ISSN2394-3777 (Print)
ISSN2394-3785 (Online)
Available online atwww.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology
(IJARTET) Vol. 5, Special Issue 14, April 2018

REDUCING APPLICATION UPDATES SIZE FOR ANDROID DEVICES

P. RAKSHITHA[#1]
Department of Computer Science and Engineering
C Byregowda Institute Of Technology
Srinivaspur Road- 563101,Karnataka,India
p.rakshitha7296@gmail.com

NESARA SINDHURI.V[#2]
Department of Computer Science and Engineering
C Byregowda Institute Of Technology Kolar-
Kolar-Srinivaspur Road- 563101,Karnataka,India
nesara1697.v@gmail.com

AMTHULLA AYESHA[#3]
Department of Computer Science and Engineering
C Byregowda Institute Of Technology
Road- 563101,Karnataka,India
amthullaayesha1234@gmail.com

SUBHASHINI. R[#4]
Assistant Professor
Department of Computer Science and Engineering
CByregowda Institute Of Technology Kolar-Srinivaspur
Kolar-SrinivaspurRoad-
563101,Karnataka,India
subhashini050@gmail.com

RADHA .R[#5]
Assistant Professor
Department of Computer Science and Engineering
C Byregowda Institute Of Technology
Kolar-Srinivaspur Road- 563101,Karnataka,India
itzradha@gmail.com

ABSTRACT

Application updates result in traffic on the cellular network and generate load to data centers that serve these updates.To solve this problem, in this paper we propose an improved Application update mechanism called DELTA++(Delta Encoding for Less Traffic for Apps) to achieve an additional traffic reduction and an additional size reduction of android application updates. The main idea of DELTA++ is to determine the difference between the application files within an APK to reduce the size of the android application updates.We use compression tool to reduce the usage of network bandwidth. We use bzip2 compression tool to implement our method.We can achieve 50% additional traffic reduction compared to Google Smart Application Update. Increased battery discharge from DELTA++ was found to be negligible.

Keywords- Android applications, data compaction, data compression, delta encoding, bzip2

I. INTRODUCTION

The introduction of new features and bug fixes make it usual for an application to have updates released every few weeks. Figure 1 shows a system level view of application updating for smartphones. Application updates result in traffic on the cellular network and generate load to data centers that serves

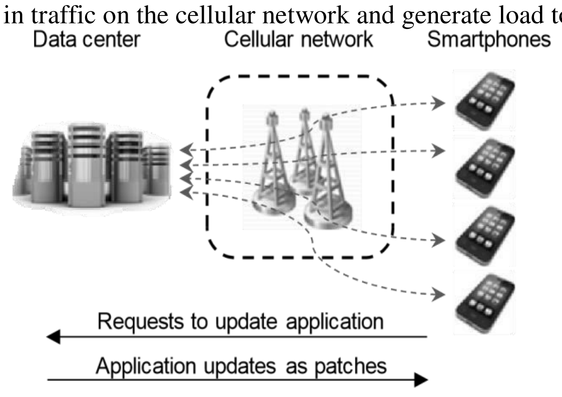
Data center
Cellular network
Smartphones
Requests to update application
Application updates as patches

Figure 1. System view of application updating

these updates. Mobile operators spend billions of dollars on network upgrades every year in order to keep up with the increasing amount of mobile traffic.

In June 2012 Google announced a new technology – Google Smart Application Update – that reduces application update traffic. Google play store has 2.7 million applications available in 2017,82 billion downloads to date.This technology enables savings in cellular networks, decreases the load on application servers in data centers that serve applications, and increases battery lifetime in mobile devices. In this article we demonstrate a new mechanism,that provides further reduction of traffic generated by application updates. DELTA++ is an extension of our previous work (we note that the initial submission of predated Google's Smart Application Update release).

Delta encoding is a technique that is used to compute the difference, or "diff", between two files. This difference can be used to construct the newer version of a file from the old one. Thus, a smartphone application can be updated by transferring only the difference between the old and new versions and then applying the delta patch locally in the smartphone. A key difference between Google Smart Application Update and our method is that we decompress the Android APK package and perform compression on individual modules within the APK. Google's method, to the best of our observational knowledge, does not do this. Our experimental results show that application updates can be reduced in size by 77% on average with DELTA++ compared to a reduction in size of 55% on average for Google Smart Application Update. Such reduction in network bandwidth use comes with a trade-off. Because of the increased patch complexity, more time has to be spent to deploy the application patch on the smartphone when using DELTA++. This delay can be tolerated for smartphone application updates as users typically do not need an update immediately after its release. Additional battery use is shown to be negligible.

Our contributions in this article include an implementation of DELTA++, a comparison of DELTA++ with Google Smart Application Update, evaluation of the additional energy use of DELTA++, and estimation of largescale savings that could be achieved with a full-scale deployment of DELTA++.

I. Google Smart Application Update

At the Google I/O developer conference held in June 2012 Google announced that their Smart Application Update technology had been introduced to the Google Play Store and would be seamlessly used to update all applications on Android devices. Google Smart Application Update is transparent to application developers and Android users. To enable Google Smart Application Update, changes were made in the Google Play application and to the server software that handles users' requests. The Google Play application is now able to construct new versions of updated applications by applying a received patch to an old version of an application installed on an Android device.In our work in this paper, we assume that Google Smart Application Update is only performing a clean update process as a single task with no other statistics collection, accounting, sanity/integrity check, and alike.

II. RELATED WORK

Delta encoding is a well-known method of traffic reduction. [1] showedthatdelta encoding algorithms can be used to download only the difference between the two versions of an app. Size of the update files is reduced and the app update traffic is reduced by about 50% which leads to significant cost and energy savings. According to J. Wortham [2], AT&T says that the majority of the nearly $18 billion it will spend this year on its networks will be diverted into upgrades and expansion to meet the surging demands on the 3G network. An average iPhone owner can use ten times the network capacity used by the average smart phone user. As a result, more data is used. S.Musil [3] showedGoogle is now offering the ability to download delta updates from its google play for app users.The new smart downloads allow users to avoid downloading the entire app send only the incremental difference between the old and new versions thereby saving data for the app user and Google. [4]In mobile devices, delta encoding based Over-The-Air(OTA) wireless downloads are widely used to distribute operating system update. With dynamic OTA download mobile devices can connect to any type of wireless network, download the required radio software and reconfigure on demand.

Developers can use Update Direct for Android to update their android apps. [5] With Update Direct, a new library is included in application, which allows it to update itself using a delta encoding based method. Support for a variety of update interfaces, including mandatory updates, notice-only and update/ignore option for clients. [6] Google developed Courgette for encoding patches for its chrome browser.It benefits from exploring specifics of the transferred binary executable files, which makes patches 10 times smaller when compared to other delta encoding techniques.[7] The authors showed that Potential Benefits of Delta Encoding and Data Compression for HTTP:The Delta Encoding can be successfully used to reduce HTTP traffic by eliminating redundancy between the cached copy of file and its new version that needs to be downloaded. He also reported that 85% byte savings can be achieved for cached files. C.Percieval proposed [8] The Naive method produces competitively small patches for any executable files. If changes are done to the files data pointers will be modified throughout the file,to solve this pointer problem, bsdiff is used.The bsdiff probably attains close to the best possible performance from a platform independent tool.

III. THE DELTA++ METHOD

In previous work we showed that DELTA (Delta Encoding for Less Traffic for Apps) based on the bsdiff delta encoding tool can be successfully used to decrease application update traffic and enable savings in mobile networks and datacenters. Here we introduce a new DELTA++ method and implementation that further decreases the transmitted package size and thus achieves even greater savings.

The size of a patch computed by a delta differencing algorithm primarily depends on the amount of difference between two files. However usage of compression in files also affects the resulting patch size. If two files have very few differences it is possible that the compressed versions of these files might be highly different on a binary level because of the ways they were processed during compression. The same happens with the APK application package which is basically just a compressed archive of all the files that comprise an Android application. The main idea of DELTA++ is to determine the difference between the application files within an APK and not between the compressed APK packages themselves. Our original DELTA method generates a patch as a delta difference between the application’s old version APK file and the new version APK. The bsdiff delta encoding tool is used to produce this delta patch in the server and in the smartphone the patch is deployed using the bspatch tool. DELTA does not unpack the APK file and works in a generally similar fashion, we believe, to Google Smart Application Update. DELTA++ improves on DELTA by decompressing APK package and exploiting its specific structure. This allows it to produce much smaller patch sizes. The DELTA++ method can be divided into two parts, which are 1) patch computation and 2) patch deployment. Patch computation is done on the server side in the data center and needs to be done only once for each application patch version. Patch deployment is done on the user smartphone and is done each time an application is updated. The DELTA++ patch procedure is as follows:

1) The APK packages of the old and the new versions of an application are decompressed.

2) The manifest files of both versions are traversed to get the names, paths, and SHA-1 hash digests of all the files in two APK packages.

3) The files contained in the new version are marked as NEW (if the file is present in the new version but not present in the old one), UPDATED (if the file is present in both versions but its SHA-1 sums differ), SAME (if the file is present in both versions and the SHA-1 digests are the same) or DELETED (if the file is present in the old version but was deleted in the new one).

4) The files from the latest version that are marked as NEW are copied into the constructed patch.

5) The files from the latest version that are marked as UPDATED are given as input to the bsdiff delta encoding algorithm to compute differences between the old and new versions. This difference is then copied into the constructed patch. Sometimes the difference between small files can be greater than size of the files themselves because of the overhead associated with the delta file creation. In such cases, the new file is remarked as NEW and is copied into the patch.

6) The files that are marked as SAME remain untouched.

7) PatchManifest.xml file is created and included in the patch. It serves as a patch description and comprises information about which application version can be updated using the patch and what NEW files and delta differences between UPDATED files are contained in the patch. Information about files marked
DELETED is also included in PatchManifest.xml.

8) Finally, the constructed patch is compressed into a ZIP archive using bzip2. The compressed patch is then ready to be sent to an Android device for deployment.

DELTA++ patch deployment in the user smartphone (Android device) is as follows:
1) The received patch is decompressed into a temporary directory.
2) The APK package of the current application version is loaded using ApplicationInfo class.
3) The PatchManifest.xml file contained in the patch is used to delete all the files that are no longer required from the old application version.
4) All the differences in the patch are applied to the proper files thus updating them.
5) All NEW files from the patch are copied to the old application version. At this point, the old version contains exactly the same files as the new version of application.
6) The APK package is constructed by compressing all the files into a ZIP archive with an .apk extension.
7) Finally, the resulting APK package is installed using the Android PackageInstaller built-in application completing the application update.

We implemented DELTA++ as server side software, which constructs patches and serves them by request, and an Android application that deploys the received patches and updates the installed applications.

IV. EXPERIMENTAL EVALUATION

We compared DELTA++ to Google Smart Application Update by conducting an experiment. We took the 110 most popular Google Play Store applications in November 2012 generated and deployed delta patches based on previous versions of the apps. The most popular apps are tabulated in the Google Play store. We manually collected previous versions and archived them locally. The summary statistics for the 110 apps are: average app size = 6.21 MB, average number of downloads = 58 million, and average time since last update = 29 days.We generated and deployed patches using both DELTA++ and Google Smart Application Update. We used a PC with an Intel Core i5 2.30 GHz processor and 8 GB RAM to generate delta patches

Figure 2 shows the size of the patch, as generated by DELTA++ and by Google Smart Application Update for the top 110 most popular apps (ordered by the total number of app downloads). The patch size is compared to the size of the latest version of the application. For Google Smart Application Update (patch size shown by green bar), the average patch size was 45% of the latest application version's size, the minimum was 4% (for Bike Race Free application) and the maximum was 100% (for the Adobe Air application). For DELTA++ (patch size shown by red hash mark) the average patch size was 23% of the latest version size, the minimum was 0.1% (for Brightest Flashlight Free application) and the maximum was 81% (for WatchESPN application.The size of other patches is significantly affected by the differences in the application code between versions.Experimental results show that DELTA++ outperforms Google Smart Application Update in terms of reduction of patch size, which correlates directly to reduction of transmitted data.The average measured savings was 50%, the minimum was -75% (that is, there was an increase in patch size compared to the size of the full application itself), and the maximum was 97%. It can be SEEN that DELTA++ significantly reduces application update size and increases data savings from 55% (for Google Smart Application Update) to 77% (DELTA++). Application updating can be divided into four steps, which are 1) patch construction, 2) transmission of patch, 3) patch deployment on the device, and 4) installation of the updated application version. DELTA++ decreases the transmission time by reducing the transferred file size but requires more time to deploy a patch. Figure 3 shows the time to apply a DELTA++ patch and install updated application compared to the same time for Google Smart

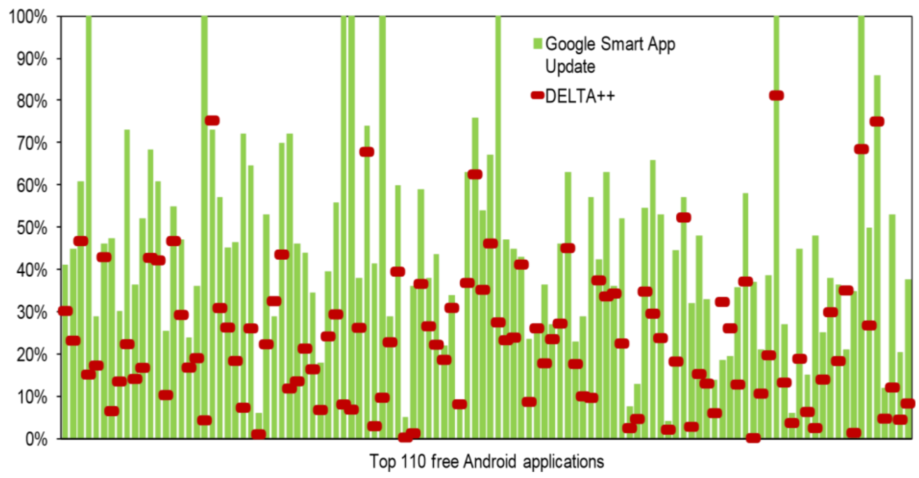

Figure 2. Patch size for DELTA++ compared to Google Smart Application Update

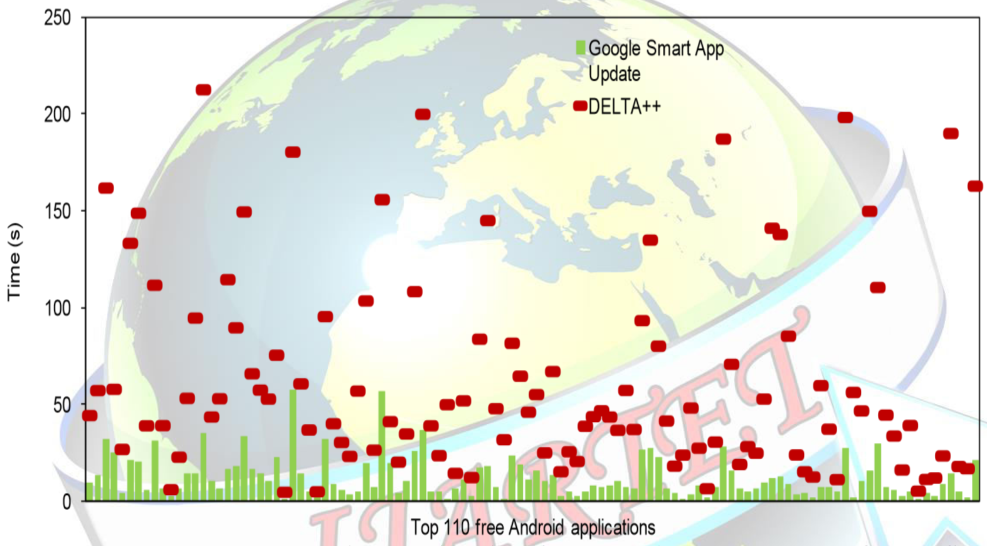

Figure 3. Patch deployment and installation time for DELTA++ compared to Google Smart Application Update

Application Update. For DELTA++, the average time was 62.5 seconds, the minimum was 4.6 seconds (for Barcode Scanner app) and the maximum was 212.3 seconds (for Angry Birds app). For Google Smart Application Update, the average time was 12.3 seconds, the minimum was 1.0 seconds (for ESPN Fantasy Football app) and the maximum was 57.5 seconds (for Temple Run app). The average patch deployment and app installation time for Google Smart Application Update is consistent with our assumption that Google's method does not compress or decompress APK files, which often takes tens of seconds (for the average APK size of 6.2 MB) in smartphone due to its limited resources.

V. USER CHARACTERIZATION

To estimate how much actual savings could be achieved with DELTA++ it is necessary to understand how users update applications on their devices. The average number of apps per Android smartphone in the U.S. at the end of 2011 was 32 and growing at 10% per year. Cisco reported that 33% of global mobile traffic was offloaded to Wi-Fi networks in 2012.

The 20 devices were owned by students at the University of South Florida. We found that the average number of apps per smartphone was 47, the average number of days between updates was 41, and the fraction of updates deployed via Wi-Fi was 37%.

VI. BANDWIDTH SAVINGS ESTIMATE

Table 1. Estimate of annual traffic reduction in the US

Measurement	Estimate
Number of Android smartphones [8]	60 million
Number of apps per smartphone[6]	32
Average size of an app update*	6.2 MB
Average days between updates*	29 days
App update traffic per year per phone	2.4 GB
Total app update traffic	146 PB
Total app update traffic w/ Google Smart	66 PB
Total app update traffic w/ DELTA++	34 PB
Extra savings with DELTA++	32 PB
Extra savings with DELTA++ in Cellular	21 PB

*Derived from Google Play Store

Table 1 shows a first order estimate of the traffic generated by app updates in the U.S. and the savings that could be achieved with full deployment of our proposed DELTA++ scheme in the Google Play store.. The average application size for the top 110 free apps in Google Play is 6.2 MB, which leads to approximately 2.4 GB in yearly applications update traffic for each user considering the average of 32 applications on an Android smartphone updated every 29 days. Google's Smart Application Update provides 55% savings and reduces this traffic down to 66 PB, while DELTA++ enables 77% savings that further decreases application updates traffic by 32 PB resulting in 34 PB yearly traffic. If 33% of updates are done using Wi-Fi then the extra savings in cellular networks is then approximately 21 PB.

VII. BATTERY DISCHARGE EVALUATION

DELTA++ reduces Android application update traffic by half compared to Google Smart Application Update. However, it takes DELTA++ approximately 50 seconds longer on average to deploy and install the received patch. Both methods use the Android PackageInstaller application to install the update so the additional 50 seconds are spent on patch deployment.

This application allows every Android user to get a first order estimate of how much power is used by certain device components (such as CPU, 4G, screen, etc.). Power consumption is estimated by maintaining certain conditions (for example, screen turned on) and measuring battery level every 30 seconds during a long period of time (in our case, 40 minutes for experiments with 4G radio and 60 minutes for other experiments). To understand the extra power draw during DELTA++ updates we measured power consumption during the following activities:

- *Idle*. Device is awake with its screen turned off, only background routines are running.
- *Screen*. Device is idle, but its screen is turned on (maximum brightness).
- *4G*. Device downloads a file using 4G radio.
- *Patch Deployment*. Device applies delta encoded patch.

The results showed that the extra 50 seconds spent on deployment of a single DELTA++ patch consumed about 0.15% of the smartphone's battery. We found that this same amount of battery charge is consumed by the screen in under 30 seconds. Considering that an average user updates about one application per day (average of 32 apps per smartphone updated every 29 days), DELTA++ consumes a negligible portion of the daily battery use of a smartphone.

VIII. SUMMARY AND FUTURE DIRECTIONS

As there is a exponential growth of mobile devices, this leads to huge network traffic for Android and Application updates. Delta++ have remarkable improvement in generating 50% smaller patches. Delta++ for android application could reduce yearly traffic in cellular networks by about 1.8% .This result would lead to remarkable savings for mobile operators and data centers. Hence less resources such as network bandwidth and number of servers would be require for serving patches. This has both economic and environmental benefits.

If we consider also Apple iPhone applications the overall savings could be much larger. Apple market share in 2012 was 33% of all smartphones. Our study of the top 110 application in the Apple App Store shows that the average iPhone application size is 26 MB with 64 days from the last update this leads to 6.8 GB annual update traffic for each user. Multiplied by the number of iPhones this results in 247 PB yearly traffic in the U.S. Initial experiments show that DELTA++ can be successfully used for iPhone apps (distributed using an IPA file, which is similar to an APK file) and can achieve 70% smaller iPhone updates on average. This means that approximately 180 PB in overall cellular traffic could be saved with DELTA++ if it were to be applied to iPhones. We note that changing technologies (for example, a move to microcells) and user behaviour's (for example, a move to using Wi-Fi more often) may change the estimates made in this paper for traffic savings in the cellular infrastructure. The savings, however, are still of benefit to microcells and to the data centers that serve updates.

IX. REFERENCES

[1] N. Samteladze and K. Christensen, "DELTA++: Reducing Application Updates size for Android Devices ," *Proceedings of IEEE Conference on Local Computer Networks*, pp. 212-215, October 2012.

[2] J. Wortham, "Customers Angered as iPhones Overload AT&T," September 26, 2009. URL: http://www.nytimes.com/2009/09/03/technology/companies/03att.html.

[3] S. Musil, "Google Play Enables Smart App Updates, Conserving Batteries," CNET News, August 16, 2012. URL: http://news.cnet.com/8301-1023_3-57495096-93/google-play-enables-smart-app-updatesconservingbatteries/.

[4] B. Bing, "A Fast and Secure Framework for Over-the-Air Wireless Software Download Using Reconfigurable Mobile Devices," *Communications Magazine, IEEE,* 44, no. 6, pp. 58-63, 2006. [14] Update Direct for Android, Pocket Soft. URL: http://pocketsoft.com/android_updatedirect.html.

[5] Update Direct for Android, Pocket Soft. URL: http://pocketsoft.com/android_updatedirect.html.

[6] "The Chromium Project, Software Updates: Courgette," URL: http://dev.chromium.org/ developers/design-documents/software-updates-courgette.

[7] J. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential Benefits of Delta Encoding and Data Compression for HTTP," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 4, pp. 181-194, ACM, 1997.

[8] C. Percival, "Naive Differences of Executable Code," draft paper dated 2003. URL: http://www.daemonology.net/bsdiff.