WebMedic: Disentangling the Memory–Functionality Tension for the Next Billion Mobile Web Users

Usama Naseer
Brown University

Theophilus A. Benson
Brown University

Ravi Netravali
UCLA

ABSTRACT

Users in developing regions still suffer from poor web performance, mainly due to their unique landscape of low-end devices. In this paper, we uncover a root cause of this suboptimal performance by cross-analyzing longitudinal resource (in particular, memory) profiles from a large social network, and the memory consumption of modern webpages in five regions. We discover that the primary culprit for hitting memory constraints is JavaScript execution which existing optimizations are ill-suited to alleviate. To handle this, we propose *WebMedic*, an approach that trades-off less critical functionality of a webpage to directly address memory and performance problems.

CCS CONCEPTS

• **Information systems** → Browsers; • **Networks** → *Network performance modeling*.

KEYWORDS

Mobile web, Memory performance, Web optimizations.

ACM Reference Format:
Usama Naseer, Theophilus A. Benson, and Ravi Netravali. 2021. *WebMedic*: Disentangling the Memory–Functionality Tension for the Next Billion Mobile Web Users. In *The 22nd International Workshop on Mobile Computing Systems and Applications (HotMobile '21), February 24–26, 2021, Virtual, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3446382.3448652

1 INTRODUCTION

The mobile web is crucial for enabling upward social mobility within developing (e.g., Africa) [25] and transitioning (e.g., Asia) regions [19]. Yet despite the growing importance of the mobile web, anecdotal evidence suggests that memory constraints of low-end devices, which are popular within these regions, limit both the performance and usability of web pages. Surprisingly, few details are known about the memory profiles of mobile devices in these regions, and their interactions with their local, region-specific websites.

The focus of this paper is on understanding the regional characteristics of mobile devices and mobile web performance. However, unlike prior studies which have focused on understanding the network [1, 30] or CPU limitations [12, 26] of low-end phones, our focus is primarily on memory (RAM) resources and constraints. Further, in contrast to recent small-scale explorations of mobile characteristics in developing regions [12], our study is performed at a global scale and over 4.3 years. Our findings are that:

- Our study of the data collected from a global social network from 2015-2019 shows that the prevailing assumption of shrinking memory gap between devices in developing and developed regions due to the influx of cheap but resource-rich devices, does not hold true. In fact, the gap is *widening*, with a 2× difference at the median currently.
- Despite the disparity in resources, websites specific to developing regions consume more memory than those in developed regions. The corresponding memory requirements exceed the capabilities of devices in developing regions, resulting in poor performance and browser crashes.
- Current mobile web optimizations are limited in their ability to alleviate memory constraints, either due to adoption challenges (e.g., custom OSes [5], page rewriting strategies [20]) or an inability to effectively balance memory consumption and page utility (e.g., offload proxies [30]).
- Our analysis of 250 websites from 5 geographic regions shows that JavaScript (JS) execution (and the cascading effects, e.g., rendering) are the primary contributors to high memory usage. We further highlight the key aspects of JS and browsers that contribute to memory.
- We show that semantic content alteration is crucial towards alleviating memory overheads, but obvious strategies, e.g., removing ads or trackers, are not effective (no memory improvement at the median, ∼10% at the p75). Removing non-origin JS is more effective (12% at the median, 35% at the p75) but removes up to 61% of the interact-able elements.

Focusing on the intricate relationship between a website's JS, memory and user-centered utility (page functionality and appearance), we investigate the design points for automatically transforming websites to alleviate memory constraints. In particular, we focus on designs that free the website developers from having to explicitly reason about and tackle the nuances and implications of the variations between different classes of low-end phones and their memory constraints. Our work is orthogonal to existing efforts, e.g., AMP [20], which tackle an identical problem by forcing web developers to rewrite their websites within the AMP context. We argue that this places a burden on developers and propose an alternative approach for an automated, on-the-fly transformation of webpages. We present a strawman approach, *WebMedic*, that addresses memory constraints by intelligently removing JS function(s) — called a surgery — while ensuring that the impact to crucial user-desired functionality is minimized: in short, *WebMedic* trades off functionality for stability and performance. *WebMedic* acts as a web-server extension and leverages user device information, captured through User-Agent and JS APIs [39], to determine the optimal transformations that maximize utility while minimizing memory usage. Our evaluation shows that the surgery

HotMobile '21, February 24–26, 2021, Virtual, United Kingdom
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8323-3/21/02...$15.00
https://doi.org/10.1145/3446382.3448652

impact is very website-specific and JS-memory can be reduced by up to 80%, while giving up only 20% of the functionality, for a number of developing region news websites.

Our contributions are:

- We present, to the best of our knowledge, the largest longitudinal study of Android phone RAM resources across four years and four months (2015-April 2019) across the globe.
- We present a methodology for analyzing and dissecting the memory footprint of JS and analyze 250 regional pages across developing and developed regions to understand the critical aspects of sites concerning memory usage.
- We explore the feasibility of multiple optimization techniques and discuss their scope and limitations for alleviating memory overheads.
- We propose a data-driven system for optimizing memory usage and explore the trade-off between memory and functionality using the proposed approach.

2 BACKGROUND

In this section, we provide a brief overview of Android's memory management and pageload process in a browser.

Android Memory Management: Unlike traditional Linux, Android does not use virtual memory techniques (e.g., swapping) to ease memory pressure, and instead relies on Garbage Collection (GC) and, in extreme situations, terminating processes. GC operations have two broad implications on the page load performance: (i) GC incurs compute overhead and thus reduces CPU cycles available to page loading process, (ii) GC may momentarily pause applications, e.g., browser, to reclaim space and thus inflates page load time. When GC fails to reclaim required memory, Android's Out-of-Memory (OOM) killer process free up memory by identifying and killing a process, based on pre-configured priorities.

Browser: Chrome loads a webpage by parsing the HTML, fetching additional objects (e.g., images, CSS or JS), parsing the objects, updating the Document Object Model (*DOM*) [10], and rendering the DOM. DOM is an object-oriented representation of a webpage, containing its structure, content and styles, and JS interacts with DOM to modify the internal structure.

3 UNDERSTANDING THE MEMORY-WEB RELATIONSHIP

In this section, we measure the evolution of device RAM over the 2015-2019 period, highlight the resource constraints and their implications on the web, and conclude by exploring the efficacy of several existing and conventional optimizations.

3.1 Large Scale Study of Device Memory Gap

We begin by analyzing the memory resources of mobile devices used to access *BigContent*, one of the largest online social networks in the world. *BigContent* runs a website and mobile applications that provide users with the ability to post content on a "wall", interact with other users' content, and send messages. Our dataset covers *BigContent*'s Android application users from the 5 most populous continents and spans from January 2015 to April 2019, with the total unique user devices in the order of 100s of millions in each year; taken together, the dataset presents a microcosm of the global mobile Internet users. The dataset consists of Android device characteristics, e.g., device model, total device memory, captured by *BigContent*'s mobile application using standard Android OS APIs [13, 14]. The users in the dataset explicitly agreed to share their location and the logs do not report any private (or identity-revealing) information about a user or their activities within the application. Based on classification from [36], we cluster all countries into **Developing** (primarily African, Latin American, and Asian countries), **Transitioning Regions** (Eastern and Central Europe, and some Asian countries), and **Developed** (North America, Western Europe, Australia, Japan, New Zealand) regions. In addition, we classify devices based on their 2019 RAM resources: (i) **low-end** (0.5-1 GB), (ii) **middle-end** (1.5-3 GB), and (iii) **high-end** (4-8 GB).

Figure 1 presents the evolution of device RAM between 2015 and 2019, with each box representing the distribution for the given region. Although overall we observe a consistent increase in RAM over this timespan, the rate of increase is not uniform across regions. More specifically, *the divergence in memory resources across regions has steadily increased over time*. Median RAM values were comparable (around 1 GB) across regions in 2015 but the landscape is quite different in 2019: devices in developing regions have 1-2 GB less RAM than their counterparts in developed regions at the median and p75. Even in 2019, low-end devices have 3× more market share in developing regions (57% are low-end) than in developed regions (20% are low-end). Comparing the 10 most popular devices per region highlights an economic factor behind these trends. High-end Samsung Galaxy variants (e.g., Note 9, S9) with 4-6 GB RAM and $500-900 price take nine of the top 10 places in North America. Whereas in Africa, a mix of far cheaper phones ($100-250) from Samsung, Huawei, Infinix, Condor, and OPPO with 0.5-3 GB RAM make up the top 10.

Digging deeper into the datasets reveals that there is a clear distinction in the life-cycle of 512MB devices. The market share for these devices has reduced at a far slower rate in developing regions (38% reduction over the 4.3 years) compared to transitioning (57%) and developed (83%) regions. For context, these devices still account for 20% of the market share in developing regions (<2% in developed regions).

3.2 Implications on the Mobile Web

Reports suggest that Internet users in developing regions use mobile phones as the primary gateway to the web [7]. A user-study from Nielsen Norman Group shows that due to storage and data cost concerns, users in India prefer accessing web through the browser, provided that the quality of the web service is good [17]. This indicates that there is an incentive for content providers to optimize the web experience for low-end devices. Next, we present results for memory utilization of 250 popular websites across multiple regions to understand the implications on web.

Experiment setup: Our testbed comprises of three phones: LG G5 (high-end, 4GB RAM, 2x2.15, 2x1.6 GHz CPU, 5.3in screen), QMobile Q Infinity B (low-end, 1GB RAM, Quad-core 4x1.2 GHz CPU, 4.95in screen), and Nokia 2 (low-end, 4x1.3 GHz, 5in screen). The devices run the vendor's Android OS distribution and ADB is used to automate any activity (e.g., loading website). Chrome v69.0 is used for the measurements, with no background applications or tabs. Each measurement is repeated thrice, and Chrome is terminated and its application cache is cleared before each trial. To ensure reproducibility, we use *mitmproxy* [11](a popular record-and-replay

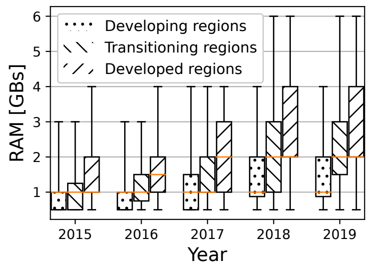

Figure 1: Device memory evolution.

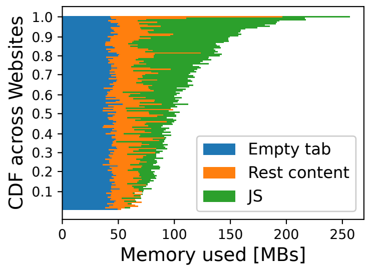

Figure 2: Memory usage of 250 websites.

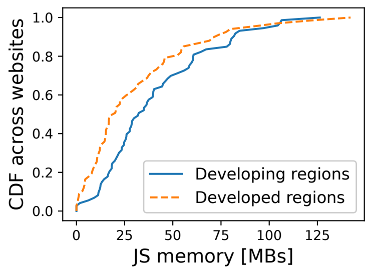

Figure 3: JS memory for regional websites.

tool) to record regional webpages and later replay them to devices. The last-mile link (i.e., the device access link) is set to 100Mbps, 20ms RTT during replay. We measure Alexa top 50 websites from 5 countries: Egypt, Nigeria, Pakistan, U.S., and U.K. (250 websites in total). The five countries provide coverage for websites exclusive to developing regions (31% of the 250 websites unique to Egypt, Nigeria, Pakistan), as well as globally popular ones. We use a VPN while recording websites to ensure that the correct regional version is recorded (e.g., google.com vs. google.com.pk may be different).

Measuring memory: We quantify the memory footprint of a website by measuring the increase in device's memory usage, captured through *dumpsys* [15] which measures memory usage as the difference of *MemAvailable* from *TotalMemory*. We take three snapshots: first, before loading Chrome; second, after opening an empty tab; and the third is the peak memory usage in the next 60 seconds[1]. The memory footprint of a website is calculated as the difference between the third and the first snapshot[2]. We calculate JS's contribution to the memory by measuring the memory footprint of a website running with and without the JS code, and taking a difference between the two. For removing the js code, a custom Python script at *mitmproxy* empties the *.js* file and <*script*> code from the response. We also classify a webpage's JS code based on their iFrame source URL and classify them as origin, non-origin, advertisement or tracker (using publicly available lists).

Measuring utility: We measure two user-centric utility metrics for the webpage's JS: (i) *functionality*, e.g., event listener attached to a button by JS, and (ii) *appearance*. The number of *event listeners* added by JS to the DOM quantifies the former (measured using *Timeline Profile* [21]), while taking a screen-shot of the initial view-port of a fully rendered page and generating a fingerprint – Perceptual Hash or *pHash* [24] – measures the latter. Comparing the two metrics for the original vs modified webpage (i.e., all or a subset of JS removed), quantifies the loss of functionality and the change in page's appearance[3] respectively, i.e., the utility impact of removing the corresponding JS. We selected pHash over alternative visual hashing algorithms because of its computational simplicity and success in web use cases [4].

Measurement results: Figure 2 plots memory breakdown for the 250 websites. While the median website uses 100MB, websites take as much as 250MB at the tail. Most of the memory-heavy (>150MB) websites are news; elwatannews.com, elbalad.news, youm7.com, dailymail.co.uk, premiumtimesng.com, to name a few. One key observation is that JavaScript (JS) by far dominates the "other" objects (e.g., HTML, CSS) on a page (up to 128MB of memory at the tail). It is important to note that this memory may not be altogether allocated *directly* by JS (e.g., allocations by the V8 engine), but other components of the browser, e.g., when a JS modifies the CSS style of a DOM node, it leads the browser to recalculate styles for DOM, render the changes, store the rasterized resources etc., and these operations have memory consequences of their own. This complexity also makes pinning down the actual cause of JS memory quite challenging, e.g., traditional tools [6, 21] focus on JS-heap (responsible for storing JS arrays, strings etc) to investigate the memory problems and we observe the JS-heap to significantly underestimate the total JS memory: overall JS memory is 3X the JS-heap for the median (over 6X for tail website).

Figure 3 breaks-down the JS's memory footprint on a regional basis. We consistently observe a higher footprint for the websites in developing regions. Though developing regions abundantly carry memory-constrained devices, we do not observe this trend to reflect in their websites' memory footprints, indicating a lack of focus on optimizing memory and catering for their unique device landscape. We observe developing region websites to contain heavier DOM at median and tail (higher number of nodes) and a higher number of ads/tracker iFrames (2 vs 5 for the tail website in both regions), though we do observe that ads/tracker are not the sole reason for the memory usage discrepancy (§ 3.3).

Memory-intensive attributes of JS: Next, we perform a brute-force search for each website (cut each JS file and <script> one by one as further explained in § 5) and perform the Pearson's Correlation Test [3] between JS file's memory and interactions with DOM/browser to understand the dynamics on an individual JS file level. The test reports a coefficient between -1 to 1, with ≥0.6 value indicating moderate to strong positive correlation [3].

- We observe that the number of event listeners and nodes contributed by a JS to the DOM correlates moderately to strongly with its memory footprint (Pearson coefficient of 0.6 and 0.54 respectively with p<0.000). This indicates that the more functionality or elements (e.g., "div") that a JS adds to the page, the more likely the JS is to have a heavier memory footprint.
- We further analyze the JS's interactions with the browser through the use of *Timeline Profile* that captures the internal Chrome events triggered as a result of executing a JS. We observe that the memory-heavy JS are more likely to trigger frequent updates to the visual appearance of the webpage, with the top 3 events being *blink.animations, updateLayoutTree, styleRecalculation* (Pearson coefficient of 0.62, 0.57, 0.57 respectively with p<0.000). We observe a similar trend

[1]Tail onLoad time is 31.5s and 60s gives room for JS execution to finish.
[2]The difference of 3rd and 2nd underestimates the total footprint as it ignores memory allocated to the empty tab beforehand.
[3]*hamming distance* [23] computes how much similar two pHashes are.

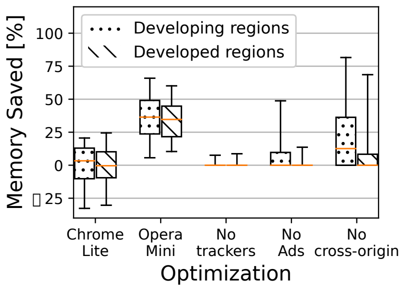

Figure 4: Current and custom optimizations.

when analyzing HTML DOM API calls, with the CSS style related API calls among the top contributors.

Implications: In the context of the developing regions where devices with 512 MB are common, we observe that these extreme websites are significantly problematic. Up to 48% of the available memory is allocated to the native OS (measured for Android 7.1), i.e., without any foreground or background apps. These memory-heavy websites (and the OS together) can force the device to use over 90% memory. In the best case, the page load performance is poor [12]. One reason is the added stalls for more frequent garbage collection. We measure that at the median, GC paused the browser for a 2.6× longer duration (472ms vs 181ms) on a low-end vs mid-end device for the Alexa top 50 US pages (measured through *logcat*). Total reported GC time (pause plus background GC) differed by 4.8s at the median. In the worst case, page loads can overwhelm the available memory and such contention can crash the browser/tab itself via OOM killer. We found the browser to consistently crash when more than 90% of the available device memory was consumed. We also observed that loading heavy (regional) news pages led to crashing all of the background tabs; even for a device with 1 GB of RAM.

Takeaway: The widening memory gap between the regions poses a challenge, motivating web optimizations for low-end devices. In essence, we expect a bifurcation of the web ecosystem, evident from efforts by Uber, Facebook, and Google to redesign their mobile platforms [5, 16, 18].

3.3 Web Optimizations and Memory Impact

Next, we evaluate the efficacy of several currently deployed, as well as, some conventional content-alteration optimizations.

OS Optimizations: Android Go [5] is designed for low-end phones and consumes fewer resources than default Android distributions. Leveraging our testbed, we loaded the same set of regional pages on two Nokia devices with the same RAM and screen size; one using Android 7.1 and the other Android Go. We measured their eventual memory footprint to be nearly the same on both devices – the median ratio of memory with and without Android Go was 0.96. Though Android Go does free memory resources for mobile web browsing (due to smaller OS footprint), relying on OS updates to tackle memory constraints is practically challenging due to network costs (hindering the download of updates) [17, 35] and out-of-date device configurations (complicating the application of updates).

Browser and Proxy-Based Optimizations: We focus on two systems: 1) Lite Mode in Chrome that compresses web objects using Google's Flywheel proxy [1], and 2) Opera Mini [30] that offloads the page load and JS execution to a proxy and requires clients to only render a simplified version. We use both live systems to load pages in our testbed and compare pageloads with and without the optimizations.

Figure 4 plots the memory savings. We do not observe any significant memory impact of Lite Mode optimization's (e.g., compression, minification) across the websites. In contrast, Opera Mini's offload does result in significant savings, with up to 35-37% improvement at median across both regions. However, they come at a cost: JS offload can degrade the webpage's interactivity. We also observe the webpage's appearance to significantly degrade. Across all webpages, we observe median, p75 and p95 pHash score differences of 24, 30, and 36 units, respectively. For reference, these numbers for back-to-back loads of our pages without a proxy are 0, 8, and 26, suggesting that Opera's proxies significantly alter visual page content beyond intentional variance (e.g., different ads). We further observe more severe cases with 7% of African websites resulted in missing content or page formatting errors that made the pages nearly unusable. Interestingly, several websites prompted the user to use a proxy-less browser or refused to load due to built-in AdBlock.

Specialized solutions: A number of solutions have been proposed in the recent years. Accelerated Mobile Pages (AMP) [20], a recent initiative by Google, aims to address CPU, bandwidth, and memory constraints by providing web developers with an alternative ecosystem for developing lightweight pages. In AMP, pages are developed using an alternative and lightweight image format and JS is forbidden. Taken together, the constraints ensure that the webpages are lightweight. Similarly, *Adaptive Loading* [31] provides a framework to optimize memory for React applications by selectively loading website functionality based on device resources. *Client Hints* [38] provides an interface through which browsers can communicate their resource limitations with the servers as an indicator to optimize the content. [34] discusses heuristics for optimizing pageloads for low-end devices. While these solutions have potential, they are either too specialized for specific frameworks (e.g., React) and assume that the developing region websites are using these frameworks, or expect the developer to rewrite/redesign their websites in a specific way (e.g., AMP). These limitations can introduce barriers for adoption and the ideal solution should optimize the existing websites without any additional work/changes to the website from the developer.

Custom solution #1: Remove ads/trackers The most apparent content-altering transformation is to eliminate ads and trackers related JS, both of which have proven time and time again to reduce performance [33]. In Figure 4, we dissect JS-memory footprint for websites to highlight the memory used by ads and trackers. We observe that across both regions, ads and trackers consume an insignificant amount of memory for most websites: in fact, there's no benefit to using eliminating ads/trackers at the median.

Custom solution #2: Remove cross-origin content Next, we focus on the impact of cutting cross-origin content, assuming that crucial content is hosted on the origin, and the auxiliary, non-critical content is hosted on non-origin sites. In Figure 4, we observe that cutting cross-origin content does provide significant reductions. However, we note that these changes significantly impact the utility or usability of the webpage (8% functionality loss at the median, 61% at the tail). One of the key reasons why the memory reductions translate into a loss of utility is because many websites host content on third-party CDNs, e.g., fonts and third-party JS libraries (e.g., JQuery).

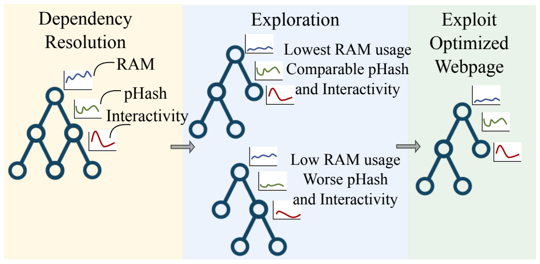

Figure 5: WebMedic workflow

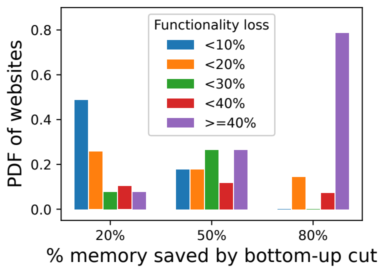

Figure 6: Memory and functionality trade-off.

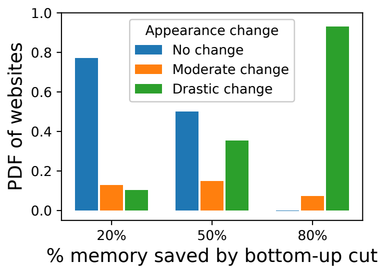

Figure 7: Memory and appearance trade-off.

Takeaway: While existing optimizations show potential for some websites; they are too specialized, force the developers to use specific frameworks or re-implement the website, or suffer from utility degradation. The ideal solution should free the developers from curating a specialized version and also preserve the useful utility of the website.

4 *WEBMEDIC*

Given the potential of semantic content alteration, we present *WebMedic*, a data-driven framework that optimizes page loads for low-end mobile phones by judiciously removing JS functions — performing *surgery* on web pages — to ensure that page load does not crash the browser and has minimal memory footprint. Intuitively, *WebMedic* trades-off functionality for performance and, in making this trade-off, aims to minimize the impact of lost functionality on end-users by removing "less-useful" functionality. We believe that a stripped version of the webpage is preferable to one that loads slow or crashes on low-end phones (i.e., no functionality at all).

Figure 5 presents the high-level workflow of *WebMedic*. Working as a server extension, *WebMedic* creates an abstract representation of the page (i.e., a DAG to capture object dependencies), performs "surgery" on this abstract representation, applies changes from the abstract representation to the concrete webpage, and returns the webpage to the user. The design of *WebMedic* introduces several interesting research questions:

(i) Who performs the surgery? The surgery can be performed either at the server or the client side. Though client-side transformation preserves privacy, it requires additional client resources – a feat that may be impossible for low-end phones. Our system adopts a server-side approach, thus freeing the client of such burdens, improving network traffic as the cut JS is not sent over the network, and enabling large-scale learning. Server-side is the perfect venue for the surgery, given that the servers today can capture unique client-side characteristics through *User-Agent* and JS APIs [39], and current CDNs have visibility into their HTTPS traffic since they are configured with the required SSL/TLS certificates.

(ii) What is a surgery? Given a website, *WebMedic* needs to determine the minimal subset of memory-heavy JS functions that can be removed without drastically impacting the functionality and appearance. This requires profiling the memory and utility aspects of JS for each website, as well as the complex interdependencies within the web objects. Naively cutting JS can break the webpage, e.g., removing a JS block may remove the state (e.g., shared array) that a subsequent function may use and thus raise errors. Fortunately, existing works [27] capture data-flow dependencies between web objects in form of *dependency DAG* (G). Our key contribution is to enrich existing work with annotations expressing memory and utility implications of each JS function (node), thus enabling a dependency graph to incorporate both utility and performance.

(iii) How is a surgery evaluated? The annotations in G serve as the weights within the graph and allow *WebMedic* to analytically reason about the impact of surgeries on both performance and functionality, while freeing *WebMedic* from having to reason about the webpage's complexity. However, evaluating the impact on memory is challenging at scale: device memory usage is not accessible to JS scope (recall that memory usage is measured from OS). We advocate for a hybrid approach: (i) an offline phase measures the JS impact on memory and models it based on a set of predictable features that are accessible to JS (§ 3) and thus can be communicated to the server-side, (ii) an online phase where *WebMedic* on the server-side groups similar devices together based on their hardware resources, builds a distinct G for each website and device group, and uses the predictable features for assigning memory to G's nodes. Though the online phase can measure high-level functionality (event listeners) for the webpage, capturing important user-desired functionality is a challenge. Fortunately, existing works [22, 29] rank the importance of webpage aspects (e.g., above-the-fold event listeners and objects) from the user's perspective and we plan to leverage these works for assigning weights to utility in G. We further plan to incorporate developer-defined heuristics (e.g., higher weights to forms) to further enrich annotations.

We plan to investigate the design of a contextual-multi-armed bandit which uses the device class as the context to determine the types of surgeries to explore or exploit. To capture user preferences and rewards, we further plan to create a link that allows a user to visit the original version, if the transformed version is inadequate – similar to the desktop version option in mobile browsers.

(iv) How are surgeries performed? Given the dependency graph G, deciding the appropriate JS to cut can be formulated as a linear programming problem that determines the minimal cut across the G that maximizes utility and usability (i.e., minimizes the probability that the user will opt for the original version) and while ensuring that the website fits within the device's memory constraints. Such an optimization problem is akin to the set-covering problem which is NP-complete. We plan to investigate the design of an algorithm which leverages a unique feature of the domain, namely that the dependency graph captures page load order and cutting a non-leave node automatically cuts all its children. Thus, we explore cutting in a bottom-up fashion. The formulation can be solved directly with a bin packing heuristic.

5 PRELIMINARY EVALUATION

Next, we present a preliminary evaluation of our approach. We perform brute-force surgeries for the 250 websites where starting from the bottom of G for each website, JS node (and its children) are cut and the memory and utility impact is measured (following the methodology in § 3). The process is repeated until all the nodes (i.e., all the JS code) are cut. As websites can have 100s of JS functions or files, this brute-force exploration can be painstakingly slow (8 hours for a website with 35 JS files). To speed up the process, we limit 10 cuts to each website by greedily grouping the JS neighboring nodes and cutting them at once, such that each cut removes an equivalent size of code.

Figures 6 and 7 plot the impact on functionality and appearance, respectively, for a cut that leads to a specific percentage saving in JS memory footprint. We observe that for a subset of websites (~18%), there exists a cut that improves JS-memory by 80%, without a significant loss of functionality (<20%). We observe that these websites comprise mostly news websites from developing regions, e.g., express.pk, premiumtimesng, elbalad, elwatannews, worldometers etc., and have multiple JS with 10-15% memory contributions. However for a majority of the sites, only 20% of the memory can be saved if there is a desire to maintain usability (i.e., preserve 80% of the functionality). Similarly, Figure 7 plots that impact on visual appearance and we observe little to no change in appearance for these websites. Producing these results involved a manual step where a human subject analyzed the screen-shots to define a threshold on pHash distance for the three categories in Figure 7. Following the guidelines for setting thresholds through empirical analysis [24], we identified <0.19, <0.27 and ≥0.27 as the threshold bounds for the categories respectively, with low false positives and false negatives (<0.05). Taken together, we observe that the functionality aspect of utility is more sensitive to JS cuts, with up to 60% of the websites showing no or moderate change in appearance for cuts saving up to 50% of memory.

6 RELATED WORK

Mobile Web Optimizations: Prior approaches on improving mobile web page loads have mostly focused on alleviating 1) network bottlenecks by reducing the number of serial round trips [1, 27, 35] or shifting costly round trips to proxy servers [30], and 2) computation bottlenecks by preprocessing pages [28] or rewriting them to use restricted HTML, JS, and CSS [20]. In contrast, our goal is to understand how device memory (not network and CPU) varies across global regions, and how page loads in these regions consume memory resources. More recently, JSCleaner [9] removes non-critical JS from web pages to improve performance. Our results suggest that removing non-critical JS from the webpage is not enough to tackle memory constraints in developing regions. Further, as JSCleaner [9] aims to keep the page structure/functionality the same, we expect the memory overheads associated with Chrome events to persist.

Measurement Studies: Recent work [2, 12, 26] explored energy usage and the impact of (low-end) device and network resources on QoE. Our work differs from these studies in two key ways. First, our study is the first (to our knowledge) large-scale analysis of device characteristics across multiple regions in the world. Second, we focus on memory availability and constraints, with an emphasis on web browsing. The focus on region-specific experiments on mobile differentiates us from prior work [2, 12], and enables us to make region-specific observations about the implications of device memory constraints.

7 DISCUSSION AND FUTURE WORK

Below, we briefly discuss some open challenges:

User-centric Metrics: Our metrics currently treat all visual and interactive elements equally; however, fundamentally, users place different utility on different parts of the page. As part of future work, we plan to decompose the webpage into several regions and assign different weights to different regions based on user preferences extracted via user studies. Building on existing approaches that capture human perception, either through crowd-sourced experiments [37] or user eye-tracking [22, 32], we plan to design user studies for understanding web pages and tailor them to our domain.

Incorporating User Preferences Our system assumes that every user is willing to make the tradeoff between performance and correctness. However, in specific scenarios, a user may be unwilling to make this tradeoff either because our metric inaccurately captures their internal preferences or because our system's performance savings may be too minimal. We plan to address this by providing the user with a method to load the full version of the page, i.e., via a link, or by providing users with a means to specify a minimum memory savings threshold for *WebMedic* and altering *WebMedic* to only transform a page if this threshold is met.

Tackling Additional Resource Constraints: This paper currently focuses on memory; however, JavaScript also imposes non-trivial CPU, energy, and networking overheads. As part of future work, we plan to extend *WebMedic*'s philosophy to optimize these dimensions by extending on recent works [8, 26] and designing new tools to capture each dimension's resource overheads accurately. Extending the set of target metrics is expected to spring new research challenges, particularly related to the representativeness of the offline profiles for the host of diverse user devices, and root-cause analysis of the aspects of JavaScript that contributes to a specific resource.

8 CONCLUSION

There has been a significant effort to improve mobile web performance for users in developing regions. However, as a community, we lack holistic studies on the user device characteristics in those regions and the set of optimizations available to them. In this work, we address this void by leveraging a dataset of mobile phone resource profiles from *BigContent*, a global social network, to study the memory usage of regional web pages. Overall, we find significant discrepancies in the memory resources for the devices present in the different global regions, and discover the negative impacts of the memory constraints for the page loads common in those regions. Motivated by the ineffectiveness of existing memory optimizations, we present *WebMedic*, a system that carefully cuts page content in a way that balances page functionality with memory usage. Our work motivates content providers to tailor their services to fit the needs of the next billion users by presenting an approach to automatically address memory constraints.

9 ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Chulhong Min, for their invaluable comments. This work is supported by NSF grants CNS-1814285 and CNS-1943621.

REFERENCES

[1] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. 2015. Flywheel: GoogleâĂŹs data compression proxy for the mobile web. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 367–380.

[2] Sohaib Ahmad, Abdul Lateef Haamid, Zafar Ayyub Qazi, Zhenyu Zhou, Theophilus Benson, and Ihsan Ayyub Qazi. 2016. A view from the other side: Understanding mobile phone characteristics in the developing world. In *Proceedings of the 2016 Internet Measurement Conference*. 319–325.

[3] Haldun Akoglu. 2018. User's guide to correlation coefficients. *Turkish journal of emergency medicine* 18, 3 (2018), 91–93.

[4] Ashok Anand, Aditya Akella, Vyas Sekar, and Srinivasan Seshan. 2010. A case for information-bound referencing. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 4.

[5] android.com. 2020. Android (Go edition). Powering entry-level devices. (2020). Retrieved 2020 from bit.ly/369t5SA

[6] Kayce Basques. 2020. Fix Memory Problems. (2020). bit.ly/3iK1o7Y

[7] Ananya Bhattacharya. 2017. Internet use in India proves desktops are only for Westerners. (2017). Retrieved 2020 from bit.ly/34TDu4J

[8] Yi Cao, Javad Nejati, Muhammad Wajahat, Aruna Balasubramanian, and Anshul Gandhi. 2017. Deconstructing the energy consumption of the mobile page load. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 1 (2017), 1–25.

[9] Moumena Chaqfeh, Yasir Zaki, Jacinta Hu, and Lakshmi Subramanian. 2020. JSCleaner: De-Cluttering Mobile Webpages Through JavaScript Cleanup. In *Proceedings of The Web Conference 2020*. 763–773.

[10] MDN contributors. 2020. Introduction to DOM. (2020). Retrieved 2020 from mzl.la/2Ibauwv

[11] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. 2010. mitmproxy: A free and open source interactive HTTPS proxy. (2010). mitmproxy.org [Version 6.0].

[12] Mallesham Dasari, Santiago Vargas, Arani Bhattacharya, Aruna Balasubramanian, Samir R Das, and Michael Ferdman. 2018. Impact of device performance on mobile internet QoE. In *Proceedings of the Internet Measurement Conference 2018*. ACM, 1–7.

[13] Android Developers. 2020. ActivityManager.MemoryInfo. (2020). Retrieved 2021 from bit.ly/2Ml8h3o

[14] Android Developers. 2020. Build. (2020). Retrieved 2021 from bit.ly/3oaUOIB

[15] Android Developers. 2020. Dumpsys. (2020). Retrieved 2020 from bit.ly/32iQtv7

[16] Facebook. 2020. Facebook Lite. (2020). Retrieved 2020 from facebook.com/lite/

[17] Nielsen Norman Group. 2016. Mobile User Behavior in India. (2016). https://bit.ly/2HPOwPO

[18] Uber Technologies Inc. 2020. Uber Lite. (2020). Retrieved 2020 from ubr.to/3sRTjTp

[19] Muhammad Ittefaq and Azhar Iqbal. 2018. Digitization of health in Pakistan. *Digital health* (2018).

[20] Byungjin Jun, Fabián E Bustamante, Sung Yoon Whang, and Zachary S Bischof. 2019. AMP up your Mobile Web Experience: Characterizing the Impact of Google's Accelerated Mobile Project. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–14.

[21] Meggin Kearney and Kayce Basques. 2020. Analyze Runtime Performance. (2020). Retrieved 2020 from bit.ly/36cpbIu

[22] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. 2017. Improving User Perceived Page Load Times Using Gaze.. In *NSDI*. 545–559.

[23] Evan Klinger and David Starkweather. 2013. Design and Validation - Hamming distance. (2013). Retrieved 2020 from phash.org/docs/design.html

[24] Evan Klinger and David Starkweather. 2013. pHash, The open source perceptual hash library. (2013). Retrieved 2020 from phash.org

[25] Murithi Mutiga and Zoe Flood. 2016. Africa calling: mobile phone revolution to transform democracies. (2016). Retrieved 2020 from bit.ly/2VXSuKX

[26] Javad Nejati and Aruna Balasubramanian. 2016. An in-depth study of mobile browser performance. In *Proceedings of the 25th International Conference on World Wide Web*. 1305–1315.

[27] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster page loads using fine-grained dependency tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association.

[28] Ravi Netravali and James Mickens. 2018. Prophecy: Accelerating mobile page loads using final-state write logs. In *NSDI*.

[29] Ravi Netravali, Vikram Nathan, James Mickens, and Hari Balakrishnan. 2018. Vesper: Measuring time-to-interactivity for web pages. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 217–231.

[30] Opera. 2020. Opera Mini. (2020). Retrieved 2020 from opera.com/mobile/mini

[31] Addy Osmani and Anton Karlovskiy. 2019. React Adaptive Loading Hooks and Utilities. (2019). Retrieved 2020 from bit.ly/34RX6Gb

[32] Alexandra Papoutsaki, Patsorn Sangkloy, James Laskey, Nediyana Daskalova, Jeff Huang, and James Hays. 2016. WebGazer: Scalable Webcam Eye Tracking Using User Interactions. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI, 3839–3845.

[33] Behnam Pourghassemi, Jordan Bonecutter, Zhou Li, and Aparna Chandramowlishwaran. 2020. adPerf: Characterizing the Performance of Third-party Ads. *arXiv preprint arXiv:2002.05666* (2020).

[34] Sarah S. 2020. Web performance risks: spotlight on JavaScript vs. low-end mobiles. (2020). Retrieved 2020 from bit.ly/2I2Ge6W

[35] Shailendra Singh, Harsha V Madhyastha, Srikanth V Krishnamurthy, and Ramesh Govindan. 2015. Flexiweb: Network-aware compaction for accelerating mobile web transfers. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 604–616.

[36] UN. 2014. Country classification - World Economic Situation and Prospects Report. (2014). Retrieved 2020 from bit.ly/3mReb9o

[37] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. 2016. EYEORG: A Platform For Crowdsourcing Web Quality Of Experience Measurements. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. Association for Computing Machinery, New York, NY, USA, 399âĂŞ412. https://doi.org/10.1145/2999572.2999590

[38] Jeremy Wagner. 2020. Client Hints. (2020). Retrieved 2020 from bit.ly/327QHVQ

[39] Philip Walton. 2017. Device Memory API. (2017). Retrieved 2020 from bit.ly/2IfhYy9