BatteryLab, A Distributed Power Monitoring Platform For Mobile Devices

https://batterylab.dev

Matteo Varvello†, Kleomenis Katevas⋄, Mihai Plesa†, Hamed Haddadi†⋄, Benjamin Livshits†⋄
† Brave Software, ⋄ Imperial College London

ABSTRACT

Recent advances in cloud computing have simplified the way that both software development and testing are performed. Unfortunately, this is not true for battery testing for which state of the art test-beds simply consist of one phone attached to a power meter. These testbeds have limited resources, access, and are overall hard to maintain; for these reasons, they often sit idle with no experiment to run. In this paper, we propose to *share* existing battery testing setups and build BatteryLab, a distributed platform for battery measurements. Our vision is to transform independent battery testing setups into *vantage points* of a planetary-scale measurement platform offering heterogeneous devices and testing conditions. In the paper, we design and deploy a combination of hardware and software solutions to enable BatteryLab's vision. We then preliminarily evaluate BatteryLab's accuracy of battery reporting, along with some system benchmarking. We also demonstrate how BatteryLab can be used by researchers to investigate a simple research question.

1 INTRODUCTION

The mobile device ecosystem is large, ever growing, and very much "location-based", *i.e.,* different devices and operating systems (Android and iOS) are popular at different locations. Advances in cloud computing have simplified the way that mobile apps are tested, today. Device *farms* [5, 23] let developers test apps across a plethora of mobile devices, in real time. Device diversity for testing is paramount since hardware and software differences might impact how an app is displayed or performs.

To the best of our knowledge, no existing device farm offers *hardware-based* battery measurements, where the power drawn by a device is measured by directly connecting its battery to an external power meter. Instead, few startups [19, 24] offer *software-based* battery measurements where device resource monitoring (screen, CPU, network, etc.) are used to infer the power consumed by few

devices for which a calibration was possible [12]. This suggests a demand for battery measurements, but a prohibitive cost for deploying hardware-based solutions.

In the research community, hardware-based battery measurements are instead quite popular [10, 11, 20, 34]. The common research approach consists of buying the required hardware (often an Android device and a Monsoon power monitor [26]), set it up on a desk, and then use it sporadically. This is because such battery testbeds are intrinsically *local*, *i.e.,* they require a researcher or an app tester to have physical access to the device and the power meter.

In this paper, we challenge the assumption that a battery testbed needs to be local and propose *BatteryLab* [8], a distributed platform for battery measurements. Similarly to PlanetLab [30], our vision is a platform where members contribute hardware resources (e.g., some phones and power monitor) in exchange of access to the hardware resources offered by other platform members. As new members join over time and from different locations, BatteryLab will naturally grow richer of new and old devices, as well as of devices only available at some specific locations.

BatteryLab's architecture consists of an *access server* — which enables an end-to-end test pipeline while supporting multiple users and concurrent timed sessions — and several *vantage points*, *i.e.,* the local testbeds described above. Vantage points are enhanced with a lightweight *controller* — hosted on a Raspberry Pi [32] — which runs BatteryLab's software suite to enable remote testing, e.g., SSH channel with the access server and *device mirroring* [15] which provides full remote control of test devices, via the browser.

We first evaluate BatteryLab with respect to the *accuracy* of its battery readings. This analysis shows that the required extra BatteryLab's hardware has negligible impact on the power meter reporting. It also shows a non-negligible cost associated with device mirroring, suggesting that it should only be used when devising a test. Such *headless* mode is not always possible, e.g., if usability testing is the goal. In this case, the extra battery consumption associated with mirroring should be accounted for.

Finally, we demonstrate BatteryLab usage investigating a simple research question: *which of today's Android browser is the most energy efficient?* To answer this question, we automated the testing of four popular browsers (Chrome, Firefox, Edge, and Brave) via BatteryLab. Our results show that Brave offers minimal battery consumption, while Firefox tends to consume the most. We further augment this result across multiple locations (South Africa, China, Japan, Brazil, and California) emulated via VPN tunneling.

2 RELATED WORK

This work was mainly motivated by the frustration of not finding a tool offering easy access to battery measurements. Several existing tools could leverage some of BatteryLab's ideas to match our capabilities in a paid/centralized fashion. For example, device farms such as AWS Device Farm [5] and Microsoft AppCenter [23] could extend their offer using our hardware and software components.

The same is true for startups like GreenSpector [19] and Mobile Enerlytics [24], which offer software-based battery testing.

To the best of our knowledge, MONROE [1] is the only measurement platform sharing some similarities with BatteryLab. This is a platform for experimentation in operational mobile networks in Europe. MONROE currently has presence in 4 countries with 150 *nodes*, which are ad-hoc hardware configurations [25] designed for cellular measurements. BatteryLab is an orthogonal measurement platform to MONROE since it targets real devices (Android and iOS) and fine-grained battery measurements. The latter requires specific instrumentation (bulky power meters) that cannot be easily added to MONROE nodes, especially the mobile ones. In the near future, we will explore solutions like BattOr [33] to potentially enhance BatteryLab with mobility support.

Last but not least, BatteryLab offers full access to test devices via mirroring. This feature was inspired by [2], where the authors build a platform to allow an Android emulator to be accessed via the browser, with the goal to "crowdsource" human inputs for mobile apps. We leverage the same concept to allow remote access to BatteryLab, but also further extend it to actual devices and not only emulators.

3 BATTERYLAB

This section details the design and implementation of BatteryLab, a distributed measurement platform for device battery monitoring (see Figure 1(a)). We currently focus on mobile devices only, but our architecture is flexible and we thus plan to extend to more devices, e.g., laptops and IoT devices.

One or multiple test devices (a phone/tablet connected to a power monitor) are hosted at some university or research organization around the world (*vantage points*). BatteryLab members (*experimenters*) gain access to test devices via a centralized *access server*, where they can request time slots to deploy automated scripts and/or ask remote control of the device. Once granted, remote control of the device can be shared with *testers*, whose task is to manually interact with a device, e.g., search for several items on a shopping application. Testers are either volunteers, recruited via email or social media, or paid, recruited via crowdsourcing websites like Mechanical Turk [4] and Figure Eight [13].

3.1 Access Server

The main role of the access server is to manage the vantage points and schedule experiments on them based on experimenter requests. We built the access server atop of the Jenkins [21] continuous integration system which is free, open source, portable (as it is written in Java) and backed by an active and large community. Jenkins enables end-to-end test pipelines while supporting multiple users and concurrent timed sessions.

BatteryLab's access server runs in the cloud (Amazon Web Services) which enables further scaling and cost optimization. Vantage points have to be added explicitly and pre-approved in multiple ways (IP lockdown, security groups). Experimenters need to authenticate and be authorized to access the web console of the access server. For increased security, this is only available over HTTPS.

The access server communicates with the vantage points via SSH. New BatteryLab members grant SSH access from the server to the vantage point's controller via public key and IP white-listing (see Section 3.4). Experimenters can access vantage points via the access server, where they can create *jobs* to be deployed in their favorite programming language. Only the experimenters that have

been granted access to the platform can create, edit or run jobs and every pipeline change has to be approved by an administrator. This is done via a role-based authorization matrix.

BatteryLab's Python API (see Table 1) is available to provide user-friendly device selection, interaction with the power meter, etc. The access server will then dispatch queued jobs based on experimenter constraints, e.g., target device, connectivity, or network location, and BatteryLab constraints, e.g., one job at the time per device. By default, the access server collects logs from the power meter which are made available for several days within the job's workspace. Android logs like `logcat` and `dumpsys` can be requested via the `execute_adb` API, if available.

We have developed several jobs which manage the vantage points. These jobs span from updating BatteryLab wildcard certificates (see Section 3.4), to ensure the power meter is not active when not needed (for safety reasons), or to factory reset a device. These jobs were motivated by our needs while building the system, and we expect more to come over time and as the system grows.

3.2 Vantage Point

Figure 1(b) shows a graphical overview of a BatteryLab's vantage point with its main components: controller, power monitor, test devices, circuit switch, and power socket.

Controller – This is a Linux-based machine responsible for managing the vantage point. The machine should be equipped with both Ethernet and WiFi connectivity, a USB controller with a series of available USB ports, as well as with an external General-Purpose Input/Output (GPIO) interface. We use the popular Raspberry Pi 3B+ [32] running the latest version of Raspbian Stretch (April 2019) that meets these requirements with an affordable price.

The controller's primary role is to manage connectivity with test devices. Each device is connected to the controller's USB port, WiFi access point (configured in NAT or Bridge mode), and Bluetooth. USB connectivity is used to power each testing device when not connected to the power monitor and to instrument it via the Android Debugging Bridge [17] (ADB), if available. WiFi connectivity is used to allow automation without the extra USB current, which interferes with the power monitoring procedure. (De)activation of USB ports is realized using `uhubctl` [28]. Bluetooth connectivity is used for automation across OSes (Android and iOS) and connectivity (WiFi and cellular). Section 3.3 will discuss several automation techniques supported by BatteryLab.

The second role of the controller is to provide *device mirroring, i.e.,* easy remote access to the device under test. We use VNC (`tigervnc` [35]) to enable remote access to the controller. We further use `noVNC` [29], an HTML VNC library and application, to provide easy access to a VNC session via a browser without no further software required at an experimenter or tester. We then *mirror* the test device within the noVNC/VNC session and limit access to only this visual element. In Android, this is achieved using `scrcpy` [15], a screen mirroring utility which runs atop of ADB. No equivalent software exists for iOS, but a similar functionality can be achieved combining AirPlay Screen Mirroring [6] with (virtual) keyboard keys (see Section 3.3).

Figure 1(c) shows a snapshot of the graphical user interface (GUI) we have built around the default `noVNC` client. The GUI consists of an *interactive area* and a *toolbar*. The interactive area (bottom of the figure) is the area where a device screen is mirrored. As a user (experimenter or tester) hovers his/her mouse within this area, (s)he gains access to the device currently being mirrored, and each
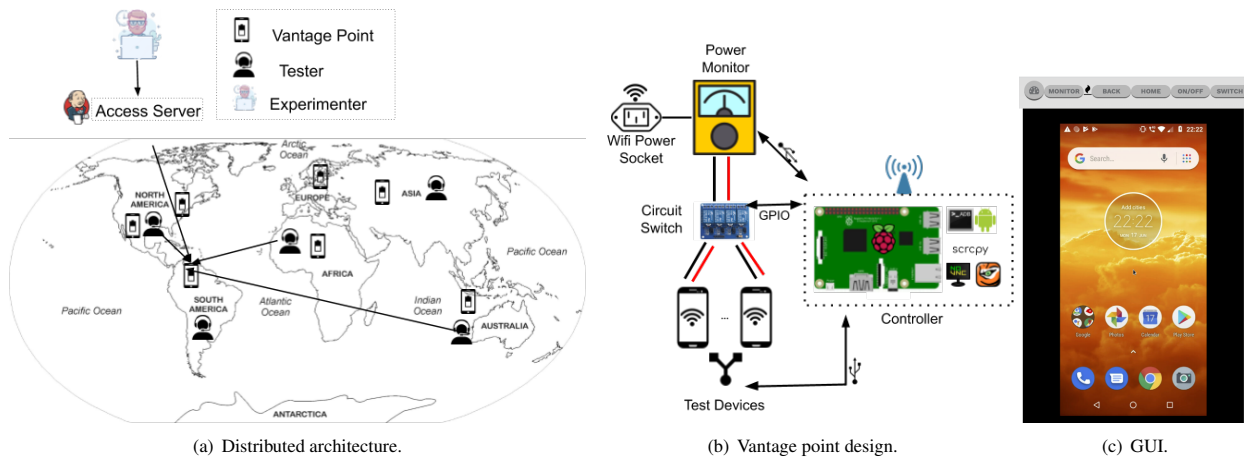
(a) Distributed architecture.   (b) Vantage point design.   (c) GUI.

Figure 1: BatteryLab's infrastructure.

action is executed on the physical device. The GUI connects to the controller's backend using AJAX calls to some internal restful APIs. The toolbar occupies the top part of the GUI, and implements a convenient subset of BatteryLab's API (see Table 1). Even though the toolbar was initially thought as a visual helper for an *experimenter*, it is also useful for less experienced *test participants*. For this reason, BatteryLab allows an experimenter to control the presence or not of the toolbar on the webpage to be shared with a test participant.

Power Monitor – This is a power metering hardware capable of measuring the current consumed by a test device in high sampling rate. BatteryLab currently supports the Monsoon HV [26], a power monitor with a voltage range of 0.8V to 13.5V and up to 6A continuous current sampled at $5KHz$. The Monsoon HV is controlled using its Python API [27]. Other power monitors can be supported, granted that they offer APIs to be integrated with BatteryLab's software suite.

Test Device(s) – It is a mobile device (phone or tablet) that can be connected to a power monitor. While we recommend phones with removable batteries, more complex setups requiring to (partially) tear off a device to reach the battery are possible. Note that, on Android, device mirroring is only supported on devices running API 21 (Android ≥ 5.0).

Circuit Switch – This is a relay-based circuit with multiple channels that lies between the test devices and the power monitor. The circuit switch is connected to the controller's GPIO interface and all relays can be controlled via software from the controller. Each relay uses the device's voltage (+) terminal as an input, and programmatically switches between the battery's voltage terminal and the power monitor's `Vout` connector. Ground (-) connector is permanently connected to all devices' Ground terminals.

This circuit switch has two main tasks. First, it allows to switch between a direct connection between the phone and its battery, and the "battery bypass"—which implies disconnecting the battery and connecting to the power monitor. This is required to allow the power monitor to measure the current consumed during an experiment. Second, it allows BatteryLab to concurrently support multiple test device without having to manually move cables around.

API	Description	Parameters
list_devices	List ADB ids of test devices	-
device_mirroring	Activate device mirroring	device_id
power_monitor	Toggle Monsoon power state	-
set_voltage	Set target voltage	voltage_val
start_monitor	Start battery measurement	device_id, duration
stop_monitor	Stop battery measurement	-
batt_switch	(De)activate battery	device_id
execute_adb	Execute ADB command	device_id, command

Table 1: BatteryLab's API.

WiFi Power Socket – This is used to allow the controller to turn the Monsoon on and off, when needed. It connects to the controller via WiFi and it is controlled with some simple API. The current BatteryLab software suite only supports Meross power sockets by integrating the following APIs [14]. In the near future we will replace this power socket by extending the capabilities of the circuit switch.

3.3 Automation

BatteryLab supports three mechanisms for test automation, each with its own set of advantages and limitations.

Android Debugging Protocol (Android) – ADB [17] is a powerful tool/protocol to control an Android device. Commands can be sent over USB, WiFi, or Bluetooth. While USB guarantees highest reliability, it interferes with the power monitor due to the power sent to activate the USB micro-controller at the device. This is solved by sending commands over WiFi or Bluetooth. However, using WiFi implies not being able to run experiments leveraging the mobile network, and ADB-over-Bluetooth requires a rooted device. Based on an experimenter needs, BatteryLab can dynamically switch between the above automation solutions.

UI Testing (Android and iOS) – This solution uses UI testing frameworks (e.g., Android's user interface tests [18] or Apple's XCTest framework [7]), to produce a separate version of the testing app, configured with automated actions. The advantage of this solution, compared with ADB, is that it does not require a communication channel with the Raspberry Pi. The main drawback is that it restricts the set of applications that can be tested since access to an app source code is required.

Bluetooth Keyboard (Android and iOS) – This approach automates a test device via (virtual) keyboard keys (e.g., locate an app, launch it, and interact with it). The controller emulates a typical keyboard service to which test devices connect via Bluetooth. This approach is generic and thus works for both Android and iOS devices, with no rooting needed. Since it relies on Bluetooth, it also enables experiments on the cellular network. The limitations are twofold. First, Android device mirroring is not supported as it requires ADB. This is not an issue for automated tests which can and should be run in *headless* mode to minimize noise on the battery reporting (see Figure 2). It follows that this limitation only applies to usability testing (with real users) on a mobile network.

The second limitation is that the level of automation depends both on the OS and app support for keyboard commands. In Android, it can be challenging to match ADB's API with this approach. It should be noted though that, when available, ADB can still be used "outside" of a battery measurement. That is, operations needed before and after the actual battery measurement (e.g., cleaning an app cache) can still be realized using ADB over USB. When the actual test starts, e.g., launch an app and perform some simple interactions, we can then switch to Bluetooth keyboard automation.

3.4 How to Join?

Institutions interested in joining BatteryLab can do so by following our tutorial [9]. In short, we recommend the hardware to use and its setup. It is important for the controller to be publicly reachable at the following configurable ports: 2222 (SSH, access server only), 8080 (GUI's backend), 6081 (noVNC). Members will provide a human readable identifier for the vantage point which will be added to BatteryLab's DNS (e.g., `node1.batterylab.dev`) provided by Amazon Route53 [3]. Our wildcard `letsencrypt` [22] certificate will be provided at this point. Renewal of this certificate is managed by the access server which also automatically deploys it at each vantage point, when needed.

The next step consists of flashing the controller (Raspberry Pi) with BatteryLab's image. This will setup the most recent Raspbian version, along with BatteryLab's required code and its configuration. Few manual steps are required to verify connectivity, grant pubkey access to the access server, and connect at least one Android device. At this point, the controller should be visible at the access server, and the device accessible at `https://node1.batterylab.dev`.

4 PRELIMINARY EVALUATION

This section preliminarily evaluates BatteryLab using its first vantage point deployed at Imperial College London, UK. This consists of a Monsoon power meter, a Samsung J7 Duo (Android 8.0), a Raspberry Pi 3B+, and a Meross power socket. We first evaluate BatteryLab's *accuracy* in battery measurements reporting. Next, we demonstrate its usage investigating a simple research question. We further use this demonstration to benchmark BatteryLab's system
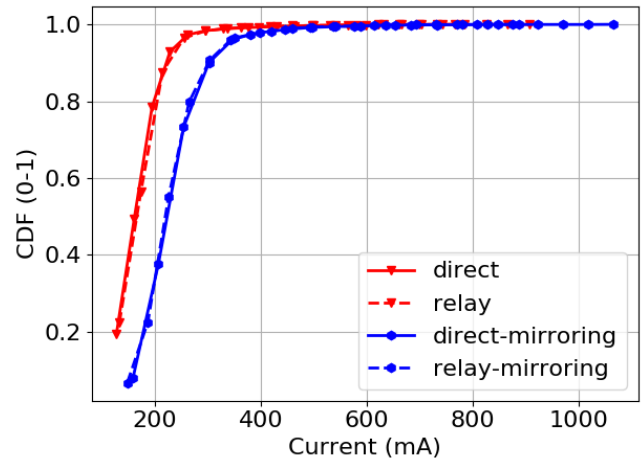
Figure 2: CDF of current drawn
(direct, relay, direct-mirroring, relay-mirroring).

performance. Finally, we experiment with the impact of multiple device *locations* emulated via a VPN.
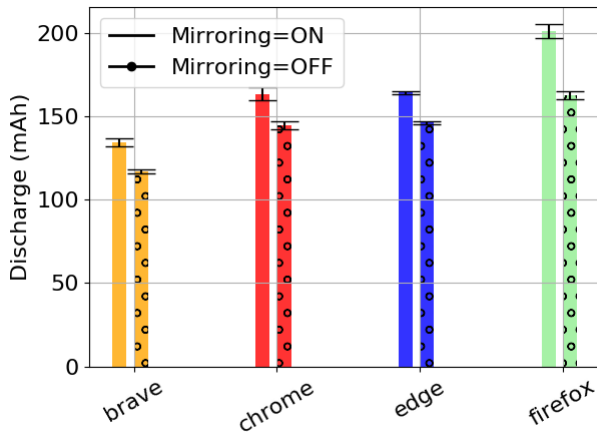
4.1 Accuracy

Compared to a classic *local* setup for battery measurements, BatteryLab introduces some hardware (circuit relay) and software (device mirroring) components that can impact the *accuracy* of the measurements collected. We devised a simple experiment where we compare three scenarios. First, a *direct* scenario consisting of just the Monsoon power meter, the testing device, and the Raspberry Pi to instrument the power meter. For this setup, we strictly followed Monsoon indications [26] in terms of tape, cable type and length, and connectors to be used. Next, we introduce two additional scenarios: a *relay* scenario where the relay circuit is used to enable BatteryLab's programmable switching between multiple devices as well as between battery bypass and regular battery operation (see Section 3.2). Finally, a *mirroring* scenario where the device screen is mirrored to an open noVNC session. While the relay is always "required" for BatteryLab to properly function, device mirroring is only required for usability testing.

Figure 2 shows the Cumulative Distribution Function (CDF) of the current consumed in each of the above scenarios during a 5 minutes test. For completeness, we also consider a *direct-mirroring* scenario where the device is directly connected to Monsoon and screencasting is active. During the test, we play an mp4 video preloaded on the device sdcard. The rationale is to force the device mirroring mechanism to constantly update as new frames are originated. The figure shows negligible difference between the "direct" and "relay" scenario, regardless of the device mirroring being active or not. A larger gap (median current grows from 160 to 220mA) appears with device mirroring. This is because of the background process responsible of screencasting to the controller which causes additional CPU usage on the device (Figure 4).

4.2 Demonstration

We demonstrate BatteryLab's usage assuming an experimenter asks the following question: *which of today's Android web-browsers is the most energy efficient?* The experimenter writes an automation script which instruments a browser to load a webpage and interact
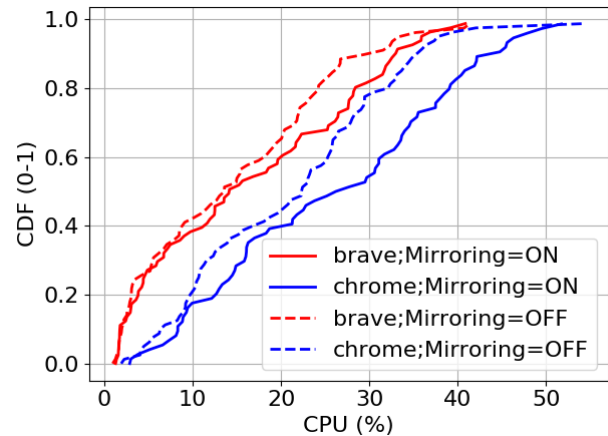
Figure 3: Per browser energy consumption (Brave, Chrome, Edge, Firefox).

Figure 4: CDF of CPU consumption (Brave and Chrome).

Figure 5: CDF of CPU consumption at the controller (Raspberry Pi 3B+).

with it. Scripts are deployed via BatteryLab's Jenkins interface, and phone access is granted via device mirroring in the experimenter's browser. When satisfied with the automation, the experimenter can launch a real test with active battery monitoring. The experiment is added to Jenkin's queue and will run when the right conditions are met, *i.e.,* no other test is running (required) and low CPU utilization (optional). When an experiment completes, logs can be retrieved via the Jenkins interface.

We build browser automation using bash and BatteryLab's ADB over WiFi automation procedure. We automate a few popular Android browsers: Chrome, Firefox, Edge, and Brave. Our experiments are WiFi only since the device under test is not rooted. Each browser is instrumented to sequentially load 10 popular news websites. After a URL is entered, the automation script waits 6 seconds – emulating a typical page load time (PLT) for these websites under our (fast) network conditions – and then interact with the page by executing multiple "scroll up" and "scroll down" operations. Before the beginning of a workload, the browser state is cleaned and the required setup is done, e.g., Chrome requires at first launch to accept some conditions, sign-in into an account or not, etc. We iterate through each browser sequentially, and re-test each browser 5 times. We repeat the full experiment with both active and inactive device mirroring.

Browser Performance Figure 3 shows the average battery discharge (standard deviation as errorbars) measured for each browser, considering both active and inactive device mirroring. The figure shows that, regardless of device mirroring, the overall result does not change, *i.e.,* with Brave offering minimal battery consumption and Firefox consuming the most. This is because device mirroring offers a constant extra cost (~20mAh) regardless of the browser being tested. This result is in line with the constant gap observed between active and inactive device mirroring in Figure 2.

This additional battery consumption caused by device mirroring is due to an increase of the CPU load on the device under test. Figure 4 shows the CDF of the CPU utilization for Chrome and Brave with active and inactive device mirroring, respectively. A similar trend is observed for the other browsers, which have been omitted to increase the plot visibility. The figure shows two results. First, Brave's lower battery consumption comes from an overall lower CPU pressure, e.g.,
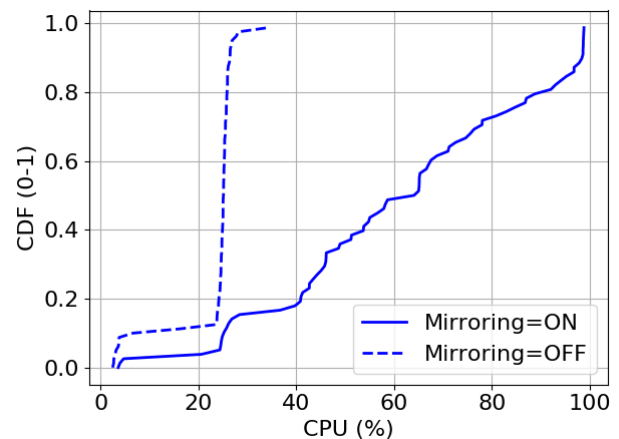
a median CPU utilization of 12% versus 20% in Chrome. Second, device mirroring causes, for both browsers, a 5% CPU increase. This is more noticeable at higher CPU values which is when the browser's automation is active. This happens because of the increasing load on the encoder when the screen content changes quickly versus, for example, the fixed phone's home screen.

System Performance Overall, higher CPU utilization is the main extra cost caused by device mirroring (extra 50%, on average). The impact on memory consumption is minimal (extra 6%, on average). Overall, memory does not appear to be an issue given less than 20% utilization of the Raspberry Pi's 1 GB. The networking demand is also minimal, with just 32 MB of upload traffic for a ~7 minutes test. Note that we set `scrcpy`'s video encoding (H.264) rate to 1 Mbps, which produces an upper bound of about 50 MB. The lower value depends on extra compression provided by `noVNC`.

Evaluating the responsiveness of device mirroring is challenging. We call *latency* the time between when an action is requested, either via automation or a click in the browser, and when the consequence

Speedtest Server (kms)	D (Mbps)	U (Mbps)	L (ms)
South Africa Johannesburg (3.21)	6.26	9.77	222.04
China Hong Kong (4.86)	7.64	7.77	286.32
Japan Bunkyo (2.21)	9.68	7.76	239.38
Brazil Sao Paulo (8.84)	9.75	8.82	235.05
CA, USA Santa Clara (7.99)	10.63	14.87	215.16

Table 2: ProtonVPN statistics. D=down/U=up/L=RTT.

of this action is displayed back in the browser, after being executed on the device. This depends on many factors like network latency (between browser and test device), load on device and/or controller, and software optimizations. We estimate such latency recording audio (44,100 Hz) and video (60 fps) while interacting with the device via the browser. We then manually annotated the video using ELAN multimedia annotator software [36] and compute the latency as the time between a mouse click (identified via sound) and the first frame with a visual change in the app. We repeat this test 40 times while co-located with the vantage point (1 ms network latency) and measure an average latency of 1.44 (±0.12) sec.

Next, we dig deeper into CPU utilization at the controller. Figure 4 shows the CDF of the CPU utilization during the Chrome experiments with active and inactive device mirroring — no significant difference was observed for the other browsers. When device mirroring is inactive, the controller is mostly underloaded, *i.e.,* constant CPU utilization at 25%. This load is caused by the communication with the Monsoon to pull battery readings at highest frequency. When device mirroring is enabled, the median load instead increases to about 75%. Further, in 10% of the measurements the load is quite high and over 95%.

4.3 Location, Location, Location

BatteryLab's distributed nature is both a *feature* and a *necessity*. It is a feature since it allows battery measurements under diverse network conditions which is, to the best of our knowledge, an unchartered research area. It is a necessity since it is how the platform can scale without incurring high costs. We here explore the impact of network location on battery measurements. In the lack of multiple vantage points, we emulate such network footprint via a VPN.

We acquired a basic subscription with ProtonVPN [31] and set it up at the controller. We then choose 5 locations from where to tunnel our tests. Table 2 summarizes the locations chosen, along with some statistics derived from SpeedTest (upload bandwidth, download bandwidth, and latency). VPN vantage points are sorted by download bandwidth, with the South Africa node being the slowest and the California node being the fastest. Since the speedtest server is always within 10 km from each VPN node, the latency here reported is mostly representative of the network path between the vantage point and the VPN node.

Next, we extend the automation script to also activate a VPN connection at the controller before testing. Figure 6 shows the average battery discharge (standard deviation as errorbars) per VPN location and browser — for visibility reasons and to bound the experiment duration, only Chrome and Brave were tested. Overall, the figure does not show dramatic differences among the battery measurements
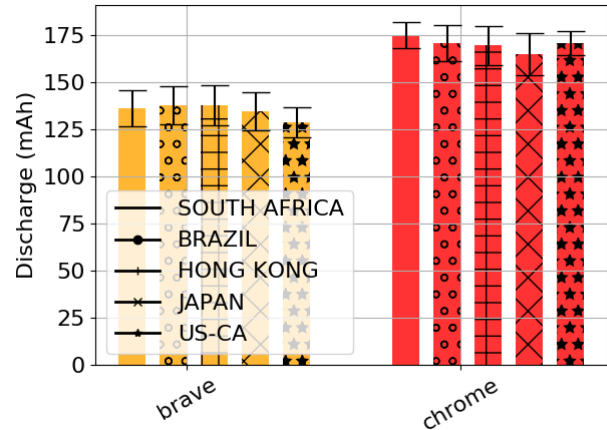
Figure 6: Brave and Chrome energy consumption measured through VPN tunnels.

as a function of the network location. For example, while the available bandwidth almost double between South Africa and California, the average discharge variation stays between standard deviation bounds. This is encouraging for experiments where BatteryLab's distributed nature is a *necessity* and its noise should be minimized.

Figure 6 also shows an interesting trend when comparing Brave and Chrome when tested via the Japanese VPN node. In this case, Brave's energy consumption is in line with the other nodes, while Chrome's is minimized. This is due to a significant (20%) drop in bandwidth usage by Chrome, due to a systematic reduction in the overall size of ads shown at this location. This is an interesting result for experiments where BatteryLab's distributed nature is a *feature*.

Anecdotally, we also noticed that Google's lite pages [16] were on by default in South Africa and Japan, for Chrome. Google mentions that this decision is driven by low bandwidth rather than location, which does not necessarily match our measurements (see Table 2). While we turned this feature off to ensure comparable tests, we also noticed that none of the tested pages currently support this feature.

5 CONCLUSION AND FUTURE WORK

In this paper we have proposed BatteryLab, a distributed measurement platform for battery measurements. We have also started building and experimenting with BatteryLab, to the point that our system is ready to accept new members. We specifically focused on Android because of ease of integration and availability of testing tools. However, we discussed iOS solutions which we soon plan to experiment with. Similarly, while we focus on mobile devices there is no fundamental constraint which would not allow BatteryLab to support laptops or IoT devices. We designed BatteryLab to enable remote access and human-controlled tests; we plan to facilitate such tests via integration with platforms like Mechanical Turk [4] and Figure Eight [13]. Our vision is an open source and open access platform that users can join by sharing resources. However, we anticipate potential access via a credit system for experimenters lacking the resources for the initial setup.

ACKNOWLEDGMENTS

REFERENCES

[1] Ö. Alay, A. Lutu, M. Peón-Quirós, V. Mancuso, T. Hirsch, K. Evensen, A. Hansen, S. Alfredsson, J. Karlsson, A. Brunstrom, et al. Experience: An open platform for experimentation with commercial mobile broadband networks. In *Proc. ACM MobiCom*, pages 70–78, 2017.

[2] M. Almeida, M. Bilal, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Varvello, and J. Blackburn. Chimp: Crowdsourcing human inputs for mobile phones. In *Proc. of WWW*, pages 45–54, 2018.

[3] Amazon Inc. A reliable and cost-effective way to route end users to Internet applications. https://aws.amazon.com/route53/.

[4] Amazon Inc. Amazon Mechanical Turk. https://www.mturk.com/.

[5] Amazon Inc. AWS Device Farm. https://aws.amazon.com/device-farm/.

[6] Apple Inc. How to AirPlay video and mirror your device's screen. https://support.apple.com/HT204289.

[7] Apple Inc. XCTest - Apple Developer Documentation. https://developer.apple.com/documentation/xctest.

[8] BatteryLab. A Distributed Platform for Battery Measurements. https://batterylab.dev.

[9] BatteryLab. Batterylab tutorial for new members. https://batterylab.dev/tutorial/blab-tutorial.pdf.

[10] D. H. Bui, Y. Liu, H. Kim, I. Shin, and F. Zhao. Rethinking energy-performance trade-off in mobile web page loading. In *Proc. ACM MobiCom*, 2015.

[11] Y. Cao, J. Nejati, M. Wajahat, A. Balasubramanian, and A. Gandhi. Deconstructing the energy consumption of the mobile page load. *Proc. of the ACM on Measurement and Analysis of Computing Systems*, 1(1):6:1–6:25, June 2017.

[12] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone energy drain in the wild: Analysis and implications. In *Proc. ACM SIGMETRICS*, 2015.

[13] Figure Eight. The Essential Data Annotation Platform. https://www.figure-eight.com.

[14] A. Geniola. Simple Python library for Meross devices. https://github.com/albertogeniola/MerossIot.

[15] Genymobile. Display and control your Android device. https://github.com/Genymobile/scrcpy.

[16] Google. Lite page by Google. https://www.ghacks.net/2019/03/14/chrome-lite-pages/.

[17] Google Inc. Android Debug Bridge. https://developer.android.com/studio/command-line/adb.

[18] Google Inc. Android Developers - Automate user interface tests. https://developer.android.com/training/testing/ui-testing.

[19] Greenspector. Test in the cloud with real mobile devices. https://greenspector.com/en/.

[20] C. Hwang, S. Pushp, C. Koh, J. Yoon, Y. Liu, S. Choi, and J. Song. Raven: Perception-aware optimization of power consumption for mobile games. In *Proc. ACM MobiCom*, 2017.

[21] Jenkins. The leading open source automation server. https://jenkins.io/.

[22] Let's Encrypt. A free, automated, and open Certificate Authority. https://letsencrypt.org.

[23] Microsoft, Visual Studio. App Center is mission control for apps. https://appcenter.ms/sign-in.

[24] Mobile Enerlytics. The Leader In Automated App Testing Innovations To Reduce Battery Drain. http://mobileenerlytics.com/.

[25] MONROE - H2020-ICT-11-2014. Measuring Mobile Broadband Networks in Europe. https://www.monroe-project.eu/wp-content/uploads/2017/12/Deliverable-D2.2-Node-Deployment.pdf.

[26] Monsoon Solutions Inc. High voltage power monitor. https://www.msoon.com.

[27] Monsoon Solutions Inc. Monsoon Power Monitor Python Library. https://github.com/msoon/PyMonsoon.

[28] Mvp - github. uhubctl - USB hub per-port power control. https://github.com/mvp/uhubctl.

[29] noVNC. A VNC client JavaScript library as well as an application built on top of that library. https://novnc.com.

[30] PlanetLab. An open platform for developing, deploying, and accessing planetary-scale services. https://www.planet-lab.org/.

[31] ProtonVPN. High-speed Swiss VPN that safeguards your privacy. https://protonvpn.com/.

[32] Raspberry Pi. Raspberry Pi 3 Model B+. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[33] A. Schulman, T. Schmid, P. Dutta, and N. Spring. Phone power monitoring with battor. In *Proc. ACM MobiCom*, 2011.

[34] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery?: Analyzing mobile browser energy consumption. In *Proc. of WWW*, 2012.

[35] TigerVNC. A high-performance, platform-neutral implementation of VNC (Virtual Network Computing). https://tigervnc.org.

[36] P. Wittenburg, H. Brugman, A. Russel, A. Klassmann, and H. Sloetjes. Elan: a professional framework for multimodality research. In *Proc. of LREC*, volume 2006, page 5th, 2006.