Reducing Size and Deployment Time of Android Application Update using DELTA++

Shweta Kale
Siddhant College of Engineering,
Sudumbare

Abhaya Bulbule
Siddhant College of Engineering,
Sudumbare

ABSTRACT
In this paper, we describe how to reduce size of android application as well as reduce the deployment time in installation of android application update using DELTA++. We are executing this by using the basic DELTA encoding algorithm .for this we need to pack and unpack the APKs that is the executable file of google smart application .To modify and update the patches DELTA++ modifying technique is used. Patches provided by google smart application are firstly constructed and then deployed. In this size reducing technique we are taking the application update in consideration hence we update the patches called files using deltas the advance feature of this is to decode the compressed patches into the delta again. Because of this we are able to reduce the deployment time And user need not to be decode the latest version every time. And can install faster.

General Terms
In this paper we used some algorithm as differential algorithm and SHA-I algorithm.

1. INTRODUCTION
In late 2014 more than 114 million people owned a smartphone whichis approximately half of all mobile device users. The existence of a sole application market for each operating system and the availability of low-cost high-speed wireless networks provided an opportunity to update mobile applications more frequently than desktop softwarea systemlevel view of application updating for smartphones showing an application market with its data centers that serve application updates, wireless networks that transportinformation between data center and end-user, and user smartphones.The opportunity todeliver updates easily to all the application users is used by developers to introduce newfeatures, add new content, or fix bugs and security vulnerabilities.Frequentapplication updates increase smartphone users download traffic, which is usually limited by the mobile operator. These updates also increase traffic in wireless networks, andoutgoing traffic in data centers that serve app updates.

The current amount of network bandwidth and require more deployment time is a big challenge for data centers and mobile operators,Smartphone applications are mostly updated incrementally – by extending functionality, adding new content, fixing existing problems, etc. Thus, the significant part of then application remains unaffected during an update.Delta encoding is a technique that is used to compute the difference, or patch, between two files.

This patchis then can be used to construct the newer file version from the old one. Delta encodingcan be used to shrink the size of an application update by transmitting only the changesbetween the old application version and the new one.Two delta encoding based methods – called DELTA and DELTA++ – are presented. They significantly decrease the traffic generated by application updates and enable savings for mobile operators, data centers, andsmartphone users.

The primary purpose of developing a better method to update mobile applicationsis to reduce the bandwidth generated by updates and reduce the deployment time require for installation of updates.it is not enough to shrink the update size as much as possible. Any savings achieved by reduction of traffic can become negligible due to the costly changes that need to be made in the current infrastructure or the software used to distribute application updates. Thus, it is very important to develop new updating methods that can be easily implemented on top of the existing infrastructure and with minimum changes in the software used.The Android application package file (APK) is the file format used to distribute and install applications in Google's Android operating system. APK files are based on the JAR file format used in Java applications. An APK is essentially a ZIP archive that contains all parts of an Android application such as program byte code, resources, assets, certificates, and manifest file. The APK file is created by compiling the application's code and resources and compressing all of its files into one package

The primary purpose of developing a better method to update mobile applications is to reduce the traffic generated by updates. However, it is not enough to shrink the update size as much as possible. Any savings achieved by reduction of traffic can become negligible due to the costly changes that need to be made in the current infrastructure or the software used to distribute application updates. Thus, it is very important to develop new updating methods that can be easily implemented on top of the existing infrastructure and with minimum changes in the software used.

Along with low implementation costs, a better updating method should not exacerbate user experience both for smartphone users and application developers as it can cause their transition to another application market or mobile operating system. Thus, any new method to update smartphone applications should not require any additional effort from application developers or affect the way users update applications installed on their smartphones.

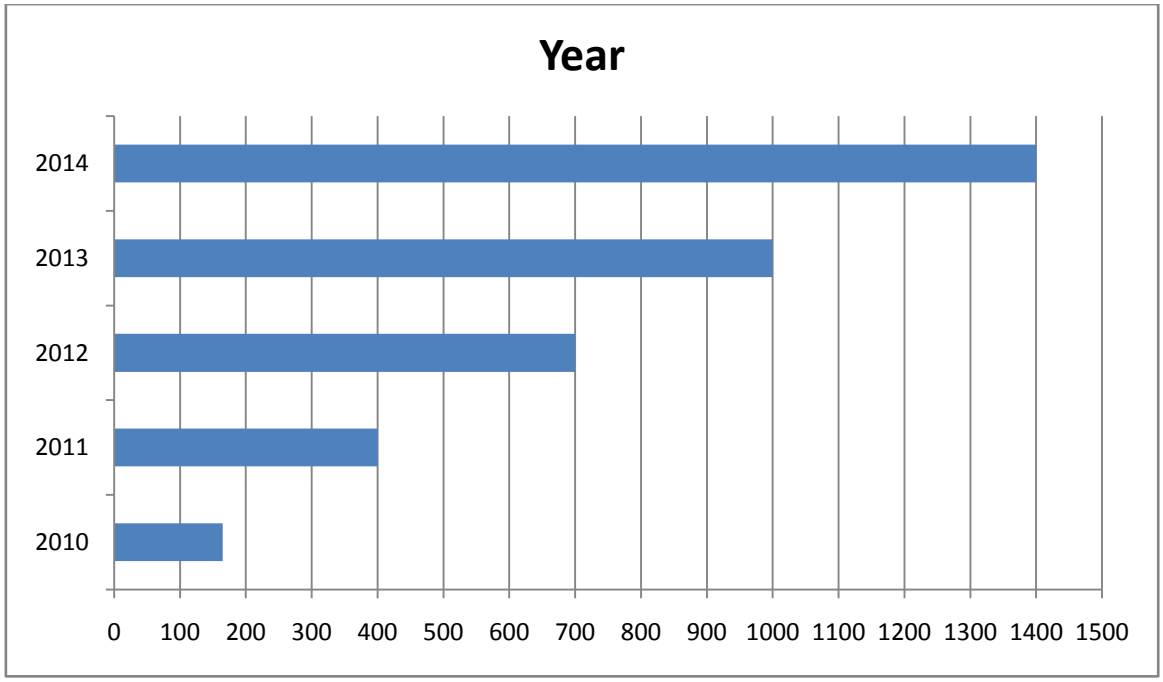

Fig 1.1: this figure shows Growth of applications in Google Play from 2010 to June 2014. Updates add a considerable percentage to number of application downloads. Data fromhttp://en.wikipedia.org/wiki/Google_Play

Figure 1.1 shows the growth of the number of applications in Google Play storein just five years. To reduce application update traffic, GoogleSmart Application Update got developed, which uses a compression method transparent toapplication developers and Android users. Modifications to the Google Playapplication and the server software enable Google Play to construct new versionsof updated applications by applying a patch to the application versioninstalled on the user's Android device. Although this solution has made inroads into traffic reduction, its compression methods are not optimal. Notably, deltaencoding is at the Android Application Package (APK) level only, which limitsthe possible reduction in patch size.

1.1 Patch Generation

The size of the patch that the delta differencing algorithm computes depends primarily on the extent to which the old and new file versions differ, but compression can also affect that size. If two files have only a few differences, the

Compressed file versions might differ considerably on a binary level because of how they were processed during compression. The same is true of the APK, which, as the sidebar "Inside an Android Application Package File" makes clear, is basically a compressed archive of all the files comprising an Android application. DELTA++ aims to determine the difference between the applicationfileswithin an APK, as opposed to the APKs themselves. DELTA generated a patch as a delta difference between the application's old and new APK versions. The bsdiff delta encoding tool produces this delta patch in the server, and the bspatchtool deploys the patch in the smartphone. DELTA works generally like Google Smart Application

Update in that both use delta encoding and neither unpacks the APK. DELTA++ improves on both methods by decompressing the APK and exploiting its specific structure. The result is a much smaller patch. The method underlying DELTA++ has two main parts: patch construction and patch deployment. Patch construction takes place on the server side in the data center and is done only once for each application patch version. Patch deployment, which takes place on the user's smartphone, is done each time an application updates.

1.2 Construction

DELTA++ patch constructionconsists of eight steps: _ DELTA++ _rstdcompresses the old and new APKversions and_ traverses the manifest files to get the names,paths, and SHA-1 hash digests for all the files intwo APKs. It then marks the les in the newversion. If the le is in the new version but notin the old one, it is marked NEW. If the le isin both versions but its SHA-1 sums differ, it ismarked UPDATED. If the le is in both versions,and the SHA-1 digests are the same, it is markedSAME. Finally, if the le is in the old version butdeleted in the new one, it is marked DELETED.After marking is complete,DELTA++ copies the les marked NEW into the constructed patch.To compute differences between the old andnew APK versions, DELTA++ inputs the remarked as UPDATEDto the bsdiff delta encodingalgorithm and copies this difference intothe constructed patch. Because of the overheadin creating the delta le, the difference betweensmall les can sometimes exceed the size of theAPK les themselves. In these cases, DELTA++re-marks the updated le as NEW and copiesit into the patch. The les marked SAMEremain untouched.After marking the les and computing versiondifferences, DELTA++ creates the Patch-Manifest.xml le and includes it in the patch.

The le is essentially a patch description,comprising information about which applicationversion can be updated using the patch, what NEW les and delta differences betweenUPDATED les are in the patch, and informationabout les marked DELETED.Computation concludes with patch compressioninto a zip archive using bzip2. Thecompressed patch is then ready to be sent to anAndroid device for deployment.

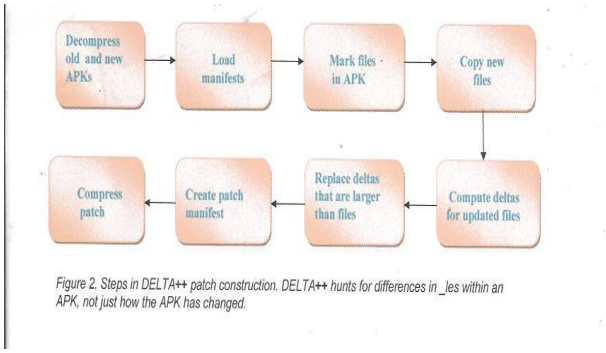

Figure 2. Steps in DELTA++ patch construction. DELTA++ hunts for differences in _les within an APK, not just how the APK has changed.

1.3 Deployment

Figure 3 shows the seven steps in DELTA++patch deployment to the user's Android smartphone. Deployment begins by decompressing the received patch into a temporary directory. DELTA++ then uses the Application Info class to load the APK of the current application version and uses the PatchManifest.xml file inthe patch to delete all the files that are no longer required. By applying all the differences in the patch to the proper files, DELTA++ updates them. It then copies all the NEW files from the patch to the old application version. At this point, the old and new application versions contain exactly the same files. The next step is to constructthe APK by compressing all the files into a zip archive with a .apk extension. Finally, DELTA++ uses the Android Package Installer built into the application to install the resulting APK.

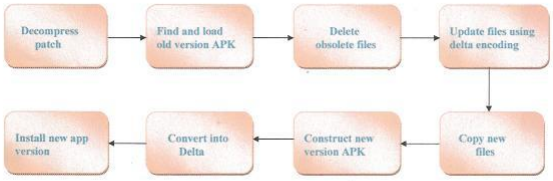

Figure 3. Patch deployment with DELTA++. Relative to Google Smart Application Update, DELTA++ constructs smaller patches, which decreases download time.

1.4 Patch Size

Figure 4 shows relative patch size ordered by total number of application downloads and evaluated relative to the size of the application's latest version. In some cases, both methods produced patches that were only slightly smaller than the application's full version — typically when the developer had added numerous resource files (images, video files, third-party libraries, and so on) in the updated application. Differences in application code between versions significantly affect patch size in part because tools such as ProGuard obfuscate byte code, deliberately making it harder to decompile. Such obfuscation introduces file differences on the binary level, causing the delta encoding algorithm to produce larger patches. Figure 4 shows that DELTA++ outperforms Google Smart Application Update in reducing patch size, which correlates directly to less transmitted data. The average measured savings was 50 percent, the minimum was –75¦percent (the patch size increased relative to the application size), and the maximum was 97 percent. DELTA++ significantly reduced application update size and increased data savings: 77 percent reduction for DELTA++ versus 55 percent for Google Smart Application Update.

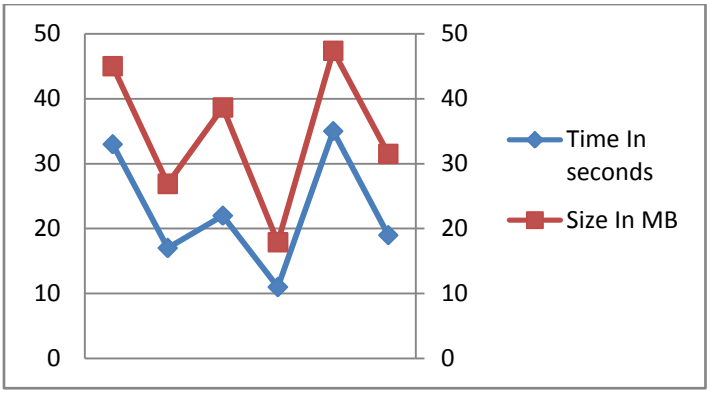

Figure 4 :This figure shows the size and the deployment time of the application in the smartphone before using the delta++ techniques.

1.5 Deployment Time

DELTA++ decreases transmission time by reducing the transferred file size but requires more time to deploy a patch. Figure 5 shows the time to apply a DELTA++ patch and install the updated application compared to the same time for Google Smart Application Update. The average patch deployment and application installation time for Google Smart Application Update is consistent with our assumption that Google's method doesn't compress or decompress APK files, which often takes tens of seconds in a smartphone (for an APK of 6.2 Mbytes) because of its limited resources.

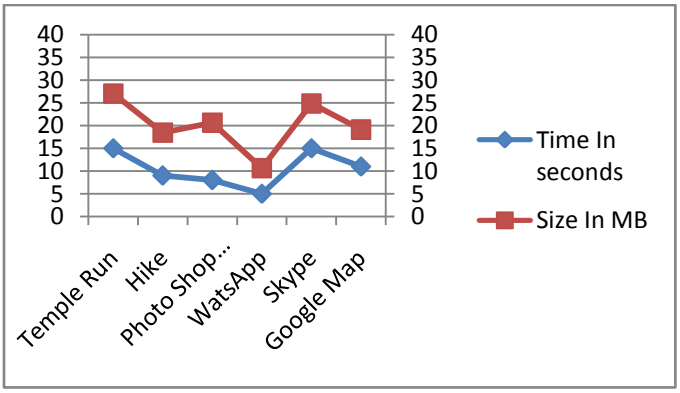

Figure 5 :This figure shows the reduced size of application and deployment time while installation after using DELTA++ techniques.

1.6 ALGORITHM:

1.6.1 DELTA++ Patch Construction Algorithm:

1. The APK packages of the old and the new versions of an application are decompressed. During decompression APK files are treated like ZIP archives.
2. The manifest files of both versions are traversed to get the names, relative paths, and SHA-1 hash digests of all the files (in the APK) included in both versions.
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) or SAME (if the file is present in both versions and the SHA-1 digests are the same). It is worth noting that the application's metafiles (contained in

META-INF directory inside APK package) are not presented in manifest, but are also marked as UPDATED.

4. The files contained in the old version are marked as DELETED if the file is present in the old version but was deleted in the new one.
5. The files from the latest version that are marked as NEW are just copied into the constructed patch.
6. 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 re-marked as NEW and is copied into the patch.
7. The files that are marked as SAME remain untouched.
8. 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.
9. 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.

1.6.2 Delta Encoding Algorithms:

Delta encoding algorithms aim to compute a difference between files and efficiently encode this difference in a way that allows using the resulting patch to construct the target file from the reference file. File differencing problem arises from the string-to-string correction problem.

1.6.3 SHA-1 algorithm:

SHA1 stands for "Secure Hashing Algorithm.160-bit hash function. This was designed by the National Security Agency (NSA) to be part of the Digital Signature Algorithm. Cryptographic weaknesses were discovered in SHA-1, and the standard was no longer approved for most cryptographic uses after 2010.

2. ACKNOLEDGEMENT

We would also like to take this opportunity to express my profound gratitude to express my profound gratitude and deep regard to my project guide Mr. Deepak Gupta , for her exemplary guidance, valuable feedback and constant encouragement throughout the duration of the project. His valuable suggestions were of immense help throughout my project work. His perceptive criticism kept me working to make this project in a much better way. Working under him was an extremely knowledgeable experience for me.

I would also like to give my sincere gratitude to all the friends and colleagues who filled in the survey, without which this research would be incomplete.

3. REFFERNCES

[1] Nikolai Samteladze, "Delta Encoding Based Methods to Reduce the Sizeof Smartphone Application Updates",January 2013. URL: *University of South Florida*, nikolay.samteladze@gmail.comFollow this and additional works at: http://scholarcommons.usf.edu/etd.

[2] TorstenSuelNasirMemon*,"*Algorithms for Delta Compression and Remote Files Synchronization*"*CIS DepartmentPolytechnic UniversityBrooklyn, NY 11201.

[3] J. Wortham, "Customers Angered as iPhones Overload AT&T," 26 Sept. 2009; www.nytimes.com/2009/09/03/ technology/companies/03att.html.

[4] S. Musil, "Google Play Enables Smart App Updates, Conserving Batteries," *CNET News*, 16 Aug. 2012; http://news.cnet.com/8301-1023_3-57495096-93/google playenables- smart-app-updates-conserving- batteries/.

[5] C. Percival, "Naive Differences of Executable Code," draft, 2003; www.daemonology.net/bsdiff.

[6] N. Samteladze and K. Christensen, "DELTA: Delta Encoding for Less Trafic for Apps," *Proc. IEEE Conf. Local Computer Networks*, 2012, pp. 212–215.

[7] "State of the Media: Mobile Media Report Q3 2011," Nielsen, 15 Dec. 2011; www.nielsen.com/us/en/reports/2011/stateof-the-media--mobile-media-report-q3-2011.html.

[8] "Cisco Visual Networking Index: Global Mobile Data Trafic Forecast Update, 2012–2017," Cisco, 6 Feb. 2013;

[9] www.cisco.com/en/US/solutions/collateral/ns341/ns525/ ns537/ns705/ns827/white_paper_c11-520862.html.