Smarter pre{fetch,rendering} for the mobile web

igrigorik@google.com

*Our goal is to enable **instant delivery of pages** - from initial click to useful content on the screen within a few hundred milliseconds. The problem is, on mobile, and even with the latest 4G networks, just the network latency required to set up the radio context, and all the associated DNS, TCP, and TLS roundtrips, easily add up to 500 ms+, and multiple seconds for 3G.*

*As a result, to meet our goal, we **must use** additional techniques, such as prefetching and prerendering, that can anticipate the next user action and initiate the necessary resource fetches to "hide" and minimize latency experienced by the user - otherwise, we'll all be staring at blank white pages for seconds at a time.*

Status of prefetching and prerendering

The good news is that we have (kinda, sorta) the necessary tools: modern browsers (Chrome, FF, IE) have built in mechanism to initiate DNS pre-resolves, TCP preconnect, and other tricks. Similarly, we have several mechanisms that site authors can use to hint the browser:

- **Prefetch:** `<link rel="prefetch" href="/next_page.html">`
  - Fetches linked resource and places it in cache. Prefetch is meant as a hint for resources that may be used for **next** navigation. The actual resource is not parsed or interpreted in any way - e.g. HTML is placed in cache but not parsed.
- **Prerender:** `<link rel="prerender" href="/next_page.html">`
  - Fetches linked document and renders it in a background tab. All dependent resources are downloaded, parsed, and executed. In effect, this is a full render, which is then swapped in later.

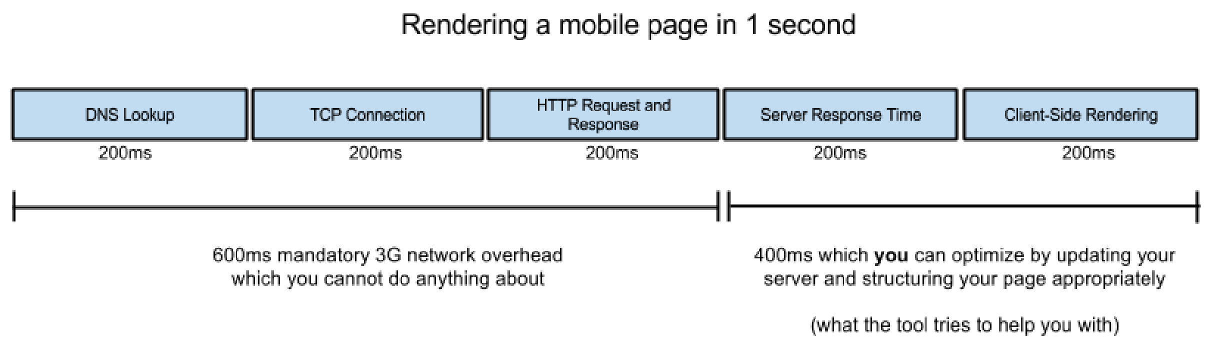
Prefetch has "pretty good"™ browser support - in theory, should work, in practice there are implementation bugs (e.g. in Chrome) that need to be addressed. Prerendering is supported by Chrome and IE11, except that, it has a number of gotchas for mobile - e.g. Chrome mobile keeps prerendering as a "Wi-Fi only" option and depending on who you talk to, the reasons are: high network costs (we're downloading the entire page and all its assets), and high resource costs (CPU, GPU, memory) on the device. In fact, even on desktop browsers, the number of prerender's is capped at a low (1..3) number to keep the overhead under control.

The net result is that neither prefetch nor prerender hints do us much good on mobile -- exactly where we need them most. Having said that, it's not just a matter of enabling both...

Smarter prerendering for mobile

Prefetching is not enough and prerendering is too much - we need something in between. Or, more specifically, the browser should be much smarter about how much (speculative) work it is willing to do upfront to help accelerate the time to render.

First, let's illustrate the problem. On mobile, we have two problems: latency and resources.



Rendering a mobile page in 1 second
DNS Lookup
200ms
TCP Connection
200ms
HTTP Request and Response
200ms
Server Response Time
200ms
Client-Side Rendering
200ms
600ms mandatory 3G network overhead which you cannot do anything about
400ms which you can optimize by updating your server and structuring your page appropriately
(what the tool tries to help you with)

1. Radio negotiation can take anywhere from 100 ms (3.75-4G) to multiple seconds (3G)
2. DNS, TCP handshake, TLS negotiation: 2 to 4 roundtrips
    - Sprinkle in some redirects to complicate matters further
3. Server response time (hopefully fast)
4. Get the HTML, discover critical resources (can't render yet)
    - Start fetching critical CSS/JS
        - Repeat steps 1-3 if on different host
        - Slightly better performance if on same host, and if we're lucky we may be able to reuse an existing connection.
5. Once we have CSS / JS, parse it, build the DOM, paint some pixels (hooray!).

Not breaking any news here, but **latency is the issue** - a typical mobile site is blocked on a dozen+ roundtrips before we can paint some useful pixels to the screen. That sucks, and we can do better.

- If the user is sensitive to extra data downloads (e.g. roaming, on a metered data plan, etc.), then we can still perform steps 1 and 2: perform DNS pre-resolution, TCP preconnects, even negotiate the TLS tunnel if necessary. This kind of logic exists in most browsers today (e.g. Chrome predictor), but the developer should be able to hint the browser what these hosts are.

    - We have dns-prefetch, but there is no “preconnect” hint. Having said that, I don’t think we should add “preconnect”... the browser should be smarter.
- If the user is OK with some speculative downloads (aka, trading bytes for rendering speed), then we can go further and download HTML and other resources as necessary.

In short , we need “smart prefetch for mobile”, which is able to fetch the HTML, the critical resources, and optionally parse and execute some of them, or even prerender the entire page:

1. Fetch and cache the HTML (i.e. prefetch)
2. Scan the prefetched HTML for critical resources (CSS, JS) and prefetch those
3. Parse / execute some of the critical resources (i.e. start construction of the DOM, CSSOM, etc)
4. Render the full page, fetch associated images and other non-critical assets (i.e. prerender)

We have (1) and (4), but what we need is for the browser to implement (2) and (3). **The browser is in the best position to determine how far down it needs to go, based on current connection speed, battery life, available CPU and memory, user preferences, etc.** Some example scenarios and strategies:

- If there is only one “prefetch” hints on the page, the browser may want to fetch just the HTML and place it in its cache.
    - The browser could also open the connection (DNS lookup, TCP handshake, TLS setup), and fetch some fraction of the document (e.g. first X KB), after which it can stall the connection until the navigation is triggered (and if not, just RST the connection).
    - Depending on amount of time the user is spending on previous page, the browser could dial up/down its prefetch logic, and so on.
- If there are multiple “prefetch” hints on the page, perhaps do less work for each one, but still perform some speculative work - e.g. page embeds 3 hints, each of which issues a preconnect, but doesn’t download any content.
- Next, the browser may decide to run a “smart scanner” on the HTML to find critical resources and dispatch requests for those.
    - Scanner can be run on the full document, or just a portion -- this would allow the browser to fetch a fixed amount of data for each page and discover other critical resources in the head of the document (CSS, JS), such that it can begin other preconnects and prefetches.
- If there is sufficient memory, CPU, and network resources, and it deems the page as a “high likelihood next navigation” it can go further and initiate the full prerender.
- … *and so on... This logic can vary based on UA implementation.*

Focus developers on “smart prefetch”, defer logic to UA

Current browsers support three prefetch hints: dns-prefetch, prefetch, prerender. There is no reason to remove these, but **the browser should be smarter about which is triggered and when, with ability to decide how far it wants to go** - e.g. prefetch hint can serve as a "low priority render hint", but if the UA thinks it will help, it shouldn't be blocked from upgrading that to prerender. Conversely, a prerender hint could be downgraded to a prefetch, or just dns-prefetch based on local context: resource requirements, network speeds, etc.

Ultimately, it is the UA that has the most information to make the optimal decision on which type of prefetch should be run based on current network conditions, device type, available resources, user preferences, and so on:

- Some users may be OK with higher BW consumption vs more aggressive prefetch
- UA can change logic depending on network type and network weather
- UA can change logic based on other environmental factors - e.g. preserving battery, vs limited CPU resources, etc.

The web developer should be able to provide one or more prefetch hints, and the UA should do the rest. The UA can also vary its logic based on number of provided prefetch hints - e.g. use a "lighter" strategy for each hint when multiple prefetch hints are specified.

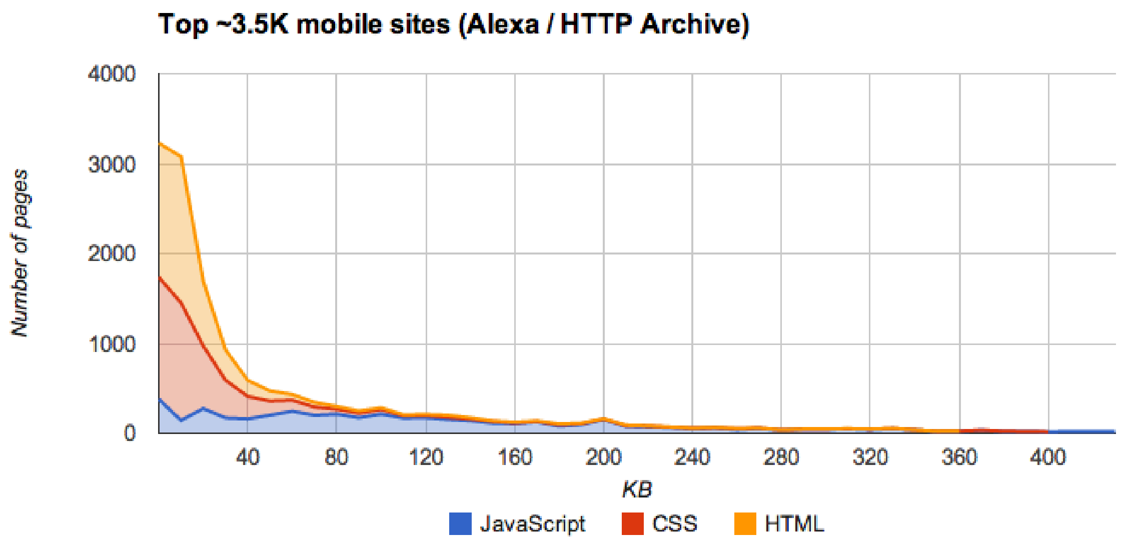
Further, "smart prefetch" hint should support any content-type. There is no reason to restrict it to HTML documents. The UA should be smart enough to prefetch and cache an image, or prefetch, parse, and execute an HTML doc, CSS file, and so on.

The developer / owner of the site has the best knowledge on the most likely next destination - prefetch allows them to specify this. The browser has the context about the device, network, and user accessing the site, and it should determine how to balance speed vs. other criteria.

Can we measure the benefits?

Based on 3000+ top mobile sites, tracked by HTTP Archive.

	HTML	**CSS**	**JavaScript**
Median	9 KB	12 KB	108 KB
75%	21 KB	30 KB	201 KB
90%	42 KB	65 KB	340 KB

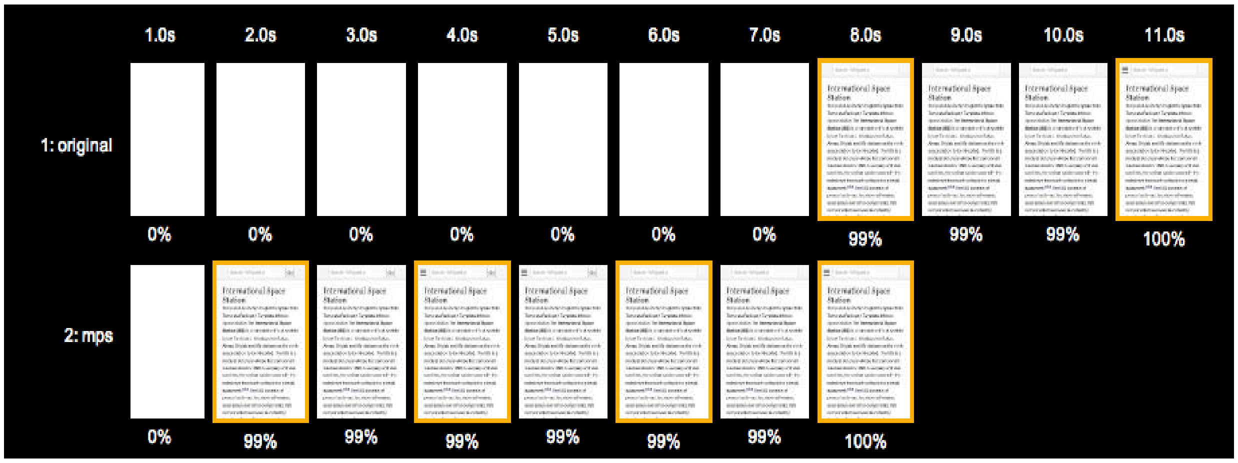


At a minimum, if we restrict ourselves to HTML and CSS (JS can be deferred by the site), we can fetch the critical resources for 50% of the mobile pages with as little as 21KB! However, to do so, we would need the browser to scan the prefetched HTML and initiate the CSS fetch.

By fetching a portion of the HTML -- let's say, one CWND's worth -- we can also cap the HTML download and allocate the rest to CSS and JavaScript. *<insert own strategy here based on current bandwidth, user preferences, and so on>.*

The important point is: the critical resources are relatively small compared to the rest of the page! We don't need to prerender the full page, but we do need something smarter than a simple resource prefetch. **By implementing this logic in the browser, we also eliminate the need for the site owners to manually defer assets, split their HTML, and apply similar hacks.**

For sake of an argument, let's imagine we had the "half-way prerender" implemented in the browser - aka, smart prefetch of critical assets, with no actual rendering - what could we get? Here's an example of current mobile Wikipedia page (8.0s render on 3G profile), which is optimized (via mod_pagespeed) to meet the mobile criteria: inlined critical CSS, defer JS...



From 8.0s down to 2.0s for first render - check out the video here. In effect, what above optimization illustrates is the case where critical assets are placed in cache and are available immediately - we get a 4X in rendering speed, without any actual background rendering.

(For bonus points, note that the "optimized case" in above example is still blocked on HTML download, so with an actual prefetch of the HTML, we can get an even better improvement in rendering time).

By making prefetch/prerender smarter, there is no reason why above can't be achieved directly in the browser, without any site or layout modifications: prefetch the HTML, identify critical blocking resources, begin prefetch of those. It's all about eliminating the blocking roundtrips...

Get to the point already...

What we need is "smarter prefetch". The UA is in the best position to determine how far it is willing to go, based on local context, user settings, and other criteria.

Current definition:
http://www.w3.org/html/wg/drafts/html/master/single-page.html#link-type-prefetch

Proposed definition:
*The prefetch keyword indicates that preemptively fetching, caching, **and processing** the specified resource is likely to be beneficial, as it is highly likely that the user will require this resource.*

From there, let the UA do the rest.