Minimizing the Size of Android Application Updates and Network Traffic

M.Sangamithra, V.N.Sowmya and S.SriVidhyalakshmi, Ms.V.Sathiya
Department of Computer Science and Engineering,
Panimalar Engineering College,
Poonamallee.
,

Abstract- **The android Smartphone applications are of large in number and each of the applications receiving periodical updates create network traffic. The Google Smartapp Update reduces the traffic size up to 49 percent and this method further reduces the data traffic by applying compression to the files that contribute more to the size of the APK. By further providing an overall compression to the APK, the updates travel consumes much lesser bandwidth than the original APK. These compression techniques results in lighter application loads into the server and reduces the cellular network bandwidth up to 72 percent. Similar methods can be integrated in the end user Smartphone to reduce the usage of memory.**

Index Terms- Updates, APK, Server, Network traffic, Compression.

I. INTRODUCTION

Today, Android has become one of the most popular operating systems that is being used in the Smartphone. Its fame has led to the development of more than 1.5 million applications of which 1 million apps are regularly used and 14 percent are of low quality. Such massive amount of applications is supported in android as it is developed in Java which is open source software. Making applications compatible with android is much easier. Many of the applications develop newer versions and periodically update them. These applications in the Google server consume much space as there is repetition of the original file as updates are uploaded along with the original app file. Also these files move to application server consuming larger network bandwidth. The end-users where angered over the high data traffic [1]. Google Play is the application server which is the store house of android compatible applications. Initially, loading the app packages (APK). It is a zip archived form which consisted of six main elements containing the whole application. These APK's reduced the traffic to a considerable amount. Later, Google introduced a more versatile method of reducing the size of the updated versions to be loaded into the server. 'Delta encoding' was the method first proposed as "Google Smartapp updates" [2] in the application server side and they updated only the differences in the old and new APK files to the application server. This reduced the US cellular network traffics to 1.7 percent i.e., by nearly 50 percent. Further enhancement to Delta was Delta++ which has been proposed to unpack the APK and compress the individual modules so that the cellular traffics reduce up to 77% [9]. Such techniques are not available in iOS and even the technique at the scratch in the android can be applied to iPhone application servers to make tremendous traffic reduction.

II. EXISTING SYSTEM

Many researchers have conducted experiments to reduce network traffics where one of them was RTpatch. It is a shareware that extracted only the difference between the old and new versions which reduced the version size by 99% but they were compatible only with the windows OS [12]. The versatile form for android was converting the application to Android Application packages (APK) which is a zip archived form that contains all parts of an Android application, including program byte code, resources, assets, certificates, and the manifest file. They contain six main elements META-INF directory, Classes.dex, lib directory, Resources.arsc, res directory, AndroidManifest.xml [4]. These files form a single APK. Further Google has employed "Delta Encoding" as 'Google Smartapp Update' which finds the difference between the old APK and the new APK using the bsdiff algorithm and the difference is taken as a 'patch' which is update to the end-user. This method of delta encoding has reduced the traffic up to 49 percent [4].

III. PROPOSED SYSTEM

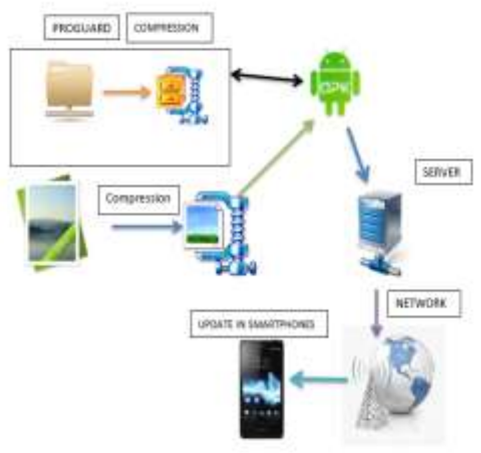
In our paper, we provide another turn of idea to reduce the network traffic due to updates a little further. Studies show that image files consume more space than text files. Thus compression over text files which contains the java code, .xml files and other such files are compressed using 'proguard' which relies inside the ADT (Android Developer Tool). The proguard.cfg file is directly created in the root directory on creating an android app using ADT and it removes all unused and debug classes from the code and shrinks it.

We invoke this in the APK and shrink the java file. The second compression we propose is the compression of images [13]. It is experimentally proved that images occupy more memory than text [11] and thus compress image files using compression tools like jstrip, PUNYpng, etc., This provides lossless compression of the image up to 16.1 percent less than the original size [11]. This compressed file is integrated into the APK (if image files are necessary for the app) and then uploaded to the server for updating. By these methods we reduce the network traffic to about 20 percent along with Delta encoding, thus making lighter app loads.

IV. METHODOLOGY DESCRIPTION

Delta++ is a new technique for reducing the size of android application updates [9]. It provides substantial reduction in network traffic produced by application updates. It does so by computing the difference between the older and newer versions of the application files. This difference is utilized to create the patch which contains the relevant files for updating. A smart phone application is updated by downloading only this difference of old version and the new one and applying the delta patch in smart phones. This process includes decompression of Android APK package and compression is performed on each modules of the APK [9].

V. ARCHITECTURE DIAGRAM



The compression is done by the administrator before uploading the updated versions into the server. Text files contribute only a minor amount to the size of an APK. Files containing graphics like images and video increase the file size drastically. So here we apply various techniques to reduce the size of APK by compressing the images and text file.

Fig. A overall process flow that will provide lesser network bandwidth usage when apps are updated. Due to this the size of an APK reduces even with an improved feature. On downloading, a file size less than the older version is installed in the smart phone.

VI. APK CREATION

Here we act as the administrator and create our own application along with its updated feature. A cloud acts as a server where we upload the newer versions for updating the application. Here we make use of an android database named SQLite... The traditional File/Open operation calls sqlite3_open () to attach to the database file [9]. Updates happen automatically as application content is revised so the File/Save menu option becomes superfluous. The File/Save As menu option can be implemented using the backup API. Before the files are bundled in to application package, they are individually compressed using various tools which will be discussed below.

VII. COMPRESSION OF AN APK

In order to reduce the size of an apk we attempt to compress the files present inside the application package[14]. This compression is done at the server by the content provider. This process uses the advanced delta encoding technique called delta++.This uses the bsdiff algorithm[3] which calculates the difference between two files by unpacking the apk’s.

A. PROGUARD

The ProGuard tool shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names. The result is a smaller sized .apk file that is more difficult to reverse engineer. ProGuard is integrated into the Android build system, so you do not have to invoke it manually [13]. As soon as an apk file is created, proguard.cfg file is automatically generated in the root directory of the project. This file defines how ProGuard optimizes and obfuscates your code, so it is very important to understand how to customize it for your needs[15]. To enable ProGuard so that it runs as part of an Eclipse build, set the proguard.config property in the <project_root>/project. Properties file. The path can be an absolute path or a path relative to the project's root [13]. Android compiles Java source files to Java byte code. The Java byte code is next converted to Dalvik byte codes by the “dx” tool. In between these two steps we can apply Proguard to shrink the Java byte codes. The result is impressive: a reduction of about 30% or more in .apk size. This results in the application downloads faster, taking up less space on the limited flash storage on the phone and starting up faster. The only drawback, it makes analyzing an application crash more difficult, as the stack trace of an exception has only the new short names and thus is pretty obscure[16].

B.*IMAGE COMPRESSION*

The images are compressed before constructing an apk file. Here we use lossless compression algorithms. The image remains the same in its quality before and after compression. The Lempel-Ziv-Welch algorithm[14] provides an encoding and decoding of images without any loss.

B.1 ENCODING

A high level view of the encoding algorithm is shown here:

1. Initialize the dictionary to contain all strings of length one.
2. Find the longest string W in the dictionary that matches the current input.
3. Emit the dictionary index for W to output and remove W from the input

B.2 DECODING

1. Read the value from the encoded input encoded input and output the corresponding String from the initialized dictionary.
2. The decoder proceeds to the next input value.
3. Concatenates this string with the first character of the next input after decoding it.
4. The process is repeated until there is no more input.

VIII. DEPLOYMENT

After the user had successfully registered to enter in to the application an update founder is provided. User can check all the applications installed into their Smartphone. They can also check whether
an update for the respective application is available in the server. If the updates are found then the size of the application is checked. If the user wishes to update then he can download the newer version and install it in to their Smartphone.

IX. FUTURE ENHANCEMENT

The compression can be still improved by unpacking the apk’s and comparing the size of old and new versions of an application. As the behavior of an apk is clearly understood we can also involve horizontal matching .An update which can be compatible to two different versions of an app could also be created.

X. CONCLUSION

Due to compression applied to the apk’s the entire size of an application is small even with an updated feature. So it eliminates the delay in downloading in a heavy traffic network. It also saves the power as installation is fast and based on the files Compressed, memory is also efficiently used. Downloading the app a smaller application package travels through the network which reduces the amount of bandwidth required per MB. This reduces the traffic in the network. Due to this saving in memory the speed and efficiency of the mobile remains the same even though it is updated with so many applications. This would contribute more advantages to iphone.

REFERENCES

[1] J. Wortham, “Customers Angered as iPhones Overload AT&T,” 26 Sept. 2009; www.nytimes.com/2009/09/03/technology/companies/03att.html.
[2] 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/.
[3] C. Percival, “Naive Differences of Executable Code,” draft, 2003; www.daemonology.net/bsdiff.
[4] N. Samteladze and K. Christensen, “DELTA: Delta Encoding for Less Traffic for Apps,” Proc. IEEE Conf. Local Computer Networks, 2012, pp. 212–215.
[5]“State of the Media: Mobile Media Report Q3 2011,” Nielsen,15Dec. 2011; www.nielsen.com/us/en/reports/2011/stateof-the-media--mobile-media-report-q3-2011.html.
[6] “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012–2017,” Cisco, 6 Feb. 2013;www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.
[7] “comScore Reports July 2012 US Mobile Subscriber Market Share,” comScore, 4 Sept. 2012; www.comscore.com/Insights/Press_Releases/2012/9/comScore_Reports_July_2012_US_Mobile_Subscriber_Market_Share.
[8] “Background on CTIA’s Semi-Annual Wireless Industry Survey,”CTIA,2012; http://_les.ctia.org/pdf/
[9] N. Samteladze and K. Christensen, “Delta++: reducing the size of Android application updates”, IEEE Computer society, Apr. 2014.
[10]Compression tool, http://www.creativebloq.com/design/image-compression-tools-1132865
[11]image files http://stackoverflow.com/search?q=reducing+apk+size.
[12] Pocket Soft®'s RTPatch Binary Diff, www.pocketsoft.com
[13] Proguard, http://developer.android.com/tools/help/proguard.html.
[14] http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
[15] http://java.llp.dcc.ufmg.br:8080/apiminer/static/docs/tools/help/proguard.html
[16] http://blog.javia.org/android-and-proguard-the-perfect-pair/