APPDYNAMICS

An AppDynamics Business White Paper

Mobile app performance explained

Developing a mobile strategy has never been more important to companies than today. According to a report from Kleiner Perkins, mobile applications now account for 15% of all Internet traffic, which represents 1.5 billion users worldwide; a Pew Internet survey in May 2013 concluded that 91% of American adults own a cell phone and 56% of American adults own a smart phone; and in 2014, mobile Internet usage should surpass desktop Internet usage. Whether you have a brick-and-mortar store or you are an Internet-based business, the need to go mobile is greater than ever. Figure 1 shows the pattern of mobile application usage versus desktop application usage.

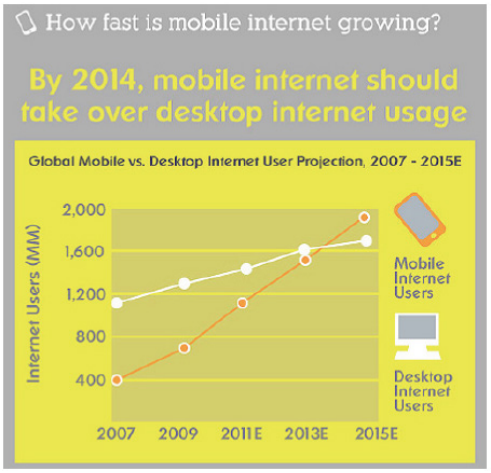
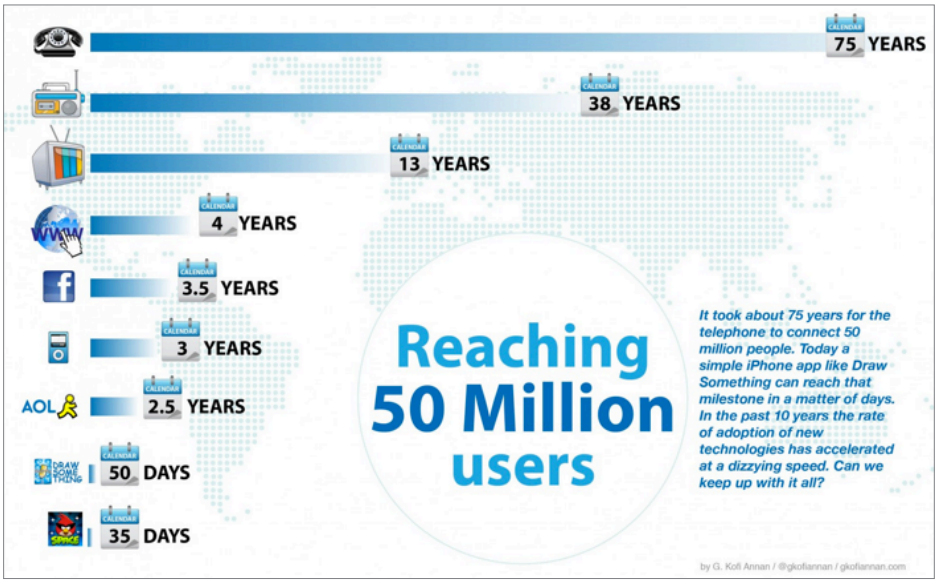


Figure 1. Mobile internet usage to overtake desktop internet usage in 2014

"Mobile app users have already established a set of expectations that you need to meet or you might just find yourself with a negative App Store review."



The process of bringing a mobile application to market, the drivers, requirements, and goals are typically very different from traditional modern desktop and web-based applications and consequently so are the challenges faced in doing this. This paper reviews some of the challenges that mobile application developers face, presents strategies to overcome these challenges, and then discusses the need to measure mobile application performance and capture performance and demographic data in order to identify and resolve performance issues.

Mobile app performance challenges

Developing a high-performing mobile application presents unique challenges from developing a traditional or web application. It's still important to write clean, clear, and highly optimized code, but there are a variety of new performance concerns that mobile applications introduce. Specifically you need to be aware that you are running on a device that has limited resources and you need to be respectful of those resources. Furthermore, mobile app users have already established a set of expectations that you need to meet or you might just find yourself with a negative App Store review. This section reviews the following topics:

– Performance concerns with mobile applications
– Native vs. Mobile web applications
– The challenge of targeting a variety of mobile devices

Performance concerns with mobile applications

Mobile application performance is defined by the user's perception of how well the application performs. This means that the performance of your application is measured by how responsive it is, how quickly it starts up, how well it uses device memory, how well it uses device power, and, in the case of an animation or game, how high its frame rate is, or in other words, how smooth the animation or game behaves.

"How many mobile apps that take an excessive amount of time to startup have you stopped using?"

Figure 2. Perception of responsiveness

A users expectation of responsiveness alters depending upon the device they are using. Consider how you use your phone and how you use your web browser: are they the same? No. When you click on a button on your web browser you understand that the browser is making a call to a server or at least "the internet", so you are prepared to wait for a response. But when you tap a button within an application on your phone, you expect it to respond immediately – if you experience even a second or two delay you'll be tempted to press the button again or, after four or five seconds, you'll probably kill the app and restart it. Therefore, when building a mobile application, the user experience should be of main focus. All networks calls and complex computations need to be performed in a background thread and, if you do need wait for a server response, then display a busy indicator while you do so to inform the user that the application is working. Furthermore, you should try to load just enough data to draw a screen in your application and allow your user to start working while you load the remaining data in the background.

A tangential aspect of responsiveness is startup time. How many mobile apps that take an excessive amount of time to startup have you stopped using? If your application needs to perform several tasks before starting, how can you mitigate the time required to perform those tasks and present your user interface in a timely manner? We can gauge startup time in three ways, as shown in figure 3.

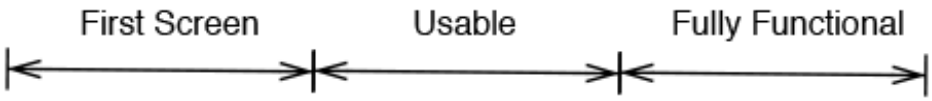


Figure 3. Measuring startup times

Figure 3 can be summarized as follows:

- **First screen:** the time required to show the user something on the screen
- **Usable:** the time that the application becomes usable and interactive
- **Fully functional:** the time when the application has loaded all of its resources and is fully functional

Because we are focusing on the user's perception of the application responsiveness, the first two points are the most critical. You have a choice when deploying your application to the App Store of bundling resources with your application or loading them from a server later. Bundled resources require a larger download and installation time, but facilitate a faster startup time. Therefore it is important to bundle commonly used resources, and most importantly, the resources you want to show your user when the application starts up. Any other lesser-used resources are better loaded from a server in a background thread while your application is running. It is a delicate balancing act between startup time, device memory storage, and memory usage, but you need to have your core resources available and loaded in memory as your application starts. The key is to show your user something as quickly as you can and then make your application interactive. Once your application is interactive you can perform the rest of your tasks in the background.

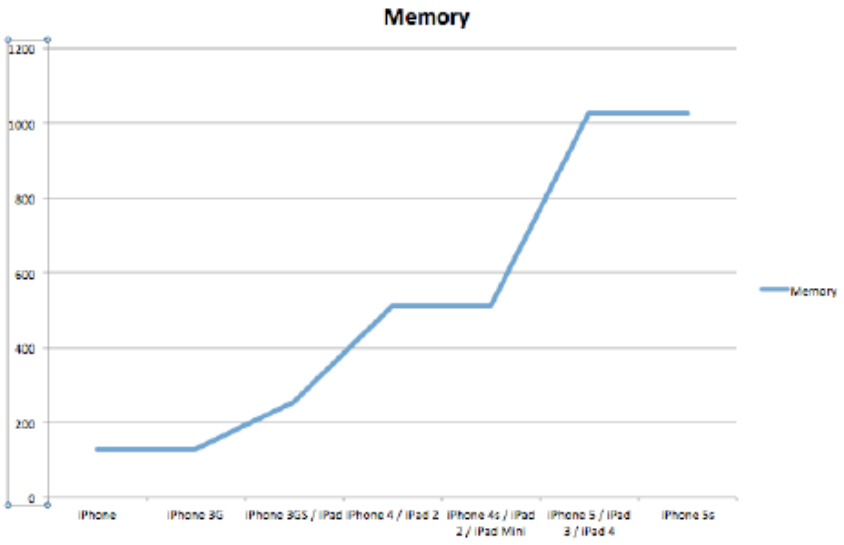


Figure 4. iPhone memory capabilities

Next we need to be aware of how much memory our application is using. Mobile devices are increasing in memory specs every year: at the time of this writing, my wife has a phone with 3 gigabytes of RAM! But mobile device memory specs will never be comparable to laptops of desktop computers, so you must consider memory usage when designing your application. While the iPhone is certainly not the only game in town, Figure 4 shows the memory capabilities of Apple's various i-Devices. The most recent devices have 1GB of memory, but devices less than two years old only have 512MB of memory, and there are plenty of them still around.

"The cost of making network calls is high, and so high in fact that it can drain your battery in a matter of a few hours."

So how can we minimize mobile device memory usage? The key to managing memory is to maintain the user state that you need in memory, but no more. You should not be afraid to use memory when you need it because loading data from storage on each interaction would hurt the perceived user experience, but you should not load a whole lot more than you need. If you have user state that is required to build screens in your application, then by all means, load it into memory, but just be cognizant that memory is a precious resource and you do not need to load everything your application will ever need into memory at once.

Our next topic is power usage. The biggest causes of battery drain are:

– CPU intensive calls and complex video rendering
– Network calls

Depending on the nature of your application, you may not have much control over the complexity of your user interface and algorithms required to build your screens, such as with video games. As CPU power increases it is easier to write less efficient code and still achieve adequate results, but at the cost of CPU usage. If you expend effort to optimize your code and choose the best performing algorithms for your objective, then you will minimize your use of the mobile device's CPU and hence improve battery life. Think about writing mobile applications like you wrote desktop applications a decade ago: you want to optimize every CPU cycle you're using.

The other major cause of power consumption is network usage. I remember a time recently when my phone battery, that typically lasts all day, was draining in three hours. After investigating the problem, I found an email stuck in my outbox and the phone was attempting to send it every couple minutes. After deleting the message, my battery usage returned to normal. What happened? The cost of making network calls is high, and so high in fact that it can drain your battery in a matter of a few hours. So minimize your network calls, consider queuing multiple network calls and sending them together in a single request, and optimize the messages that are passed between your mobile device and your server by using more succinct messages, such as by opting for JSON over XML and gzipping your payload. Additionally, think long and hard about the ads that you show and the frequency with which you show them because ads are a primary cause of network usage and hence battery drain.

Finally, be sure to test the behavior of your application on supported devices with the least specifications to ensure that all of your users have a good experience. If the experience is not satisfactory then you'll need to employ strategies such as the following:

- Optimize algorithms: return to your computer science roots and analyze the order (big Oh) of your algorithms and try running your application inside a code profiler that will identify your bottlenecks.
- Choose resources with lesser resolution (easier to paint) for devices with screens of lesser resolution. For example, the latest Samsung Galaxy S5 has a 1920x1080 resolution, but the two-year-old Galaxy S3 only has a 1280x720 resolution. There is no benefit to sending a device images with greater resolutions than the device itself.
- Gracefully degrade complex functionality: remove some of the nice-to-have features such as anti-aliasing, complex texture mapping, and so forth from your painting algorithms if you identify performance issues. Users with powerful devices will see all of the niceties, but users with less powerful devices will still have a good user experience.

This section presented an overview of several important factors that influence a user's perception of the performance of your application. Now let's review some additional factors you need to consider when building mobile applications.

Native vs. mobile web apps

When building a mobile application, you have a few choices:

- Build a native application for specific devices
- Build a mobile web application that can run across devices
- Build a native applications that display HTML interfaces

And as you might guess, each option has its pros-and-cons. The benefit to building a native application for your target platforms, such as iOS and Android, is that the performance of the application will be much better. The drawback is that you will need to maintain separate code-bases for each platform, and possibly for each version of each platform, and you might need to hire different individuals with different skillsets to build different versions of your application. Furthermore, you need to decide whether or not you are going to support other platforms, such as BlackBerry and Windows Phone, and, if so, are you going to build native applications for them too or find a generic framework for these lesser-used platforms. This all boils down to a longer time-to-market and more complexity in writing and maintaining different versions of the same application on different platforms.

An important input to your decision of whether or not to build a native app is the overall breakdown of mobile operating system market share. According the IDC's analysis of 2013 mobile device shipments, here are the top 5 operating systems and their percentage of market share:

- Android: 78.6%
- iOS: 15.2%
- Windows Phone: 3.3%
- BlackBerry: 1.9%
- Other: 1.0%

"If you want to maximize your reach while supporting many of the latest operating system features, you will need to maintain multiple versions of your app for multiple operating systems."

Furthermore, if you decide to develop a native app, you need to identify the versions of each operating system you are going to support. According to a December 2013 article in Apple Insider, iOS operating system versions are distributed as follows:

- iOS 7: 74%
- iOS 6: 22%
- Older: 4%

Therefore, if your application supports iOS 6 then you will be able to reach 96% of the iOS market. The Android market, however, is more fragmented. According to Google, at the time of this writing, Android operating system versions are distributed as follows:

- 4.4 (KitKat): 8.5%
- 4.3: 8.5%
- 4.2.x: 18.8%
- 4.1.x (Jelly Bean): 33.5%
- 4.0.3 – 4.0.4 (Ice Cream Sandwich): 13.4%
- 3.2 (Honeycomb): 0.1%
- 2.3.3 – 2.3.7 (Gingerbread): 16.2%
- 2.2 (Froyo): 1%

In order to reach 82.7% of the Android market you need to support Ice Cream Sandwich and in order to reach 99% of the Android market you need to support Gingerbread.

In other words, if you want to maximize your reach while supporting many of the latest operating system features, you will need to maintain multiple versions of your app for multiple operating systems. The user experience is better, but the amount of work is substantial!

Building a mobile web application using HTML, CSS, and JavaScript and compiling it with a framework that deploys to multiple platforms has the following benefits:

- Quicker time-to-market because one code base supports multiple platforms (build it once rather than multiple times)
- Reuse existing web development skills in your current organization
- Easier maintenance because bug fixes and feature enhancements only need to performed in one place

But it is not a panacea: one code base that targets multiple platforms does so at the expense of performance. Most of these frameworks provide a virtual machine, so-to-speak, that sits between your application and the underlying operating system. This virtual machine interprets actions executed on your application and translates those to native system calls. As a result this intermediate layer can slow down your application.

Popular frameworks for building native applications that abstract the low-level native code implementation include:

- PhoneGap: PhoneGap, which is sponsored by Adobe, is a free and open source framework that allows you to build your application using HTML, CSS, and JavaScript and deploy natively to iOS, Android, Windows Mobile, BlackBerry, WebOS, and more. At run-time, it launches a native web browser on the mobile device and runs your web pages inside that web browser. It provides a JavaScript "bridge" that allows you to send messages to native components.
- Appcelerator Titanium: Titanium, which is sponsored by Appcelerator, is a platform that allows you to develop mobile apps using JavaScript and deploy natively to iOS and Android. It is a framework for developing native apps that do not run inside of a mobile device web browser, but instead run using native controls. Your JavaScript is interpreted at runtime and the Titanium engine manages the native version of your application with the logic in your JavaScript. It aims to provide you with the best of both worlds: a JavaScript abstraction of your mobile application with access to native functionality. The learning curve for Titanium is higher than PhoneGap, but, depending on what your application is doing, the performance may be better.
- Sencha Touch: Sencha Touch, which is sponsored by Sencha, is a high performance JavaScript framework for building HTML5 mobile applications that can be compiled into native applications using PhoneGapor Sencha's command line tool.

So what is the best solution? The answer depends on several factors:

- How complex is your application? Is it little more than a web application running in a mobile format or is it complex and presenting a unique user experience?
- How important is your application to your business? Is it mostly a marketing tool or are your users performing core business functionality with it?
- How frequently will your users be using your application? Is it something they might refer to once a week or once a month or will then be using it several times a day?

In short, if your application is complex, significant to your business, or frequently used then you want to guarantee the best experience for your users, which probably necessitates a native application. If, on the other hand, your application is simple and not core to your users' daily lives then you can gain valuable time-to-market and ease of maintenance by standardizing on a single code base. You need to measure your return on investment (ROI) and answer the question of whether it is worth it to your business to invest the time to build native applications.

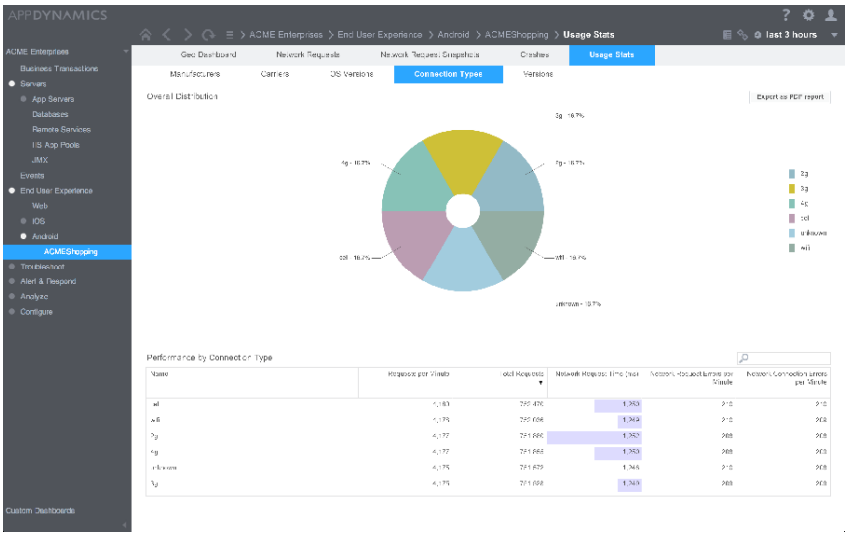
Variety of mobile devices

In order to be successful in building high-performance mobile applications, you need to identify your target platforms. Various devices have various limitations on their capabilities. Specifically, devices are different in terms of:

- CPU/GPU processing capabilities
- RAM
- Screen Size / Form Factor
- Power
- Sensors
- Connectivity

While it may be a considerable amount of work, user-perceived high-performance applications will take each of these factors into consideration and provide an experience that works well in different scenarios. For example, on a powerful CPU/GPU device, your application might opt to overlay beautiful, yet complex, textures on objects to enhance the visualization of a game. But that same application might use simpler textures on a less powerful device. The end result is that both users will have a positive experience, but the visualization will be different. A strategy that I have employed is to interrogate the device's capability as a "calibration" step and then adjust the complexity of my application based on the capabilities of the device. Or stated another way, a user with a lesser performing device does not want to see a beautifully rendered screen that is so choppy it is not useable. This approach is somewhat analogous to a strategy we employ in web development: graceful degradation. While graceful degradation has deeper roots than front-end application behavior, the idea is simple: ensure that your application looks and performs well based on the capabilities of the device. Not all users will have the same experience, but all users will have a good experience.

In addition to the device-specific capabilities, you also need to be cognizant of carrier-specific capabilities upon which the user's device is running. This is not to say that you need to have profiles for different carriers, but your graceful degradation strategy is applicable to network speed. Just as video players support adaptive bitrate streaming, in which the player detects the user's bandwidth and CPU capabilities in real-time and adjusts the quality of the video stream accordingly, if your application loads data from a server, then you can improve user perceived performance by maintaining different quality resources and load the resources most appropriate for both the device as well as the available bandwidth. It is highly recommended that you detect your bandwidth and adjust your network communications accordingly. If your bandwidth is low then be sure to show less ads, report status back to your server less often, and download smaller resources. It is important to note that this is an ongoing process as your application is running because mobile devices are "mobile" and may move in and out of low coverage zones.

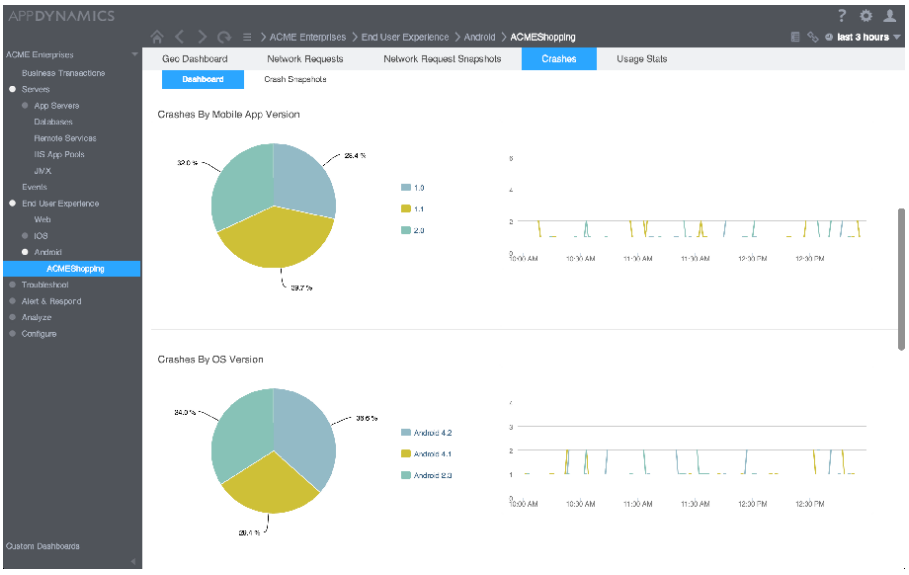


Managing mobile app performance

You've employed the advice in the previous section to develop and test your application on multiple devices and it is rock solid, but how do you avoid receiving a bad review on the App Store because your application is not performing well on certain devices or crashing in certain circumstances? The answer is that there is nothing you can do unless you know what is happening in your application. Therefore the key is to assess the performance of your mobile application, capture error and crash reports, and correlate mobile app performance with server performance. Specifically, you should be capturing the following information from your application at runtime:

- Crash and Error Reports
- Device performance (CPU usage, memory usage, etc.)
- Application response time
- Server response time
- Demographic data (operating system, device type, carrier, etc.)

Whenever an application crashes, a crash or error report is generated and your application has access to it. You need to capture this report and send it to your server with relevant demographic data, such as the device operating system, device type, connection type, carrier, etc. With this information you can learn what was happening during the crash and, when correlating with other crashes, you can determine whether or not there is a device or operating system specific root cause of the issue. Figure 6 shows a sample crash report.



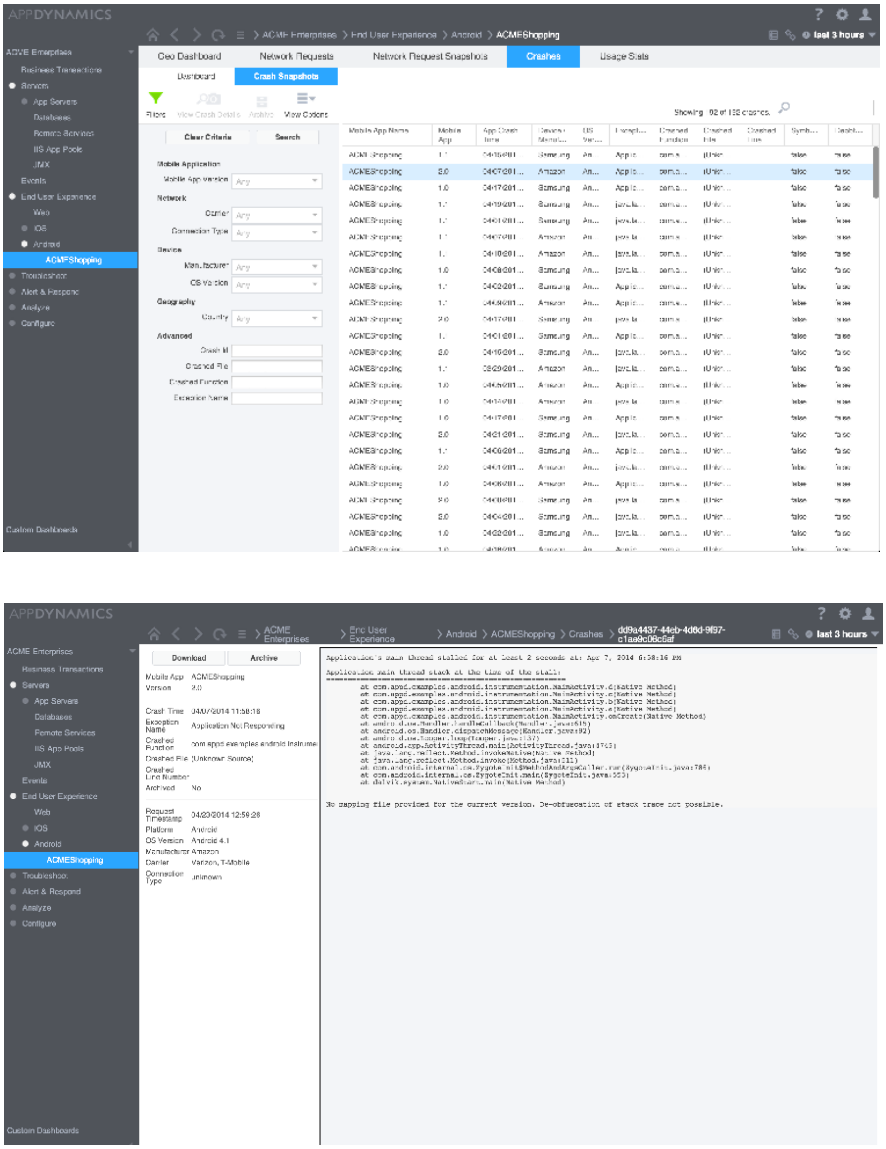


Figure 6. Sample crash report

The crash report in figure 6 can be used to identify exactly where the application crashed and it shows the device type (iPhone 5,1) as well as the operating system version (iOS 6.1.4).

When an application crashes or starts performing poorly, it is important to correlate the behavior of the application with any server calls it is making. Depending on the nature of your application, it may very well be the performance of your server rather than the performance of your application itself. Figure 7 show an example that traces a mobile device server call.

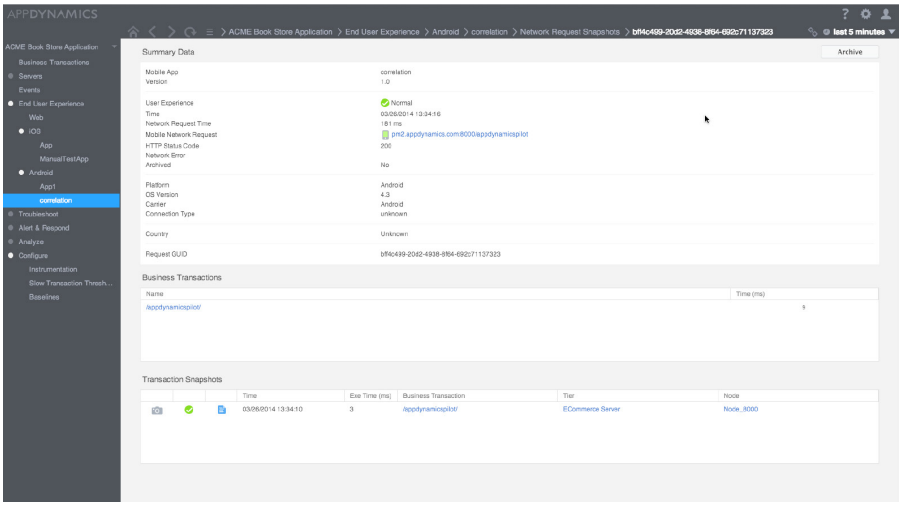


Figure 7. Mobile device–to–Server call

Finally, individual crashes and performance snapshots are helpful, but more importantly, you want to identify any systemic issues that might be occurring in your application. To do this you will need to organize your data with the following demographics:

- Operating System and version
- Device Type
- Network Carrier
- Connection Type (3G, 4G, Wifi, etc.)
- Application Version
- User Geography

For example, figure 9 shows a screenshot that lists the business transactions executed on the server side, correlated with the performances of those business transactions on both Android and iOS, and grouped by geographical location.

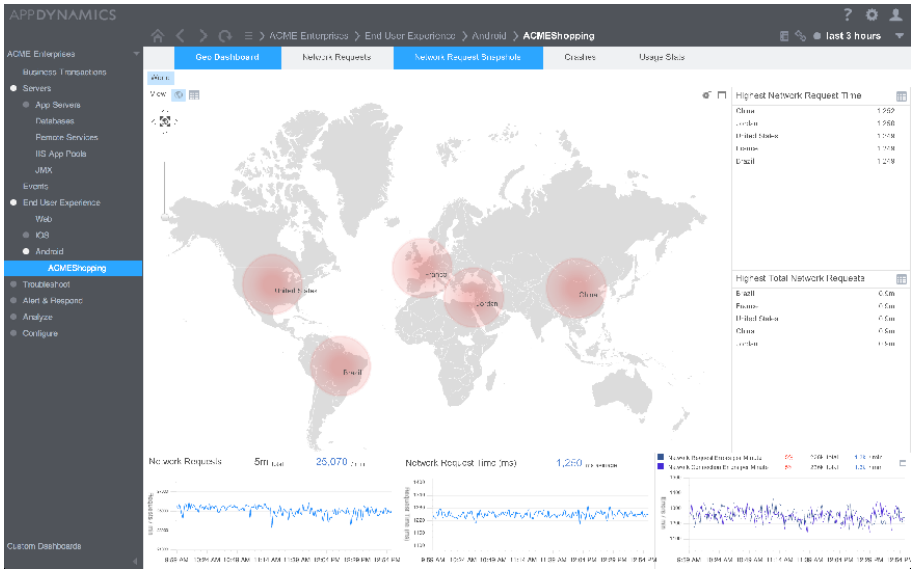


Figure 9. Device response time correlated with server response time and augmented with device type and geographical demographics

With this granularity of information, you will be well equipped to identify systemic issues originating from geographical location, device type, carrier, operating system, and so forth. When you detect a performance anomaly in your environment, first check to see if it is across all of you clients or a subset. If you do determine the problem is not global, but across a subset of devices, locations, etc., then you can use demographic filtering to identify the problem areas. And if there are crash reports then they can help you diagnose the root cause of many problems.

Conclusion

Managing the performance of mobile applications is different from managing the performance of traditional desktop or web-based applications. Mobile devices introduce a new set of constraints, such as lesser powered CPUs and lower memory, that require us to think about application development more like we did a decade ago when resources were more scarce than on the servers we work on today. Furthermore, mobile application performance is determined by user perception moreso than by quantitative measurement. The difference between a good user experience and a postive App Store review and a bad user experience and a negative App Store review might be a couple of seconds.

This paper presented strategies to improve the user perceived performance of your application by reviewing both the lifecycle of a running mobile application as well as understanding how users interact with your application. Simple strategies, such as optimizing your algorithms with a code profiler and gracefully degrading performance with smaller resources and less nicities list anti-aliased drawings when performance is slow, can equate to a good user experience, which will pay dividends in the App Store.

Finally this paper emphasized the importance of measuring and managing the performance of your mobile application. Specifically, it recommended capturing crash and error reports, correlating client interactions and device performance with server calls, and capturing important demographics such as device type, operating system, connection type, carrier, and so forth.

Empowered with good device diagnostics correlated with server performance and augmented by smart demographics you can identify performance issues and hopefully resolve them before you receive a negative App Store review.