Android App Update Timing: A Measurement Study

Ed Novak and Chris Marchini
Franklin and Marshall College
Lancaster, PA
{enovak,cmarchin}@fandm.edu

***Abstract*—Over the past decade the pace of software publication has increased dramatically. Thanks to the advent of "app markets" software distribution on mobile devices is centralized and software updates are, by default, fully automated. In this work we study the pace of software updates on Android smart mobile devices. Specifically, we measure the rate at which users experience software updates, and the delay between the time an app update is made available, and when it is actually installed. Our data shows that, of the top 12 most popular apps in our dataset, over 10 of them are updated more frequently than once every two weeks. On average users install an update for at least one app every 56hrs.**

***Index Terms*—mobile, software engineering, software update cycles, software evolution, android, app**

I. Introduction

Software distributed through centralized app markets has been extremely advantageous allowing for easier discovery and installation of software, improved security, and more streamlined updating of software. App markets reduced friction in software deployment. This, catalyzed by agile development practices, has caused developers release much more frequently [1]. Software updates are now published at an unprecedented rate. However, this change in pace has exacerbated preexisting user concerns about updates. Our results show that software updates now occur on a daily basis. At the same time, we see that updates do not propagate to user devices rapidly for a variety of reasons (some unknown!) such as sporadic update polling intervals, and limited connectivity options. In our work we aim to measure the objective user experience of Android application software updates on smart mobile devices. We've also done some preliminary work towards a predication mechanism that will allow users to better handle the tidal wave of software changes they are expected to mediate.

Much traditional research in security and updates has focused on the operating system itself. While keeping the operating system up-to-date is prudent, it is now equally clear that keeping individual applications up-to-date is just as important [2], [3]. Unfortunately, because of the multitude of apps a typical user has, and the rate at which apps are updated, this task is onerous. In concept, the increased rate of software updates should be a positive change bringing improved functionality, bug fixes, and security enhancements. However, in practice it has become clear in recent years that users feel burdened by numerous software updates that they do not fully understand. Occasionally features are removed, user interfaces are modified, and in general changes are made that don't strictly benefit the user. Because of this trend, some users choose to turn off updates entirely [4], [5]. Alternatively, we have found that even users who have enabled fully automatic updates don't always receive updates as quickly as might be expected. Both these situations can lead to users running software that is out of date. Ultimately this can result in several problems, including compatibility issues, missing functionality, or worse, missing critical security updates. Users are faced with a paradox. Industry best practices dictate that they should directly mediate software updates, individually, in order to make informed decisions about when to install updates and when to avoid them. However, at the same time users are expected to run fully updated software in order to be most secure and, in theory, avoid software bugs. This duality, combined with the increased rate of updates is proving to be too large of a burden for users who do not have the time, expertise, or motivation to make informed decisions.

We aim to measure software update trends such as *saturation* and *publication rates* directly on users' devices. Similarly to previous work [6], we also produce some tools which empower users to better manage these updates. Our work incurs several challenges such as implementing a data collection methodology that works across multiple Android OS versions, and gathering users to harvest data. Although our data collection mechanisms are fully automated, we do require that Android users install our data collection application. Providing tools for users to better manage software updates incurs the challenge of somehow making predictions about the timing of future updates. We find that, by their nature, updates are somewhat chaotic and unpredictable even for very well established, professional development teams.

Previous studies have only analyzed update propagation via app markets directly [7], [8]. Our study uniquely gathers data directly from users. This gives us an entirely new perspective for analyzing the nature of updates within the Android space. Also, we are the first to make some preliminary steps towards a recommender system that helps inform users about software updates coming in the future. Our contributions are summarized below:

- We build and promote an android application used to record software updates on user devices as they happen in real time. Our app is installed on more than 70 devices and has gathered more than 27,000 data points representing real user experiences over approximately six months.

- Using the data from our application, we make a careful

analysis and study to form several interesting conclusions and insights.
- We design and implement a prototype update predictor, which aims to predict when an app will most likely recieve an update.

II. Related Work

There exist several measurement papers which analyze application markets directly. Work shows trends such as saturation rate of app updates (about 7 days) [9], the overall composition of the market [8], and what percentage of popular apps update frequently [10], [11]. A substantial survey of many other application market research studies was recently published by Martin, Sarro, Jia Zhang, and Harman [7]. Our work is focused on the typical, individual user experience. This cannot be ascertained by examining the application market in its entirety. We find that using a modern smart mobile device requires users to be comfortable with a state of constant change due to frequent software updates of apps.

There is active research which aims to provide tools to empower users to better manage software updates. Work from authors Tian et. al. [6] suggests augmenting the update manager with user reviews about the same app from users that have already installed the update in question. This allows the user to see if others favor the newest update or not. Thomas, Beresford, and Rice [12] study the propagation (or lack thereof) of Android Operating System updates. While there is some delay that seems to be ubiquitous across device manufacturers, users can be informed by their "FUM" metric to choose a manufacturer that has a good track record of keeping a minimum number of open vulnerabilities. Wagner, Rice, and Beresford also have a recent work which measures a plethora of metrics directly on user devices, similar to our work here. However, software updates are notably absent [13]. Our work is the first (to the best of our knowledge) which aims to predict the timing of future software updates.

Recent surveys into the user experience of updating software shows that users view the process as burdensome and overwhelming. In general there is a negative sentiment, because updating software requires expert knowledge, as well as significant time and energy. Furthermore, it occasionally results in unwanted changes [4], [5], [14], [15].

In light of this, the Android platform has almost fully automated app updates. However, our work shows the ways in which this automation breaks down such as "lag-time," which is explored in Section IV-B. Also, because of this automation, we show the excessive frequency of updates. Because updates effect the user experience heavily by bringing UI and functionality changes, full automation is maybe not the best solution.

III. Data Collection

Because we are trying to measure the software update experience of users, our work must collect data from user devices directly. This is in contrast to much of the related literature, which usually targets large software repositories online such as SourceForge, GitHub, and smartphone application markets themselves [8]. Our approach, inspired by Wagner et. al. [13], gathers information directly from user devices. This is done via an app we authored called "Update Timing Collector." Our app is installed (by the volunteer participants) on the participant's device. Our application is available for free on the Google Play Store [16]. The source code is licensed under the GPL v3.0 and is also distributed free online [17]. To gain users we promoted our app to computer science students on our local campus. For supplementary users we made postings on three crowd-sourcing Internet marketplaces; Amazon Mechanical Turk [18], Job Boy [19], and microworkers [20]. Our postings indicated that users should install the app on their primary personal device.

In all promotions the purpose of our app is clearly stated; to record information about when applications on [the user's] device are installed, uninstalled, or updated. We don't collect any personally identifiable or sensitive user information. Our app is only available stand-alone and is not embedded into any other app as a third party library. Therefore, users can choose to enter into, or remove themselves from, the study simply be installing or uninstalling the app at any time. Users can request their data be fully deleted from the study at any time by contacting the authors.

Our app enters a record into a log file whenever a package manager event occurs on the device. This log is then uploaded to a companion back-end service written in python running on a remote, publicly available server. Random IDs are generated to protect user privacy.

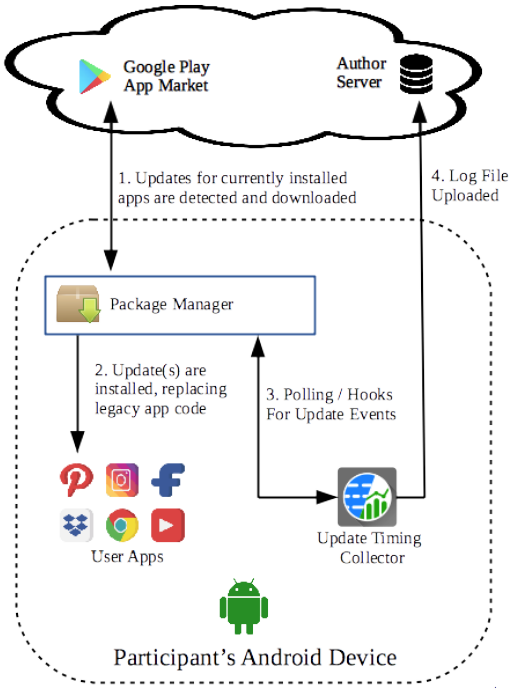

Fig. 1. Data collections system diagram.

We began our data study in May of 2018 and continued to

collect data until December 31st 2018. At the time of writing we have collected 27,673 data points from 95 unique users. A graphical representation of our data collection system design can be seen in Fig. 1

A. Implementation Details

Name	Description
android.intent.action.PACKAGE_ADDED	App installed
android.intent.action.PACKAGE_REMOVED	App uninstalled
android.intent.action.PACKAGE_REPLACED	App updated

TABLE I
IMPLICIT BROADCAST RECEIVER EVENTS

For devices running Android version 7.x and lower, our data acquisition relies on implicit broadcasts. Broadcasts for various events on a user's device are generated by the Android framework and are sent out to all apps that have registered a "Broadcast Receiver" class.

Although there are many events that may be hooked, we are interested in software changes and therefore execute logging code when certain events from the package manager occur. Therefore in our broadcast receiver implementation we specify (in the app's $manifest.xml$) the three events listed in Table I.

For Android 8.0+ devices, apps can no longer receive implicit broadcasts in the background. This was a significant change implemented by Android OS development team, at the OS level, in order to help reduce background processes and thereby reduce extraneous battery usage.

As a workaround, our app must poll the package manager Android API, which provides access to the information necessary and, critically, the time of the last update for all apps on that device. We poll the package manager API every 15 minutes to check for differences in the list of installed apps. Based on the differences we can infer the events that must have occurred and append to the log file accordingly. Using the JobScheduler API our app can wake-up and perform the poll entirely in the background without any user interaction.

B. Limitations

Some applications existed on the device long before our logging app was installed and the *install* events for those apps were not logged. Therefore, we do not have access to a complete list of applications installed. However, we can closely approximate this list by inspecting all of the applications that have been updated, (but never uninstalled) in the log of a particular device.

Pre-installed apps were installed on the device even before the user purchased it. The "last update time", which the package manager maintains for every app, is usually some placeholder value for applications like this (e.g., "Jan 1st, 2009"). In all of our analysis we remove all log entires that have dates before our study began.

IV. DATA ANALYSIS

In this section we perform some analyses of the data gathered from the logs to gain some understanding of the objective user experience of software updates in the modern Android ecosystem. We are most interested in the rate at which software is changing on a typical user's device.

1) Update Frequencey: The inter-arrival time (i.e., the time between) of PACKAGE_REPLACED events on each device can be seen in Fig. 2. Most users seem to experience software updates as frequently as every few hours, but usually not more than 200hrs (about eight days) apart. On average across all devices an update will be installed for at least one app approximately every 54hrs (as shown by the black line in Fig. 2). However, it is clear to see that most users have a personal average below 54hrs. The median inter-arrival time between software updates is only 35.7hrs. Furthermore, we observe that software updates are typically installed in batches of more than one at a time. Using the DBscan clustering algorithm we measured the average batch size to be 6.0 apps.

Counter to our data, other recent literature [7], [10] has shown that the vast majority of apps are updated much less frequently than once a week. Our conclusion is that, although most apps are not updated on a daily (or even monthly) basis, the apps on a particular user's device skew heavily towards wildly popular apps. Apps with millions of active users generally have more resources and publish periodic *weekly* updates. Additionally, the various developers of these apps do not coordinate their update schedules.

2) New Installation Frequency: We measured the inter-arrival time between new app installs as indicated by PACKAGE_ADDED events. As can be seen in Fig. 3, the variance is generally wider and the average time is much higher, about 189hrs. According to the same DBScan cluster analysis, users install new apps in batches of only 3.0 apps at a time. Compared to software updates, users install new apps much less frequently and in much smaller batches. Users install a single new app only about once per week. One user, perhaps extreme, exhibited almost four months between installing any new application! This is primarily explained by the fact that software updates are automated and new software installation is entirely manual. The user must seek out and actively install new software. Therefore, this contrast hints at the rate at which users seek change in their software.

3) Outliers: On some devices, we noted a few extreme cases in which there are thousands of hours between two PACKAGE_REPLACED events or two PACKAGE_ADDED events. While these cases may be interesting as case studies, they are very rare. We treated them as outliers using the "$3 * IQR$" method and removed them from Fig. 2 and 3.

A. Most Common Applications

The applications that each user has installed will have a dramatic impact on the experience that user has in terms of how many updates per day will be made available to them. Using our data we are able to analyze the most popular apps across our users. As mentioned in our limitations in Section III-B this data will only include apps that are detected by our app. We calculated which applications occur in the log files of the most users. The results are presented in

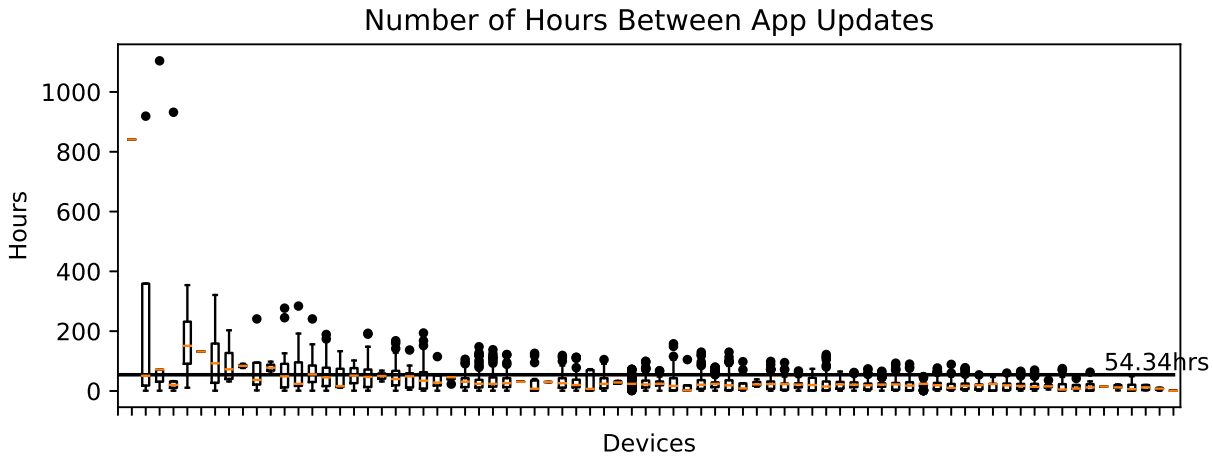

Fig. 2. Number of hours between software updates (any app) on each device.

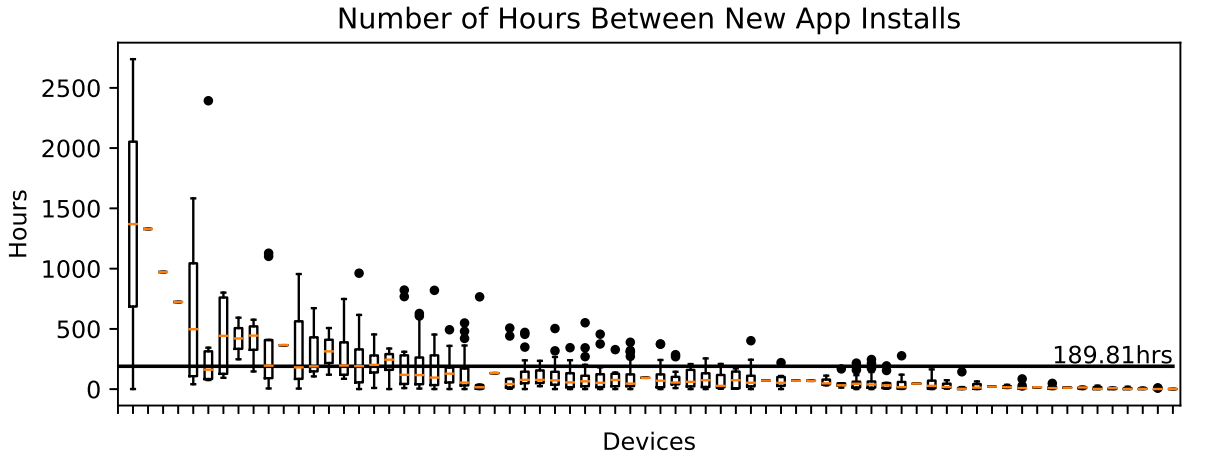

Fig. 3. Number of hours between installing new apps (any app) on each device.

Table II. Almost all the apps are very well known with a rapid update schedule of several days. Noteably, the list is dominated by Google's own first party apps. The most common app *com.android.vending* (the Google Play Store app) occurs in 65% of our participant's device. We assume that this app is installed on all of the devices, this data shows that it has been *updated* on only 65% of devices.

B. Update Lag Time

We found that not all devices install a particular update at the same time. For a particular app (e.g., the GMail app), an update may be published by the developer, in theory making it available to all users. However, according to our data, some users will install this update quickly (within a few hours) and others may have significant lag time before they install the update. This "lag time" impacts the saturation of the new update. We can measure it by identifying the first user and last user that recieves a particular update for a particular app. The average lag time of an app can be calculated as that over several different update cycles and users. In Fig. 4 we calculated the average lag time for each of the apps compiled in Table II. Interestingly, the average lag time spans anywhere from 25hrs to 140hrs.

Package Name	Install %	Update Rate
com.android.vending	65.3%	14.9 days
com.google.android.youtube	60.0%	18.1 days
com.google.android.apps.maps	54.7%	24.5 days
com.google.android.googlequicksearchbox	54.7%	21.8 days
com.google.android.gm	52.6%	29.7 days
com.google.android.apps.docs	52.6%	25.0 days
com.facebook.orca	48.4%	16.2 days
com.android.chrome	46.3%	35.0 days
com.google.android.apps.photos	46.3%	25.1 days
com.google.android.videos	42.1%	38.0 days
com.instagram.android	41.1%	10.4 days
com.google.android.music	40.0%	49.3 days
com.facebook.katana	40.0%	11.1 days
com.google.android.instantapps.supervisor	35.8%	19.0 days
com.google.android.play.games	34.7%	42.8 days

TABLE II
TOP 15 MOST COMMON APPS LISTED IN DESCENDING ORDER ACCORDING TO THE AMOUNT OF DEVICES THEY APPEAR ON (OUT OF 95 DEVICES). UPDATE RATE IS THE AVERAGE NUMBER OF DAYS BETWEEN UPDATES.

One app, Google Maps, was identifed as an outlier and

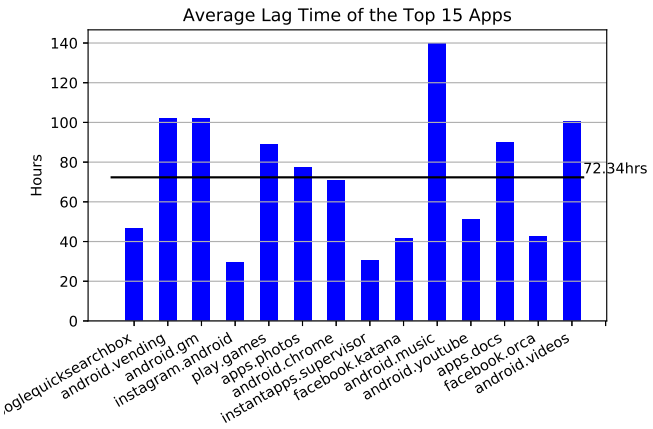

Fig. 4. "Lag Time" the average delay between the earliest and latest installation of the same update for each app in Table II.

removed from the figure. On average, it has a lag time of 821hrs; over one month! In general this result seems to contradict the average time between updates, shown in Fig. 2. How can an update be delayed 800hrs if users are installing updates approximately every 54hrs?

In fact, many factors impact installation time, such as WiFi availability, time since the previous update, and developer configurations. Upon further investigation, we found that updates will sometimes be missed entirely. Even if an update for a particular app is available, there is no guarantee that the update will be installed. In fact, it's common for several batches of updates for *other* apps to be installed first! We are still investigating this phenomenon and, at the time of writing, are unsure about the root cause. Missed or lagging software updates are a serious concern for privacy and security.

C. Edge Cases

A few applications perform updates which are not federated by the Google Play Store app in the typical way. Fortunately, they are still logged by the package manager.

Two interesting edge cases are "Google Play Services" which operates under the package name `com.google.android.gms` and the Google Play store app itself (`com.android.vending`). Google Play Services is a low level app that is tightly integrated into the Android platform, managing things such as authentication to Google services, and even app updates. It cannot be found in the Google Play Store market. In regards to `com.android.vending`, the design of the android package manager does not allow the Play Store app to update itself through the normal mechanisms. Independent update mechanisms are implemented directly into both of these apps.

V. Update Predication and Recommender

A useful software update recommender will enable users to optimize when their app updates are installed in order to conserve their device's resources, to improve the user experience, and to help ensure that their device receives updates they care about with minimal delay. Predicting the future timing of updates is the foundation of such a recommender. There are several scenarios in which it may be beneficial to influence the timing of update installation [6].

A. App Case Study

In order to design a predictor we first analyze the typical update timing of one specific app on one specific device. We selected a device at random and measured the YouTube App (`com.google.android.youtube`). This app is very common in our dataset and receives updates at a medium pace. Fig. 5 presents the various measurements we took. In the top left we see the actual raw timing of the updates for this app. The y-axis shows the update sequence number (the first update is seq. number 1, the second is seq. number 2, etc.) and the x-axis shows the hour this update occured, since recording began. The bottom left is a histogram of the inter-arrival time between updates.

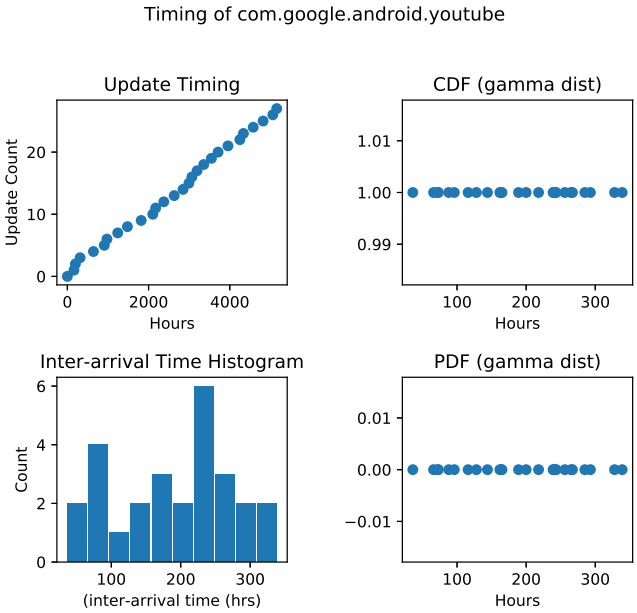

Fig. 5. Timing analysis of the YouTube App from one randomly selected device. Top-left: app update arrival time. Bottom-left: histogram of update inter-arrival times. Right: CDF and PDF of inter-arrival times for gamma distribution.

We modeled this data using the gamma disruption as is customary with inter-arrival time data. However, the distribution is nearly flat, which means that no particular observed time is any more likely than any other. Therefore, the CDF and PDF of the gamma distribution (right side of Fig. 5) are flat as well. In terms of making a predication for the timing of an update for this app, any time between the minimum (36hrs) and maximum (339hrs) observed are equally likely.

Deep analysis of other apps is omitted for brevity, but we did not find any app that did not have a flat (uniform) distribution of inter-arrival times.

B. Prototype Predictor

In this work we implement a prototype predictor, which aims at predicting when an app will update next. Of course our endeavor is to achieve both *accuracy* and *precision*. We experimented with two different prediction methods; the first using linear regression and the second using a trie data structure. A trie (also known as a digital tree) is a tree-like structure that is based on the patterns of the given data. Our approach builds the trie based on the timing data and it then attempts to match the current trend to one of those previously seen. Both approaches can be seen in Fig. 6 predicting the YouTube app.

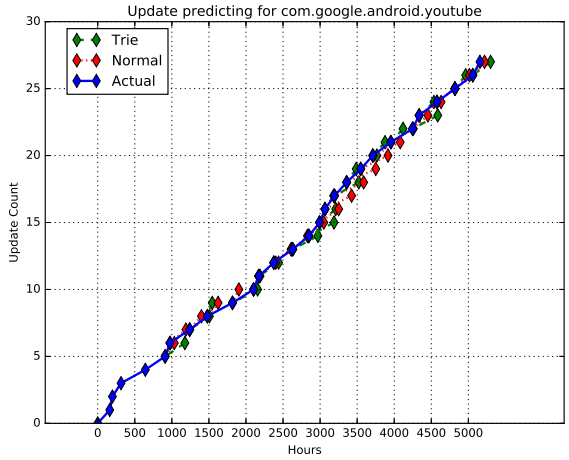

Fig. 6. Prediction of youtube app update timing. Trie: prediction using trie datastructure. Normal: prediction using linear regression. Actual: real timing of YouTube updates.

The results show that each approach is close to the actual timing. Our linear regression model is slightly more accurate, but less precise. It achieves at best 17hrs error. Occasionally an update will come more quickly than normal, for example at 3000hrs, and then the regression will fall away from the actual further. Meanwhile our Trie is very capable of closely predicting the next update, often getting within 24 hours and occasionally getting under 12 hours. However our Trie approach suffers greatly when it misidentifies the current timing trend. Misidenifications of the trend sometimes result in errors of over a week. Such errors can be seen in Fig. 6 at 1200hrs and 4500hrs. Table III summarizes the error of these two approaches. We leave improvement of our predictor to future work.

Method	Best Error	Worst Error	Average Error
Linear Reg.	15 Hours	229 Hours	237 Hours
Trie	2 Hours	-273 Hours	225 Hours

TABLE III
PREDICTING 20 CONSECUTIVE UPDATES TO THE GOOGLE YOUTUBE APP

VI. CONCLUSION

Our work is the first to provide a measurement of the rate at which software updates occur for typical users of Android devices. We show that apps update roughly every 54hrs and in batches of six at a time. Our study is the first to examine the objective user experience gathering data directly on user devices. Our predictor estimates the arrival time of future application updates, per app, and achieves an average accuracy of approximately 9 days (225 hours) in predicting the YouTube app. In the best case we can predict the future timing within only 2 hours.

REFERENCES

[1] F. Henderson, "Software engineering at google," *CoRR*, vol. abs/1702.01715, 2017.

[2] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith, "Sok: Lessons learned from android security research for appified software platforms," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 433–451, May 2016.

[3] A. Mendoza and G. Gu, "Mobile application web api reconnaissance: Web-to-mobile inconsistencies and vulnerabilities," in *2018 2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 646–659.

[4] K. Vaniea and Y. Rashidi, "Tales of software updates: The process of updating software," Proceedings of CHI '16, (New York, NY, USA), pp. 3215–3226, ACM, 2016.

[5] K. E. Vaniea, E. Rader, and R. Wash, "Betrayed by updates: How negative experiences affect future security," in *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, (New York, NY, USA), pp. 2671–2674, ACM, 2014.

[6] Y. Tian, B. Liu, W. Dai, B. Ur, P. Tague, and L. F. Cranor, "Supporting privacy-conscious app update decisions with user reviews," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15, (New York, NY, USA), pp. 51–61, ACM, 2015.

[7] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering*, vol. 43, pp. 817–847, Sept 2017.

[8] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, pp. 221–233, June 2014.

[9] A. Mller, S. Diewald, L. Roalter, T. U. Mnchen, F. Michahelles, and M. Kranz, "Update behavior in app markets and security implications: A case study in google play," in *In Proc. of the 3rd Intl. Workshop on Research in the Large. Held in Conjunction with Mobile HCI*, pp. 3–6, 2012.

[10] S. Mcilroy, N. Ali, and A. E. Hassan, "Fresh apps: An empirical study of frequently-updated mobile apps in the google play store," *Empirical Softw. Engg.*, vol. 21, pp. 1346–1370, June 2016.

[11] R. Potharaju, M. Rahman, and B. Carbunar, "A longitudinal study of google play," *CoRR*, vol. abs/1802.02996, 2018.

[12] D. R. Thomas, A. R. Beresford, and A. Rice, "Security metrics for the android ecosystem," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15, (New York, NY, USA), pp. 87–98, ACM, 2015.

[13] D. T. Wagner, A. Rice, and A. Beresford, "Device analyzer: Understanding smartphone usage," 09 2014.

[14] R. Wash, E. Rader, K. Vaniea, and M. Rizor, "Out of the loop: How automated software updates cause unintended security consequences," in *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, (Menlo Park, CA), pp. 89–104, USENIX Association, 2014.

[15] M. Fagan, M. M. H. Khan, and R. Buck, "A study of users' experiences and beliefs about software update messages," *Comput. Hum. Behav.*, vol. 51, pp. 504–519, Oct. 2015.

[16] E. Novak, "Updatimg timing collector app (google play store)." https://goo.gl/8isMFv.

[17] E. Novak, "Updatimg timing collector app (source code)." github.com/fmresearchnovak/updatetimingcollector.

[18] "Amazon mechanical turk." https://www.mturk.com/.

[19] "Job boy." https://www.jobboy.com/.

[20] "Micro workers." https://www.microworkers.com/.