Qualcomm Technologies Incorporated

WHEN MOBILE APPS USE TOO MUCH POWER

A Developer Guide for Android App Performance

December 2013

Qualcomm Technologies Incorporated

Qualcomm Technologies Incorporated

5775 Morehouse Drive

San Diego, CA 92121

U.S.A.

Table of Contents

Introduction

One-star ratings.

Plummeting download stats.

Placement on a carrier's list of high-risk Android apps.

As users depend more on their smartphones, they become more protective of battery life and more inclined to give poor reviews on apps that use too much power. Smart developers focus on power consumption as part of the user experience, not as an afterthought. They use tools that help them answer important questions about power consumption and their Android apps:

- Which system components and hardware blocks does your app tie up?
- What kind of tool can you use to profile your app and locate the parts of it that are most power-hungry, so you can debug them?
- How can you see the effect of your programming choices on power consumption and performance?
- Which characteristics are most valuable in a profiling tool?

This paper identifies many common power consumption problems and possible solutions that affect battery life. Readers will gain an understanding of the current landscape of Android app performance profiling tools – including Trepn™ Profiler and Trepn Plug-in for Eclipse from Qualcomm Technologies Inc. – and they will take away important criteria in selecting a tool that meets their needs.

Main Takeaways

- Mobile apps use too much power – and frustrate users – when they needlessly run system resources such as the CPU, GPU, display and wireless (mobile network, Wi-Fi, GPS, Bluetooth) radios.
- Developers can make programming choices to reduce battery drain, but they must first identify power problems and associate them with the portions of their code that cause them.
- Most power profiling tools are designed without taking developer workflow into consideration and do not support the developer's straightforward goal of analyzing battery drain and modifying code to reduce it.
- Trepn Profiler and Trepn Plug-in for Eclipse are developer tools designed to identify and locate power consumption problems in Android apps.

Why is power consumption important to mobile app developers?

On the desktop, power consumption is almost a non-issue: hardware resources are nearly abundant and power is as close as the nearest AC outlet. But on mobile devices, power is precious and power consumption is important for several reasons:

1) Power is part of the user experience.

Like design and ease of use, power deserves to be a big part of the overall user experience. It is short-sighted for a developer to focus on intuitiveness, responsiveness, performance, graphics and all the other traditional priorities if the app consumes too much power, drains the battery and leaves users scrambling for the phone charger. Users will uninstall the app and never be able to appreciate the hard work on design and ease of use.

Power consumption is a back-burner consideration for most developers because it's not a checkbox-feature like rich graphics or social connectivity. It's difficult to incorporate power profiling during app development, and even more difficult to be certain the app is consuming power moderately. But it's too big a part of the user experience to ignore.

2) Poor power management leads to poor ratings.

If an app is unnecessarily power-hungry, then user ratings and reviews soon reflect it. Heavy battery use is a frustration that would drive 55 percent of surveyed U.S. users to write a bad review of an app.[1] Favorable reviews are the life-blood of any mobile app, and developers are keen to avoid and address unfavorable reviews promptly.

It's not only small companies and startups whose apps use too much power and are subject to poor ratings; even some of the largest social networking apps have received poor reviews for default configurations that drain the battery by constantly updating and checking status. As a result of user reaction, their one-star reviews outnumber four-star reviews,[2] and the blogosphere and user forums light up with recommendations to uninstall the app and use the mobile website instead.[3]

When apps use too much data and start affecting battery life, who is on the front line of complaints? Not the developers, not the device manufacturers, but the wireless carriers, and they have no choice but to insulate themselves from customer complaints about data usage and power consumption by conducting their own reviews of the most popular apps.[4,5] Their desire to satisfy customers has even led to listing high-risk apps, some of which drain the battery up to six times faster than normal.[6] This is their way of putting customers on notice that some apps use too much power, and putting developers on notice that they must include power-efficiency among their app development priorities.[7]

[1] *"Users Reveal Top Frustrations That Lead to Bad Mobile App Reviews," Apigee Survey, November 2012.*

[2] *Google Play, retrieved August 2013.*

[3] *"Facebook Apps Draining Your Batteries? Try Uninstalling And Using Its Mobile Site," AllFacebook blog, June 2013.*

[4] *Android™ App Reviews by Verizon, verizonwireless.com, retrieved August 2013.*

[5] *"Using outdated Facebook app causes battery drain: Android," retrieved August 2013*

[6] *High Risk Android™ Apps, verizonwireless.com, retrieved August 2013.*

3) Power is important to users.

But most of all, power consumption is important to app developers because it's important to users. Battery life is of the highest priority to smartphone users – higher than ease of operation, physical design and features – yet it is the least satisfactory element of the mobile user experience.[8]

The transition to mobile computing has heightened user sensitivity to power consumption and placed a premium on battery life.

5 Common Ways Mobile Apps Use Too Much Power

Apps do not drain the battery directly; they drain it indirectly by using the device's hardware components. Among the most common ways in which apps use too much power are through the inefficient use of:

- On-device resources, such as CPU, GPU, memory/bus and display.
- Network resources, such as GPS, Bluetooth, Wi-Fi and especially radios/modems for transmitting and receiving data. Examples include social networking apps that send and receive frequent updates or take too many GPS location fixes.

The role of mobile network states in battery drain

Mobile 3G/4G networks are designed with a transition to an idle state when returning from active to inactive states. Developers who understand this transition, the reasons behind it and the amount of

[7] *"Which apps will drain your battery and data plan? Verizon's got a list," Kevin Fitchard, Gigaom, Feb 2013.*

[8] *"J.D. Power: Consumers most dissatisfied with smartphone battery life," Jeff Saignor, Digital Trends, March 2012.*

power consumed in each state can optimize their apps accordingly.

Figure 1 depicts a typical progression through network states. (3G and 4G vary slightly, but the figure is representative of either network model.) The y-axis shows the approximate current consumed for various network states over time on the x-axis.

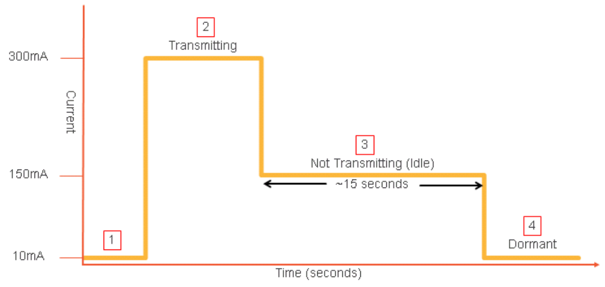


Figure 1 - Mobile network states

1. The radio starts dormant, consuming less than 10mA. It consumes very little power but can quickly connect to receive and make phone calls and to transmit/receive data.
2. When the device transmits/receives data, the radio connects and enters the active state, consuming approximately 300mA.
3. After transmitting/receiving, the radio remains connected but drops to idle, still consuming about 150mA.
4. Following a timeout interval of 8 to 15 seconds (depending on the wireless carrier), the radio returns to the dormant state.

The area under the curve represents the total power consumed by the radio for a single connection to the mobile network, including the 8 to 15 seconds in the idle state. This points to two important considerations for mobile app developers:

- Over a series of very brief transmit/receive operations, the idle state can drain the battery more than the active state.
- It is battery-expensive to bring up the network, so apps should take full advantage of every connection, once established.

Thus, a certain, irreducible margin of inefficiency is inherent to mobile network states (see sidebar), and developers must accept that margin as a given. Still, they can look for and avoid five common programming issues that lead to battery drain.

1. Hanging sockets

An app does not automatically close its connection to the network after transmitting/receiving data; it must issue an instruction to close it. Otherwise, after a given interval without network activity, during which the radio has usually returned to the dormant state, the server will time out and close the socket by sending a FIN packet (see Figure 2). This wakes the radio from dormant all the way up to the active state, merely to execute the TCP socket close handshake with the server. The radio follows the normal curve through 8 to 15 seconds in the idle state then returns to the dormant state.

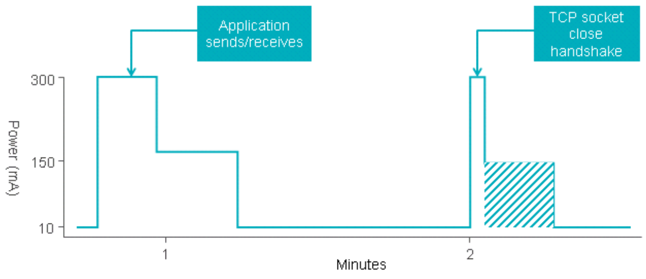


Figure 2 - Hanging socket

Thus, the phone has to bring up the radio for a simple, easily avoidable handshake because the server has to ask the device for something that the app should have provided. If no other traffic moves between the device and the network, the connection is a complete waste of several hundred mA.

Assuming that the app uses the network four times in an hour, the simple fix of having the app close the socket when finished can reduce network power consumption by about 20 percent, which would be the difference between eight and ten hours of standby power.

Tip #1:

Program the app to close sockets when done. Otherwise, the phone consumes power to bring up the radio for a simple, easily avoidable handshake with the server.

2. Ungrouped network activity

Every connection to the network consumes power, so spacing connections out unnecessarily is wasteful.

Consider an app that needs to connect 3 times, with one-minute intervals between the start of each connection. The network state transitions might resemble this:

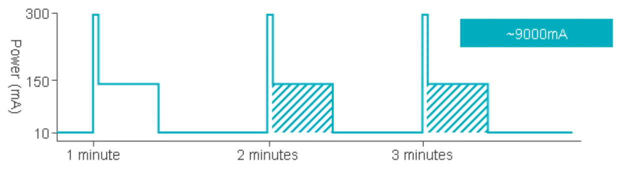


Figure 3 - Ungrouped network activity

The shaded area under each curve depicts power-inefficiency, when the radio is in the idle state. The device consumes a total of approximately 9000mA.

Contrast that with this curve, in which the app groups its transactions and brings up the network a single time:

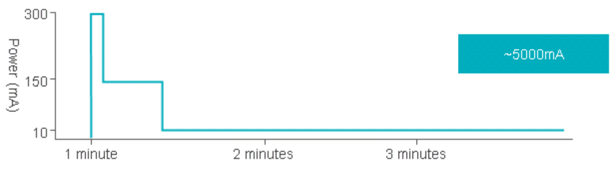


Figure 4 - Grouped network activity

Even though the radio is active for almost three times as long, the same app optimized to group network activity together and allow the radio to remain dormant could eliminate the two unnecessary idle states and consume only about 5000mA, or about 45 percent less power.

Tip #2:

Reorganize code to group the app's network connections together. This may require deferring some transactions and moving others up.

3. Unreleased Wakelocks

The Android API Reference describes a wakelock as "a mechanism to indicate that your application needs to have the device stay on." However, it also notes, "Device battery life will be significantly affected by the use of this API."[9]

An app may acquire wakelocks to temporarily keep the display illuminated or the CPU running. Wakelocks also allow an app to stay awake even when the power button is pressed. But if the app does not release them, wakelocks can prevent the device from going to sleep, quickly draining the battery.

In a typical scenario of unreleased wakelocks, the app switches to running in the background or closes altogether, yet resources like the display and CPU continue to run – and drain the battery – as if the app were still active.

To keep the screen illuminated, power-efficient apps use the FLAG_KEEP_SCREEN_ON constant of `WindowManager.LayoutParams` instead of using wakelocks. This keeps the screen on when the app is in the foreground without the need for additional code to release wakelocks when the app goes into the background. As for using wakelocks to handle other, temporary background tasks, it is best to heed the advice of the Android API Reference: "Do not acquire `PowerManager.WakeLocks` unless you really need them, use the minimum levels possible, and be sure to release them as soon as possible."

Tip #3:

For most background tasks, ensure the app releases wakelocks as soon as they are no longer needed. To keep the screen illuminated, use the FLAG_KEEP_SCREEN_ON constant instead of a wakelock.

4. Keep-alives

Some apps require a persistent socket connection to the mobile network; e.g., for incoming voice calls and instant messages. However, network nodes are configured to time out and tear down connections like these when they remain inactive for too long.

---

[9] PowerManager, Android API Reference, retrieved August 2013.

Thus, apps that send keep-alive messages do so at the cost of starting up radios again and using more power.

Like the ungrouped network activity described above, frequent keep-alive messages can drain the battery, simply to deal with a non-essential connection. They are often characterized by a radio transition to the transmitting state without any accompanying meaningful usage.

Operations such as status updates can tolerate delays, but processes like real-time communication cannot. In both cases, a reasonable Android alternative to keep-alives is Google Cloud Messaging (GCM), which accepts messages from third-party app servers, queues them and sends them to corresponding apps when the device is online. Instead of the app on the client asking, "Is there new data?" and being told "no" most of the time, GCM holds the "yes" and can even wake the app with Intent broadcast when it is time to send it.

Tip #4:

Use Google Cloud Messaging instead of keep-alive messages between the app and the network.

5. Unnecessary GPS fixes

Besides pure-play location-based apps, many apps use location simply because they can. A social networking or fitness app may supplement its appeal and functionality with location, which makes sense when users are posting and sending data, but does not make sense when they are merely reading.

Unnecessary GPS fixes are akin to unreleased wakelocks. Since each GPS fix starts the radio, developers can reduce power consumption by thinking carefully about the role of location in their app. They can eliminate non-essential fixes by balancing between complete location-awareness and energy efficiency.

Tip #5:

Use GPS only when necessary for the user experience. Allow users to configure between greater location-awareness and reduced battery drain.

Android App Profiling Tools

To help developers identify how much of a given hardware resource an app uses, a category of Android app profiling tools has evolved.

Criteria for useful profiling tools

For developers to adopt and benefit from an app profiling tool, several criteria stand out:

- Real-time view – Developers should be able to simultaneously use an app and profile it on the mobile device. To be this close to the app, the profiler must run on the device.
- Overlay mode – During profiling, developers should be able to see exactly how and when user interaction and on-screen events affect the current status of hardware resources.
- Data retention – The tool should store the data gathered during profiling and make it available for subsequent analysis and presentation.
- Graphical and statistical format – Profiling data should be visible in both numeric and graphical format.
- Integration with development environment – Since most developers work in an IDE running on a PC, they should be able to control and analyze profiling from there, without attaching other instruments to the device to be tested.
- Code correlation – The tool should afford developers a way to connect the dots between system events captured by the profiler and areas to examine in code.
- Minimal observer effect – To avoid skewing the results of profiling, the tool should be as unobtrusive as possible, occupying the minimal practical footprint on the device to be tested.

What should app profiling tools measure?

Because battery drain can originate in a variety of hardware resources, ideal tools collect and present the greatest practical amount of system usage data, including:

- **Power at battery** – Battery usage over time, with relative battery drain of system components.
- **CPU** – CPU frequency, state by core, wakelock state and details.
- **Network** – Mobile and Wi-Fi network states, Wifilocks, and network usage per app.
- **Display and GPU** – State and brightness, and GPU usage and frequency.
- **GPS and Bluetooth** – State.

It is possible for profiling tools to collect many more data points from the device. But ideal tools focus the developer's attention on the metrics most likely to be of use in increasing the power-efficiency of the app.

Current landscape of profiling tools

As summarized in the following table, developers have several options for profiling power consumption in their Android apps (in descending order of accessibility and ease of use):

Battery Usage, an indicator included among Android's Settings, shows how components and apps consume power on the device. Since users refer to this in assessing whether an app drains the battery, it is important for the app to present favorably here. However, the tool is limited to high-level, user-oriented estimates, without details that developers can use in optimizing their apps.

Wakelock Detector from UzumApps presents system usage statistics around a single problem very clearly. But this and other, similar utilities do not provide a full overview of system resource usage.

PowerTutor collects and graphs by app the power states of a variety of hardware components. However, the tool is designed to give accurate power consumption figures on only a few devices.

Intel® Power Monitoring Tool for Android Devices presents real-time performance data related to power consumption at the component level and exports system usage metrics to a file. But developers must parse the file and create their own graphs to identify and find problems.

Application Resource Optimization (ARO) is a PC-based utility from AT&T that charts network traffic at each layer in the protocol stack. However, the tool's diagnostics take developers beyond the realm of app development onto a potentially steep learning curve of mobile network modeling.

Hardware power measurement tools present a very accurate picture of power consumption right at the device battery. Still, the equipment is generally expensive, it needs to be attached to the device by leads, and developers cannot easily correlate test results to problems in code.

In short, most existing tools have been designed without considering developer workflow. Also, most existing tools simply present system usage statistics without providing enough data or correlating it in such a way that developers can see how to fix problems in their code and improve the app.

Introducing Trepn™ Profiler and Trepn Plug-in for Eclipse

Trepn Profiler and Trepn Plug-in for Eclipse are specially designed, easy-to-use Android developer tools from Qualcomm Innovation Center Inc. (QuIC) that are designed to demonstrate where apps use power inefficiently so that developers can easily find and modify code that causes battery drain and ruins the user experience.

Trepn Profiler

Trepn Profiler is a standalone tool that runs on Android devices. It is designed to profile in real time how a mobile app uses CPU, network and hardware resources, and chart the data on the device in both instant gauges (see Figure 5) and historical graphs (see Figure 6). Developers can also save and export profiling data for later analysis.

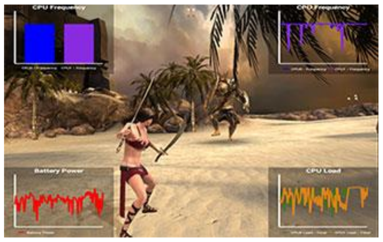


Figure 5 - Trepn Profiler, overlay mode

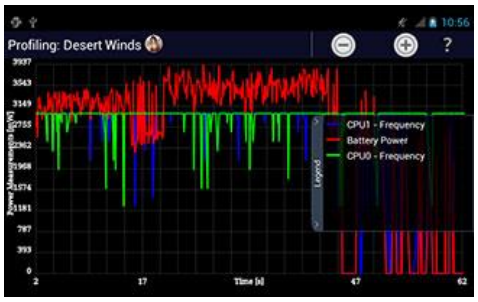
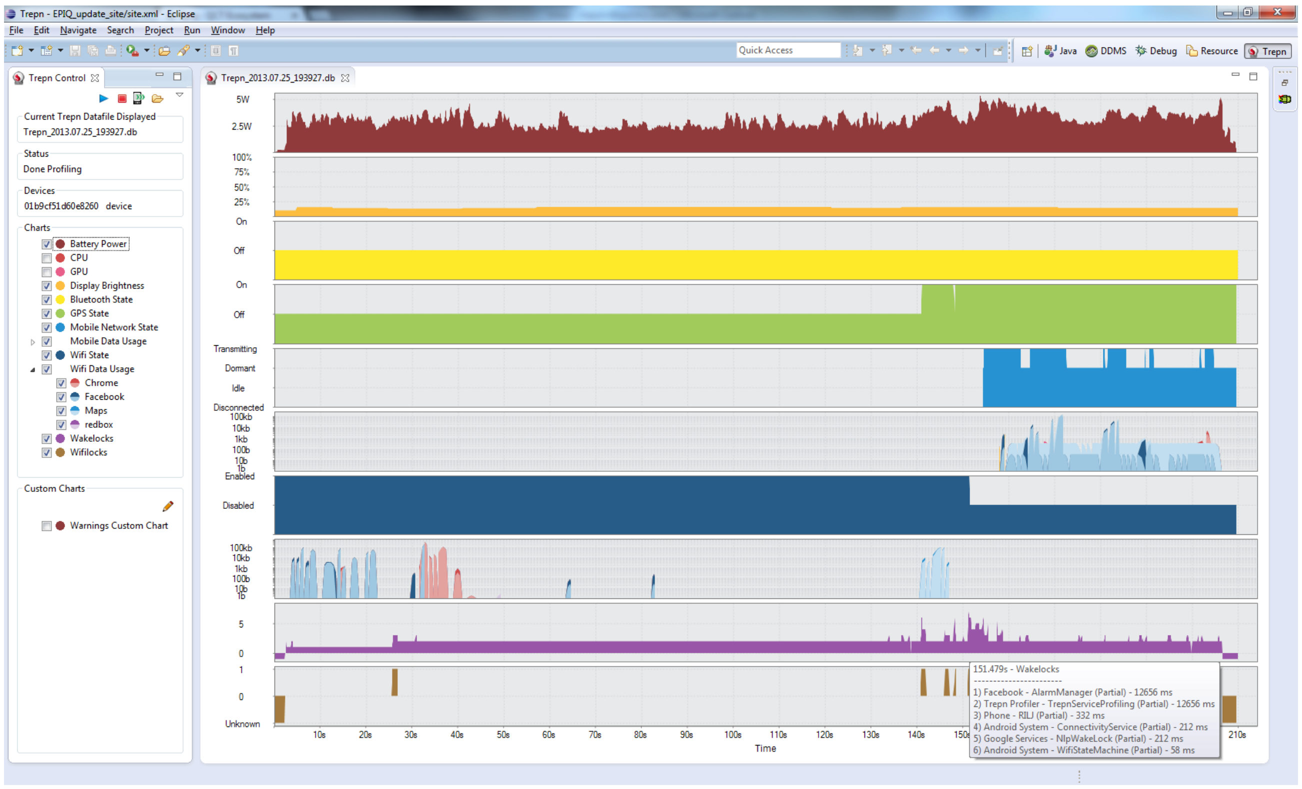


Figure 6 - Trepn Profiler, graph display

Trepn Plug-in for Eclipse

Trepn Plug-in for Eclipse goes one big step further by controlling Trepn Profiler from inside the development environment and displaying data from the Android device on the PC. The plug-in charts the profiling data collected on the device for analysis in an Eclipse perspective (see Figure 7), where developers can easily modify their code to resolve power consumption problems.


Trepn - EPIQ_update_site/site.xml - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Java DDMS Debug Resource Trepn
Trepn Control
Trepn_2013.07.25_193927.db
5W
2.5W
100%
75%
50%
25%
On
Off
Transmitting
Dormant
Idle
Disconnected
Enabled
Disabled
Unknown
Time

Current Trepn Datafile Displayed: Trepn_2013.07.25_193927.db

Status: Done Profiling

Devices: 01b9cf51d60e8260 device

Charts:
- ✓ Battery Power
- CPU
- GPU
- ✓ Display Brightness
- ✓ Bluetooth State
- ✓ GPS State
- ✓ Mobile Network State
- ✓ Mobile Data Usage
- ✓ Wifi State
- ✓ Wifi Data Usage
  - ✓ Chrome
  - ✓ Facebook
  - ✓ Maps
  - ✓ redbox
- ✓ Wakelocks
- ✓ Wifilocks

Custom Charts:
- Warnings Custom Chart

```
151.479s - Wakelocks
------------------------
1) Facebook - AlarmManager (Partial) - 12656 ms
2) Trepn Profiler - TrepnServiceProfiling (Partial) - 12656 ms
3) Phone - RILJ (Partial) - 332 ms
4) Android System - ConnectivityService (Partial) - 212 ms
5) Google Services - NlpWakeLock (Partial) - 212 ms
6) Android System - WifiStateMachine (Partial) - 58 ms
```

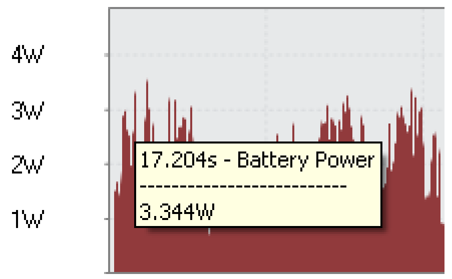
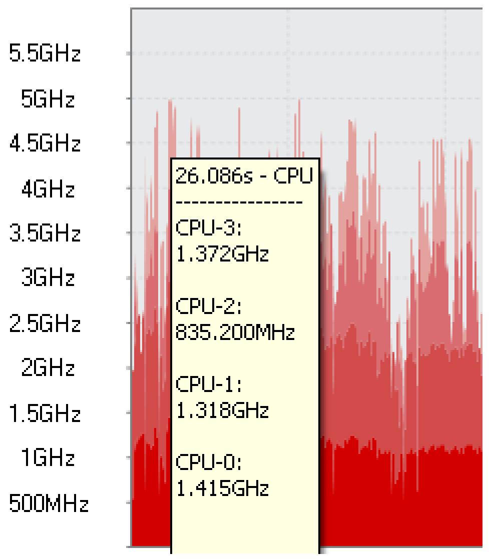
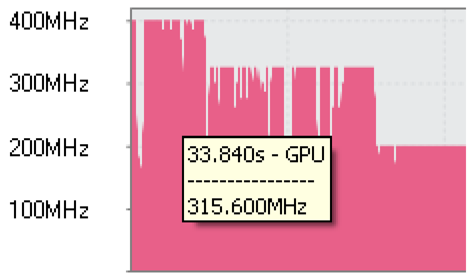
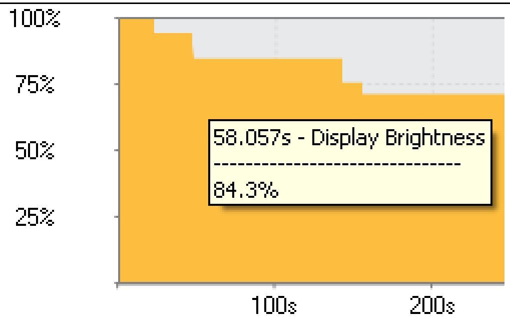
Figure 7 - Trepn Plug-in for Eclipse charts

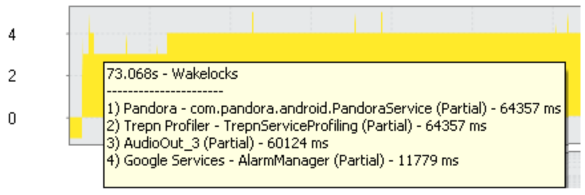
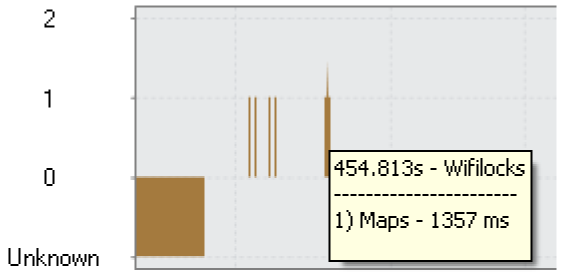
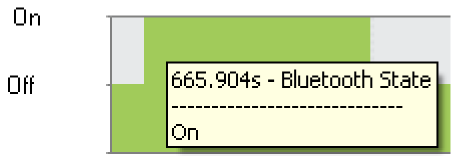
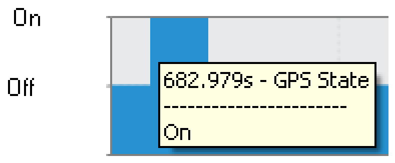
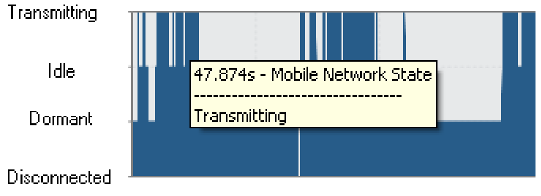
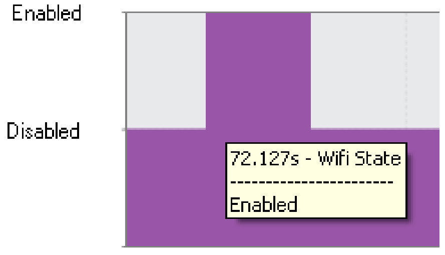
How the tools work

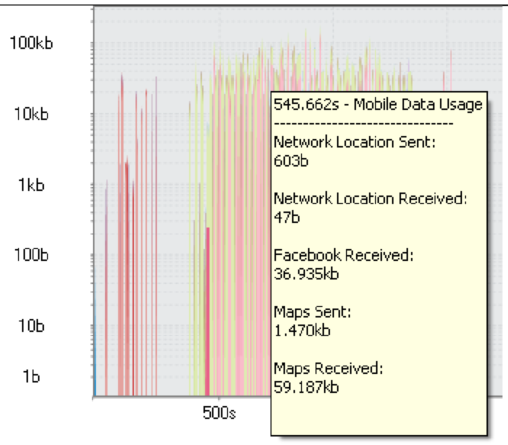
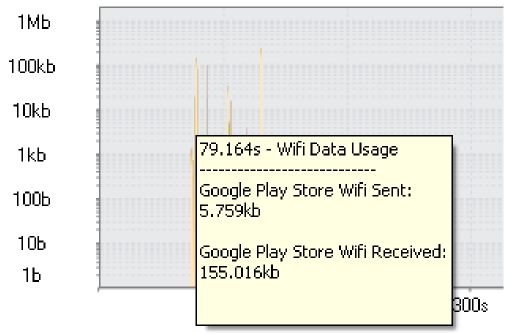
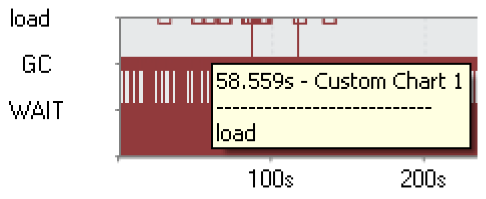
The developer installs Trepn Plug-in for Eclipse as new software inside Eclipse. The JAR file contains the plug-in, Trepn Profiler and all associated files required on both the development computer and the mobile device. Once Trepn Plug-in for Eclipse is installed, it pushes and installs Trepn Profiler to any connected Android device when the developer starts profiling.

On the computer, the developer launches the plug-in as an Eclipse perspective, starts/stops profiling, and analyzes charts based on the data that Trepn Profiler collects. On the device, the developer can also run Trepn Profiler as a standalone app. (For more details on installation, visit the Trepn Plug-in for Eclipse page on the Qualcomm Developer Network site.)

Trepn Plug-in for Eclipse charts data on the hardware and network resources most likely to cause battery drain. As shown in the following table, developers can see graphically the impact of system events and user interaction on these resources, and begin to address them in code.

Battery Power presents a general idea of the amount of power consumed by the entire phone. Trepn charts output at the battery in watts at 100ms intervals. (Profiling battery power requires a device powered by a Snapdragon™ mobile processor. See the FAQ for more details.)	
CPU usage displays the load on each and all CPUs. Developers can see when a system event results in a spike or drop in cycles.	
GPU usage demonstrates whether, when and to what extent the system is offloading work to the GPU.	
Display Brightness charts the device's single greatest battery drain, showing when the display is illuminated and to what degree.	

As described above, **Wakelocks** and **Wifilocks** prevent the device from sleeping, resulting in rapid, unintended battery drain. These charts identify the locks and demonstrate when they are acquired/released so that developers can analyze their impact on power consumption.	 73.068s - Wakelocks -------------------- 1) Pandora - com.pandora.android.PandoraService (Partial) - 64357 ms 2) Trepn Profiler - TrepnServiceProfiling (Partial) - 64357 ms 3) AudioOut_3 (Partial) - 60124 ms 4) Google Services - AlarmManager (Partial) - 11779 ms  454.813s - Wifilocks ------------------------ 1) Maps - 1357 ms
Bluetooth State and **GPS State** show the points in time at which Bluetooth and GPS are on (consuming power) and off.	 665.904s - Bluetooth State ------------------------------ On  682.979s - GPS State ------------------------ On
Mobile Network State charts the points at which the device moves among four states: transmitting, idle, dormant and disconnected. **WiFi State** charts the transitions between disabled and enabled status. These help developers to locate and identify common network problems.	 47.874s - Mobile Network State -------------------------------- Transmitting  72.127s - Wifi State ------------------------ Enabled

Beyond charting the state of the radios, Trepn Plug-in for Eclipse charts **Mobile Data Usage** and **WiFi Data Usage** by app. This shows developers exactly which apps are transmitting and receiving data at any given time and helps them correlate the traffic to specific events during profiling.	 545.662s - Mobile Data Usage ------------------------------ Network Location Sent: 603b Network Location Received: 47b Facebook Received: 36.935kb Maps Sent: 1.470kb Maps Received: 59.187kb  79.164s - Wifi Data Usage --------------------------- Google Play Store Wifi Sent: 5.759kb Google Play Store Wifi Received: 155.016kb
Developers can create their own **Custom Charts.** Trepn Plug-in for Eclipse includes a regular expression utility for matching strings captured in the system logs (LogCat) during profiling and charting them along the same timeline as all other system resources. This example shows a chart of the occurrences of “load,” “GC” and “WAIT” among the messages in the system logs.	 58.559s - Custom Chart 1 --------------------------- load

Typical use cases

- A developer discovers that a game drains the battery when in standby mode. Using Trepn Plug-in for Eclipse, the developer sees that the CPU never sleeps after the game is installed because the app doesn’t release wakelocks when the application stops. Once the problem is pinpointed, the developer can iterate and profile the game with modified code to alleviate the power consumption issue.
- Testers report that an app is draining battery power when the device is in standby mode. Under Mobile Network State, Trepn Plug-in for Eclipse reveals an additional, unexpected 3G network connection two minutes after the app synchs. Aware of the impact that new connections have on power consumption, the developer modifies the app to group mobile network activity and close all connections cleanly, then profiles the app again to measure the improvement.
- Charts in Trepn Plug-in for Eclipse show several intervals of excessive CPU usage and power consumption, while the GPU remains inactive. Seeing an opportunity to offload tasks to dedicated hardware, the developer introduces trace statements around several routines, profiles the app again and creates custom

charts that correlate the routines to CPU and GPU performance.

Meeting the needs and criteria of mobile app developers

Given the importance of reducing unnecessary battery drain before apps are released, Trepn Plug-in for Eclipse and Trepn Profiler meet mobile app developers' needs and criteria in ways that would otherwise require several tools used together.

- They put the control and results of the profiling process right in the Eclipse development environment.
- At no cost, they allow developers to analyze, modify and profile power consumption in their apps quickly and with a short learning curve.
- They offer easy-to-read charts, graphs and gauges that display both real-time and historical data on the system resources most affected by battery drain.
- In spite of their light footprint, they provide developers with exceptional tools for identifying and addressing power consumption problems in mobile apps.

How to get Trepn

Trepn Plug-in for Eclipse is available under Install New Software… inside Eclipse. The installation package includes Trepn Profiler and all associated files required on both the development computer and the mobile device.

Users without Eclipse can install Trepn Profiler by following the instructions on the Trepn Profiler page of the Qualcomm Developer Network site.

Conclusion

Trepn Plug-in for Eclipse and Trepn Profiler from Qualcomm Technologies Inc. are developer tools for identifying and locating power consumption problems in Android apps before users discover the problems and give the apps poor ratings. The tools are integrated directly into the Eclipse development environment, work on all commercial Android devices, present data in easy-to-read formats and enable developers to associate system events with the code that causes them.

By showing how and where apps drain the battery, Trepn Plug-in for Eclipse and Trepn Profiler provide developers with valuable tools for reducing power consumption in their apps, and with the opportunity to release more power-efficient apps to their users.

Follow Us

Since 2000, Qualcomm has grown a developer community around its mobile processors and innovations. The Qualcomm Developer Network (QDN) centralizes access to the tools and resources needed for building on Qualcomm Snapdragon processors, whether for writing applications, extending APIs or working with our technologies.

Visit QDN for developer tools, announcements and support. Find us on YouTube, Facebook, Twitter and other points of contact on the Web.

Additional Resources

"Power Optimization for Android Developers," Little Eye Labs, September 2012.

"Transferring Data Without Draining the Battery" and "Optimizing Battery Life," Android Developer Training.

"Smarter Apps for Smarter Phones," GSMA, February 2012.