, however, if you hit an AMP link using an activation script like `https://www.gq.com/story/gq-eye-exclusive-fatherhood-style-series---todd-snyder/amp` it will take you to the AMP version, regardless of platform or form factor. I find this even more interesting, why restrict some ways to access the content from some form factors and not others? Am I missing the point of the URLs?

Performance

People have been working with performance for years. The early work Ilia Grigorik and Addy Osmani did and continue to do at Google, tools like UNCSS, the fact that we can precache and preload content (but not provide automatic preloading like Google does with AMP search results), the fact that we have PWA and its component technologiestechnologiies available within and outside PWAs really makes me wonder how lazy we've become as developers.

I'll take three examples of things we can do now to make experiences outside of an AMP environment: Image Optimization, Resource PrioritizationPioritization and Service Workers.

We know that images make up for a large fraction of what gets pushed into a web page, we don't evangelize tools like Imagemin and workflows that use these tools to generate smaller images.

Does `amp-img` really reduce the bloat problem? If you make the images smaller that means people will put even more images in a page because the size of each individual image is smaller so we can use more of them, we don't need to worry about the individual image, AMP will take care of that.

It is not hard to create a good resource prioritization strategy using a combination of H2 Push, link preload, prefetch, prerender and dns-prefetch to load resources the way we want them to. If we compare this with lazy loading content so they will only load when needed, we have a much better web experience.

There is no current 100% foolproof way to get H/2 push to work consistently in all browsers as discussed in Jake Archibald's HTTP2 is tougher than I thought. As Jake points out in the article's conclusion:

> There are some pretty gnarly bugs around HTTP/2 push right now, but once those are fixed I think it becomes ideal for the kinds of assets we currently inline, especially render-critical CSS. Once cache digests land, we'll hopefully get the benefit of inlining, but also the benefit of caching.
>
> Getting this right will depend on smarter servers which allow us to correctly prioritise the streaming of content. Eg, I want to be able to stream my critical CSS in parallel with the head of the page, but then give full priority to the CSS, as it's a waste spending bandwidth on body content that the user can't yet render.

We're not there yet, but we can do something to make the whole load faster with a little help from the server. We create server-side user agent sniffing strings based on case-insensitive matching with BrowserMatchNoCase directive, part of the mod_setenvif module.

```
# Windows 10-based PC using Edge browser
BrowserMatchNoCase edge browser=Edge

# Chrome OS-based laptop using Chrome browser (Chromebook)
BrowserMatchNoCase CrOS browser=ChromeOS

# Mac OS X-based computer using a Safari browser
BrowserMatchNoCase safari browser=Safari

# Windows 7-based PC using a Chrome browser
BrowserMatchNoCase chrome browser=chrome

# Linux-based PC using a Firefox browser
BrowserMatchNoCase firefox browser=firefox
```

We can then test if the content has been pushed to the client using cookies as described in Web Performance Improvement and serve resources based on the client browser and its current limiations.

We need to be aware that the cookie we set from the server as we push the resources is not completely reliable after the first time we push resources; that may cause us to miss resources or to load them more than once.

Service workers can also help improve performance of your web content after the initial visit. We can use it to keep a persistent cache of content for our site or application that can be populated and expired as needed; we can also modify the requests as needed and ameliorate the problems from h/2 push alone.

Libraries like Workbox can help us make this process easier.

A more radical solution is to server-side render your application. This will take care of perceived performance issues at the cost of potentially longer initial load times. Since the server has to process all the Javascript to render the content on the page. Eric Bidelman presents a way to use Puppeteer to SSR an Express Node application. Other frameworks like React and Vue have their own systems to server side render applications. It is up to you to decide if the extra work and the extra time it'll take for your application to render on the server is worth it.

One of the ways AMP makes content load faster is by pre-rendering content from the Google Search Result Page. Note how the pre-render only works for valid AMP pages. Malte Ubl, the tech lead for AMP, wrote a Medium post outlining why they chose to do so. While it's nice that they can be so aggressive in their prerendeering of their custom elements it still raises some questions about implementation and presents some interesting questions when talking about prerendering resources outside the AMP environment.

If you have more than one resource that the `amp-img` element is supposed to load, will it prerender all of them? For example in the following element, how will the AMP runtime know which image to prerender for browsers that may or may not support WebP?

```
<amp-img alt="Mountains"
  width="550"
  height="368"
  src="images/mountains.webp">
  <amp-img alt="Mountains"
    fallback
    width="550"
    height="368"
    src="images/mountains.jpg"></amp-img>
</amp-img>
```

Similar but even more worrisome is the case of responsive images. The example

below offers ten different options for this one single image (not counting 2x and 3x images for retina displays). Will the browser download all of them if the image is above the fold? Are images subject to the restriction of placement from the top of the page?

```
<amp-img src="/img/amp.jpg"
  srcset="  /img/amp.jpg 1080w,
            /img/amp-900.jpg 900w,
            /img/amp-800.jpg 800w,
            /img/amp-700.jpg 700w,
            /img/amp-600.jpg 600w,
            /img/amp-500.jpg 500w,
            /img/amp-400.jpg 400w,
            /img/amp-300.jpg 300w,
            /img/amp-200.jpg 200w,
            /img/amp-100.jpg 100w"
  width="1080"
  height="610"
  layout="responsive"
  alt="AMP"></amp-img>
```

We do have the options of prerendering content outside of the AMP ecosystem but the question becomes: How much? Is it worth taking what Steve Souders calls “the nuclear option” of preloading your content? How do you pick what content to prerender? Do we prerender all the links in our page? All the internal links? External?

Google has the resources to proactively prerender the AMP pages it serves from the AMP cache and that’s the only way we can take full advantage of AMP, using their technology and their delivery mechanism. This doesn’t mean that AMP’s prerender mechanism is the best way to go as it trades speed for bandwidth.

I recently came accross Tim Kladlec’s How Fast Is Amp Really? performance analysis of AMP content served in different ways:

1. How well does AMP perform in the context of Google search?
2. How well does the AMP library perform when used as a standalone framework?

3. How well does AMP perform when the library is served using the AMP cache?
4. How well does AMP perform compared to the canonical article?

His findings mirror my opinion about AMP not doing anything that we can't do in the open web. To me, the most, important section of Tim's article is when he talks about page weight reduction:

> The 90th percentile weight for the canonical version is 5,229kb. The 90th percentile weight for AMP documents served from the same origin is 1,553kb— a savings of around 70% in page weight. The 90th percentile request count for the canonical version is 647, for AMP documents it's 151. That's a reduction of nearly 77%.
>
> AMP's restrictions mean less stuff. It's a concession publishers are willing to make in exchange for the enhanced distribution Google provides, but that they hesitate to make for their canonical versions.
>
> If we're grading AMP on the goal of making the web faster, the evidence isn't particularly compelling. Every single one of these publishers has an AMP version of these articles in addition to a non-AMP version.
>
> From: How Fast Is Amp Really?

AMP restricts what you can do with JavaScript and what you can do with custom AMP elements restricts what you can do on your pages and, regardless of the reasons why you put the restrictions in place, it shouldn't be up to a library or framework to dictate how much stuff you put in a page but it's up to the designers and developers to slim down their content... easier said than done, I know, but AMP hasn't sold me as the solution or even a good solution to this problem and neither do publishers, otherwise AMP would be the only version of all published web content.

The other interesting aspect of this conversation is what happens to search engines like Bing or other platforms that don't support cannonical links?

The final, technical, question I have is how well AMP works in low end and feature phones or for people using Proxy browsers such as Opera Mini, UC

browser or Chrome for Android in their most aggressive data saving modes? How do publishers in those markets address the needs of their users while still remaining compliant with AMP requirements?

Conclusion

I wrote the tweet below as a response to my original tweet.

> No, @AMPhtml hasn't fully sold me on it. Too many open issues in AMP that are easy to do in non-amp. Looking at component creation and how hard it is
>
> — Carlos Araya (@elrond25) April 6, 2018

As I've mentioned throughout this post the technology behind AMP is not revolutionary. It's a reaction to a problem on the mobile web and it has been taken as a way to gate keep and restrict what can be done on the web.

There are many unanswered questions about AMP and the direction it's going on for me to be comfortable with it. AMP email and Stories are out, what comes next?

Attempting to make AMP the canonical version of your web content worries me, it means we're moving away from having an HTML version altogether and people may choose to learn AMP but not HTML (sorry, but HTML is the spec that lives at WHATWG and W3C and pages built from that spec, not AMP) and when AMP limitations become too restrictive will not know how to build web content that doesn't require AMP and be starting from square one again.

It is tempting to draw comparisons between AMP and jQuery but the web was different when jQuery was first introduced. The use cases jQuery dealt with were about smoothing out different ways browsers did things (looking at you, Netscape and IE) and making sure you wouldn't have to write similar code 3 times (one for IE, one for Netscape and one for browsers that supported standards when those became available) but now the technology is available to make sure we do things right, it's a matter of whether we choose to use them wisely or not.

I mentioned earlier but having AMP Stories be completely different to any non-amp alternatives makes me worry as much if not more than making AMP the canonical version of my content. If there is no relation between my web content

and the AMP stories that are built around it then what's going to be the content that gets returned by the Googlebot when you search for it? I know this is not, strictly, an AMP issue and more of a search issue but since it's an AMP product it's worth pointing out here.

I'll keep my mind open towards AMP (after all I still provide AMP for my blog content to help people with slow connections or who may have data cost issues) but there are too many open questions about the platform and its direction before I can say I agree with their assessment of how they want to move the web forward.

Links and Resources

- AMP content
    - https://www.ampproject.org/docs/fundamentals/converting
    - Improving URLS for AMP Pages
    - AMP New Horizons
- AMP Elements/Components/Templates
    - amp-story element
    - AMP Story Example
    - amp-iframe element
    - News and Blogs Templates
- AMP Stories
    - Introducing AMP Stories, A Whole New Way To Read Wired
    - Wired Introductory AMP Story
    - Search Engine Land Piece on AMP Stories
    - Google Releases AMP Stories: What Are They All About?
- Governance
    - Githubb Discussion
- AMP at Condé Nast
    - The Why and How of Google AMP at Condé Nast
    - Evolving Google AMP at Condé Nast
- Questions, Commennts and Criticisms
    - Ethan Marcotte
        - I, for one
        - AMPlified
        - AMPersand
    - Jeremy Keith
        - AMPed up
        - The meaning of AMP
        - In AMP we trust

        - Tim Kladec
            - How Fast Is Amp Really?
            - The Two Faces of AMP
            - AMP and the Web
            - AMP and Incentives
            - CPP: A Standardized Alternative to AMP
- AMP for you (whether you want it or not)
    - Twitter
        - Twitter
        - Twitter Developer Docs
    - Wordpress
        - Wordpress Support
        - WordPress Sites Now Support Google's AMP To Make Mobile Pages Load Much Faster
- SSR
    - The simple guide to server-side rendering React with styled-components
    - Upgrading a create-react-app project to a SSR + code splitting setup
    - Adding state management with Redux in a CRA + SSR project
    - Server-Side Rendering in Vue
- Performance
    - Prefetching, preloading, prebrowsing
    - Prebrowsing
- Other
    - Taking AMP for a Spin
    - AMP News
    - Need to Catch Up on the AMP Debate?