**Summary of Prophecy: Accelerating Mobile Page Loads Using Final-state Write Logs**

The paper argues that mobile web browsing is expensive in terms of computation, bandwidth and energy consumption. They make the web server more intelligent so that the client can have faster and robust web experience.

They propose a prophecy web server which pre-computes much of the information that a web browser would generate during the page load time. In particular, they compute Javascript heap and DOM state that belong to a loaded version of a frame (a.k.a webpage). They make write logs out of these final states and transmit the logs upon client’s HTTP request instead of sending unprocessed HTML, JS, CSS files. They further extend the system to support caching, personalization and cookies without any privacy concerns.

A prophecy system has two phases. 1) The web server first generates a prophecy version of frame either online or offline. Upon the client request, the web server employs a headless browser to request the frame. As the frame loads, a prophecy rewriter injects instrumentation which tracks how the frame manipulates the JS heap and DOM tree. Then a prophecy frame generator uses this instrumentation to generate a prophecy version of frame which consists of three logs -- *write log for JS heap, write log for DOM, an image prefetch log and a resurrection library*. The write log for JS contains a series of statements with final states of variables. The write log for DOM contains an HTML string representation of the frame with explicit style information. The image prefetch log contains a list of images. The resurrection library is a small piece of code which orchestrates the client-side reconstruction of the frame. 2) The client upon receiving the prophecy frame, executes the resurrection library received as part of prophecy frame. The library first issues image requests asynchronously. As the browser fetches images in the background, it builds the frame in three phases. First, it reconstructs the DOM, updates the JS heap, and patches some cross-references to handle event listeners.

The extend the system to support caching by maintaining different versions of write logs or diffs at the server. Whenever the client requests for a frame, the server looks for customization id in the cookie sent by the client. If the server can’t find the id, then it assumes that the client has cold cache and sends prophecy frame by placing a customization id. If it finds an id, then it assumes that the client cached the write logs and generates diffs between the latest write logs and the ones that are cached at client and sends only the diffs. This preserves privacy in a way that the server gets cookies only a frame that the server is hosting but not the cookies of other servers i.e, cross-origins.

To evaluate their system, they use FLT which is same as PLT (onLoad) and speed index. They show improvements over default browsing, Polaris and Shandian. They further extend the work to optimize it for ready index i.e, optimizing for visual and interactive content. They divide the write logs into ATF interactive content and below the fold content. First, they build the DOM for ATF content and update JS heap then builds the remaining subtrees. Overall, they improve energy by 36%, PLT by 53% and bandwidth by 18%.