The following is the first few sections of a chapter from The Busy Coder's Guide to Android Development (/Android), plus headings for the remaining major sections, to give you an idea about the content of the chapter.

Sources of Power Drain

If you can measure power drain well yourself, that is the best way for you to determine precisely where your power consumption is going. Alas, for various reasons, you may not be able to get good power consumption data.

Which means you may have to guess.

We know the general sorts of things that consume power in a device, such as the screen and the CPU. We know that if we use these things less, we will use less power. Eventually, though, we have an app that does nothing, and while this may result in optimal power usage, we are still likely to get poor reviews, because *the app does nothing*.

What we need is some rough idea of how bad certain things are, so we can weigh our use of those system components appropriately.

This chapter will try to give you some “rule of thumb” heuristics of how to estimate power usage of various system components, plus some general recommendations of how to use less of that particular component without necessarily eliminating useful functionality from your app.

Prerequisites

Understanding this chapter requires that you have read the core chapters and understand how Android apps are set up and operate.

Also note that:

- mA = milliamps, where the ampere (or “amp”) is the SI unit of current
- mAH = milliamp-hours, which is how battery capacities are measured (e.g., 2000mAH can power a 200mA draw for 10 hours)

Screen

Screen size and battery size generally trend together. Tablets have bigger batteries and bigger screens than do phones, which in turn are bigger in both areas than are wearables.

A rough rule of thumb is to expect to consume ~10% of the device’s battery for every hour you keep the screen on. Or, to look at it another way, on a phone-sized screen, expect a power draw of ~100-200mA, depending on variations in screen size and display technology (e.g., AMOLED).

Normally, the user is in control over how long your app is in the foreground and therefore is “to blame” for the screen being on. There are a couple of cases where you can make the screen be more of a problem.

The first is if you `acquire()` a `WakeLock` (other than a `PARTIAL_WAKE_LOCK`)… and forget to ever `release()` it. Since the `WakeLock` will keep the screen on, the screen will stay on, even if your app is in the background, until such time as your process is terminated or the device shuts down due to low battery.

In fact, such `WakeLock` types have been deprecated, with the last of them being flagged as deprecated in API Level 17. The recommended alternative is to use `android:keepScreenOn` or `setKeepScreenOn()` on some `View`. This will keep the screen on, so long as the activity hosting that `View` is in the foreground. That way, just moving to the background releases the underlying `WakeLock`, allowing the device to return to sleep.

However, in some cases, even that may be insufficient. Suppose that the user is in your activity, and they get distracted, putting down their device for an extended period. Unless you somehow detect the inactivity, and manually turn off the keep-screen-on mode, the screen will stay on indefinitely, until the power is drained. Hence, if you have a decent way of determining if the user is still using your activity, consider using that as a way to determine when the device is inactive (e.g., a `postDelayed()` that gets canceled and rescheduled when the user does something, so if the `postDelayed()` `Runnable` gets invoked, you know the user has done nothing for the delay period). Then, if you know the device is inactive, call `setKeepScreenOn(false)` to return the screen to its normal operating mode.

The academic paper “How is Energy Consumed in Smartphone Display Applications?” (http://www.hotmobile.org/2013/papers/full/17.pdf) has a more extended analysis of screen power draw.

Disk I/O

The preview of this section was abducted by space aliens.

WiFi and Mobile Data

The preview of this section was accidentally identified as an Android 'tasty treat' by the Cookie Monster.

GPS

The preview of this section will not appear here for a while, due to a time machine mishap.

Camera

The preview of this section is en route to Mars.

Additional Sources

The preview of this section apparently resembled a Pokémon.