Experience: Android Resists Liberation from Its Primary Use Case

Noah Klugman[†], Veronica Jacome[†], Meghan Clark[†], Matthew Podolsky[†],
Pat Pannuto[†], Neal Jackson[†], Aley Soud Nassor[‡], Catherine Wolfram[†],
Duncan Callaway[†], Jay Taneja[¶], and Prabal Dutta[†]

[†]University of California, Berkeley, [‡]The State University of Zanzibar, [¶]University of Massachusetts, Amherst

ABSTRACT

Network connectivity is often one of the most challenging aspects of deploying sensors. In many countries, cellular networks provide the most reliable, highest bandwidth, and greatest coverage option for internet access. While this makes smartphones a seemingly ideal platform to serve as a gateway between sensors and the cloud, we find that a device designed for multi-tenant operation and frequent human interaction becomes unreliable when tasked to continuously run a single application with no human interaction, a somewhat counter-intuitive result. Further, we find that economy phones cannot physically withstand continuous operation, resulting in a surprisingly high rate of permanent device failures in the field. If these observations hold more broadly, they would make mobile phones poorly suited to a range of sensing applications for which they have been rumored to hold great promise.

ACM Reference Format:

Noah Klugman, Veronica Jacome, Meghan Clark, Matthew Podolsky, Pat Pannuto, Neal Jackson, Aley Soud Nassor, Catherine Wolfram, Duncan Callaway, Jay Taneja, and Prabal Dutta. 2018. Experience: Android Resists Liberation from Its Primary Use Case. In *MobiCom '18: 24th Annual Int'l Conf. on Mobile Computing and Networking, Oct. 29–Nov. 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3241539.3241583

1 INTRODUCTION

This paper presents our experiences with a four-month deployment of Android phone-based cellular gateways in Zanzibar, Tanzania. The deployment explored the feasibility of an accurate, inexpensive, and real-time AC power measurement system built around a commercial off-the-shelf (COTS) plug load power monitor and smartphone, shown in Figure 1. This work highlights the experiences, lessons learned, and empirical evaluation of the viability of using COTS Android phones as gateways for sensor deployments in the real world.

Cellular networks provide a global, low-cost, reliable, and high-bandwidth network backhaul that is ideally suited for the data collection, maintenance, and operational needs of remote sensor nodes [10, 23]. Unfortunately, for many applications, sensor nodes are not able to directly connect to a cellular network. Cellular modems draw considerable power, add monetary cost, impose form factor constraints, and increase implementation complexity. Additionally, they require a SIM card, which must be registered and maintained with a cellular network provider or virtual network operator. When per-node access to a cellular network is not a viable option, a gateway architecture allows multiple sensor nodes to access a cellular network by connecting to a single cellular gateway via a short-range, lower-energy, and lower-cost communications link such as WiFi or Bluetooth Low Energy [16].

Smartphones are attractive as gateways because they connect to the cellular network and offer a rich application platform, powerful processors, WiFi and Bluetooth radios, large storage capabilities, complementary integrated sensors, over-the-air updates, and a well-known and multilingual user interface, all with a price point only achievable at scale. Further, contract-enabled upgrades and rapid market growth have resulted in potentially gateway-capable phones sitting unused and unrecycled [11, 13, 27]. Repurposing smartphones as gateways could extract value from hundreds of millions of devices currently considered to be e-waste [28].

Unfortunately, our experience suggests that many of the expected benefits from building a phone-based gateway are either unobtainable or difficult to achieve in practice. Issues arise from the fundamental tension of converting a general-purpose, user-facing platform into a single-purpose, remotely-supervised device. Android optimizations, like automatic process termination, which improve the normal user experience, are problematic for long-running gateway apps. Ensuring unsupervised recovery after failures is difficult on

MobiCom'18, October 29–November 2, 2018, New Delhi, India
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5903-0/18/10.
https://doi.org/10.1145/3241539.3241583

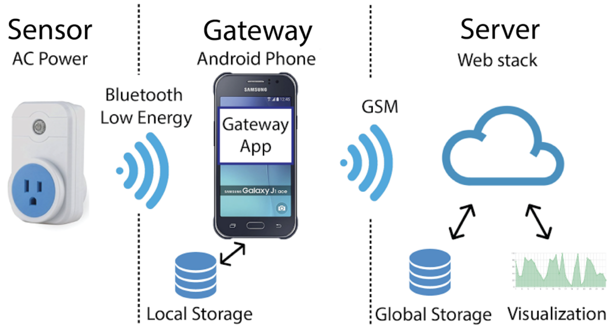


Figure 1: System Architecture. A commercial off-the-shelf plug-load power meter (WiTenergy E110) uses an unmodified Android phone (Samsung J1 Ace running Android 4.4.4) in 16 households in Zanzibar, Tanzania. A gateway app connects to the power meter over Bluetooth Low Energy and forwards the power data from the sensor to a cloud server. The gateway app also appends metadata such as GPS, system time, sensor ID, and gateway software version. The phone is encased in a plastic box and the system runs unattended.

a platform that assumes the existence of a user to initiate actions and respond to prompts. Since most phone users have a small number of devices which they manage directly, infrastructure to manage small cellular fleets can be limited, expensive, and dependent on the newest versions of a mobile OS [8, 22, 24]. Moreover, the challenges of simply keeping a cellular phone connected to a cellular network are nontrivial. We also encounter other significant issues, including bugs in the Android operating system (particularly the Bluetooth Low Energy stack) and physical failures of Android hardware. As a result, we conclude that the benefits of using COTS Android devices may not outweigh the costs of tricking Android into supporting long-running, always-on applications with no human interaction.

Even with these experiences, we are confident that cellular networks will remain a critical backhaul for sensing at scale. Dedicated cellular gateway hardware, such as the emerging class of advanced multi-radio routers [1] provide a path to support economical sensors and provide them with global connectivity. Smartphone sensing applications have also been successful when they work inside their host platform's intended multi-tenant design. For example, augmenting daily usage phones with sensing applications, such as those for wearable fitness trackers, has been widely commercially adopted [4]. Rich and well-supported application programming environments have not existed historically on gateways; however, environments like the Arm Mbed IoT Device Platform [20] and the Particle Device OS and Cloud [29] may soon close this gap.

We conclude by identifying high-level Android changes that would free discarded phones from the limitations of user-centric, multi-tenant design. These changes could be distributed using existing channels, allowing abandoned phones across the world to be more easily repurposed as single-tenant application gateways for the next generation of sensor networks.

2 RELATED WORK

Academic and commercial projects alike have successfully leveraged smartphones as gateways [3, 4, 12, 15, 21]. For example, the Pogo middleware project was deployed for 24 days on Android 2.1 and allowed phones to be more easily used as always-on sensors [3]. Similarly, Cenceme was deployed for 21 days on the Nokia 95 platform to perform continuous sensing and activity classification [21]. Piggyback Crowd Sensing ran for 25 days as a background system service on unspecified Android phones [19].

However, these applications have frequently operated under the assumption of *tight association* between a phone and a human, depending on the user to closely monitor and maintain the application's health. In 2012, Beschastnikh et. al. proposed recycling smartphones under a *loose association* paradigm [2], which relaxes the assumption of frequent interactions between the human and the phone. In a loose association context, phones are mostly stationary, and apps are continuously-running, unsupervised, and characterized by machine-to-machine interactions. Repurposing old smartphones as long-running gateways would provide new life to abandoned hardware and keep potentially hundreds of millions of phones from becoming toxic e-waste.

Prior work has not evaluated the feasibility of using phones as gateways under the assumption of loose association. Pogo, Cenceme, and Piggyback Crowd Sensing assumed each phone would be carried on a user, and relied on that user to clear messages and ensure that the phone continued to function. Additionally, each application ran for only a few weeks, potentially falling short of finding temporally distant failure states that could be catastrophic in a long-running deployment. These applications also may have benefited from running on fairly immature operating systems that had not yet begun to limit functionality of long-running processes to optimize for user experience and privacy [31, 33, 35].

Android Things, a branch of Android that supports embedded systems, may offer solutions, but is hard to port and install, does not support OTA updates through Google Play, and is on a less active development timeline [32]. PhoneLab took a promising approach to supporting loose association applications on COTS phones by modifying the Android platform to circumvent many of the issues stemming from OS features meant to protect the user experience, including garbage collection, restrictive permissions, and limited application level APIs. Even with these modifications, PhoneLab still relies on human intervention to overcome many types of failure conditions, breaking loose association [26].

In contrast, we believe that this is the first work to extensively report on the suitability of using stock Android to host gateway applications on phones deployed in a loose association context. While we did not initially set out to explore this question, we were surprised by the degree to which the loose association conditions of our field research fundamentally impacted Android's ability to function as a sensor gateway, motivating us to share our experiences and conclusions.

3 DESIGN CONSIDERATIONS

Our exploration with phone-based sensor gateways was part of a deployment called PlugWatch. The study investigated:

(i) how power utility customers without advanced metering infrastructure perceive distribution-level energy reliability,
(ii) how power quality varies over time of day and as a function of distance to a transformer,
(iii) whether a low-cost cellular meter could be a reliable source of the data needed to answer these questions.

The PlugWatch deployment periodically sensed voltage, frequency, current, and on/off power state at an AC plug, and transmitted this information to a central service over the GSM cellular network. In collaboration with the Zanzibar Electricity Corporation, sixteen PlugWatch units were deployed in two villages in Zanzibar for four months.

The PlugWatch architecture is built on a COTS Android smartphone and a COTS plug load power meter, as shown in Figure 1. The decision to build around a smartphone was made with several expected benefits:

Reduce Development Time. By using a smartphone as the primary CPU, local database, and radio, we expected to reduce development time and costs compared to designing custom hardware. A phone's components have been integrated at the hardware and system software level, leaving only app composition, which benefits from mature development tools and comprehensive OS services that expose the device's resources in a general-purpose manner.

Increase Reliability. Android and associated Google Services such as the Play Store [34] and Firebase [7] offer over-the-air updates and remote monitoring and configuration out of the box, promising better deployment management tools than could easily be written for a custom system.

Ease Deployment Hurdles. The interactive screen on a smartphone gateway is conducive to debugging and verifying local network connectivity. Signal strength is displayed in the status bar, and browser apps make checking internet connectivity as easy as opening a webpage. Additionally, local phone shops know how to manually configure Access Point Name (APN) settings to connect smartphones to the internet over a cellular provider.

Contrary to our initial expectations, we find that using an Android smartphone as a networking gateway for sensing applications *increases* development time, *decreases* reliability, and does not significantly reduce deployment hurdles. These findings are discussed in Sections 5 and 6.

3.1 Requirements

The design of the PlugWatch system, including the phone-based gateway, met a number of functional requirements:

Minimal Human Interaction. We decided to keep humans out-of-the-loop as much as possible over the course of the deployment to reduce the burden of participation and remove noise from incorrect human operator actions.

Reliably Long-Running. Our experiment was intended to measure variance in voltage and power quality over the course of many months. To do this, we designed PlugWatch to run 24 hours a day. PlugWatch restarts the app or reboots the phone if the app hangs, fails to receive sensor data, or is notified of process or wireless stack failure.

Non-Obtrusive. PlugWatch was designed to be placed into households that often had few physical barriers to separate PlugWatch from participants. Because of this, we designed PlugWatch not to make noise, display lights, or otherwise cause annoyance for the participants. We also disabled all system sounds and vibrations in Android.

Real-Time Data Stream. Our utility partners were interested in receiving plug-load information in real time. This would allow PlugWatch to act as a simple prototype of a more traditional smart meter, and would let the utility experiment with how they might ingest and react to this new data stream. Further, this data stream gave us insight into the performance of PlugWatch, which was helpful for deployment management activities.

Low Budget. Our budget was modest and targeted a complete per-unit system cost of approximately $100 USD. This partially motivated our selection of what was a economical mid-range phone available in the local geography.

4 IMPLEMENTATION DETAILS

The implementation consisted of the fully-assembled PlugWatch unit and the data endpoints, as described below.

4.1 PlugWatch Unit

Each PlugWatch unit was comprised of several components: the physical connections and enclosure, the WiTenergy plug load sensor, the phone (including the most current available version of Android, an SD card, a SIM card, and a factory included battery), and the PlugWatch gateway app.

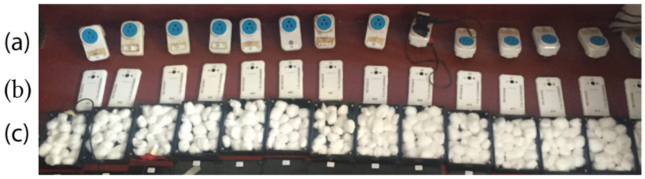
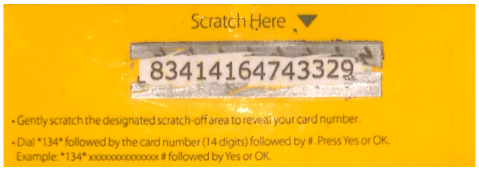


Figure 2: Plug load monitors and Android gateways before deployment. (a) The WiTenergy plug-load meters plug into an available wall outlet. (b) A phone plugs into the WiTenergy meter via a standard 5V DC USB adapter. (c) The cord from the adapter runs into a plastic box, which is padded with cotton balls and screwed shut. The phone sits in the box and runs the gateway application, forwarding data continuously from the WiTenergy to the cloud.

4.1.1 Physical Design. To discourage unintended participant interaction, we placed each phone in a black plastic box. The box was filled with cotton to prevent the phone from freely moving. Finally, we placed stickers on the box instructing participants to not open it and screwed the box shut. A standard 5 V Micro-USB charger was connected to the phone, run through a notch cut into the box, and plugged into a WiTenergy E110 plug load meter. Each plug point was taped over to reduce the likelihood of accidental disconnection. The phone, box, plug, and WiTenergy were all labeled with a common unique ID. The pre-deployment PlugWatch units are shown in Figure 2.

4.1.2 The PlugWatch Phone. We ran the PlugWatch app on a Samsung J110H phone loaded with Android 4.4.4 [39]. We selected this phone because it was relatively low cost ($90 USD at time of purchase in Q4 2016), was widely available in-country, was of mid-range quality, and had reasonable resources for its price range (1.3 GHz, dual-core, 4 GB ROM). These phones came with Android 4.4.2 pre-installed, and we upgraded them to Android 4.4.4, the most current version available for this model at the time of deployment. The phone was packaged with an 1800 mAH EB-BJ111ABE battery [38]. We purchased a homogeneous fleet of phones to minimize variance. Each PlugWatch phone also had an SD card for local logging, and was logged into a Google Account with its Play Store configured for automatic updates.

We also selected the J110H because it could be modified to turn back on if power was restored to the phone after the battery died. This functionality was necessary to recover a phone after a power outage long enough to fully drain the phone's battery. We found after evaluating five different models that this modification (which requires replacing the charging image displayed when power is applied to a phone with a script to power on the phone) works exclusively with Samsung phones. The J110H was the least expensive Samsung phone available from local stores.


Scratch Here

83414164743329

• Gently scratch the designated scratch-off area to reveal your card number.

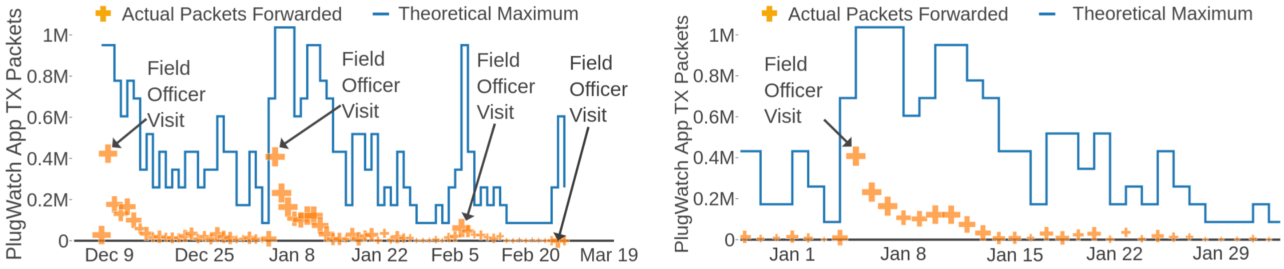
• Dial *134* followed by the card number (14 digits) followed by #. Press Yes or OK.
Example: *134* xxxxxxxxxxxxxx # followed by Yes or OK.

Figure 3: The back of an airtime card used to add airtime to a phone. A user scratches the card to reveal a code, then types in an Unstructured Supplementary Service Data (USSD) number, the # character, and the code. If the code is accepted, the user will get a notification that the account has received a set amount of airtime. If the user wishes to use this airtime to connect to the internet most economically, they enter a second USSD code and select an internet bundle, which provides data that sunsets after a certain amount of time. If the user has enough credit, the bundle will be purchased. A user also checks their account balance via USSD. This interface scales poorly but can be, as in Tanzania, the only one available, making management of a large number of cellular gateways highly labor-intensive.

We used KingoRoot [17] to root each phone. Root access allowed us to effect the aforementioned "boot from off upon restoration of power" behavior and enabled the PlugWatch app to reboot the phone, which was necessary to recover from some BLE driver failures. Finally, it allowed for us to install ClockworkMod recovery [5], a tool that enabled a gold-standard system image to be loaded onto each phone.

The smartphones were unlocked and could function on any of the mobile carriers in Zanizbar. All carriers offer prepaid data plans in which airtime is added to a SIM as needed and purchased as a code behind a scratch-off ticket, such as the one shown in Figure 3. This code is then manually entered into the phone over an Unstructured Supplementary Service Data (USSD) interface. Then USSD is used to purchase "bundles" that provide discounted rates for cellular data bandwidth and can last from a couple hours to 60 days. As all carriers had comparable feature sets and interfaces, we spoke with village residents to learn which carrier had the highest quality and most reliable service, and chose the carrier (Halotel [43]) based on these conversations.

4.1.3 The PlugWatch App. The PlugWatch app was written to keep an active long-running BLE connection to the WiTenergy sensor and to regularly collect power information from the sensor over this connection. Additionally, the app accepted API calls from our central service, logged state, automatically updated, sent periodic heartbeats, restarted on crashes, and rebooted the phone when necessary. The app's UI was designed to be inaccessible to non-technical users, and had much of its functionality hidden behind passwords so that participants could not change the state of the app.



(a) Packets forwarded over entire deployment

(b) Close-up for month of January

Figure 4: Number of power measurement packets forwarded from the WiTenergy plug load meter by the PlugWatch app to the cloud service. The area of the crosses is proportional to the number of phones reporting any packets each day. PlugWatch reports a power measurement at 1 Hz. Therefore, the daily theoretical maximum number of packets is calculated as *number of seconds in a day* × *number of phones reporting any packets during a day*. After each field officer's visit to perform smartphone maintenance activities, the number of measurements received increased dramatically, then declined rapidly over time. Field officers performed a number of tasks: rebooting the phone, checking its placement, re-establishing the Bluetooth connection, installing available updates, clearing any messages, and emptying the SD card.

Exception Handling. Each Java class in PlugWatch used the `DefaultUncaughtExceptionHandler` to catch all exceptions. On exception, a new thread would spawn and invoke a custom `restart` class, passing it the exception message and the name of the class that threw the exception. When the `restart` class woke up, it would either schedule a restart of the app or a reboot of the phone. The decisions to restart or reboot and when to do so were based on a counter, which allowed for a resetting backoff (i.e. app reset actions would be increasingly delayed until a max number of resets had occurred, at which point the phone would reboot and the counter would start over with a minimum delay).

Remote Monitoring. To facilitate remote monitoring of the deployment, we designed a management API into the PlugWatch app. This API performed three functions: querying state, requesting data, and manually resetting the system. API calls could be made over both Google Cloud Messaging [6] and SMS. This dual interface ensured that we could reach the phones as long as either data services or SMS services were online and the PlugWatch app was running.

Logs. To access in-depth system information, we maintained several different logs. Because Android's system logs are ephemeral, we implemented our own logging system, which wrote files to an SD card on the phone. Every application crash, incoming API message, and paired BLE MAC address was logged. Additionally, we kept a complete local copy of the database that contained all measurements from the WiTenergy device. Logs were remotely accessible via the PlugWatch API. This logging system was only available while the PlugWatch app was running.

Staying Alive. During development, we observed that when an app registered a background service—which is a service that runs even when the app UI is not being displayed—the OS would close this service after a period of time as part of garbage collection activities. We used four techniques to try to circumvent the automatic garbage collection process:

(i) The app periodically generated notifications in the status bar, transforming the background service that contains the BLE connection and data forwarding logic into a foreground service that is less likely to be killed by the OS under memory pressure [36].
(ii) The app forced the OS to keep the UI open on the screen by restarting the app on crash and boot, disabling navigation buttons, and registering callbacks that relaunched the app whenever the UI was hidden.
(iii) A separate watchdog process spawned on first open and periodically woke up and restarted the primary app process if it was no longer running.
(iv) The app used Android accessibility services [30] to catch and automatically close all dialog boxes launched by the OS by using a heuristic of soft clicking either the only button or the leftmost button in the pop-up. This heuristic was sufficient to dismiss all system dialog boxes encountered during testing.

We note that we are not the first to identify these techniques. These methods were described in various forum posts by developers also struggling to develop loose association applications.

4.2 Data Endpoints

Data were sent to the cloud through HTTP POST requests from the app. These were received by a Node.js server and logged. Debugging information was also sent through the Firebase Realtime Database API [7]. When the phone could not reach the network, it queued both HTTP data and Firebase packets until connectivity was restored. Received measurement data were stored in a PostgreSQL database and an InfluxDB database [14], and visualized using Grafana [18].

5 EVALUATION

We evaluate PlugWatch and determine that adding human interaction increased availability. We then consider the restart and reboot rate, which was much higher than we experienced during testing, and trace the logs captured from each of these events to glean insights. Finally, we consider the physical state of PlugWatch post-deployment and find screen burn-in and significant swelling in the batteries.

5.1 How Did PlugWatch Perform?

We analyze performance as a function of the number of packets sent daily. The WiTenergy sensor sends packets over BLE at 1 Hz, leading to 1,382,400 expected packets from our fleet daily (86,400 seconds in a day × 16 phones). As seen in Figure 4, the two highest yield days of the deployment did not achieve even half of the theoretical maximum. When we collected the WiTenergy meters at the end of the deployment, the devices were functional, leading us to conclude most failures were due to the phones. These failures were not observed during lab tests in the US.

Variable Performance. Figure 4 shows roughly monthly spikes in aggregate packet delivery. These correspond to monthly field officer visits to each household to perform maintenance tasks including rebooting the phone, checking its placement, re-establishing the Bluetooth connection, installing available PlugWatch updates, clearing any messages, and emptying the SD card. We would expect those in-app routines that restart the app and reboot the phone upon errors to reset the state of PlugWatch and lead to the same increase in performance as the activities performed by field officers. However, we did not observe this in practice.

We have not been able to explain this discrepancy based on the logs gathered by the PlugWatch app. We only have logs from times when PlugWatch was running, so we have no visibility into the system state whenever the app failed to start. This opacity is enforced by Android's design, which writes logs to a ring buffer that is reset on reboot.

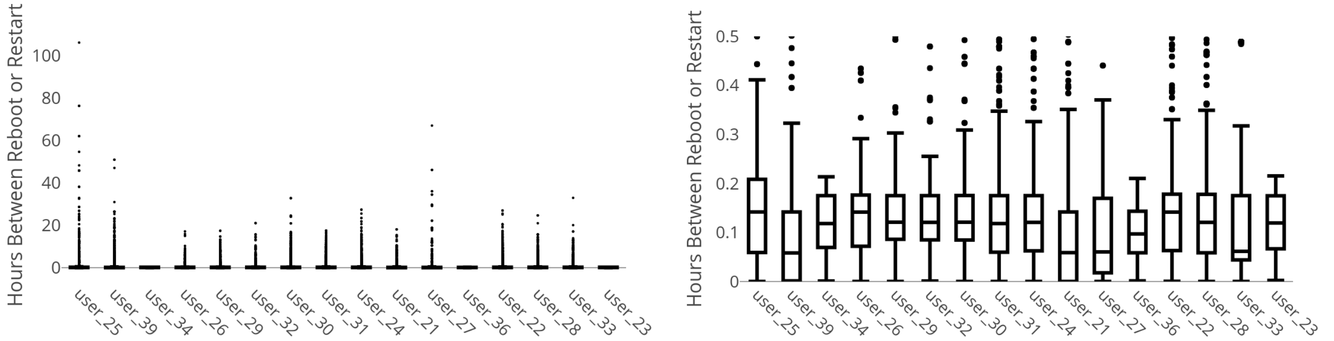
Connectivity Management. We found management of a cellular phone fleet more challenging than anticipated. The cellular network provider offered no interface to manage groups of phones or their collective data. Instead, data had to be added to each phone individually through a USSD interface. This posed a problem as USSD numbers do not work remotely, requiring us to employ a human on the ground to add data to the phones each month. Furthermore, the lack of an interface to determine if a network was experiencing transient failures or a if SIM card had insufficient credit made it difficult to distinguish network failures from app failures.

Row	Error Message	Reports
1	watchdog2 rebooting due to dead process	257,547
2	gridwatch.plugwatch.wit. PlugWatchService:bluetooth stack died	108,653
3	gridwatch.plugwatch.wit. PlugWatchService:unable to start scanning	40714
4	restarting due to timeout	30,898
5	service disconnected	28,211
6	An error occurred while executing doInBackground()	18,643
7	watchdog rebooting due to dead process	3,981
8	gridwatch.plugwatch.wit. ConnectionCheckService:restart rebooting due to max timeout	3,836
9	Exception thrown on Scheduler.Worker thread. Add ‘onError‘ handling.	1,680
10	[memory exhausted]	398

Table 1: Errors logged locally by the PlugWatch App. Logs are not independent and multiple errors may be logged for the same error event. Logs are captured only when the app is running. The most common error (Row 1) occurs when the separate *watchdog2* process finds the primary PlugWatch process no longer running and restarts it. Rows 2 and 3 indicate errors related to the BLE stack. Row 4 indicates the PlugWatch app timing out after failing to receive BLE packets. Rows 5 and 6 are exceptions in the PlugWatch background service due to implementation error. Row 7 indicates that the *watchdog* running within the primary PlugWatch process found the separate *watchdog2* process no longer running and restarted it. Row 8 is another BLE timeout error. Row 9 is an exception in the PlugWatch background service due to implementation error. Row 10 describes memory tension.

PlugWatch Lifespan. By examining the top ten most common errors logged by the PlugWatch app (see Table 1), we find that app restarts and phone reboots occurred due to the two app watchdogs catching dead processes, as well as BLE stack errors, implementation errors, and timeouts between receiving packets from the WiTenergy device. These are all behaviors that were not broadly observed in laboratory testing. Further, we observe a wide distribution of uptime across the deployment in Figure 5. The high variance suggests that several different factors shortened lifespan, making it hard to make strong conclusions about the real root cause of such frequent app restarts and phone reboots.

Over the course of the deployment, the PlugWatch phones collectively restarted the app 341,725 times and rebooted 153,719 times. An app restart event took around 1.5 seconds.



(a) Up time per-phone with all outliers

(b) Uptime per-phone with Y range near the upper quartile

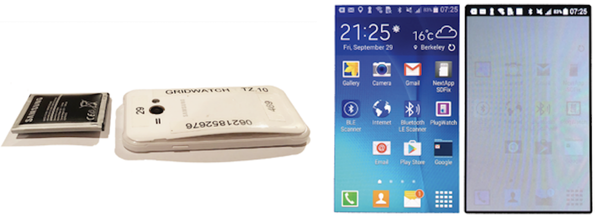
Figure 5: Uptime in hours before an app restart or phone reboot. Box-and-whisker plots showing hours between app restarts or phone reboots for: (a) the full range of hours; and (b) 0-0.5 hours. Each phone exhibits large variance in uptime, ranging from nearly zero to over 100 hours, which renders the boxes and whiskers nearly invisible in (a). However, (b) shows that the mean uptime for each phone was approximately 15 minutes or less. Unsupervised recovery from app crashes and phone reboots is therefore critical for gateway applications, but we found poor support for reliable, automatic and interaction-free recovery in Android.

A phone reboot event took an average of 1 minute and 40 seconds. The total downtime was therefore approximately 4,454 hours (142 from restarts and 4,312 from reboots), or 8% of the total deployment time (24 hours × 142 deployment days × 16 phones). Furthermore, this does not account for the time spent in error states before these restart and reboot events were triggered, nor time spent in unrecoverable states.

5.2 How Did the Phones Fare Physically?

Out of the 16 PlugWatch devices, eight phones exhibited a visibly-swollen battery by the end of the experiment (see Figure 6). We hypothesize that battery problems led to the sharp drop-off in packets around February 5th (see Figure 4). The worst battery was swollen to 270% of its normal width. Six of the batteries had swollen to the point where they had pushed the plastic phone backing out of its setting and they were no longer seated in the electrical contacts. We have not identified any recall of the J110H phone or the EB-BJ111ABE battery that suggests that these problems are model-specific. While we did not see any battery issues during an 18-hour heat test in the development phase, longer-term exposure to the hot and humid deployment climate seems to have been sufficient to cause battery swelling.

Ten of the phones experienced some degree of screen burn-in (see Figure 6). The burned-in image was most often of the home screen, indicating that the app was not correctly pinning itself to the foreground (see Section 4.1.3). The damage was not bad enough to make the phones unusable, but it does significantly detract from any future user experience.



(a) Two swollen batteries. **(b) Screen burn in.**

Figure 6: Hardware problems discovered. (a) A battery and a phone with its battery after deployment. Both demonstrate swelling, and the battery in the phone has unseated the plastic backing of the phone. (b) Screen burn-in from a device frozen in an invalid and unrecoverable state. Notice that the burn-in is from the home screen, not the PlugWatch app which should have been in the foreground.

6 LESSONS LEARNED

We seek to draw some conclusions from our experience with the PlugWatch deployment to help inform future efforts. These lessons should be read critically. They are derived from a single deployment of a modest number of phones, although one that is similar in scope to related studies [3, 19, 21]. A small team of academic researchers developed the PlugWatch application, not professional software engineers. However, future research systems are likely to be built by similarly-small research teams and by domain scientists who expect, as we initially did, that smartphone gateways would be as successful in loose association deployments as they have been in tight association contexts. It is for their benefit that we share our experiences, arguing that the current state of deploying such systems may not align with expectations, even for experts. Finally, we lay out a road map toward a plausible future in which these systems are better supported.

6.1 Human Interactions are Critical

We find that despite our best efforts to liberate the phones from human interaction, keeping humans in the loop over the course of the experiment both drastically improved the functionality of PlugWatch and was ultimately necessary to keep the phones operational on the network.

Notably, monthly maintenance by field officers had a positive impact on PlugWatch's performance. Although we are not sure which of the interactions was key, it is clear that the system benefited from a regular human touch. Unfortunately, our design choices around independence prevented us from taking advantage of this fact once we discovered it. We intentionally did not design a UI that would allow participants to troubleshoot devices themselves, did not include participant-device interaction in our IRB application, and screwed the phones into boxes for which the participants likely did not have screwdrivers. These early decisions hampered our ability later to change the experimental protocol and allow participants to interact with the phones.

6.2 Physical Environment Matters

Before deploying PlugWatch, we researched how the deployment circumstances might affect the battery. We attempted to find a datasheet for the battery but found nearly no information from the manufacturer [38]. We found no sources that described more than a slight possibility of battery swelling; most discussions focused on the regulating circuitry in the phone that keeps it safe from overcharging. Given this limited information, we heat-tested the system by placing a PlugWatch phone in a oven at 120° F for 18 hours and logging the battery temperature and capacity, as well as any system reboot events, and found no unusual behavior. Despite this test, we were not able to predict the effect of months spent in the heat and humidity of Zanzibar, and the dangerous battery-swelling that resulted presumably due, at least in part, to these conditions.

6.3 Seek Supporting Infrastructure

There are a number of tools that might have helped with provisioning phones, collecting debugging information, ensuring reliable OTA updates, and remotely accepting permissions. These include Google's own enterprise mobility management (EMM) tools [8], as well as a number of third party options [22, 24, 25, 42]. Unfortunately, the Google EMM tools are only fully supported on Android 5.0 and would not support our deployment or the nearly half-billion phones loaded with versions of Android pre-5.0, as Figure 7 shows [8]. We did a shallow search for third-party options but did not ultimately adopt any, in part due to misguided confidence that we would not need these services (as OTA updates are provided by the Play Store [34], debugging information is sent from our software, etc.), and in part due to their cost, although free tiers with limited functionality do exist for some services [22]. We are encouraged that many of these tools did support our target Android API [22, 24, 25] and we would certanly consider these tools as valuable and perhaps indispensable for a future deployment.

6.4 SIM Management is Hard

Keeping SIM cards active on the network is a non-trivial task largely due to the lack of supporting infrastructure from cellular providers. There have been many calls for research networks that change this model, although these networks are still rare [37]. Collectively, the authors have attempted cellular-based deployments (between 5 and 20 smartphone or embedded devices) in five countries: the United States, Kenya, Tanzania, Nigeria, and Ghana. We observe from these experiences that:

(i) Purchasing SIMs in high quantities can be difficult. For example, SafariCom in Kenya and MTN in Ghana limit the purchase of SIMs to 10 per person. Companies may purchase more than 10 SIMs, but only after supplying extensive documentation that requires a multi-day review. For a deployment in Ghana, we partnered with a local company to purchase 400 SIMs and were surprised to learn that each SIM then had to be hand-activated by a sales representative, a tedious and error-prone process.

(ii) SIMs may be expensive. For example, some US carriers sell pre-paid non-contract SIM cards for much more than other countries ($25.00 USD from T-Mobile [40] compared to $0.22 USD from Airtel Tanzania [41]).

(iii) SIMs can be hard to transport. For example, we found that FedEx and DHL have an policy against shipping activated SIM cards out of Ghana, making it difficult to assemble sensors out-of-country.

(iv) SIMs may be difficult to keep funded and operational. For example, in Kenya, Tanzania, Nigeria, and Ghana, the primary method for adding airtime and data was scratch-off cards, shown in Figure 3. Some web APIs for SIM management did exist, but they were not functional or scriptable. In Kenya, Tanzania, and Ghana, we asked multiple telecoms for help managing data on fleets of SIMs but all tools were cumbersome, not readily available, and not designed to support scales of hundreds of phones.

6.5 Service Restarts Often Fail

The failure of one peripheral in a gateway should not cause the whole system to fail. A common approach to recover from peripheral failures is to enable the core logic to reset or power cycle peripherals independently. Android does not support this practice, which can be problematic.

For example, the BLE system used by PlugWatch in Android 4.4.4 regularly crashed or otherwise failed to connect and scan. In a dedicated sensor node, a reasonable solution to these non-deterministic bugs would be to power cycle the BLE peripheral and restart its driver when erroneous behavior is noticed. Although the Android BLE API exposes ways to enable and disable the BLE system, we found that to reliably restart a BLE connection, PlugWatch had to either restart itself or reboot the whole phone. This greatly increased the complexity of the app and is undesirable for many traditional sensing applications, especially on devices that take minutes to reboot.

6.6 OS-Coupled Drivers Not Upgradable

Nearly all modern operating systems (including many embedded OSs) allow for drivers to be replaced and upgraded independent of upgrading the OS kernel itself. Neither of these practices is standard in Android, which leads to hardware not getting critical updates to its peripheral drivers unless the entire OS is ported to that hardware (a process that is guaranteed only for a relatively short period of time). Google's Project Treble [9], introduced in Android 8.0, attempts to address this problem. However, this does not help non-upgrade-eligable older phones that are already deployed.

6.7 User-Centric Design is Problematic

Recent changes to Android have made things better for the user but worse for loosely associative sensing applications. Android 6.0 added runtime permissions, with which a user manually approves each permission-protected function. These include many critical functions for long-running apps, such as accessing the internet, reading phone state, and writing to persistent memory. Android 8.0 (2017) and Android P (2018) both change how background operations work, limiting the OS services that can wake up an app and the sensors a background service can access. Instead, these versions promote a model of foreground services and define foreground services as services that are privileged due to users awareness of them. Ultimately, these changes improve user experience by ensuring that long-running background services do not take more than a fair share of the resources and do not invade user privacy. However, in the context of repurposing phones as long-running gateways or sensor nodes that do not have interactive users, these changes significantly and artificially constrain the application space.

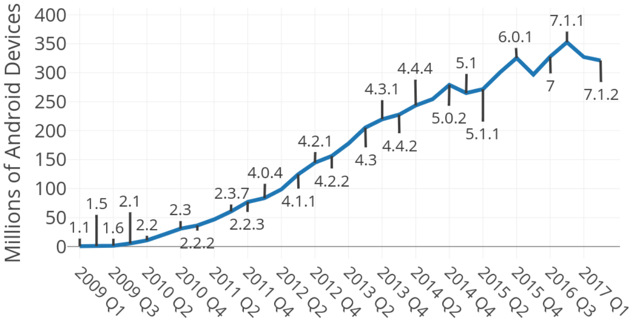


Figure 7: Global Android shipments from 2009 to 2017 and corresponding Android version number at time of shipment. Hundreds of millions of old Android phones are already deployed in the world. Changing the version cut-off for reuse by even a minor version number could potentially affect millions of phones.

6.8 You Cannot Future-Proof the Past

Even if future versions of Android were to fix all of the Android-related problems we encountered with PlugWatch, requiring more mature hardware or operating systems translates to abandoning billions of older phones already sold. This not only delays the viability of smartphone gateways until modern operating systems can penetrate the markets, but abandons the most widely-deployed computing resource across the globe. As Figure 7 shows, changing the cut-off for reuse by even a minor version could result in the consignment of millions of phones to landfills instead of productive second lives as gateways.

6.9 Recommendations for the Future

In our ideal vision of the future, once a user decides to discard a functional but dated smartphone, they can choose to donate their phone's body to science by enabling an OTA OS upgrade to a gateway-friendly version of Android. A number of changes could potentially help enable this vision of ubiquitous cellular gateways.

First, Android could be modified so that privileged processes are never killed and are always restarted, permissions are removed, logging is retained, and modal windows are suppressed or could be programmatically handled trivially. A default gateway app could be pre-installed that allows a user to pick endpoints for their data.

Further, this new OS could be made available using the existing OS OTA update mechanisms. This would allow a user to decide to install this version of Android on their old device using a familiar interface when they upgrade to newer hardware. Finally, if a cellular network existed that white-listed phones running this version of Android, cellular connections could be provided with a different pricing model to the end user. Collectively, this would make reusing phone hardware dramatically easier for many researchers.

7 CONCLUSIONS

Android smartphones are inexpensive, powerful, and ubiquitous computing systems that work in nearly every country, have a mature ecosystem for software development and distribution, support application multi-tenancy, include a rich set of sensors and wireless interfaces, and offer global positioning and network connectivity. It would seem that they are the perfect platform for an unimaginable array of simple and demanding IoT applications, from sensing to perception to wirelessly connecting peripheral devices.

And yet, like a petulant toddler, an Android smartphone requires near constant adult supervision. Users must periodically reset the devices when applications hang. Users must manually login to the phones after a reset to launch applications. Users must check and clear modal dialogues. Users must regularly restart crash-prone Bluetooth stacks to reconnect with peripherals. Users must manually upgrade applications over cellular networks. And users must ensure that batteries do not swell or explode from use or abuse.

Knowing these costs upfront might dissuade some from being lured by the very tantalizing—but perhaps ultimately unrealizable—potential of the Android smartphone platform as self-sufficient system for long-running services. After a year of sunk costs and lost time with Android, we gave up on smartphones and implemented nearly identical functionality with custom hardware in a matter of weeks, which has since spent two months deployed in the field with far better results.

8 ACKNOWLEDGMENTS

This work was supported in part by the Development Impact Lab (USAID Cooperative Agreement AID-OAA-A-13-00002), part of the USAID Higher Education Solutions Network, and in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Additionally, this material is based upon work supported by the NSF/Intel CPS Security Program under grant CNS-1822332 and the NSF CPS Program under grant CNS-1239467. We would like to thank the anonymous reviewers for their insightful and detailed feedback and our anonymous shepherd for their guidance. We would also like to extend special thanks to the field officers in Zanzibar and the employees of the Zanzibar Electricity Company who supported the PlugWatch deployment, as well as the many other collaborators who helped make this undertaking possible.

REFERENCES

[1] Arrow. 2018. Bluetooth WiFi Combo Module by WLAN+BT. https://www.arrow.com/en/categories/rf-and-microwave/rf-modules/combo-wireless-modules. (Accessed on 03/13/2018).

[2] Ivan Beschastnikh, Yuan Zhang, Zhengping Qian, and Lidong Zhou. 2012. Liberating Mobile Phones from their Primary Use Case.

[3] Niels Brouwers and Koen Langendoen. 2012. Pogo, a middleware for mobile phone sensing. In *Proceedings of the 13th International Middleware Conference*. Springer-Verlag New York, Inc., New York, NY, USA, 21–40.

[4] Keith M Diaz, David J Krupka, Melinda J Chang, James Peacock, Yao Ma, Jeff Goldsmith, Joseph E Schwartz, and Karina W Davidson. 2015. Fitbit®: an accurate and reliable device for wireless physical activity tracking. *International journal of cardiology* 185 (2015), 138–140.

[5] Koushik Dutta. 2012. Clockworkmod.

[6] Google. 2016. Google Cloud Messaging: Overview. https://developers.google.com/cloud-messaging/gcm. (Accessed on 03/13/2018).

[7] Google. 2018. Firebase Realtime Database. https://firebase.google.com/docs/database. (Accessed on 03/13/2018).

[8] Google. 2018. Get Started with Android in the enterprise. https://developers.google.com/android/work. (Accessed on 07/11/2018).

[9] Google. 2018. Treble. https://source.android.com/devices/architecture/treble. (Accessed on 03/13/2018).

[10] Vinoth Gunasekaran and Fotios C. Harmantzis. 2007. Emerging wireless technologies for developing countries. *Technology in Society* 29, 1 (2007), 23 – 42. https://doi.org/10.1016/j.techsoc.2006.10.001

[11] Kristin Hanks, William Odom, David Roedl, and Eli Blevis. 2008. Sustainable millennials: attitudes towards sustainability and the material effects of interactive technologies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 333–342.

[12] David Hasenfratz, Olga Saukh, Silvan Sturzenegger, and Lothar Thiele. 2012. Participatory air pollution monitoring using smartphones. *Mobile Sensing* 1 (2012), 1–5.

[13] Elaine M Huang and Khai N Truong. 2008. Breaking the disposable technology paradigm: opportunities for sustainable interaction design for mobile phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 323–332.

[14] InfluxData, Inc. 2018. InfluxData (InfluxDB): Time Series Database Monitoring & Analytics. https://www.influxdata.com. (Accessed on 03/13/2018).

[15] Matthew Keally, Gang Zhou, Guoliang Xing, Jianxin Wu, and Andrew Pyles. 2011. Pbn: towards practical activity recognition using smartphone-based body sensor networks. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. ACM, New York, NY, USA, 246–259.

[16] Wazir Zada Khan, Yang Xiang, Mohammed Y Aalsalem, and Quratulain Arshad. 2013. Mobile phone sensing systems: A survey. *IEEE Communications Surveys & Tutorials* 15, 1 (2013), 402–427.

[17] KingoSoft, Inc. 2017. KingoRoot for Android, the best One Click Root Tool/APK for free. https://www.kingoapp.com. (Accessed on 03/13/2018).

[18] Grafana Labs. 2018. Grafana – The open platform for analytics and monitoring. https://grafana.com. (Accessed on 03/13/2018).

[19] Nicholas D Lane, Yohan Chon, Lin Zhou, Yongzhe Zhang, Fan Li, Dongwon Kim, Guanzhong Ding, Feng Zhao, and Hojung Cha. 2013. Piggyback CrowdSensing (PCS): energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, New York, NY, USA, 7.

[20] Arm Limited. 2018. Home | Mbed. https://www.mbed.com/en. (Accessed on 07/17/2018).

[21] Emiliano Miluzzo, Nicholas D Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B Eisenman, Xiao Zheng, and Andrew T Campbell. 2008. Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, New York, NY, USA, 337–350.

[22] Miradore. 2018. Miradore. https://www.miradore.com. (Accessed on 07/12/2018).

[23] Shridhar Mubaraq Mishra, John Hwang, Dick Filippini, Reza Moazzami, Lakshminarayanan Subramanian, and Tom Du. 2005. Economic analysis of networking technologies for rural developing regions. In *International Workshop on Internet and Network Economics*. Springer, Berlin, Germany, 184–194.

[24] Mitsogo, Inc. 2018. Hexnode MDM. https://www.hexnode.com/mobile-device-management. (Accessed on 07/12/2018).

[25] MobileIron, Inc. 2018. MobileIron. https://www.mobileiron.com/en/welcome-era-modern-work. (Accessed on 07/12/2018).

[26] Anandatirtha Nandugudi, Anudipa Maiti, Taeyeon Ki, Fatih Bulut, Murat Demirbas, Tevfik Kosar, Chunming Qiao, Steven Y Ko, and Geoffrey Challen. 2013. Phonelab: A large programmable smartphone testbed. In *Proceedings of First International Workshop on Sensing and Big Data Mining*. ACM, New York, NY, USA, 1–6.

[27] Nokia. 2008. Global consumer survey reveals that majority of old mobile phones are lying in drawers at home and not being recycled. http://www.nokia.com/press/press-releases/showpressrelease?newsid=1234291

[28] J Park, L Hoerning, S Watry, T Burgett, and S Matthias. 2017. Effects of electronic waste on developing countries. *Advances in Recycling and Waste Management* 2, 2 (2017), 1–6. https://doi.org/10.4172/2475-7675.1000128

[29] Particle Industries, Inc. 2018. Particle - Welcome. https://www.particle.io. (Accessed on 07/17/2018).

[30] Android Open Source Project. 2018. Accessibility Service. https://developer.android.com/reference/android/accessibilityservice/AccessibilityService.html. (Accessed on 03/13/2018).

[31] Android Open Source Project. 2018. Android P Behavior Changes. https://developer.android.com/preview/behavior-changes.html#input-data-privacy. (Accessed on 03/13/2018).

[32] Android Open Source Project. 2018. Android Things. https://developer.android.com/things/index.html. (Accessed on 03/13/2018).

[33] Android Open Source Project. 2018. Background Execution Limits. https://developer.android.com/about/versions/oreo/background.html. (Accessed on 03/13/2018).

[34] Android Open Source Project. 2018. Google Play Store | Android Developers. https://developer.android.com/distribute/google-play. (Accessed on 07/17/2018).

[35] Android Open Source Project. 2018. Request App Permissions. https://developer.android.com/training/permissions/requesting.html. (Accessed on 03/13/2018).

[36] Android Open Source Project. 2018. Services. https://developer.android.com/guide/components/services.html#Foreground. (Accessed on 03/13/2018).

[37] Research and Markets. 2018. Global M2M/IoT Communications Market 2017-2022: Cellular M2M Subscribers Increased by 56% During 2017. https://www.prnewswire.com/news-releases/global-m2miot-communications-market-2017-2022-cellular-m2m-subscribers-increased-by-56-during-2017-300581334.html.

[38] Samsung. 2018. GH43-04527A - INNER BATTERY PACK-EB-BJ111ABE, 1800MAH. https://www.samsungparts.com/GH43-04527A-BATTERY.html. (Accessed on 07/13/2018).

[39] Samsung. 2018. Samsung Galaxy J1 Ace – Price, Specs & Features. http://www.samsung.com/in/smartphones/galaxy-j1-ace-j110h. (Accessed on 03/11/2018).

[40] T-Mobile USA, Inc. 2018. SIM Card Starter Kit | 3 in 1 SIM Starter Kit | T-Mobile. https://www.t-mobile.com/cell-phone/t-mobile-3-in-1-sim-starter-kit. (Accessed on 07/19/2018).

[41] Airtel Tanzania. 2018. FAQs. http://africa.airtel.com/wps/wcm/connect/africarevamp/Tanzania/Home/Personal/Customer_care/Help/FAQs. (Accessed on 07/19/2018).

[42] ProMobi Technologies. 2018. MobiLock. https://mobilock.in. (Accessed on 07/12/2018).

[43] Viettel Tanzania Ltd. 2018. Halotel Home Page. http://halotel.co.tz. (Accessed on 03/13/2018).