Revisiting Network Energy Efficiency of Mobile Apps: Performance in the Wild

Sanae Rosen, Ashkan Nikravesh, Yihua Guo, Z. Morley Mao, Feng Qian†, Subhabrata Sen*
University of Michigan, †Indiana University, *AT&T Labs – Research
{sanae,ashnik,yhguo,zmao}@umich.edu,
fengqian@indiana.edu, sen@research.att.com

ABSTRACT

Energy consumption due to network traffic on mobile devices continues to be a significant concern. We examine a range of excessive energy consumption problems caused by background network traffic through a two-year user study, and also validate these findings through in-lab testing of the most recent versions of major mobile apps. We discover a new energy consumption problem where foreground network traffic persists after switching from the foreground to the background, leading to unnecessary energy and data drain. Furthermore, while we find some apps have taken steps to improve the energy impact of periodic background traffic, energy consumption differences of up to an order of magnitude exist between apps with near-identical functionality. Finally, by examining how apps are used in the wild, we find that some apps continue to generate unneeded traffic for days when the app is not being used, and in some cases this wasted traffic is responsible for a majority of the app's network energy overhead. We propose that these persistent, widespread and varied sources of excessive energy consumption in popular apps should be addressed through new app management tools that tailor network activity to user interaction patterns.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: wireless communication; C.4 [**Performance of Systems**]: measurement techniques, performance attributes

Keywords

4G LTE; smartphones; cellular network performance; mobile energy consumption

1. INTRODUCTION

Fueled by powerful mobile devices and ubiquitous cellular data network access, smartphone applications (*apps*) have become an indispensable part of modern life. There have been more than 100 billion mobile app downloads from the Apple App Store as of June 2015 [2]. However, battery life remains a scarce resource. Over the past 15 years, CPU performance has improved 250x while Li-Ion battery capacity has only doubled [11]. It is known that inefficient app design can lead to excessive battery drain. In particular, certain app traffic patterns, like periodic requests, interact poorly with the power-hungry cellular interface [22, 8, 21]. Despite these known problems, however, apps continue to drain user batteries.

In this paper, we measure the prevalence of excessive mobile app network energy consumption by analyzing data collected from 20 real smartphone users and 342 unique apps over a period of 22 months. This unique long-term dataset allows us to examine the smartphone and app ecosystem in the wild. We focus on the impact of background traffic — traffic sent when the app has no UI element visible — which makes up 84% of the total network energy consumed across all users. Periodic background traffic is often power-hungry [21], but apps have flexibility in scheduling background traffic due to the absence of real-time user interaction, and can use strategies such as bundling traffic or reducing update frequencies to reduce energy consumption. We examine global trends across all apps and determine that energy overconsumption remains a pervasive problem, despite many apps taking steps to reduce their energy overhead. Furthermore, some of this traffic is likely unintentional and not useful to the end user.

Our key findings are as follows:

- We identify a significant new source of excessive background energy consumption (§4.1), where network traffic persists after an app transitions from the foreground to the background, sometimes for as long as a day. 30% of background traffic from one major browser is caused by this phenomenon. Over 80% of apps transmit more than 80% of their background data in the first minute after the app is sent to a background state, in total across all user devices in our study.
- We show that there is high variation in the energy overhead of apps that rely on frequent background traffic, even between apps with similar background functionality (§4.2). By examining case studies of apps that require background updates, we find that the energy consumed by similar apps can vary by up to an order of magnitude. Furthermore, we find that apps studied in previous work have often improved their energy overheads but that other new apps continue to make the same mistakes. There is substantial room for improvement by adopting energy-efficient design approaches, such as batching background updates.
- By examining apps as they are used in the wild, we find that many apps are frequently not used for days, including apps with substantial background traffic. We demonstrate that the network energy overhead of these apps can be reduced by up to a half in some cases if the OS were to proactively terminate long-running apps after three days of inactivity (§5). More generally, we emphasize the need for apps to be aware of their foreground/background

IMC'15, October 28–30, 2015, Tokyo, Japan.

DOI: http://dx.doi.org/10.1145/2815675.2815713.

state when scheduling network requests, and our findings suggest that new suggestions for managing background traffic are likely to be highly valuable.

2. RELATED WORK

Mobile network and app performance is an area that has received a great deal of interest, and prior work has examined many aspects of this problem, including papers on TCP-cellular interactions [17], accurate active performance measurements [14], app usage [27], wireless data drain [23], comparing cellular and Wi-Fi performance [25], application performance [19], energy efficiency when downloading and rendering web content in the foreground [26], and the impact of user behavior [13]. We focus on background network energy consumption specifically, specifically where it occurs in the wild.

Prior work has also examined various aspects of background network activity as well, which or work builds upon. Aucinas *et al.* examined smartphone energy efficiency through in-lab experiments with a number of major apps which maintain a continuous online presence [8]. Work by Huang *et al.* [18] examined traffic sent when the screen is off (note that background traffic can also occur when the screen is on, including from minimized apps). Other work has examined periodic transfers [21], and the disproportionate impact of small background requests [12, 9]. Tamer [20] demonstrates it is possible to modify the energy impact of app wakeups by interposing on wakeup events to better manage them. Our work is complementary, examining a broad set of apps and focusing on their behavior in the wild over a long time period, which enabled us to uncover new energy drain problems as well as understand how old problems have evolved.

There has also been a great deal of interest in the impact of background traffic in work developed concurrently with this paper. In particular, Google announced Android M after the submission of this paper, which introduces Doze and App Standby, which should decrease the energy impact of the excessive background traffic we uncovered in this paper [15, 5]. Other concurrent work includes ZapDroid [24], which automatically isolates or disables infrequently-used apps, a solution which our findings suggest would be highly valuable. Work by Chen et al [10] presents a large-scale user study of how users use their phones and how that interacts with app battery consumption. Their study is complementary, covering all sources of energy consumption and trends across categories of devices, whereas we focus on examining the role of background network transfers specifically in depth and exploring the root causes of this excessive background consumption.

3. DATA COLLECTION AND OVERVIEW

We first summarize our measurement dataset. We recruited 20 students[1] at the University of Michigan and provided each of them with a Samsung Galaxy S III smartphone with an unlimited LTE data plan. We pre-installed custom data collection software on each phone that transparently collects complete network traces. These traces include packet payloads (note we are unable to decrypt SSL traffic), user input events, and packet-process mappings. All collected data was kept strictly confidential. The data was collected over a period of 623 days (December 2012 to November 2014) with an overall raw data size of 125 GB, including cellular and WiFi packet traces and user input and context data. We focus primarily on cellular traffic in this study as it consumes far more energy than WiFi. Processes are labeled with names derived from the app package name, allowing us to straightforwardly map packets to the originating apps. In a few cases, requests are delegated to some system services such as the Media Server. We label this traffic according to the service from which it originated rather than the app which triggered it, as mapping back to the originating app in this specific case cannot always be done with high accuracy.

[1]This user trial was approved by University of Michigan IRB-HSBS #HUM00044666.

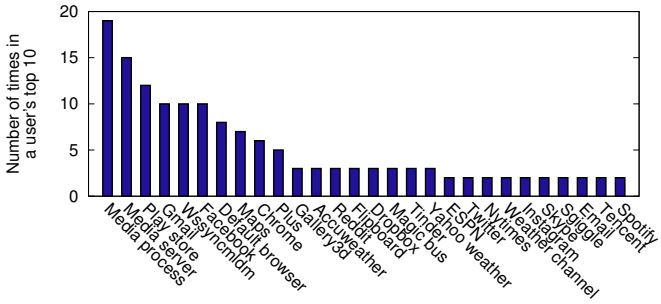

Figure 1: Number of times each app appears in a user's top 10 apps, ranked by total data consumption.

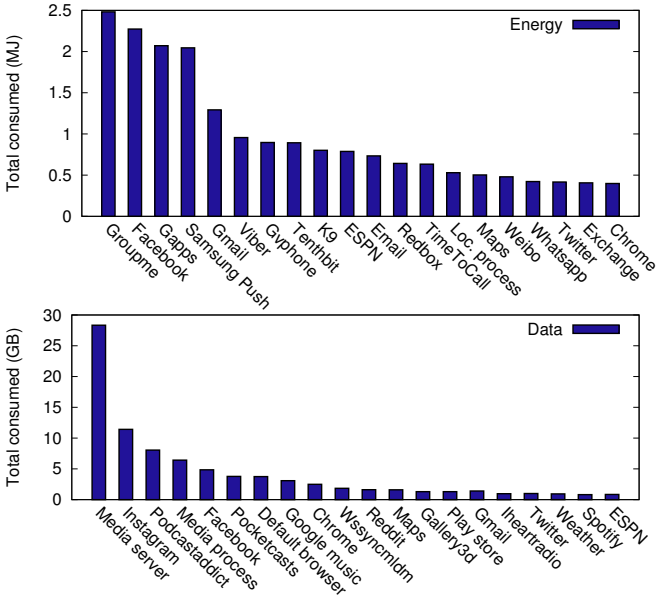

Figure 2: Highest cellular data and network energy usage by app across all users.

3.1 Measurement Data Overview

We next give an overview of this 22 month dataset before focusing on specific apps.

App Popularity and Diversity. Users differ greatly in the apps they use. Figure 1 shows apps that appear in at least two users' top-10 lists (by total data consumption). While a handful of apps are popular among all users (*e.g.,* the built-in media player, Facebook, and Google Play), users' top-ten lists otherwise exhibit significant diversity. Similar diversity of app usage was observed in previous work [13, 27].

Data- and Energy-Hungry apps. First, we examine trends in applications that consume a large amount of energy or data. We use a standard power model for LTE [16, 22] supported by measure-

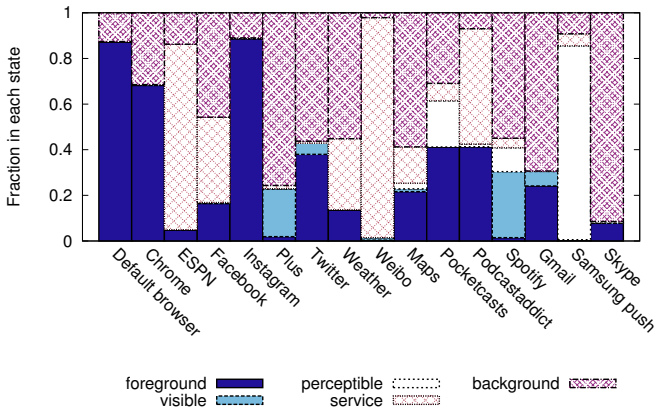

Figure 3: Fraction of energy in each foreground/background state, based on process codes assigned by the Android operating system.

ments gathered with a Monsoon power monitor to compute radio energy consumption[2].

We summarize the top energy and data consumers in Figure 2. Note the top energy consumers and the top data consumers are not necessarily the same. For example, the default email app consumes network energy disproportionate to its data usage, whereas the built-in media server consumes significantly less energy per byte. In cellular networks, the radio remains active for several seconds after a data transmission, consuming additional energy called *tail energy* [9]. Since tail energy consumption is dependent on the number of traffic bursts and the time between them rather than the amount of data sent, apps sending data intermittently incur a disproportionate amount of energy.

As we evaluate the impact of each app in the wild, rather than the impact of apps in isolation, we assign any tail energy to the last packet sent during the tail period to avoid double-counting energy when there are multiple concurrent flows. In this way, the total cellular network energy consumed by each device is the sum of the energy assigned to each app.

Longitudinal Trends. We examined trends in network usage and energy consumption over time. However, the diversity of apps, the smaller user set, users' propensity to change the apps they use over time, and changes in user behavior, obscured the overall impact of app design changes over time or any trends towards more energy efficiency. Background energy fluctuated by up to 60% from week to week throughout the study. Examining specific apps, we did determine that some apps have become more energy-efficient due to adjusting the inter-packet intervals of background traffic, which we discuss in more detail in §4.2.

4. BACKGROUND ENERGY CONSUMPTION

Energy consumption in the background makes up 84% of the total network energy, and is thus the focus of this study. An app running in the background may run until either the user kills it manually or Android does (such as when more memory is needed). Many apps sync with a server, receive push notifications, or run updates in the background. Since no user interaction is present, these processes have much more freedom to determine when they transmit data than when running in the foreground, where they may be subject to time constraints to meet user expectations. Furthermore, there is often a tradeoff between ensuring updates are timely and avoiding wasted background updates the user never looks at. For this reason, apps vary greatly in the amount of energy that they consume in the background, even when providing similar functionality. In this section, we analyze the resource efficiency of app background network activities through detailed case studies, identifying large disparities between apps due to diverse design approaches. We also identify several cases where large numbers of network requests are sent unnecessarily, verified through in-lab testing.

Our definition of "background" traffic is based on five main process states defined by Android [6]: *foreground*, where the process is responsible for the main UI; *visible*, where the process is responsible for a secondary UI element; *perceptible*, where a process not visible to the user may still affect the user experience (e.g. playing music); *service*, where a background process should not be terminated if possible; and *background*, where the OS will kill the app if system memory is low. We summarize the cellular energy in each of these five states for twelve data- or energy-hungry apps in Figure 3. We refer to the first two categories as "foreground" processes and the last three as "background" processes for the remainder of the paper. Note that for all but three of these apps, background energy of some sort contributes more than half of the overall network energy consumed by the app. Across all apps, 84% of cellular network energy is consumed in a background state. This included only 8% of energy consumed by "perceptible traffic", as only a few users used streaming services and it is apparent from Figure 3 that not all apps made use of this feature when expected. 32% was consumed by "service" traffic.

We focus on two main categories of background transfers. In §4.1, we examine background traffic that occurs when an app switches from the foreground to the background. In §4.2 we investigate traffic initiated automatically in the background, such as that for periodic updates, push notifications, or music streaming. We supplement our longitudinal traces with in-lab measurements to validate our findings and determine the context and purpose of the traffic in our traces.

4.1 Foreground Traffic not Terminated

While it is expected that some apps will transmit data in the background, such as when checking email, updating a social networking app or streaming music, other apps such as browsers are expected to mainly transmit data when the app is in the foreground. However, we find some such apps appear to inadvertently transmit data in the background. As shown in Figure 3, about 30% of the Chrome browser's network energy is consumed while the app is running in the background. To understand why, we examine a representative trace from the user study dataset in Figure 4. We have highlighted the time period after Chrome switches to the background in grey. During this time packets continue to be sent for several minutes: note that some websites also generate periodic background requests.

To validate our hypothesis that Chrome allows web pages to continue sending periodic traffic after the app is minimized, we first created a custom web page that only sends XMLHttpRequest asynchronously to a server every second. We found that the Chrome app allows this web page to transfer data when tabs are not selected and thus invisible to the user; when the screen is off; and even when the app has been sent to the background. To further confirm this problem exists in the wild, we also opened several web pages, minimized Chrome, and recorded the resulting network traces. In general, any web page which automatically refreshes content has this problem, including some ad and analytics content. In one partic-

[2]In the rest of the paper, "energy" refers to the network energy unless otherwise noted.

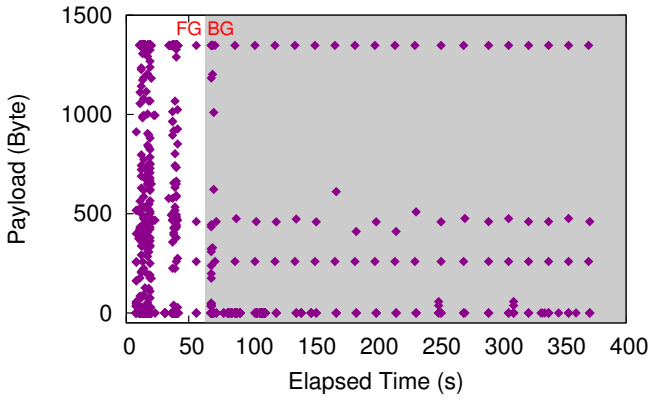

Figure 4: Chrome allows webpages to continue sending and receiving data in the background.

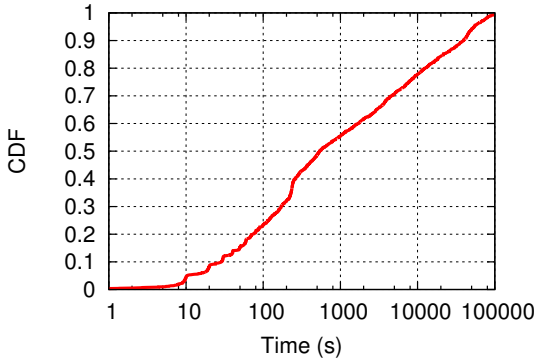

Figure 5: Duration for which traffic continues to be sent/received after the app is sent to the background. Each data point represents one transition to the background.

ularly egregious case, a popular local transit information webpage sends background requests roughly every 2 seconds, indefinitely, keeping the cellular radio alive and draining the battery until the app is killed or the tab is closed.

To quantify the severity of the problem on a larger scale, we plot the distribution of the length of time during which Chrome continues to transmit data after being sent to the background in Figure 5. This includes flows that were started when Chrome was in the foreground but persist after Chrome is sent to the background. Each data point represents one instance of the app being minimized. In some cases background traffic flows persist for more than a day! While updating pages in the background may have some benefit to users who then revisit that page, in most cases continuing to send data for so long is likely not intended or useful behavior. Note that our data points do not include cases where the app remains in the foreground but a tab other than the one being viewed is sending data, and so the scope of the problem is likely even bigger.

We compared this behavior against that exhibited by Firefox and the default Android browser. Neither allow data to be sent when the app is in the background or the screen is off, and Firefox blocks data from being sent by inactive tabs. To estimate the prevalence of this problem among other apps, we examine the data sent by apps in the background as a function of the time since the app was last in the foreground. As we show in Figure 6, the more recently the app was

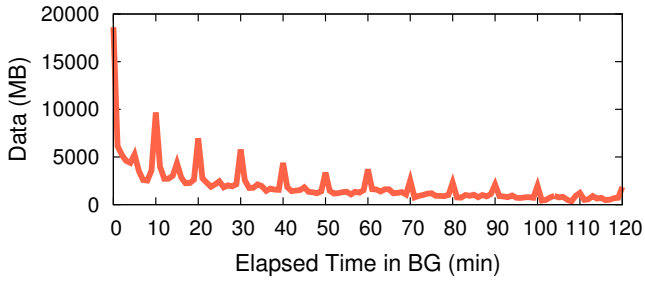

Figure 6: Total background data sent by all apps, as a function of the time since switching from a foreground state. Note the periodic spikes at 5 and 10 minute intervals, the large amount of traffic in the first minute, and the long tail of persisting, continuous flows.

sent to the background, the more traffic is sent, with substantially more traffic being sent in the first few minutes than any other time. Some of this traffic is periodic: there are peaks at 10 minute and 5 minute intervals, which are common time intervals for intentional periodic background traffic. However, there is also a non-periodic pattern, where the overall volume of background traffic falls off rapidly in the first few minutes. To estimate the prevalence of this problem, we look for apps where 80% of the background traffic is sent within 60 seconds of the app going to the background than any other time. 84% of apps meet this criteria.

There are some apps, like Dropbox, which may have valid reasons to upload content immediately after the app is closed, but in many other cases transmitting a large amount of traffic after the app is closed is undesirable. Developers of apps which send a large amount of data immediately after sending the app to the background should determine if this is expected or necessary behavior.

4.2 Transfers Initiated in the Background

We next evaluate data transfers that intentionally occur in the background. Even though these transfers may be beneficial to the user, depending on the frequency of user interaction with the app, the overhead of these transfers can be disproportionate. We examine a number of energy-hungry apps in depth, as well as some energy-conserving apps with similar functionality. Finally, we study a number of apps examined in previous work to evaluate how background update energy efficiency has improved over time. The energy efficiency of background transfers are primarily determined by their frequency, with small updates incurring a disproportionate amount of tail energy. Large transfers are known to be more efficient, as they make better use of available bandwidth [17]. As a result, apps with similar functionality can have very different overheads depending on the traffic pattern used.

We summarize key findings in Table 4.2, focusing on five classes of apps that are responsible for a substantial fraction of background updates in the user study: social media apps, widgets, music streaming, podcasts and services that provide background updates. We break down the energy overhead into average per-day energy consumption and average per-flow energy consumption. Note that it is not always the case that there is only one flow per periodic update, nor that for periodic updates that the updates necessarily continue for the entire day, as background applications may be forced to close for a variety of reasons.

First, we examine social media apps. These apps generally ask for updates from a central server periodically, regardless of user activity, and can thus potentially consume a large amount of energy.

	MJ/ day	MJ/ flow	MB/ flow	Avg. J/B	Update frequency	Notes
Social media						
Weibo	3500	57	0.3	190	5-10 min	Frequent, nearly-empty requests
Twitter	220	11	17	0.65	1h	
Facebook	930	14	7.7	1.8	5 min ⇒ 1 h	Previously every 20-60s [21] in 2012
Plus	180	11	8.9	1.2	1h	Rarely actively used but installed by default
Periodic update services						
Samsung Push	1500	140	2.2	64	15 min to 15h	
Urbanairship	2000	310	1.9	163	5-30 min	Library; period varies by app
Maps	190	21	55	0.38	20-30 min	Decreased to a few hours near the end
Gmail	410	20	10	2	30 min ⇒ varying	30 min in 2012 [21]; updates appear to become discontinuous.
Widgets						
Go Weather widget	300	12	1.6	7.5	5 min	
Go Weather	220	2.8	5.6	0.5	5 min ⇒ 40 min	Switched push notification approaches
Accuweather	1500	51	3.2	16	7 min but high variation	
Accuweather widget	33	1.7	18	0.094	~3h	More efficient than the app
Streaming						
Spotify	310	50	220	0.23	5 min ⇒ 40 min	
Pandora	35	3.9	45	0.087	1 min ⇒ 2h	Previously every 1 min [21] in 2012
Podcasts						
Pocketcasts	36	4.3	2200	0.002	~2h average	0.4 mJ per minute running.
Podcastaddict	92	2.9	750	0.004	12 min average	3.7 mJ per minute running.

Table 1: Case studies. Energy per flow and per day are averages over time, and one flow may not correspond to one periodic update. These can vary as apps change over time or as background apps are forced to close, and energy consumption values vary by device and carrier.

Apps with small, periodic background traffic (such as Weibo) have very high energy overhead and send little data, whereas apps with similar functionality (such as Twitter) have a much smaller footprint. Facebook, which had previously been identified as a heavy energy user, improved its energy efficiency over the course of our study by decreasing its background update frequency from 5 minutes to 1 hour, which is much longer than the 1-minute periodicity measured in 2012 [21].

Applications oriented towards providing periodic background updates, such as certain push notification services, may consume a lot of energy compared to the amount of data they send. In an in-lab experiment, one third-party library transmitted nearly empty HTTP requests every five minutes for hours, but only provided one user-visible notification during this time. Another example is Google Maps, which by default provides a background location service which continuously collects anonymous location data. This service consumed up to 90% of the app's total energy usage at the beginning of the study, but the frequency decreased to once every few hours by the end. GMail also leverages periodic updates using push notifications: it has actually increased its inter-update intervals, but updates appear to no longer be periodic, arriving only on demand, leading to an overall low degree of energy consumption.

Widgets are a class of apps that appear on the home screen and have little or no direct user interaction. In many cases their functionality revolves around periodic background updates (such as to keep the user updated on the latest weather). There is a tradeoff between timeliness of information and energy consumption. However, even just examining weather widgets, the difference in update frequency between two apps (and the resulting energy overhead) varies by an order of magnitude. Note also that the Accuweather app is far less efficient than the corresponding widget, as the widget updates itself less frequently. Widgets and apps made by the same developers may have very different behavior.

We also examined several multimedia streaming apps. Music streaming apps were not as popular in our dataset as in prior work, but their update frequency was generally much lower than before [21], having apparently moved away from a continuous streaming model to larger batch downloads, although particularly long update frequencies may reflect users who only intermittently use these apps. Podcasts were far more popular, and we compare two popular apps. Podcastaddict consumed more energy overall, as Pocketcasts downloads an entire podcast in one chunk whereas Podcastaddict downloads smaller chunks as needed. While the latter approach may reduce data consumption if users don't finish listening to a podcast, it consumes more energy.

5. WHAT-IF ANALYSIS: PREEMPTIVELY KILLING IDLE BACKGROUND APPS

In §4 we determined that background traffic has a substantial impact on energy consumption, and in some cases much of this traffic is from apps users are not frequently using. We propose having the OS kill background apps that have remained in the background for several days. A new permission or whitelist could address corner cases where apps (such as widgets) have a legitimate need to run in the background for an extended period of time, and OS feedback on background energy consumption could disincentivize unnecessary use of this functionality. In fact, the preview of Doze in Android M, a project announced after our submission, appears to add such functionality [15, 5]. We have identified a number of apps where this type of functionality has the potential to greatly reduce background traffic, although we do not evaluate Doze itself in this paper.

To evaluate the effectiveness of this approach, we simulate restricting background traffic after three days, and highlight six apps

	Plus	Weibo	Maps	ESPN	Accuweather	Skype
A: % days with only background traffic	42	83	70	13	43	62
B: Max consecutive background days	40	24	84	10	18	49
C: Disable after 3 days: avg.% energy reduction	14	54	39	6.2	22	45

Table 2: Example trends in background traffic when apps are infrequently used, and simulated energy savings from suppressing this traffic.

in Table 5. In row A we show the fraction of days where we see only background traffic from the app, and in row B we show the maximum number of such days that we see occurring consecutively, considering only time periods where there is foreground traffic at the beginning and end of the time period. These apps are rarely used by certain users, creating energy savings opportunities if the apps were to be preemptively killed. Row C summarizes the average savings per user of killing the app after three consecutive days. Note in particular that Weibo, which we showed was very energy-hungry, can have its network energy consumption more than halved this way.

Due to the large number of apps users in our study had installed on their phone, the impact of each app individually on a user's total network consumption was small. Thus, this would have resulted in total network energy savings of less than 1% on average overall. However, we found that for the users running Weibo, disabling Weibo alone after just three days of inactivity could have reduced their total network energy consumption by 16% on those days. Overall, how much users benefit from this functionality depends greatly on the set of apps involved and on user behavior, so it is hard to draw definite conclusions on the average benefits of our proposed system for or other systems such as Doze, but such an approach seems especially promising in protecting users from poorly optimized or buggy apps, and reducing the worst-case energy consumption generally.

6. CONCLUDING REMARKS AND RECOMMENDATIONS

Excessive energy consumption by mobile apps has long been known to be a significant problem, and background traffic continues to be a major battery drain. We have examined a significant but previously unstudied phenomenon where network traffic initiated in the foreground persists unnecessarily when the app is sent to the background. Furthermore, we have shown that improvements for known inefficiencies have not been universal, even for professionally developed apps with a large user base. While we recommend that app developers continue to carefully consider the cost of the traffic they send, more is needed to improve the situation, especially for background traffic.

We make several recommendations. First, apps should be designed to explicitly account for their foreground/ background state and adjust network transfers accordingly. Most crucially, apps should ensure network transfers are terminated when the app is minimized where possible. In §5 we demonstrate that more aggressively killing apps that run in the background for days could greatly reduce the energy impact of infrequently used apps. Since submitting this paper, Google announced Android M, where all background activity is disabled when the device is idle, and users are able to manually specify exemptions for specific apps. Our findings suggest this is likely a very positive step towards improving battery life. Similar tools to manage other aspects of network content overconsumption would also be valuable, such as to terminate flows meant to only occur in the foreground.

Finally, the impact of periodic background transfers can also be reduced in many cases. We have seen that some of the improvements described in prior work, such as batching requests, have been implemented with positive effect [18, 21], but background data continues to have a large relative overhead. As proposed in prior work, app developers should continue to batch traffic to minimize the frequency of background updates, as well as tailor updates to reflect the frequency with which useful, new data is provided. Our findings also emphasize the importance of previously proposed approaches to reducing the energy consumption of background traffic at the OS level, such as by providing explicit OS support for periodic updates or by using radio-layer energy saving features such as fast dormancy [7]. Even if some apps improve energy consumption, new apps will likely emerge that make the same mistakes.

In this paper, we have focused on background energy issues on Android. Other systems take different approaches: IOS, for instance, has the OS manage background tasks for applications, restricting the potential impact of suboptimal app designs [1]. Although we have not evaluated these systems, we can speculate as to how our findings might apply to these systems. OS management allows transfers to be batched, providing opportunities for energy consumption optimization. However, some of their design approaches and guidelines may suggest scheduling approaches inconsistent with our findings. For instance, prioritizing apps who download a small amount of data in the background quickly may incentivize the wrong behavior. Windows Phones also put restrictions on background apps [3], restricting resource-intensive tasks to WiFi and limiting the frequency with which background apps can run, but there have still been consumer reports of specific apps draining the battery when running in the background [4]. Overall, there is a tradeoff between developer flexibility and the potential for excessive energy consumption; we leave examining the energy impact of these alternate approaches to future work.

7. ACKNOWLEDGEMENTS

We would like to thank our anonymous reviewers as well as Erich Nahum, our shepherd, for their valuable comments. This research was supported in part by NSF under CNS-1059372 and CNS-1345226, as well as by an NSERC Canada PGS D scholarship.

8. REFERENCES

[1] App Programming Guide for iOS — Background Execution. https://developer.apple.com/library/prerelease/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html.

[2] Apple's app store has passed 100 billion app downloads. http://www.theverge.com/2015/6/8/8739611/apple-wwdc-2015-stats-update.

[3] Background agents for Windows Phone 8. https://msdn.microsoft.com/en-us/library/windows/apps/Hh202942(v=VS.105).aspx.

[4] Conserve windows phone battery life by managing background apps. http://www.windowscentral.com/conserve-windows-phone-battery-life-managing-background-apps.

[5] Developer preview - power-saving optimizations. https://developer.android.com/preview/features/power-mgmt.html.

[6] ActivityManager.RunningAppProcessInfo documentation. https://developer.android.com/reference/android/app/ActivityManager.RunningAppProcessInfo.html.

[7] P. K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V. N. Padmanabhan, and G. Varghese. RadioJockey: Mining Program Execution to Optimize Cellular Radio Usage. In *Proc. ACM MobiCom*, 2012.

[8] A. Aucinas, N. Vallina-Rodriguez, Y. Grunenberger, V. Erramilli, K. Papagiannaki, J. Crowcroft, and D. Wetherall. Staying Online while Mobile: The Hidden Costs. In *CoNEXT*, 2013.

[9] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proc. ACM IMC*, 2009.

[10] 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. Sigmetrics*, 2015.

[11] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proc. ACM MobiSys*, 2010.

[12] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A First Look at Traffic on Smartphones. In *Proc. ACM IMC*, 2010.

[13] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in Smartphone Usage. In *Proc. ACM MobiSys*, 2010.

[14] A. Gember, A. Akella, J. Pang, A. Varshavsky, and R. Caceres. Obtaining In-Context Measurements of Cellular Network Performance. In *Proc. ACM IMC*, 2012.

[15] R. Holly. Checking out Doze and App standby on the Android M Developer Preview. http://www.androidcentral.com/checking-out-doze-android-m-developer-preview.

[16] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Proc. ACM MobiSys*, 2012.

[17] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An In-Depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *ACM SIGCOMM Computer Communication Review*, volume 43, 2013.

[18] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck. Screen-off Traffic Characterization and Optimization in 3G/4G Networks. In *Proc. ACM IMC*, 2012.

[19] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing Application Performance Differences on Smartphones. In *Proc. ACM MobiSys*, 2010.

[20] M. Martins, J. Cappos, and R. Fonseca. Selectively Taming Background Android Apps to Improve Battery Lifetime. In *Proc. Usenix ATC*, 2015.

[21] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization. In *Proceedings of the 21st international conference on World Wide Web*, pages 51–60, 2012.

[22] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Profiling Resource Usage for Mobile Applications: a Cross-layer Approach. In *Proc. ACM MobiSys*, 2011.

[23] A. A. Sani, Z. Tan, P. Washington, M. Chen, S. Agarwal, L. Zhong, and M. Zhang. The Wireless Data Drain of Users, Apps, & Platforms. *ACM SIGMOBILE Mobile Computing and Communications Review*, 17(4), 2013.

[24] I. Singh, S. V. Krishnamurthy, H. V. Madhyastha, and I. Neamtiu. ZapDroid: Managing Infrequently Used Applications on Smartphones. In *Proc. UbiComp*, 2015.

[25] J. Sommers and P. Barford. Cell vs. WiFi: On the Performance of Metro Area Mobile Connections. In *Proc. ACM IMC*, 2012.

[26] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who Killed my Battery?: Analyzing Mobile Browser Energy Consumption. In *Proceedings of the 21st international conference on World Wide Web*, 2012.

[27] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *Proc. ACM IMC*, 2011.