DELTA++: Reducing the Size of Android Application Updates

This method of creating and deploying update patches improves on Google Smart Application Update by first unpacking the Android Application Package and then compressing its elements individually. The smartphone user can then download a smaller patch. Experiments show that performance yields 49 percent more reduction relative to Google's solution, increasing the savings in cellular network bandwidth use and resulting in lighter application server loads. This reduction in Android application-update traffic could translate to a 1.7 percent decrease in annual US cellular traffic. Similar methods applied to iPhone application updates could yield even greater savings.

Nikolai Samteladze and Ken Christensen
University of South Florida

Application updates continue to add considerable traffic to the cellular network and increase the load on the data center servers that handle them. As of late 2013, the Google Play store was offering more than 1 million applications and processing more than 50 billion application downloads (http://en.wikipedia.org/wiki/Google_Play). Given that each application carries its own set of feature updates and bug fixes, an update release every few weeks has become the norm. Indeed, mobile operators spend billions of dollars on network upgrades every year to keep pace with the burgeoning mobile traffic generated by the explosion of applications and their upkeep.[1]

Figure 1 shows the growth of the number of applications in Google Play in just five years. To reduce application update traffic, Google developed Google Smart Application Update,[2] which uses a compression method transparent to application developers and Android users. Modifications to the Google Play application and the server software enable Google Play to construct new versions of updated applications by applying a patch to the application version installed on the user's Android device. Although this solution has made inroads into traffic reduction, its compression methods are not optimal. Notably, delta encoding is at the Android Application Package (APK) level only, which limits the possible reduction in patch size.

To address this shortcoming and reduce update traffic even more, we extended our Delta Encoding for Less

 Published by the IEEE Computer Society 1089-7801/14/$31.00 © 2014 IEEE

Traffic for Apps (DELTA), an update mechanism based on the bsdiff delta encoding tool.[3] In our experiments,[4] which predated the release of Google Smart Application Update, we showed that DELTA could successfully decrease application update traffic and enable savings in the cellular network and data centers. We then refined DELTA, creating DELTA++, which enables an even larger traffic decrease and greater savings.

Both DELTA++ and Google Smart Application Update use delta encoding, which computes the difference, or diff, between two files, enabling software developers to construct the newer version from the old file. Thus, the content provider can update a smartphone application merely by transferring the diff and then applying the delta patch locally in the smartphone.

However, unlike Google Smart Application Update, DELTA++ unpacks the APK and then compresses its individual modules. Our experimental results show that DELTA++ can reduce application update size by 77 percent on average, relative to a 55 percent average size reduction possible with Google Smart Application Update. This figure represents a 49 percent improvement in compression over Google Smart Application Update. The tradeoff is that DELTA++ requires a more complex application patch and thus more time to deploy that patch on the smartphone. Our experiments show that additional battery use is negligible.

We have implemented DELTA++ as server-side software, which constructs update patches and serves them by request, and as an Android application that deploys the received patches and updates the installed applications. Our software is a free download available at https://github.com/NikolaiSamteladze.

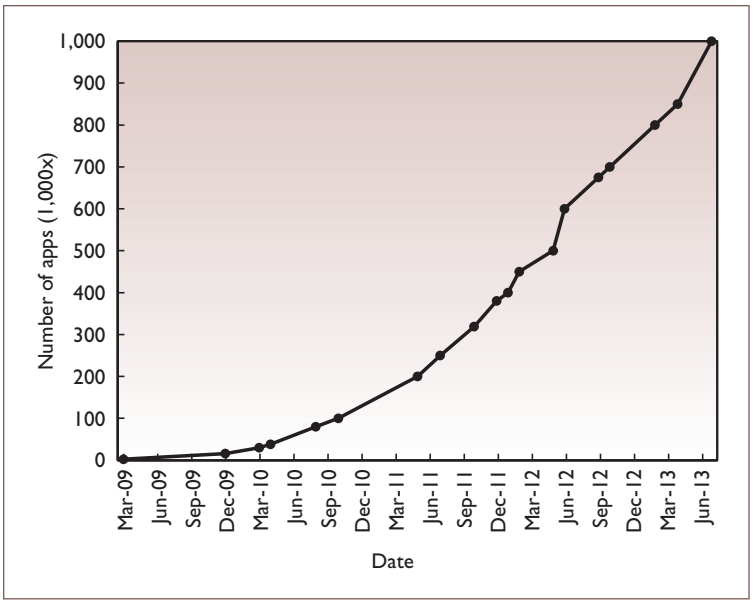

Figure 1. Growth of applications in Google Play from March 2009 to June 2013. Updates add a considerable percentage to the already skyrocketing number of application downloads. Data from http://en.wikipedia.org/wiki/Google_Play.

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 application *files* within 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 bspatch tool 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.

Construction

As Figure 2 shows, DELTA++ patch construction consists of eight steps: ❶ DELTA++ first decompresses the old and new APK versions and ❷ traverses the manifest files to get the names, paths, and SHA-1 hash digests for all the files in

Inside the Android Application Package File

The Android Application Package (APK) is the format for distributing and installing applications in the Android operating system. An APK is essentially a zip archive that contains all parts of an Android application, including program byte code, resources, assets, certificates, and the manifest file. The application's author creates an APK file by compiling the application's code and resources and compressing all its files into one package. An APK file usually contains six main elements:

- The *META-INF* directory contains the application's manifest (MANIFEST.MF), certificate (CERT.RSA), and resources list (CERT.SF). The manifest lists all the files in the APK as well as their checksums (SHA-1 digest).
- *Classes.dex* is the compiled application's code in the in.dex file format designed for the Android operating system. It is usually half the size of a .jar (Java Archive) file derived from the same code, deriving its reduction in part from the use of shared strings pools.
- The *lib directory* includes the processor-specific compiled code.
- Resources.arsc contains compiled application resources, such as XML files.
- The *res directory* has all the application's resources that cannot be compiled into resources.arsc, such as icons or pictures that the application uses.
- *AndroidManifest.xml* contains information about the distributed application and serves as an additional Android manifest file.

Working at the level of these elements, and viewing the APK as more than a single file, can allow a far greater degree of compression and thus a smaller patch size with all the benefits that this brings to reducing network bandwidth use and reducing load on the application servers.

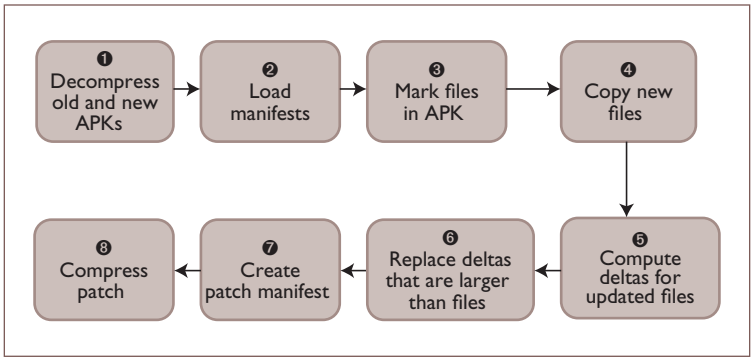

Figure 2. Steps in DELTA++ patch construction. DELTA++ hunts for differences in files within an APK, not just how the APK has changed.

two APKs. It then ❸ marks the files in the new version. If the file is in the new version but not in the old one, it is marked NEW. If the file is in both versions but its SHA-1 sums differ, it is marked UPDATED. If the file is in both versions, and the SHA-1 digests are the same, it is marked SAME. Finally, if the file is in the old version but deleted in the new one, it is marked DELETED. After marking is complete, DELTA++ ❹ copies the files marked NEW into the constructed patch.

To compute differences between the old and new APK versions, DELTA++ ❺ inputs the files marked as UPDATED to the bsdiff delta encoding algorithm and copies this difference into the constructed patch. Because of the overhead in creating the delta file, the difference between small files can sometimes exceed the size of the APK files themselves. In these cases, DELTA++ ❻ re-marks the updated file as NEW and copies it into the patch. The files marked SAME remain untouched.

After marking the files and computing version differences, DELTA++ ❼ creates the PatchManifest.xml file and includes it in the patch. The file is essentially a patch description, comprising information about which application version can be updated using the patch, what NEW files and delta differences between UPDATED files are in the patch, and information about files marked DELETED.

Computation concludes with ❽ patch compression into a zip archive using bzip2. The compressed patch is then ready to be sent to an Android device for deployment.

Deployment

Figure 3 shows the seven steps in DELTA++ patch deployment to the user's Android smartphone. Deployment begins by ❶ decompressing the received patch into a temporary directory. DELTA++ then ❷ uses the ApplicationInfo class to load the APK of the current application version and ❸ uses the PatchManifest.xml file in the patch to delete all the files that are no longer required. ❹ By applying all the differences in the patch to the proper files, DELTA++ updates them. It then ❺ 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 ❻ construct

the APK by compressing all the files into a zip archive with a .apk extension. Finally, DELTA++ ❼ uses the Android PackageInstaller built into the application to install the resulting APK.

Comparative Evaluation

To evaluate the traffic reduction possible with DELTA++, we constructed and deployed delta patches for 110 of the most popular Google Play applications in November 2012 (measuring popularity by number of downloads), and compared DELTA++'s performance to that of Google Smart Application Update. The delta patches were based on previous versions of the applications, which we manually collected and archived locally.

For the 110 applications, we assumed an average application size of 6.21 Mbytes, average download number of 58 million, and an average time since last update of 29 days.

For all aspects of our evaluation, we used a PC with an Intel Core i5 2.30-GHz processor and 8 Gbytes of RAM to generate delta patches, and we deployed the patches on an HTC Thunderbolt smartphone with a single core, 1,000-MHz Snapdragon processor, and 768 Mbytes of RAM.

Application updates consist of four steps: patch construction, transmission, and deployment on the device, and finally installation of the updated version.

Our response variables of interest were *patch size* and *deployment time*. For patch size, we looked at how DELTA++ compared with both the original application size and the patch size that the Google Smart Application Update generated. Both these variables relate to patch construction and transmission. Deployment time includes both download and installation. For each time measurement, we performed 10 repetitions and took the average value.

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.

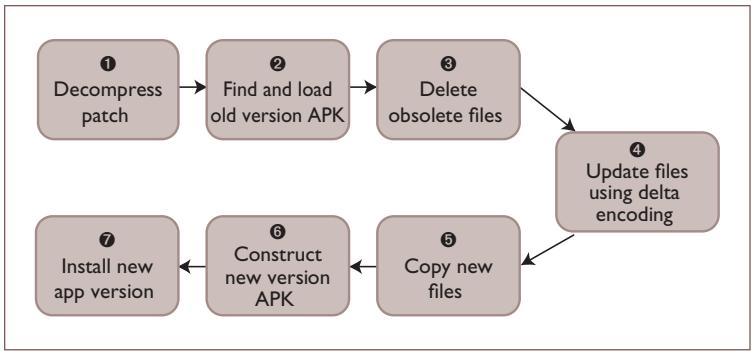

Figure 3. Patch deployment with DELTA++. Relative to Google Smart Application Update, DELTA++ constructs smaller patches, which decreases download time.

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.

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.

Estimated Savings

To understand the bandwidth and battery discharge savings possible with DELTA++, we first looked at how many applications an average user might have and how users behave in updating applications. According to a December 2011

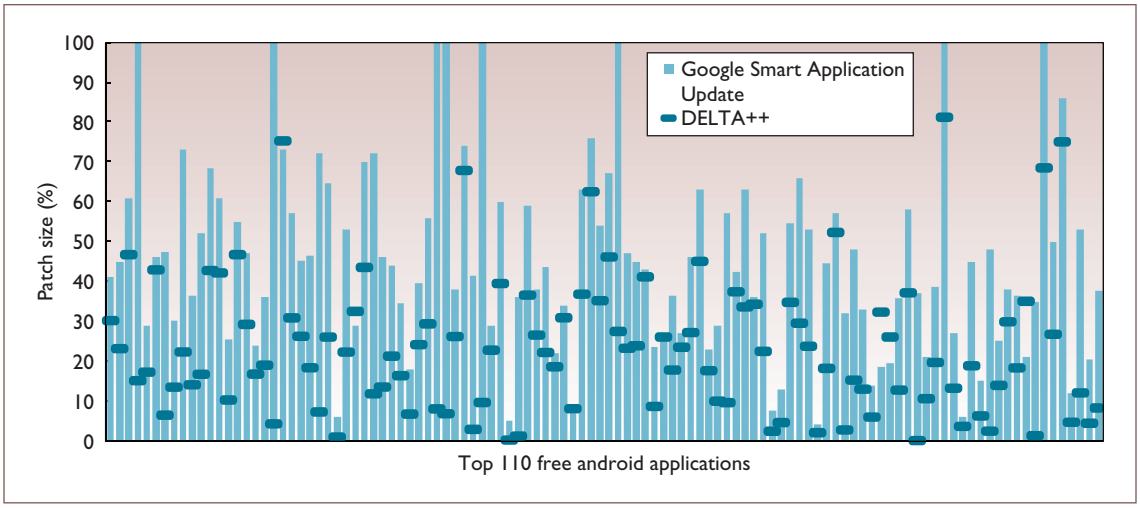

Figure 4. Comparative patch size for Google Smart Application Update and DELTA++ for 110 applications. For Google Smart Application Update (thin blue bar), the average patch size was 45 percent of the latest application version's size, and the minimum and maximum sizes were 4 percent (Bike Race Free) and 100 percent (Adobe Air). For DELTA++ (dark blue hash mark) the average patch size was 23 percent of the latest version size, and the minimum and maximum sizes were 0.1 percent (Brightest Flashlight Free) and 81 percent (WatchESPN).

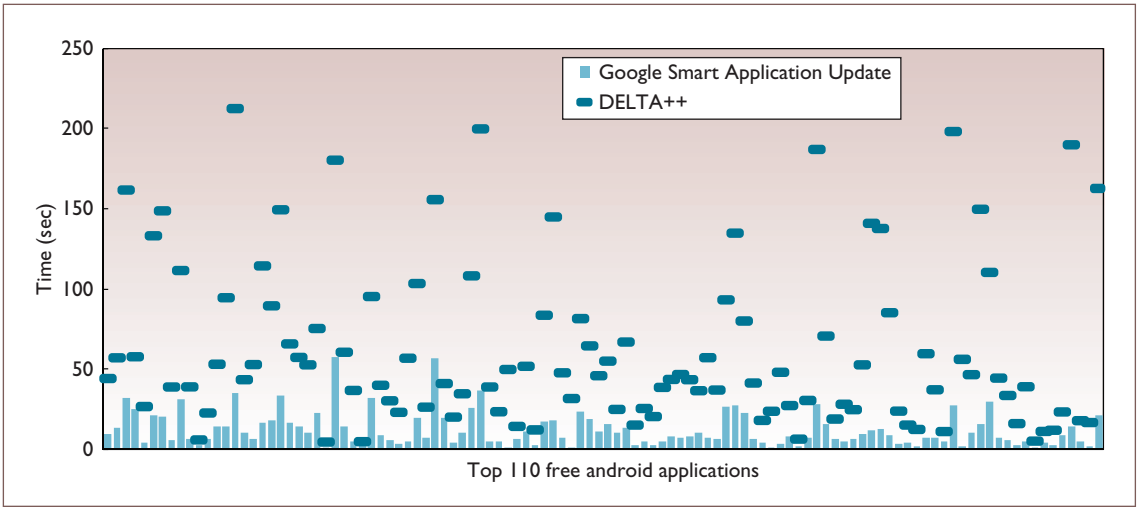

Figure 5. Patch deployment and installation time for DELTA++ and Google Smart Application Update. For DELTA++, the average time was 62.5 sec, the minimum was 4.6 sec (Barcode Scanner), and the maximum was 212.3 sec (Angry Birds). For Google Smart Application Update, the average time was 12.3 sec, the minimum was 1.0 sec (ESPN Fantasy Football), and the maximum was 57.5 sec (Temple Run).

Nielsen report,[5] an Android smartphone in the US averaged 32 applications, with a projected annual growth of 10 percent. Users also tend to rely more on Wi-Fi networks — Cisco's networking index showed that, in 2012, users offloaded 33 percent of global mobile traffic to Wi-Fi networks.[6]

To characterize user behavior locally, we conducted a cursory study of students at the University of South Florida from November 2012 to January 2013. We created and installed DELTA Statistics, an application to collect user behavior data, on 20 Android devices (the application is free through Google Play). Results showed that users averaged 47 applications, the average days between updates was 41, and 37 percent of those updates were deployed through Wi-Fi. Our local results were very close to those in the Nielsen report and Cisco projections.

Related Work in Delta Encoding

Delta encoding is a well-known traffic reduction method. In 1997, Jeff Mogul and colleagues[1] showed that delta encoding could reduce HTTP traffic by eliminating redundancy between the cached file copy and the new version to be downloaded. At that time, delta encoding enabled approximately 85 percent byte savings for cached files.

Targeting a specific file format during delta encoding can also improve the compression rate and further reduce traffic. Google's Courgette,[2] which encodes patches for the Chrome browser, benefits from exploring the specifics of the transferred binary executable files (see http://dev.chromium.org/developers/design-documents/software-updates-courgette). The resulting patches are 10 times smaller relative to those possible with other delta encoding techniques.[2] DELTA++ in turn targets Android Application Packages (APKs) and explores their internal structure to achieve smaller patch sizes.

In mobile devices, over-the-air wireless downloads based on delta encoding[3] are a popular way to distribute operating system updates. Besides Google Smart Application Update, which focuses on applications distributed through Google Play, developers can use Update Direct for Android to update their Android applications (see http://pocketsoft.com/android_updatedirect.html). Applications compatible with Update Direct include a new library that lets the application update itself using a method based on delta encoding. Although, Update Direct reduces applications update traffic, applications can't be distributed through Google Play, and the mechanism doesn't support the Google App Engine. In contrast, DELTA++ reduces update traffic, and developers can distribute their applications through Google Play at no additional cost.

References

1. J. Mogul et al., "Potential Benefits of Delta Encoding and Data Compression for HTTP," *ACM SIGCOMM Computer Communication Rev.*, vol. 27, no. 4, 1997, pp. 181–194.
2. "The Chromium Project, Software Updates: Courgette," http://dev.chromium.org/developers/design-documents/software-updates-courgette.
3. B. Bing, "A Fast and Secure Framework for Over-the-Air Wireless Software Download Using Reconfigurable Mobile Devices," *IEEE Comm.*, vol. 44, no. 6, 2006, pp. 58–63.

Traffic Reduction

Table 1 shows a first-order estimate of the traffic that application updates generate in the US, as well as the savings possible with DELTA++ if it were available through Google Play. In 2012, the US had more than 114 million smartphone users, with 52 percent (60 million) running the Android operating system.[7] Using an average application size of 6.2 Mbytes (average in the 110 free applications we considered), and figuring that each user updates an average of 32 applications every 29 days, the annual application-update traffic is approximately 2.4 Gbytes per Android smartphone user. With 60 million users, these updates total 138 Pbytes of yearly traffic.

With its 55 percent reduction, Google Smart Application Update reduces traffic to 62 Pbytes (138 × 0.45), but DELTA++ at 77 percent savings pushes that reduction another 22 percent to 32 Pbytes (138 × 0.23). If 33 percent of updates are through Wi-Fi, the extra savings in cellular networks is approximately 20 Pbytes. According to a 2012 report by CTIA,[8] US wireless data traffic in the first half of 2012 was 590 Pbytes, translating to approximately 1,180 Pbytes per year. Thus, our savings of roughly 20 Pbytes represents more than 1.7 percent of all cellular traffic — a significant savings. This calculated savings percentage should still hold even as applications and users increase.

Table 1. Estimate of annual traffic reduction in the US.

Measurement	Estimate
Number of Android smartphones	60 million
Number of apps per smartphone	32
Average size of an app update*	6.2 Mbytes
Average days between updates*	29 days
App update traffic per year per phone	2.4 Gbytes
Total app update traffic	138 Pbytes
Total app update traffic w/ Google Smart Application Update	62 Pbytes
Total app update traffic w/ DELTA++	32 Pbytes
Savings with DELTA++ over Google Smart Application Update	30 Pbytes
Extra savings with DELTA++ in cellular	20 Pbytes

*Derived from Google Play store

Battery Discharge

Although DELTA++ is clearly superior to Google Smart Application Update in patch size and traffic reduction, its advantage in deployment time is less straightforward. Both methods use the Android PackageInstaller application to install the update, but it takes DELTA++ approximately 50 seconds longer on average to deploy the patch.

To determine the effect of this longer deployment on the smartphone battery discharge, we

created another Android application, DELTA Energy Profiler, and made it free through Google Play. The application enables every Android user to see how much power the device components (CPU, 4G, screen, and so on) are consuming. The application provides a first-order estimate by maintaining certain conditions (for example, screen turned on) and measuring battery level every 30 seconds during a long period. For our purposes, we set that period to 40 minutes for experiments with 4G radio and 60 minutes for all other experiments.

Energy Profiler measured power consumption as the percentage of battery consumed per second during four activities:

- *Idle*. Device is awake with its screen turned off and only background routines running.
- *Screen*. Device is idle, but its screen is turned on (maximum brightness).
- *4G*. Device is downloading a file using 4G radio.
- *Patch deployment*. Device is applying the delta encoded patch (includes patch construction and installing the newly constructed version).

In all experiments, we used the HTC Thunderbolt smartphone from our traffic-reduction evaluations, which relies on an out-of-the-box standard Li-ion 1400 mAh battery, and we installed the top 32 free applications in the 110-application sample on the device.

We conducted all experiments outside in an urban environment on a business day. During the experiments with 4G radio, we varied the downloaded file size from 512 Kbytes to 10 Mbytes and found that, although energy cost per megabyte varies across file size, energy cost per second stays approximately the same, primarily because the overhead of establishing a connection is fairly constant. Thus, it takes less energy to download 10 Mbytes of data once than to download 1 Mbyte of data 10 times.

The results showed that the extra 50 seconds spent on the deployment of a single DELTA++ patch consumed about 0.15 percent of the smartphone's battery — the same amount of battery charge that a screen consumes in less than 30 seconds. Consequently, the additional energy that DELTA++ consumes relative to Google Smart Application Update roughly equals what a smartphone screen alone consumes in 30 seconds. Moreover, the actual extra battery discharge from DELTA++ will be even less per update because of the reduced time to download the smaller patch. 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 smartphone's daily battery use.

The growing popularity of mobile devices that host multiple applications leads to significant network traffic from application updates. For Android application updates through Google Play, DELTA++ significantly improves on Google Smart Application Update. Adding Apple iPhone applications could greatly increase the 1.7 percent savings in cellular traffic (Android application updates only). According to comscore.com,[7] Apple devices constituted 33 percent of all smartphones in 2012. Our study of the top 110 applications in the Apple App Store shows that the average iPhone application size is 26 Mbytes, with 64 days from the last update.

Given these statistics, each user generates 6.3 Gbytes of update traffic annually. Multiplied by the number of iPhones, the total yearly traffic from iPhone application updates in the US is 231 Pbytes. Initial experiments show that DELTA++ can reduce iPhone application updates by 70 percent on average (applied to the iOS Application Archive file, which is the Apple iOS counterpart to the Android APK file). Consequently, applying DELTA++ to iPhone updates could save approximately 108 Pbytes in overall cellular traffic. Changing technologies or altering user behavior (for example, moving to microcells or using Wi-Fi more often) could change our traffic savings estimates, but we're confident that the benefit to the cellular network and data center servers will remain.

Acknowledgments

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 article.

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-play-enables-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, 15 Dec. 2011; www.nielsen.com/us/en/reports/2011/state-of-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://files.ctia.org/pdf/CTIA_Survey_MY_2012_Graphics-_final.pdf.

Nikolai Samteladze is employed at C1 Bank in St. Petersburg, Florida. His primary interest is in developing highly scalable performance-oriented systems. Samteladze has an MS in computer science from the University of South Florida. He's a member of IEEE and ACM. Contact him at nsamteladze@mail.usf.edu.

Ken Christensen is a professor in the Department of Computer Science and Engineering at the University of South Florida and director of the university's computer science and engineering undergraduate program. His research interests are green networks — reducing the energy use of computer and communication networks. Christensen has a PhD in electrical and computer engineering from North Carolina State University. He's a senior member of IEEE and a member of ACM and the American Society for Engineering Education. Contact him at christen@cse.usf.edu.