DELTA++: Reducing the Size of Android Application Updates

Nikolai Samteladze, Ken Christensen
Department of Computer Science and Engineering
University of South Florida
4202 East Fowler Avenue, ENB 118
Tampa, FL 33620

nsamteladze@mail.usf.edu, christen@csee.usf.edu

Correspondence author: Ken Christensen (christen@csee.usf.edu, (813) 974-4761)

Abstract

Compression can be a useful tool to reduce network bandwidth usage. We have developed an improved compression method for Android application updates called DELTA++ that achieves an additional 50% traffic reduction when compared to Google Smart Application Update. We estimate that the extra reduction in network bandwidth use from using DELTA++ would be about 1.8% of all annual cellular traffic in the US. Increased battery discharge from DELTA++ was found to be negligible. If similar methods were to be used for iPhone application updates even larger savings could be achieved.

Keywords

Mobile applications; data compaction and compression, delta encoding, Android

Introduction

In late 2012 the Google Play store had more than 675,000 applications available and over 25 billion total application downloads with more than 1.5 billion new downloads every month [1]. 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 serve 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 [2].

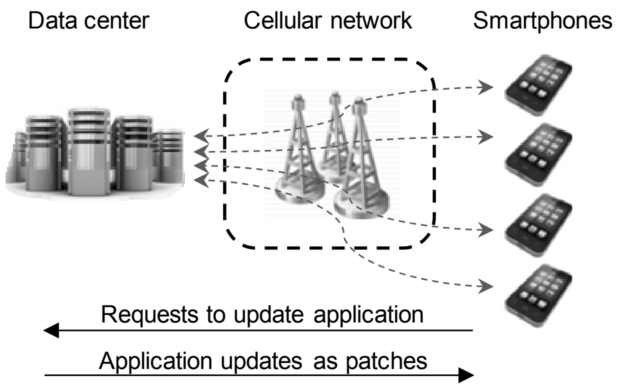
Data center
Cellular network
Smartphones
Requests to update application
Application updates as patches

Figure 1. System view of application updating

In June 2012 Google announced a new technology – Google Smart Application Update – that reduces application update traffic [3]. 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 application update mechanism, called DELTA++, that provides further reduction of traffic generated by application updates. DELTA++ is an extension of our previous work in [4] (we note that the initial submission of [4] 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 DELTA++ is that we decompress the Android APK package (see the sidebar for a quick tutorial on APK packages) 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 large-scale savings that could be achieved with a full-scale deployment of DELTA++.

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.

The DELTA++ Method

In previous work [4] we showed that DELTA (Delta Encoding for Less Traffic for Apps) based on the `bsdiff` delta encoding tool [5] 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 described in [4] 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 re-marked 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. This software is freely available from [6].

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. An HTC Thunderbolt smartphone with a single core, 1000 MHz Snapdragon processor and 768 MB RAM was used to deploy patches. Our response variables of interest were patch size (compared to both the original application size and the patch size generated by Google Smart Application Update) and deployment time including both download and installation. For each time measurement, 10 repetitions were done and an average was taken.

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). In some cases both methods produced patches that were only slightly smaller than the full version of the application. Such large patches occur when numerous new resource files (for example, images, video files, or third-party libraries) have been added in the new version of the application. The size of other patches is significantly affected by the differences in the application code between versions. This may be due to the use of tools such as ProGuard that obfuscates byte code with the goal of making it harder to decompile. Such obfuscation of byte code introduces new differences between two files on the binary level and leads to larger patches produced by a delta encoding algorithm. 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

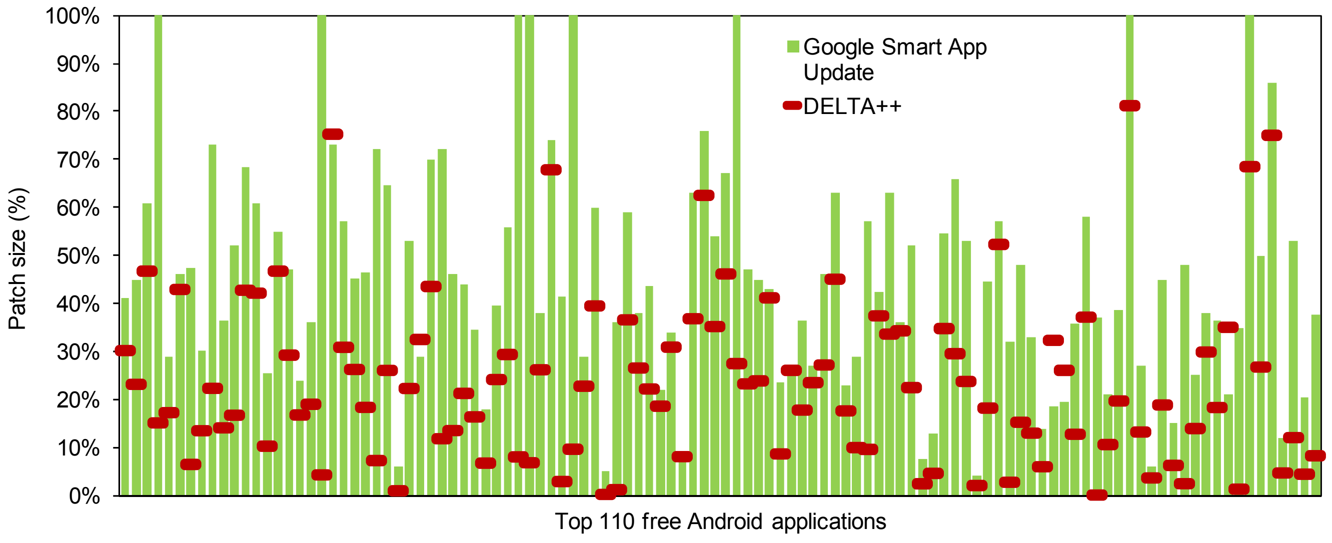

Figure 2. Patch size for DELTA++ compared to Google Smart Application Update

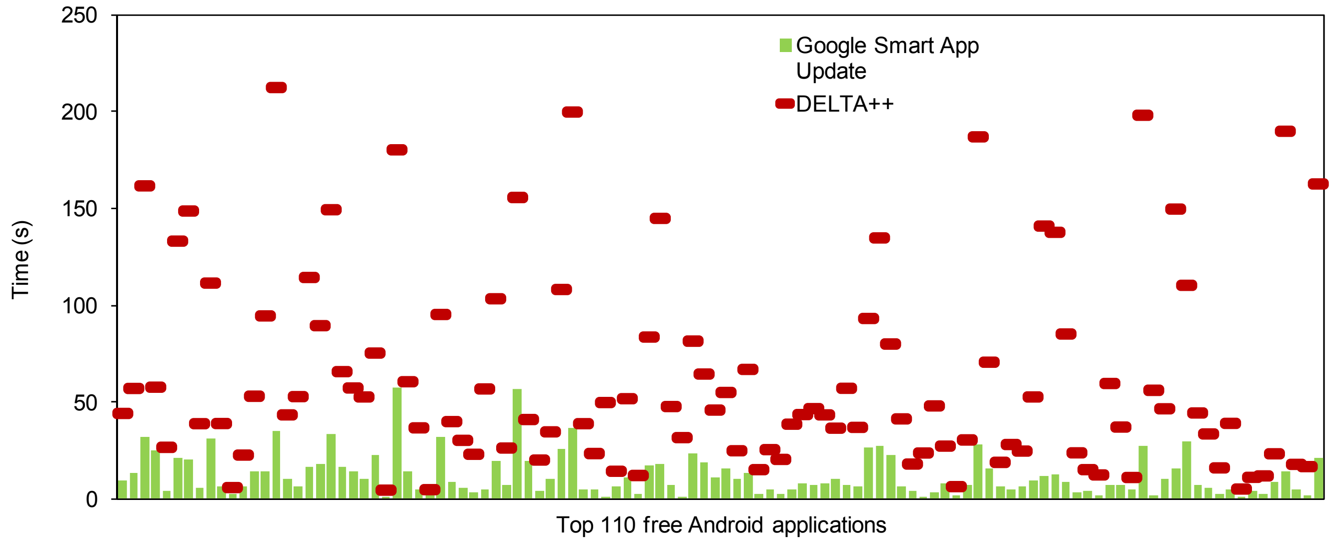

Figure 3. Patch deployment and installation time for DELTA++ compared to Google Smart Application Update

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.

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. According to [7] 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 [8].

We conducted a cursory study to characterize user behavior. We created an Android application (named DELTA Statistics) to collect user behavior data. DELTA Statistics is freely available from the Google Play store. We installed DELTA Statistics on 20 Android devices and collected information during three months – from November 2012 until January 2013. 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%. Our collected results were very similar to the results reported in [7] and [8].

Bandwidth Savings Estimate

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. There were more than 114 million smartphone users in the U.S. in 2012 with 52% of smartphones (60 million) running the Android operating system [9]. 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. These updates add up to the total of 146 PB yearly traffic for all users. 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. According to [10] the total U.S. wireless data traffic in the first half of 2012 was 590 PB (so about 1180 PB per year). *Thus, our roughly 21 PB savings represents over 1.8% of all cellular traffic – this is a significant savings.*

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

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.

A key question is, what is the impact of DELTA++ on battery discharge? To evaluate additional power consumption caused by DELTA++ we created an Android application called DELTA Energy Profiler, which is freely available from the Google Play store. 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.

In all experiments we used the same HTC Thunderbolt smartphone used for our above described DELTA++ evaluation. The device was equipped with an out-of-the-box standard Li-ion 1400mAh battery and had 32 popular applications installed. Energy use is measured in percentage of battery that is consumed per second. All experiments were done in an outside urban environment on a business day. During the experiments with 4G radio we varied the downloaded file size from 512 KB to 10MB and found that although energy cost per megabyte varies for different sizes, energy cost per second stays approximately the same.

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. It is worth noting that the actual extra battery discharge from DELTA++ will be less than 0.15% per update because of the reduced patch download time. 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.

Related Work

Delta encoding is a well-known method of traffic reduction. Mogul et al. [11] showed that 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. The authors reported that delta encoding enables approximately 85% byte savings for the cached files. Targeting specific file format during delta encoding can even further improve the compression rate and, thus, traffic reduction. Google developed Courgette [12] 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. DELTA++ in turn targets Android APK packages and explores their internal structure to achieve smaller patch sizes.

In mobile devices, delta encoding based Over-the-Air (OTA) wireless downloads [13] are widely used to distribute operating system updates. Besides Google Smart Application Update, which is currently used to update all apps distributed through the Google Play store, developers can use Update Direct for Android [14] to update their Android apps. With Update Direct for, a new library is included in application, which allows it to update itself using a delta encoding based method. Although, Update Direct for Android reduces applications update traffic, it prevents applications from being distributed through the Google Play store and does not support the Google App Engine. DELTA++ differs from Update Direct for Android because it allows developers to distribute their apps through Google Play and achieve traffic reduction with no additional cost.

Summary and Future Directions

The growing popularity of mobile devices that host multiple applications leads to significant network traffic from application updates. We focused on Android and application updates as served by the Google Play Store. Our DELTA++ significantly improves upon Google Smart Application Update in generating 50% smaller patches. We estimated that the deployment of delta encoding similar to DELTA++ for Android application updates could reduce yearly traffic in cellular networks by about 1.8%. This would lead to significant savings for mobile operators and data centers as less resources (both network bandwidth and number of servers) would be required for serving patches. This has both economic and environmental benefits.

If we consider also Apple iPhone applications the overall savings could be much larger. According to [9] 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 behaviors (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.

Acknowledgment

This work was partially supported by a Google research award (Christensen). We thank the anonymous reviewers for their helpful comments that have improved the quality of this paper.

References

[1] K. De Vere, "Android Reaches 25 Billion App Downloads, 675,000 Total Apps Available," September 26, 2012. URL: http://www.insidemobileapps.com/2012/09/26/android-reaches-25-billion-app-downloads-675000-total-apps-available/.

[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-updates-conserving-batteries/.

[4] N. Samteladze and K. Christensen, "DELTA: Delta Encoding for Less Traffic for Apps," *Proceedings of IEEE Conference on Local Computer Networks*, pp. 212-215, October 2012.

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

[6] N. Samtealdze, DELTA Software, January 2013. URL: https://github.com/NikolaiSamteladze.

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

[8] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012 – 2017", CISCO, February 6, 2013. URL: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/ white_paper_c11-520862.html.

[9] "comScore Reports July 2012 U.S. Mobile Subscriber Market Share," comScore, September 4, 2012. URL: http://www.comscore.com/Insights/Press_Releases/2012/9/comScore_Reports_July_2012_US_Mobile_Subscriber_Market_Share.

[10] "Background on CTIA's Semi-Annual Wireless Industry Survey," CITA, 2012. URL: http://files.ctia.org/ pdf/CTIA_Survey_MY_2012_Graphics-_final.pdf.

[11] 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.

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

[13] 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.

Sidebar

The Android APK file

The Android application package file (APK) is the file format used to distribute and install applications in Google's Android operating system. 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. An APK package usually contains the following:

- `META-INF` directory that contains the application's manifest (`MANIFEST.MF`), certificate (`CERT.RSA`) and list of resources (`CERT.SF`). The manifest file lists all the files included in APK packages together with their checksums (SHA-1 digest is used).
- `classes.dex` that is the compiled application's code in .dex file format, which was designed specifically for the Android operating system and is usually twice smaller than a .jar (Java Archive) file derived from the same code (such savings are partially enabled by using shared strings pools).
- `lib` directory that includes the processor-specific compiled code.
- `resources.arsc` that contains compiled application resources (for example, XML files).
- `res` directory with all the application's resources that cannot be compiled into `resources.arsc` (for example, icons or pictures used in the application).
- `AndroidManifest.xml` file that serves as an additional Android manifest file and contains information about the distributed application.

Author biographies

Nikolai Samteladze has an MS in Computer Science from the University of South Florida. He is currently employed in the banking industry in Florida. His primary interest is in developing highly scalable performance-oriented systems. He is a member of IEEE and ACM.

Ken Christensen is professor and director of the undergraduate program in the Department of Computer Science and Engineering at the University of South Florida. His primary research focus is on reducing the energy use of computer and communication networks – green networks. He has made contributions to Energy Efficient Ethernet (EEE) and network connectivity proxying. Christensen has a PhD in Electrical and Computer Engineering from North Carolina State University. He is a senior member of IEEE and a member of ACM and ASEE. Contact him at christen@cse.usf.edu.