Prophecy: Accelerating Mobile Page Loads Using Final-state Write Logs
A review.

Kevin Dackow

July 2018

1 Summary

Prophecy aims to improve mobile web browsing performance. Specifically, it seeks to reduce battery usage, decrease page load time, and decrease bandwidth usage during web browsing.

1.1 Basic Structure

To accomplish this, Prophecy utilizes server-side processing. Through the use of write-logs, Prophecy sends the client the DOM state and Javascript state after evaluating the page completely. Additionally, it sends list of images to prefetch. Once this data is sent, the client's browser begins prefetching the images. While downloading them, the client interprets these write-logs to render the page. The theory behind this strategy is that server-side processing will reduce the processing burden on the client, thus reducing battery usage while decreasing page load time. Additionally, it intends to decrease bandwidth by ensuring only one HTTP rtt is needed to get all required site data.

1.2 The Implementation

1.2.1 Write Logs:

- The JS write-logs simply store all variables in the heap as a single line: $lhs = rhs;$. That is, the pre-evaluation assigns each variable to the final value it takes after processing the site server-side.
- The DOM write-log is an HTML string representation of the site using the browser's builtin $XMLSerializer$ interface. It additionally cuts out styling and JS interactivity, which is added later.

1.2.2 Client Side Scheduling

The scheduling for Prophecy is rather simple. Specifically, the Server sends back all of the aforementioned data plus a small JS Library that contains scheduling code and the site's CSS (in case objects are added after page load). The HTML that is evaluated by the client starts with a single line that is a script tag referencing the library. This library is evaluated, and during its evaluation the DOM and JS states are reassembled as the write-logs indicate. The DOM is reinstated using the built-in $DOMParser$ interface to quickly change the $< body >$ and $< head >$ elements of the DOM tree. This cuts load time dramatically, because everything is already processed and in the correct location - i.e. rather than parsing HTML, CSS, and JS to create a DOM, it is already made and only needs to be inserted.

1.2.3 Results

The results include a 53% decrease in median PLT, a 21% reduction in bandwidth consump-

tion, and a 36% reduction in energy usage. The preceding stats were for mobile devices, but smaller improvements were also found for PLT and bandwidth for desktops.

2 Analysis

Though the paper proposes a fascinating approach to mobile browsing enhancements, it presents a particularly troublesome claim. It claims implicitly that it maintains correctness of the page through the preprocessing. Most notably, the paper lacks a formal proof of the correctness of its server-side code-to-write-log reduction. That is, the paper presents no proof that the write-logs will be correct representations of the data that the client should have received in all cases.

Additionally, even if we assume that Prophecy accurately and correctly calculates and sends the site, there is another assumption regarding the determinism of sites. To be more specific, non-determinism is not mentioned in the paper. The lack of discussion of it is problematic because non-deterministic JS is common on the web.

Without any mention of this issue, we must look at the three possible ways that could Prophecy could respond to non-deterministic code: 1) The server preprocesses all non-deterministic calls and sends results via write-logs, 2) the server halts preprocessing at non-deterministic calls and sends that code in the write-logs, 3) some middle ground (e.g. some non-deterministic code is evaluated, some is not). There are problems with each of these approaches:

1. If the server fully evaluates non-deterministic code, then the values that should have been randomly selected by the client are instead randomly selected by the server and sent to the client. This can produce security risks. For example, consider the Elliptic Curve Digital Signature Algorithm (ECDSA) which is currently in use as a method of security for Bitcoin. In brief, the algorithm fundamentally relies on the random selection of a private key for the user. Suppose the user is in the process of generating this private key, and instead of it being generated locally it is generated by the server. Then, one can envision a man-in-the-middle intercepting the Prophecy write-log [1] and getting the key. This would then give the attacker access to the victim's virtual wallet. Though this example is not necessarily possible with Bitcoin, the risk of non-deterministic code being processed server-side rather than client-side is nonetheless evident in this example.

2. The server not fully evaluating non-deterministic code is likely the best option. This is because the risks described above are mitigated and all randomness can be generated on the client side. However, this raises other issues. Specifically, how can one fully evaluate the JS state or the DOM as Prophecy tries to do if some of the code could not be evaluated until the client gets to it?

3. A mix of the two approaches is the worst option, as it makes the code more unclear and essentially impossible to predict, while still creating the security risks described above.

Overall, the way that Prophecy describes the JS state write-logs implies that they evaluate all code server-side, however the paper's lack of clarity on the issue of determinism leaves it unclear. That said, any approach will have drawbacks - either by reducing the efficacy of the Prophecy write-logs by not fully evaluating, or by adding security risk.

[1] Is the write log encrypted? I would assume not, since encryption is not mentioned in the paper.