mBenchLab: Measuring QoE of Web Applications using mobile devices

Emmanuel Cecchet, Robert Sims, Xin He, Prashant Shenoy
University of Massachusetts Amherst, CS department
Amhert MA, USA
{cecchet,rsims,xhe,shenoy}@cs.umass.edu

Abstract— **In this paper, we present mBenchLab, a software infrastructure to measure the Quality of Experience (QoE) on tablet and smartphones accessing cloud hosted Web services. mBenchLab does not rely on emulation but uses real phones and tablets with their original software stack and communication interfaces for performance evaluation. We have used mBenchLab to measure the QoE of well-known web sites on various devices (Android tablets and smartphones) and networks (Wifi, 3G, 4G). We present our experimental results and lessons learned measuring QoE on mobile devices with mBenchLab. In our QoE analysis, we were able to discover a new bug in a very popular smartphone that impacts both performance and data usage. We have also made the entire mBenchLab software available as open source to the community to measure QoE on mobile devices that access cloud-hosted Web applications.**

Keywords—benchmarking, QoE, Android, tablets, smartphones, mobile web, cloud, web applications, web services.

I. Introduction

The Cloud has become the platform of choice to host modern Web applications. Cloud services provide elasticity, reliability and scalability at a low cost by virtualizing Web applications in large data centers. Concurrent with this server side transformation, the client side has begun to change dramatically as well by shifting from traditional desktops to smartphones and tablets. Wikipedia [9], the free online encyclopedia has a page view count approaching 20 billion page views per month with more than 13% for mobile traffic. From Dec 2011 to Dec 2012, overall Wikipedia traffic has increased 24%. This increase was dominated by mobile traffic which increased 77%, while non-mobile traffic increased only 18% [13]. On the hardware side, the latest forecast for 2013 predicts that tablet sales will surpass that of notebooks this year [10]. Unlike traditional PCs, these new devices not only have limited hardware resources (such as cpu, memory, storage, battery-power) but they also have access to a wider variety of networks (such as Wifi, 3G/4G, LTE). All these factors can significantly affect the user perceived quality of experience (QoE) of cloud hosted Web services.

We argue that the complexity of interactions with modern Web applications (WebApps) requires the use of real software stacks and network infrastructure that are too hard to simulate realistically. In this paper, we present mBenchLab, an open testbed to measure the QoE of Cloud hosted WebApps using real mobile devices. Unlike other benchmarking frameworks, mBenchLab does not rely on simulation or emulation. Instead we use (i) the original software stack of smartphones and tablets including their native Web browser, and (ii) the real network infrastructure. In our previous work [6], we focused on benchmarking server and network performance using desktop browsers on wired networks. In this paper, we present our results and lessons learned in developing mBenchLab for Android mobile devices to measure the QoE of cloud hosted Web Applications over wireless networks. We have used mBenchLab to measure the QoE of well-known services such as Amazon, Craigslist, or Wikipedia. To identify issues in QoE, we focus not just on overall latency or page load times—mBenchLab can also record fine grain events such as connection establishment, DNS resolution, network send/receive and browser rendering. These finer-level insights allow us to identify issues that users may face while browsing web sites from mobile devices. Mobility information is also recorded on devices equipped with GPS devices. By tracking the location of devices during an experiment, mBenchLab can help point out QoE issues related to geolocation.

All mBenchLab experiments are deployed from a Dashboard that is implemented as a Web Application. A system designer can deploy his or her own Dashboard and record experimental results from mobile devices into the database embedded in the WebApp. This data can then be exported or directly analyzed in the Dashboard to identify QoE issues. The Dashboard can also synchronize multiple devices to participate in the same experiment to generate a workload on a particular server or set of servers. This functionality can be helpful to measure the scalability of cloud services or the performance of wireless networks. In addition to targeting system designers who measure web application performance, mBenchLab is also designed for researchers who wish to use realistic mobile devices and networks to inject workloads into realistic web applications. To that end, we have reproduced the entire Wikipedia software stack that can be deployed in private or public clouds as a realistic server backend to mBenchLab mobile clients. We were also able to get access to Wikipedia access logs to reproduce realistic workloads in research experiments.

Our contributions are the following:

- We have built mBenchLab, a software infrastructure that can benchmark the QoE of cloud hosted Web applications with Android devices. We have also rebuilt the Wikipedia software stack to deploy it on-demand in private and public clouds. The mBenchLab Android application,

Dashboard and all the Wikipedia virtual machines for private clouds and Amazon EC2 are publicly available to the community to advance research in benchmarking cloud services with mobile devices.

- We perform detailed QoE measurements with Android smartphones and tablets on popular web sites and compared it to standard desktop browsers. We show that our monitoring overhead does not affect significantly the user perceived QoE. We measure how the device hardware/software combination influence the overall user perceived QoE. We also measure how QoE is correlated with mobile network performance on multiple continents.
- We show through a series of experiments how mBenchLab can identify QoE issues either related to the network, the Web service or the mobile device itself. We were able to find a previously undiscovered bug in the native browser of the popular Samsung S3 phone (40 million sold as of January 2013) that significantly affects performance and bandwidth usage on certain Web sites.

Our paper is structured as follows: section II gives an overview of the mBenchLab platform. Section III details the specifics of QoS measurements on Android devices. We present the results of our experimental evaluation in section IV. We discuss related work in section IV.F before concluding in section VI.

II. mBenchLab overview

mBenchLab is an open testbed for Web application benchmarking from mobile devices. The load is injected from real mobile devices that run the mBenchLab Mobile App and the experiments are coordinated through the mBenchLab Dashboard (section A). mBenchLab can be used with any existing Web application without any modification. For experiments where the user wants to control a real Web Application, we provide a Wikipedia implementation as virtual appliances that can be deployed on private or public clouds (section B).

A. *mBenchLab Dashboard and MobileApp*

Fig. 1 gives an overview of the mBenchLab components and how they interact to run an experiment. The mBenchLab Dashboard is the central component that deploys and controls experiments. It is built as a Java Web application that can be deployed in any Java Web container such as Apache Tomcat. The mBenchLab Dashboard provides a Web interface to interact with experimenters that want to create experiments using mobile devices. The Dashboard gives an overview of the devices currently connected, the experiments (created, running or completed) and the Web traces that are available for replay.

Web trace files are uploaded by the experimenter through a Web form and stored in the Dashboard database. The trace file includes the list of URLs to visit and encodes the values to fill Web forms as well as buttons to click. Every element is referred to by its id or name in the HTML page that is being accessed. The trace file can either be generated using a simple CSV format or by a traditional desktop browser by recording a browsing session using the standard HTTP Archive format (HAR) [11]. Each URL is assigned to a particular session that will be replayed by a single browser. An experiment that wants to use simultaneously 10 browsers must use a trace that contains at least 10 sessions.

mBenchLab does not deploy, configure or monitor any server-side software. There are a number of deployment frameworks available that users can use depending on their preferences (Gush, WADF, JEE, .Net, etc). If the experimenter deploys her own Web Application to be tested, monitoring the server software is also the choice and responsibility of the experimenter (Ganglia and fenxi are popular choices).

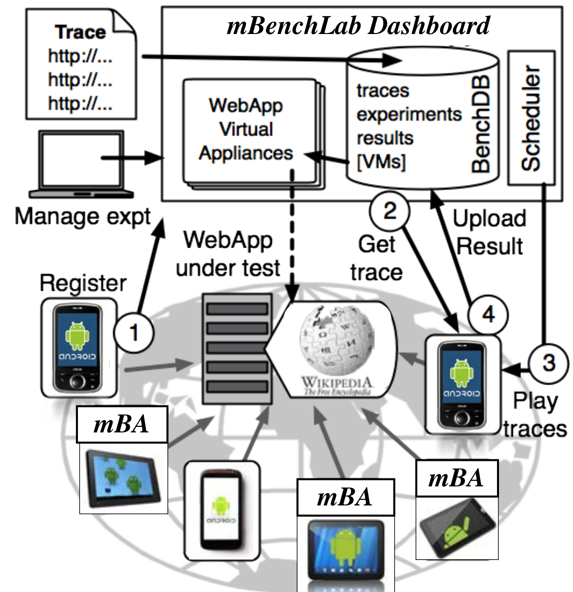

Fig. 1. mBenchLab experiment flow overview

Anyone can deploy an mBenchLab Dashboard and therefore build his or her own benchmark repository. An experiment defines what trace should be played and how. The user defines how many mobile devices should replay the sessions with eventual constraints (specific platform, version, location…). The experiment can start as soon as enough clients have registered to participate in the experiment. The Dashboard does not deploy the application on the mobile devices, rather it waits for mobile devices to connect and its scheduler assigns them to experiments.

The mBenchLab Android application (mBA) is a mobile application that starts and controls the native Web browser on the mobile device. On startup, the mBA connects the browser to an mBenchLab Dashboard (step 1 in Fig. 1). When the browser connects to the Dashboard, it provides details about the exact browser version and platform runtime it currently executes on as well as its IP address and GPS location (if enabled). If an experiment needs this device, the Dashboard redirects the mBA to a download page where it automatically gets the trace for the session it needs to play (step 2 in Fig. 1). The mBA stores the trace on the local storage and makes the Web browser regularly poll the Dashboard to get the experiment start time. There is no communication or clock

synchronization between mBAs, they just get a start time as a countdown in seconds from the Dashboard that informs them 'experiment starts in x seconds' through a Web form. The status of mobile devices is recorded by the Dashboard and stored in a database.

When the experiment start time has been reached, the mBA plays the trace through the Web browser monitoring each interaction (step 3 in Fig. 1). If Web forms have to be filled, the mBA uses the URL parameters stored in the trace to set the different fields, checkboxes, list selections, files to upload, etc. Text fields are replayed with a controllable rate that emulates human typing speed through the virtual keyboard of the device. The GPS location when the page was fetched as well as various QoE statistics (see section III for more details) are collected locally on the mobile device. The results are uploaded to the Dashboard at the end of the experiment (step 4 in Fig. 1). The mBA can also record the HTML pages and take screen snapshots of rendered pages to include in the Dashboard database. By parsing the HTML or comparing snapshot images with data from other runs, one can detect errors or rendering issues that affect user QoE. Fig. 2 shows an example of the experimental results stored in the Dashboard database.

Fig. 2. Partial screenshot of an experiment result in the mBenchLab dashboard

mBAs replay traces based on the timestamps contained in the traces. If an mBA happens to be late compared to the original timestamp, it will try to catch up by playing requests as fast as it can. A page loading timeout can also be set to prevent browsers from being stuck on particular pages.

B. *Wikipedia Virtual Appliances*

Wikipedia is available in 285 languages all relying on the same MediaWiki software stack and supervised by the Wikimedia foundation. Other satellite sites such as Wikibooks [8] (free content textbooks and annotated texts), WikiNews (free content news), Wiktionary (dictionary and thesaurus)… also rely on the same software. The server side is basically a PHP application with a number of extensions storing content in a database (MySQL by default).

We have created a Wikimedia server virtual machine that contains a preconfigured software stack including Apache 2.2.16, PHP 5.3.3, MediaWiki 1.16, as well as all necessary extensions necessary to run the Wikipedia family of web sites, including the Lucene search engine and multimedia content. We have also created a set of virtual machines with the database software and the content for particular wikis. Database dumps are freely available from the Wikimedia foundation in compressed XML format. TABLE I gives an overview of the databases we have made available as virtual appliances. Note that the English Wikipedia database (enwiki) is not available in public clouds due to its 5.5TB size and cost of storage. The English Wikibooks (enwikibooks) has a smaller number of articles but still a significant size as each article is larger than typical Wikipedia articles.

TABLE I VIRTUAL APPLICANCE DATABASES AVAILABLE (DUMPS FROM JANUARY TO MARCH 2010 TO MATCH OUR TRACES ENDING IN MARCH 2010)

Wiki name	**# of articles**	**Size on disk**	**Time to generate db and index**
dawiki	122 k	6.5 GB	14 hours
nlwiki	584 k	39 GB	3.25 days
frwiki	901 k	94 GB	7.3 days
enwiki	3.1 M	5.5 TB	>3 months
enwikibooks	32 k	4.3GB	10 hours

To prevent copyright issues with multimedia content, we use a multimedia content generator that produces images with the same specifications as the original content but with random pixels. Such multimedia content can be either statically pre-generated or produced on-demand at runtime. We have similar generators for audio and video content.

Wikipedia access traces are available from the Wikibench Web site [7]. The log can be used to reproduce read workload traces while the wiki history log can be used to reproduce the exact update workload. mBenchLab traces support both CSV and HTTP archive (HAR) formats. On top of capturing the original request, HAR also includes sub-requests, post parameters, cookies, headers, caching information and timestamps. We provide mBenchLab traces to use with our Wikipedia virtual appliances.

III. MEASURING QOE ON ANDROID DEVICES

A central contribution of mBenchLab is the ability to replay traces through real Web browsers. Major companies such as Google and Facebook already use open source technologies like Selenium [12] to perform functional testing. These tools automate a browser to follow a script of actions, and they are primarily used for checking that a Web application's interactions generate valid HTML pages. We argue that the same technology can also be used for performance benchmarking. One of the technical challenges is that Selenium is originally designed for testing from a desktop machine that conducts performance tests via an emulator or a mobile device connected to it. We have extracted the relevant core pieces of Selenium and have embedded it in a standalone mobile application on Android devices.

The mBenchLab Android application (mBA) extends the Selenium framework with mBenchLab functionalities to download a trace, replay it, record QoE statistics for each page and upload the results at the end of the replay. Unlike traditional load injectors that work at the network level, replaying through a Web browser accurately performs all activities such as typing data in Web forms, scrolling pages and clicking buttons. The typing speed in forms can also be configured to model a real user typing. This is particularly

useful when inputs are processed by JavaScript code that can be triggered on each keystroke. Through the browser, mBA captures the real user perceived latency including network transfer, page processing and rendering time.

Most desktop browsers include debugging tools such as Firebug for Firefox or the developer tools for Chrome that are able to capture the timeline of all browser interactions while pages are being loaded. This data can usually be stored into a HAR file. No Android Web browser, including the native Android browser, offers that feature. We therefore have re-implemented that functionality to obtain and log detailed QoS information on mobile devices.

An open source proxy developed by WebMetrics called BrowserMob proxy [14] offers that functionality in a standalone Java proxy. That proxy being designed for regular desktop JVMs, we had to port it and adapt it to the idiosyncrasies of the Android platform in order to integrate it in the mBA. Running a fully functional Web proxy on devices with very limited resources can impose a significant overhead and therefore the proxy is only optional if detailed HAR recording is not desired. Fig. 3 gives an overview of the architecture of the mBA.

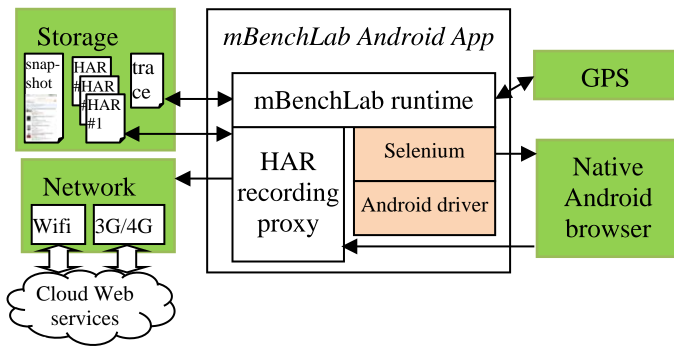

Fig. 3. mBenchLab Android application architecture

Fig. 4. HAR captured by the mBA when accessing the main page of the english Wikipedia web site

The detailed information collected by the mBA is the following: DNS resolution time, connection establishment time, request failure/success/cache hit rate, send/wait/receive time on network connections, overall page loading time including Javascript execution and rendering time. Unfortunately the current Android APIs do not provide monitoring of battery usage on a per application level. Therefore we are not able to measure the power impact of Web service designs. Fig. 4 shows a partial example of the information captured when accessing the main page of the Wikipedia web site.

Additionally the mBA can record HTML page sources and screen snapshots of rendered pages. This data is automatically uploaded at the end of the experiment and can be visualized in the dashboard. As shown on Fig. 5, the screen snapshots can also capture errors reported by the browser.

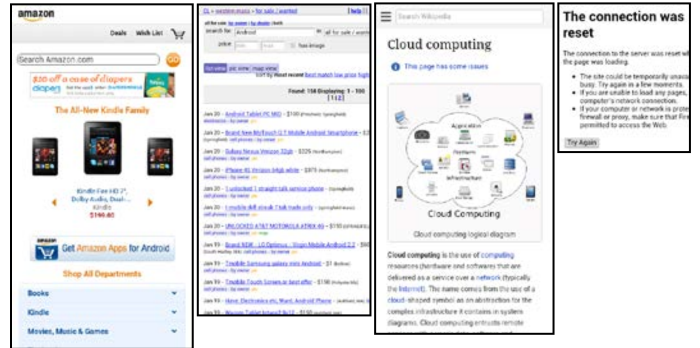

Fig. 5. Example of screen snapshots taken during experiments on Amazon, Craigslist, Wikipedia and a network connection error.

The screen snapshot functionality can also be extended to measure the QoE on streamed video content by capturing images at fixed time intervals.

IV. Evaluation

In this section we present the results of our experimental evaluation with mBenchLab. First, we describe our experimental setup and our methodology (section A). Our first set of experiments targeting well known web sites is presented in section B. We evaluate the overhead of our instrumentation mechanisms in section C. We show various use cases of QoE problem detection in section D and location-based QoE in section E. Section F summarizes our results.

A. *Experimental setup and methodology*

We have conducted our experiments on a laptop with Firefox as a baseline to compare with the native browser of our tablets and smartphones. We tested various browsers for the desktop baseline including Chrome and Internet Explorer and we obtained similar results as Firefox. Therefore we only present Firefox results for the desktop baseline. Our software only supports the native Android browser, so we cannot compare with Firefox or Chrome versions for Android.

TABLE II shows the hardware and software specifications of our devices. The Trio tablet is an entry-level Android tablet while the Kindle Fire is a higher end tablet of the same generation. The smartphones used in our experiments are the popular high-end Samsung S3 and Motorola Droid RAZR as well as an entry level HTC Desire C. While devices can have the same physical screen size, screen resolutions vary greatly impacting the amount of information provided to the user.

TABLE II ANDROID DEVICES USED IN OUR EXPERIMENTS WITH THEIR RESPECTIVE HARWARE AND SOFTWARE SPECIFICATIONS.

Device	Processor	RAM / Storage	Screen size / resolution / GPU	Network	OS version	Web browser version
MacBook Pro	2 GHz Intel Core i7	1GB / 150GB (VM)	15" / 1440x900 AMD Radeon HD 6490M 256MB	Wifi	Windows 7 x64 / VMWare Fusion	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:15.0) Gecko/20100101 Firefox/15.0.1
Trio Stealth Pro Tablet	Single core 1.2GHz ARM Cortex A8	512MB / 4GB	7" / 800x480 Mali 400	Wifi	Android 4.0.3 (official release Dec 2011)	Mozilla/5.0 (Linux; U; Android 4.0.3; en-us; SoftwinerEvb Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Amazon Kindle Fire	Dual core 1.2 GHz TI OMAP4 4430HS	512MB / 8GB	7" / 1024x600 PowerVR SGX540	Wifi	Android 4.1.2 (AOKP Otter Oct 17, 2012)	Mozilla/5.0 (Linux; U; Android 4.1.2; en-us; Amazon Kindle Fire Build/JZO54K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
HTC Desire C	Single core 600 MHz, ARM Cortex-A5	512MB / 4GB	3.5" / 320x480 Adreno 200	Wifi/ Edge/3G Orange	Android 4.0.3 (official release Dec 2011)	Mozilla/5.0 (Linux; U; Android 4.0.3; fr-fr; HTC Desire C Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Samsung S3 GT-I9300	Quad-core 1.4 GHz ARM Cortex-A9	1GB / 32GB + 64GB	4.8" / 720x1280 Mali-400MP	Wifi/3G AT&T	Android 4.1.2 (official release Dec 2012)	Mozilla/5.0 (Linux; U; Android 4.1.2; en-us; GT-I9300 Build/JZO54K) AppleWebKit/ 534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Motorola Droid RAZR	Dual core 1.2 GHz ARM Cortex-A9	1GB / 16GB	4.3" / 540 x 960 PowerVR SGX540 304 MHz	Wifi/3G/ 4G LTE Verizon	Android 4.0.4 (official release Mar 2011)	Mozilla/5.0 (Linux; U; Android 4.0.4; en-us; DROID RAZR Build/6.7.2-180_DHD-16_M 4-31) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30

We have generated 4 trace files, each targeted at a particular web site. The *Amazon* trace browses products from the amazon.com US store. The *Craigslist* trace searches for various items in the Western Massachusetts Craigslist website. The *Wikipedia* trace accesses a number of articles of varying length on the English Wikipedia web site. Finally, the *Wikibooks* trace browses articles from our implementation of the English Wikibooks website running in our datacenter with the enwikibooks database described in TABLE I.
Each experiment is repeated at least 5 times and caches are emptied between each run. All experiments are done on Wifi networks except where 3G or 4G LTE are indicated. 3G experiments in the USA use the Straight Talk data plan on AT&T and Orange in Europe. 4G LTE uses the Verizon network.

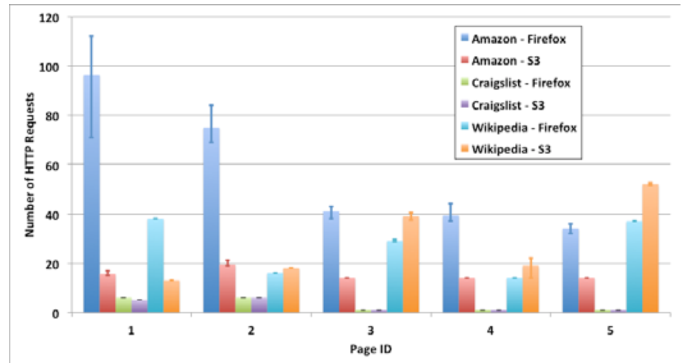

Fig. 6. Comparing the number of browser generated requests on Android/S3 and Firefox/desktop with Amazon, Craigslist and Wikipedia.

B. Measuring performance of major Web sites

When a browser is directed to a particular URL, it starts to get the main HTML page and processes it to download any additional images, style sheets or scripts required to properly display the page. Fig. 6 shows the average number of requests issued by Web browsers when trying to access the 5 first pages of our traces for Amazon, Craigslist and Wikipedia.
The desktop version of the Amazon web pages is the most complex and can require more than 100 requests to fetch. We observe a significant variability between runs especially on the home page as a lot of content is generated dynamically depending on the user and the current sales. The mobile version of the same pages never exceeds 20 requests and is consistent between consecutive runs. The more simple Craigslist Web pages only require 1 request once the browser cache is hot. The desktop and mobile versions exhibit the same behavior.

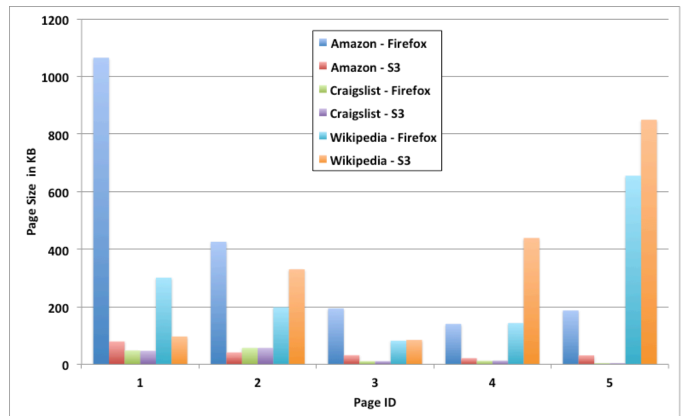

Fig. 7. Comparing the page sizes in KB on Android/S3 and Firefox/desktop for the 5 first request of our Amazon, Craigslist and Wikipedia traces.

The number of requests for Wikipedia on the Samsung S3 is inflated by a bug that is investigated in section D. The number

of requests and page sizes for Wikipedia are typically smaller on mobile devices than desktops as show on Fig. 13. Similarly, Amazon serves much smaller pages to its mobile users that are not exposed to the large number of ads displayed in the non-mobile version. Craigslist with its minimalistic design offers very small page sizes for both mobile and non-mobile clients.
Since Amazon shows significant difference in the content being fetched between consecutive runs, it is not possible to directly compare the performance between these runs. This shows that it is important to understand the details of the generated content to be able to interpret client side QoE.

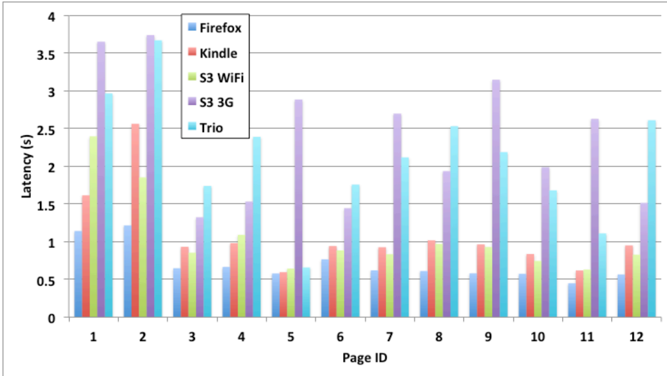

Fig. 8. Comparing average observed latency for desktop, tablets and phones on wireless networks with our Craigslist trace.

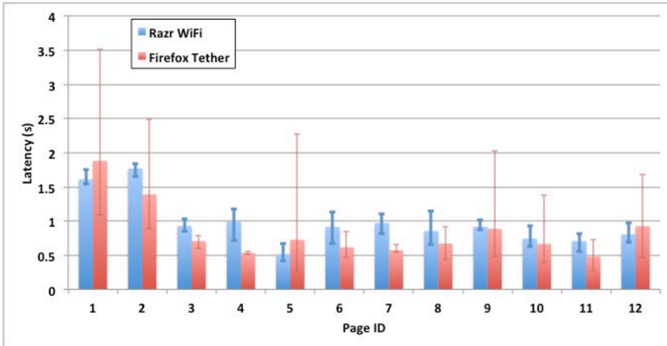

Fig. 9. Comparing average latency with our Craigslist trace for the native Razr browser on Wifi and Firefox on MacBook tethering via the Razr Wifi.

Fig. 8 shows the average latency to load pages from our Craigslist trace. The spikes on page id 2 are attributed to varying DNS resolution times. The desktop version is the fastest though it uses the same Wifi network as the tablets and phones. The processing power makes the difference in rendering even simpler pages. The higher latency of the 3G network almost doubles the page loading time on our Craigslist trace where half of time is spent in establishing the connection with the server. The Trio tablet is significantly lower both in rendering and networking. Its performance over Wifi is comparable to the one observed for 3G on the S3. 4G experiments could not be made directly run using the Razr browser due to certain limitations of the Verizon version of the phone. However, we were able to use the desktop browser and tether through the phone's 4G connection. We verified on Wifi that the performance of Firefox tethering via the phone was similar to the Razr browser performance as shown on Fig. 9. Therefore we expect our 4G results via tethering to be very close to the performance of the native Razr browser on 4G. The Razr 4G performance is similar to the Wifi performance of other devices.

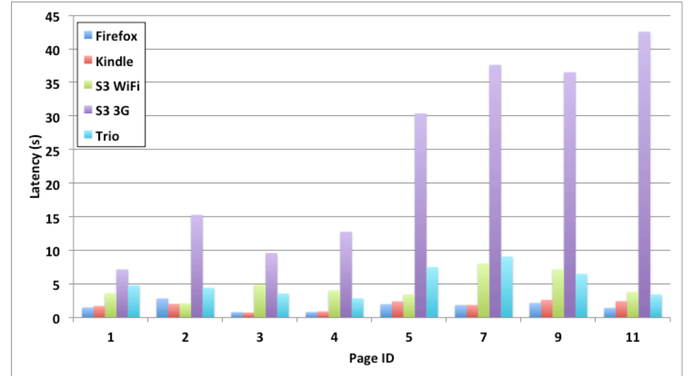

Fig. 10. Comparing average latency for desktop, tablets and phones on wireless networks with our Wikipedia trace.

Fig. 10 compares the devices' observed latency for our Wikipedia trace. While the Kindle has a performance close to the desktop browser, the Trio shows slower performance due to reduce Wifi performance and image rendering speed. The S3 results are impaired by a bug in the browser that is analyzed in more details in section D. The Razr performance on Wifi and 4G (via tether) is very similar showing how 4G brings user perceived QoE to the same level as Wifi.

C. QoE measurement overhead on mobile devices

We conducted a series of experiments using our Craigslist and Wikipedia traces with and without our QoE instrumentation on all platforms and networks. Our HAR proxy recorder intercepts all outgoing connections and collects statistics on network and system events that can help troubleshoot QoE issues as we will see in section D. Running the proxy however requires cpu and memory resources that could affect performance or the behavior of the Web browser.

TABLE III QoE INSTRUMENTATION OVERHEAD USING CRAIGSLIST AND WIKIPEDIA TRACES ON OUR DEVICES ON WIFI AND MOBILE NETWORKS.

Device	Craigslist latency (+overhead)	Wikipedia latency (+overhead)
MacBook Pro	505 (+166) ms	1576 (+156) ms
Trio Stealth Pro	2200 (-46) ms	2737 (+2781) ms
Kindle Fire	1380 (-506) ms	1709 (-14) ms
Samsung S3 3G	7571 (-2303) ms	14884 (+2167) ms
Samsung S3 Wifi	1185 (-100) ms	4823 (-450) ms
Droid Razr Wifi	978 (-75) ms	2076 (+329)
Droid Razr Wifi tether	838 (+165) ms	1875 (+746) ms
Droid Razr 4G tether	739 (+283) ms	1955 (+467) ms

TABLE III presents our findings by aggregating the latencies of all pages for a trace in a single average. The overall latency without monitoring is presented first, followed by the overhead of monitoring in parenthesis. The latencies are measured by taking a clock start before directing the browser to a URL and

the clock stops when the browser notifies that the page has been fully loaded. The instrumented latency includes the timing of all internal events in memory by the proxy. The storing of the in-memory data to the device storage is done after the page is fully loaded and therefore does not impact page loading time latency.

On most devices, the overhead of monitoring is within the natural variance observed in real conditions between multiple runs. 4G offers performances very close to Wifi on the Droid Razr. Our monitoring shows a slightly higher overhead when using tethered connections (note that the browser used is Firefox on the laptop for tethered connections whereas non-tethered connections use the native browser of the phone). The variations on 3G are more significant as base latencies are much higher and network performance varies a lot. The bigger and more complex Wikipedia pages require the proxy to relay more data and time more events but still without significantly affecting the user QoE. A notable exception is the entry-level Trio tablet that shows a significant overhead in its instrumented runs on Wikipedia.

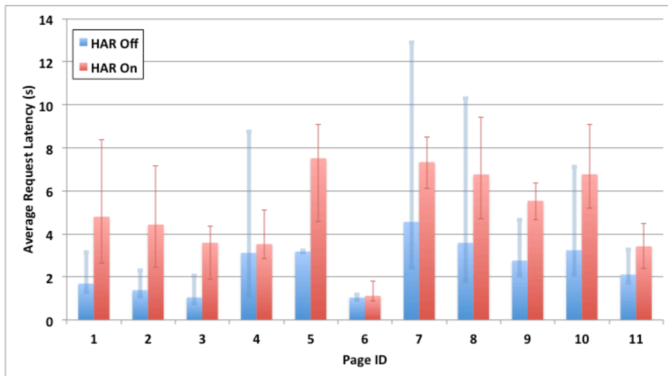

Fig. 11. Overhead of instrumentation for our Wikipedia trace on Trio tablet.

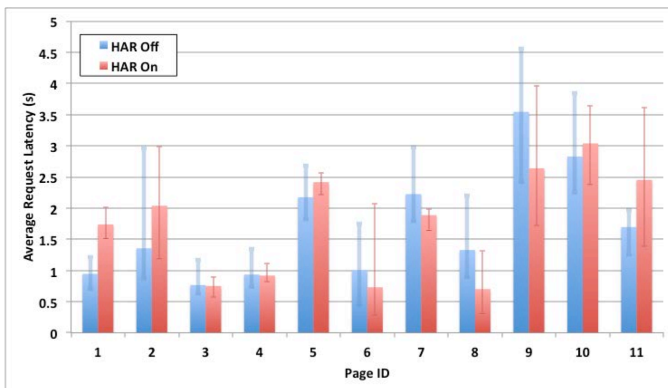

Fig. 12. Overhead of instrumentation for our Wikipedia trace on Kindle Fire.

Fig. 11 shows the average latency with min and max values on 5 runs of our Wikipedia trace on the Trio tablet. In most cases, the instrumented runs (HAR on) are 3 seconds slower than the non-instrumented ones. Page id 6 is an exception as the page size is much smaller than any other page (8KB vs 100+KB). The Trio being a single core tablet, the context switching between the Web browser threads and our proxy threads are significantly slower. Other factors that contribute to the no overhead noticed on page 6 is due to the fact that the page contains no image at all in its mobile version.

Fig. 12 shows the results for a similar set of experiments with the Kindle Fire. While the first request where all connections must be initiated and mapped through the proxy shows a clear overhead with monitoring (HAR on), all subsequent queries have similar latencies whether monitoring is enabled or not (HAR off).

Given the quick pace of technological progress in tablets and smartphones, we expect that the instrumentation overhead will not be any more significant than it is today and will most likely become even more negligible.

D. Identifying QoE issues

1) Why HAR instrumentation is important

Some aspects of the user perceived QoE are specific to the device such as the physical display size or screen resolution. However one of the main aspects considered by users is the page loading latency and of course the correct and successful completion of all operations involved in loading the page. While techniques such as HTML recording and screen snapshots can help detect some issues in the rendered page, the overall page loading time measurements is not sufficient to understand the root cause of QoE issues.

When running our experiments on the Amazon store without instrumentation, we noticed a number of abnormal page loading latencies that we were not able to explain as the recorded HTML and the screen snapshots showed properly rendered page. Instrumented runs also showed similar random events but the HAR instrumentation allowed us to identify the root cause of these issues.

Fig. 15 shows the HAR data collected while playing the Amazon trace on one of our smartphones. Out of the 14 HTTP GET requests needed to fetch the page, one subrequest blocked for almost 9.5 seconds on a DNS lookup operation. The troubled networking layer spent another half second establishing the connection with the server and nearly 2 seconds to get 57 bytes response!

The blocked DNS requests are usually caused by other DNS requests that are already being processed in the request queue and timing out. Given the limited number of threads that the DNS subsystem can use to issue requests, a small number of failing requests can block all other application requests. The slow connection establishment and data transfer is attributable to network congestion either on the Wifi network or anywhere on the path to the server. One of the limitations of the HAR recording is that it does not give us insights where on the network path the issue might be.

2) The Samsung S3 browser bug

When comparing our results on the different devices and networks for our Wikipedia trace, we noticed significantly higher latencies for our Samsung S3 smartphone on both Wifi and 3G. We first looked at the number of HTTP requests per page and the size of the pages downloaded from the server. Our findings are illustrated on Fig. 13. The number of HTTP requests is always much higher for the Samsung S3 and the page sizes are much bigger. Note that the page size for Samsung S3 on 3G is sometimes very small as we only account

for successfully transferred bytes and not expected object sizes. On a successful page load, the page sizes should be the same on both networks.

Fig. 14 gives an insight into the cause of the problem. By looking at the recorded HTML page source, we saw that Wikipedia pages use `srcset` HTML tags that indicate a list of images to pick from depending on the resolution and magnification needed by the device. It turns out that the S3 browser has a bug and systematically downloads all images in a `srcset` instead of picking only the one it needs (left most red circles on Fig. 14 show 3 different versions of the same image being downloaded). This can result in a massive amount of extra data download.

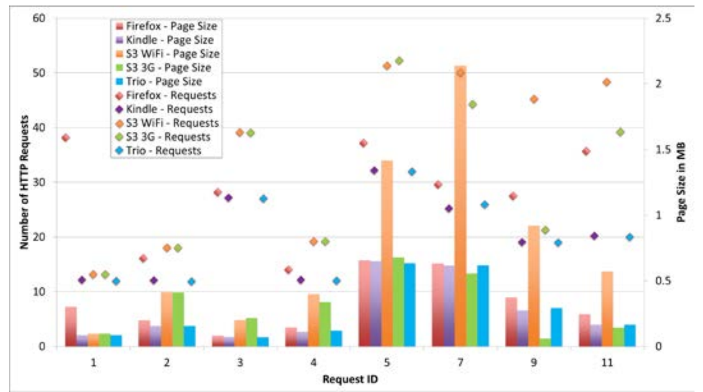

Fig. 13. Comparing number of HTTP requests and downloaded page size for our devices on Wifi and wireless networks with our Wikipedia trace.

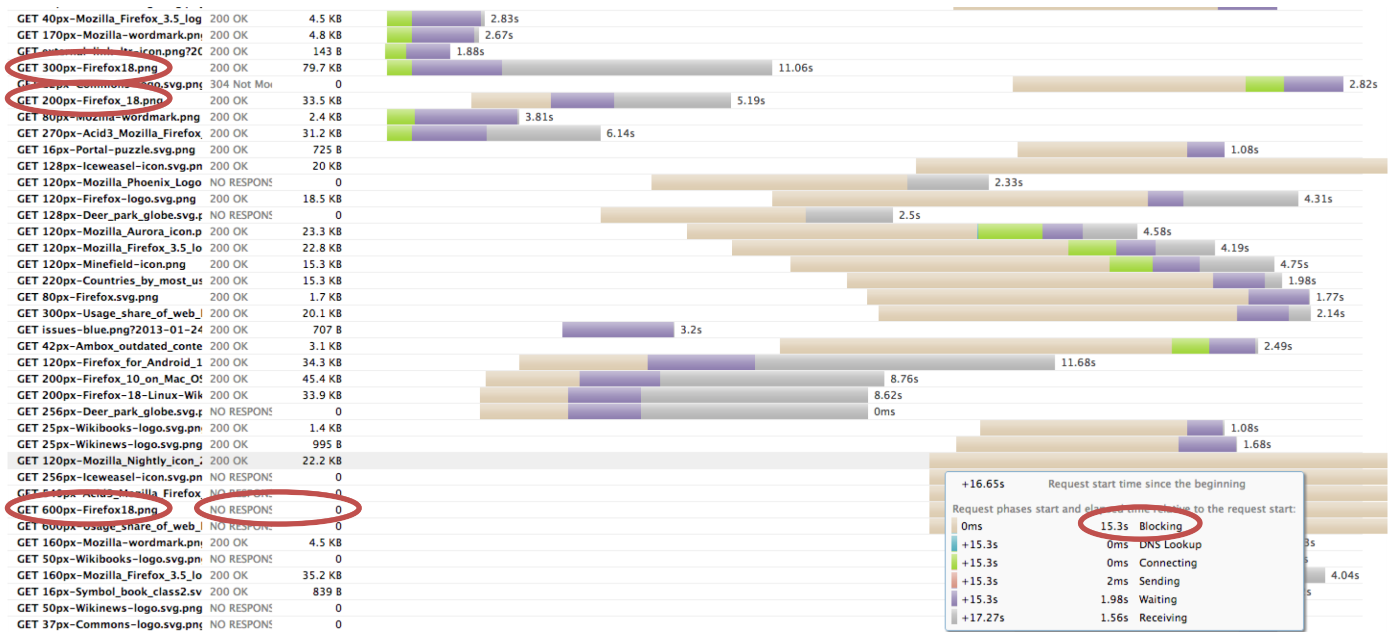

Fig. 14. Example of a Wikipedia page load on a Samsung S3 using a 3G network showing a browser issue loading all images in a srcset and network timeouts.

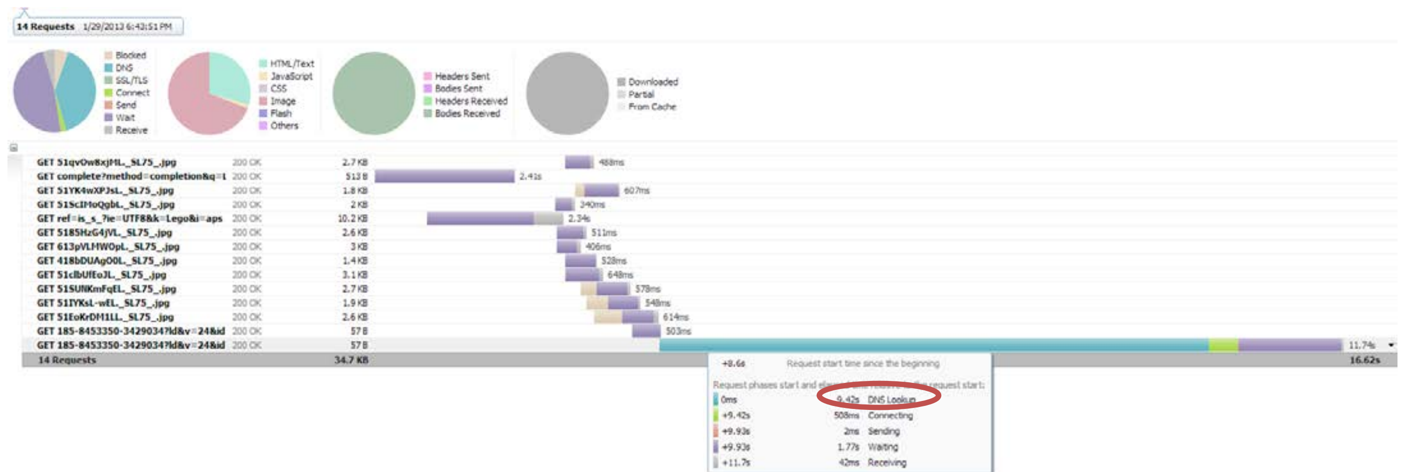

Fig. 15. Example of an Amazon page load that blocks for 9.42s on a DNS lookup operation increasing overall page loading time by more than 127%.

The Wikipedia page dedicated to the Internet Explorer browser that typically requires 600KB of data download jumped to 2.1MB on the S3. This bug significantly affects the Wikipedia performance on 3G were these massive number of requests for image downloads overwhelmed the network and ended up timing out rendering an incomplete page. This can be seen on Fig. 14 where a large number of requests are blocked for very long amount of time and many of them fail with a 'NO RESPONSE' HTTP error code.

Note that we were able to reproduce these results with the latest Android 4.2.2 for the S3 GT-I9300 (international version of the phone). The issue was also reproduced with an S3 SGH-I747 which is the AT&T US version of the phone. We believe that this problem affects all S3 versions and have contacted Samsung to report the issue.

Having a database with results from other devices helped us to quickly locate the origin of the problem and detect this previously undiscovered bug. Based on this experience, a possible direction for future work is to design tools that automatically analyze and report anomalies by comparing experience reports between devices/networks for the same trace.

E. *QoE based on location*

We have previously observed [6] that latency is very dependent on user location in wired networks. Identifying geographical regions where user QoE is poor is crucial for the design of CDNs or replicated systems. To measure the effect of location with wireless networks, we use our Wikipedia implantation running the English Wikibooks database (enwikibooks in TABLE I). The application server and database are deployed in our datacenter at the University of Massachusetts Amherst. The Wikibooks pages usually show higher loading time as they contain entire books.

Fig. 16 shows the observed latencies from a Samsung S3 phone on a Wifi network within 2 network hops of the datacenter (S3 Wifi USA), an S3 phone on AT&T 3G network within 1 mile of the datacenter and an HTC phone on a residential Wifi in eastern France (HTC Wifi France).

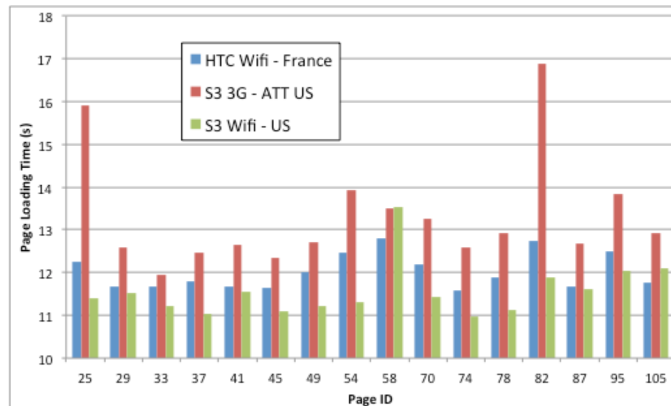

Fig. 16. Comparing latencies from an HTC phone in Europe vs an S3 phone near a data center in the US running our Wikibooks implementation.

The Wifi latencies are actually very close to each other (note that the y scale starts at 10 seconds). The 3G performance remains much slower even compared to cross-continental accesses. From this experience it looks like the QoE is more linked to the network access than the geolocation of the user.

We conducted a similar experiment using our Amazon trace on the US store. This time we experimented with access from a Wifi, Edge and 3G networks in Europe. Our results are presented in Fig. 17. The Wifi latencies are comparable regardless of the user location even though the Samsung S3 is technically a more powerful phone. Overall the Orange France 3G network offers latencies at most 2 seconds higher than the Wifi latencies. The AT&T 3G network exhibits worse performance with latencies that can more than double compared to Wifi. As expected the Edge network is the slowest though in some cases it is not that far from the AT&T 3G performance. The latency spikes observed on the Orange network are due long period of inactivity where the phone waits for network access. Once again the provider used to access the network had a dominant effect over user geolocation or device performance.

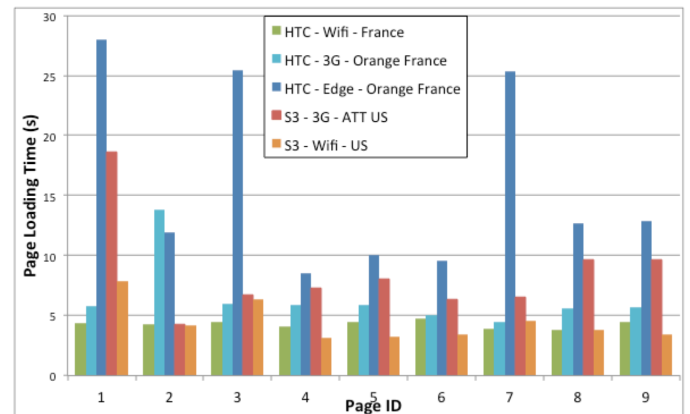

Fig. 17. Comparing average observed latency for devices in US and Europe over Wifi, Edge and 3G with our Amazon trace accessing Amazon.com (US).

F. *Results summary*

Dynamic content on complex Web Applications can introduce a lot of variability even between consecutive experiments using the same trace. This makes it hard to interpret QoE measurements without a detailed monitoring of individual page loading events. We have shown that such instrumentation can be achieved without being detrimental to the user perceived QoE while providing crucial information to isolate root causes of QoE issues. Using mBenchLab we were able to discover a new bug in the native browser of the Samsung S3 smartphone that prevented certain Wikipedia pages from loading properly.

Our set of experiments is restricted to a small number of devices and networks which limits their statistical validity. However, we can identify some trends like mobile performance is still limited by hardware resources on low-end devices, newer higher-end devices are more limited by network capabilities. 4G networks seem to approach Wifi performance on mobile devices. Unlike wired networks where the location of the user dominates latency, the performance of mobile networks largely defines the user QoE independently

of his/her location. We are working on larger scale experiments to verify these observations.

V. Related work

As Web browsing constitutes the majority of traffic on smartphones [3] it is a necessity to analyze the QoE of mobile devices at various levels. mBenchLab's approach of running unmodified software stacks is closer to the one presented in [1] various mobile apps were observed at the network level of with more than 30K users all over the world. They found that 3G performance varies according to the network provider and that browser performance increases with connection parallelism. We made similar observations on the various mobile networks we have tested with mBenchLab. The device influence was mostly perceived on Javascript execution and download performance. The authors in [15] also showed that the device storage performance could adversely affect the browsing experience. A more intrusive approach [2] instrumenting Webkit showed that network RTT was detrimental to browser performance. Also resource loading was more important than JavaScript execution, layout calculation or formatting. The device processor was still playing a significant role in overall performance. While we have seen that low-cost entry devices like the Trio tablet are still limited by their hardware performance, the playfield is being leveled with the network provider performance dominating over the device capabilities.

Other works are focusing on server side improvements to increase user perceived QoE. In [4], the authors improve Wikipedia page loading power consumption by 29% by improving JavaScript and CSS. They also found that using JPEG images over other formats improve energy savings. Mobile proxies can also improve performance by aggregating multiple small transfers [3]. The same study showed that increasing the socket buffer sizes at servers can improve throughput; and reducing radio sleep timers can reduce power consumption. mBenchLab complements these studies as it can be used to measure the QoE variations between various server side designs or detect QoE issues with particular devices or geographical locations. Complementary approaches try to rethink the networking infrastructure for mobile devices [5] and investigate how to transparently switch between networks. By recording the device GPS location throughout experiments, mBenchLab allows to build database of geolocalized performance data to explore further network influence in modern realistic mobile networking.

VI. Conclusion

In this paper, we have presented mBenchLab, an open source infrastructure to measure the QoE of Web application on mobile devices. We have shown that our instrumentation allowed us to identify accurately QoE issues with unmodified devices on real networks. We were able to identify a new bug in the native browser of a very popular smartphone that causes major issues (increased data usage, network overload, loading errors…) for users of the Wikipedia website.

We measured the performance of several tablets and smartphones and showed that mobile network performance was a dominant factor in user perceived QoE over device performance or user location. The device hardware resources only had a significant impact for low-end devices while 4G networks offered performance similar to Wifi.

All our software is freely available on our project page at http://lass.cs.umass.edu/projects/benchlab/. The software can be downloaded from https://sourceforge.net/projects/benchlab/ for anyone to use and deploy their own mobile benchmarking platform. We are actively distributing mBenchLab to collect worldwide QoE data on popular websites but we hope that other research groups will use these tools to measure the impact of their research on mobile device QoE.

Acknowledgment

The authors would like to thank Veena Udayabhanu, Camille Pierrat, Fabien Mottet and Vivien Quema for their contributions. This research was supported in part by NSF grants OCI-1032765, CNS-0916972, CNS-1117221 and CNS-1040781.

References

[1] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z. Morley Mao, Ming Zhang, and Paramvir Bahl. *Anatomizing application performance differences on smartphones*. In Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10). ACM, New York, NY, USA, 165-178.

[2] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. *Why are web browsers slow on smartphones?*. In Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile '11). ACM, New York, NY, USA, 91-96.

[3] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. *A first look at traffic on smartphones*. In Proceedings of the 10th annual conference on Internet measurement (IMC '10). ACM, New York, NY, USA, 281-287.

[4] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. *Who killed my battery?: analyzing mobile browser energy consumption*. In Proceedings of the 21st international conference on World Wide Web (WWW '12). ACM, New York, NY, USA, 41-50.

[5] Erik Nordström, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J. Freedman. *Serval: An End-Host Stack for Service-Centric Networking*. In Proc. 9th Symposium on Networked Systems Design and Implementation (NSDI '12), San Jose, CA, April 2012.

[6] Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood and Prashant Shenoy. BenchLab: An Open Testbed for Realistic Benchmarking of Web Applications. In Proc. of 2nd USENIX Conference on Web Application Development (WebApps '11), Portland, OR, June 2011,.

[7] WikiBench - http://www.wikibench.eu/.

[8] Wikibooks – http://www.wikibooks.org/.

[9] Wikipedia – http://www.wikipedia.org/.

[10] NPD DisplaySearch Reports. Tablet PC Market Forecast to Surpass Notebooks in 2013. January 7, 2013. http://www.displaysearch.com/pdf/130107_tablet_pc_market_forecast_to_surpass_notebooks_in_2013.pdf

[11] HTTP Archive specification (HAR) v1.2 - http://www.softwareishard.com/blog/har-12-spec/.

[12] Selenium - http://seleniumhq.org/.

[13] Page views for Wikipedia - http://stats.wikimedia.org/EN/TablesPageViewsMonthly.htm.

[14] WebMetrics BrowserMob proxy - http://opensource.webmetrics.com/browsermob-proxy/.

[15] Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu. *Revisiting Storage for Smartphones*. In Proceedings of 10th Usenix Conference on File and Storage Technologies (FAST'12), San Jose, CA, February 2012