ECODroid: Uma Ferramenta para Análise e Visualização de Consumo de Energia em Aplicativos Android

Francisco Helano S. de Magalhães, Lincoln S. Rocha, Danielo G. Gomes

[1]Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)
Universidade Federal do Ceará (UFC)
Av. Mister Hull, s/n – Campus do Pici – Bloco 942-A
CEP: 60455-760 – Fortaleza-CE – Brasil

{helanomagalhaes,lincoln,dgomes}@great.ufc.br

Resumo. *Nos últimos anos, os dispositivos móveis evoluíram de plataformas fechadas, contendo apenas aplicações pré-instaladas, para plataformas abertas capazes de executar aplicativos desenvolvidos por terceiros. Entretanto, essa evolução trouxe consigo diversos problemas, tais como o consumo anormal de energia. Esse problema muitas vezes é causado por aplicativos mal projetados para lidar com questões de consumo de energia. Portanto, para melhorar a qualidade dos aplicativos com relação a eficiência energética, os desenvolvedores necessitam de ferramentas adequadas. Nesse contexto, este artigo apresenta o ECODroid, uma ferramenta para análise e visualização do consumo de energia em aplicativos Android que ajuda na identificação de trechos de código que possuem problemas relacionados ao consumo anormal de energia. Uma avaliação inicial do ECODroid foi realizada com o intuito de analisar a sua viabilidade.*

1. Introdução

Os dispositivos móveis (e.g., *smartphones* e *tablets*) tornaram-se parte integrante da nossa vida cotidiana. A razão para sua crescente popularidade é a possibilidade de ter um grande número de funcionalidades em um único sistema embarcado. Por meio de aplicativos móveis, esses dispositivos podem acessar a Internet, capturar fotos, capturar e reproduzir áudio e vídeo, etc. Porém, muitos desses aplicativos não levam em conta o consumo de energia quando são projetados [Pathak et al. 2012].

Estudos recentes relatam que vários aplicativos *Android* não são energeticamente eficientes devido a dois motivos principais [Liu et al. 2014]. O primeiro está relacionado ao fato do *Android* expor para os desenvolvedores APIs que possibilitam realizar operações de *hardware* (e.g., API para controlar o brilho da tela e acionar interfaces de comunicação sem fio). Embora essas APIs tragam flexibilidade para os desenvolvedores, estes devem utilizá-las com cautela, uma vez que o mau uso do *hardware* pode facilmente acarretar um alto consumo de energia. O segundo motivo está relacionado ao fato dos aplicativos *Android* serem desenvolvidos por equipes pequenas nas quais não existe um esforço dedicado às atividades de garantia da qualidade. Esses desenvolvedores raramente realizam esforços no sentido de melhorar a eficiência energética dos aplicativos.

Segundo [Liu et al. 2014], a localização de problemas de consumo anormal de energia em aplicativos *Android* é uma tarefa difícil, necessitando auxílio ferramental adequado. Nesse contexto, esse trabalho apresenta o ECODroid (*Energy Consumption Optimizer for Android*), uma ferramenta para análise e visualização do consumo de energia

em aplicativos Android. O ECODroid foi implementado como um *plugin* da IDE *Android Studio*[1] e, a partir da utilização de um modelo analítico de consumo energético, identifica as áreas do código-fonte do aplicativo que apresentam níveis anormais de consumo de energia. Uma avaliação inicial do ECODroid foi realizada, apresentando indícios de viabilidade como ferramenta de auxílio à localização de problemas de consumo de energia.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta conceitos relacionados à medição do consumo de energia. A Seção 3 apresenta o ECODroid e a Seção 4 descreve uma avaliação inicial da ferramenta. Já a Seção 5 é dedicada aos trabalhos relacionados. Por fim, a Seção 6 apresenta as considerações finais do artigo.

2. Medição do Consumo de Energia

A medição do consumo de energia de um dispositivo móvel pode ser feita de duas maneiras: *online* e *offline*. A medição *online* é realizada por meio de um dispositivo de medição externo, utilizando dispositivos de referência para os testes. Neste tipo de medição, é recomendado fazer uso de uma fonte de corrente contínua ao invés da bateria convencional. Essa abordagem ajuda a minimizar a interferência das propriedades da bateria nos valores medidos. Já a medição *offline*, fornece estimativas auferidas por meio de um software que é executado no próprio dispositivo móvel ou por meio da utilização de valores extraídos do diagnóstico de *hardware* feito pelo fabricante do dispositivo. Portanto, as medições *online* geralmente são feitas utilizando os resultados das medições *offline* como referência.

Em cenários reais, os componentes de *hardware* do dispositivo móvel não podem operar totalmente isolados, pois o resultado de cada medição é a soma do consumo de todos os componentes que estão ativos. A abordagem recomendada pelo fornecedor da plataforma *Android*[2] é subtrair da energia total consumida o valor do consumo de *standby* dos demais componentes que não estão sendo avaliados naquele momento. No entanto, há alguns componentes do dispositivo que estão sempre operantes e não podem ser simplesmente desligados, como por exemplo a CPU (*Central Processing Unit*). Obter o consumo de energia desses componentes pode ser possível por meio do cálculo algébrico de um sistema de equações lineares, consistindo na soma dos consumos de energia de um número específico de componentes e a energia total consumida pelo dispositivo em um determinado momento em um cenário de uso específico. Entretanto, esses cálculos não são necessariamente precisos e os dados obtidos devem ser validados, pois não se pode garantir que a CPU não esteja sendo usada por outros aplicativos ou por serviços que executam em *background* [Tarkoma et al. 2014].

O modelo de consumo de energia adotado neste trabalho é mesmo utilizado em [Couto et al. 2014]. Esse modelo assume que diferentes componentes de *hardware* causam diferentes impactos no consumo de energia em um dispositivo móvel e, por esse motivo, leva em consideração características particulares de cada um desses componentes. O modelo de energia utilizado no neste trabalho foi inspirado no *PowerTutor*[3] e considera seis componentes de hardware diferentes: CPU, LCD, GPS, *Wi-Fi*, 3G e Áudio.

O consumo de energia da CPU é fortemente influenciado pela sua utilização e

[1] http://developer.android.com/tools/studio/

[2] https://developer.android.com/training/monitoring-device-state/

[3] http://ziyang.eecs.umich.edu/projects/powertutor/

frequência. O processador pode trabalhar em diferentes frequências de acordo com sua necessidade. Dependendo da atividade processada, a porcentagem de utilização pode variar entre 0 e 100. O consumo de energia deste componente em um momento específico é calculado multiplicando o coeficiente associado com a frequência em uso e a porcentagem de sua utilização. Já o modelo de energia do *display* LCD considera apenas o brilho como variável. O valor do consumo é calculado baseado na porcentagem do brilho em relação aos valores de *screen.on* e *screen.full* obtidos por meio do *profile*. O consumo do GPS depende exclusivamente do seu estado atual: *gps.active*, *gps.sleep* ou *gps.off*. A interface *Wi-Fi* possui três estados de consumo no *profile* disponibilizado pelo fabricante: (i) quando o *Wi-Fi* está ligado, mas não está recebendo ou transmitindo dados (*wifi.on*); (ii) quando está transmitindo dados (*wifi.active*); e (iii) quando está procurando por pontos de acesso (*wifi.scan*). Assim como na interface *Wi-Fi*, o 3G possui três estados de consumo: os estados *radio.active* e *radio.scanning*, que são equivalentes aos estados *wifi.active* e *wifi.scan* da interface *Wi-Fi*. O terceiro estado, *radio.on*, possui dois valores de consumo, um quando está ligado, mas sem sinal e quando está ligado, porém com sinal operante. Por fim, o componente de áudio é modelado apenas por meio do estado ligado e desligado, uma vez que o volume do áudio não interfere de maneira expressiva no consumo da energia [Carroll and Heiser 2010].

3. ECODroid - *Energy Consumption Optimizer for Android*

3.1. Visão Geral

O ECODroid foi desenvolvido como um *plugin* do *Android Studio* com o objetivo de identificar regiões do código-fonte de um aplicativo Android que possuem consumo anormal de energia. A execução do ECODroid consiste em três etapas: (i) instrumentação; (ii) execução e teste; e (iii) visualização de resultados. Na etapa de instrumentação, o código-fonte do aplicativo a ser analisado passa por um processo de instrumentação. No fim desta etapa, uma cópia do projeto contendo o código instrumentado é criada e armazenada em um diretório chamado "`_INSTRUMENTED_`". Já na etapa de execução e teste, a cópia do projeto criada na etapa anterior é compilada e executada de forma automática a fim de realizar a análise de consumo de energia. Por fim, na etapa de visualização dos resultados, os dados sobre consumo de energia gerados na etapa anterior são analisados para determinar quais regiões do código do aplicativo são responsáveis pelo o consumo anormal de energia. O resultado dessa análise é apresentado ao desenvolvedor no formato de um gráfico *Sunburst*[4]. Cada uma dessas etapas são detalhadas nas próximas seções.

3.2. Instrumentação do Aplicativo

O processo de instrumentação no ECODroid é feito de forma automática com o auxílio da biblioteca *JavaParser*. Essa instrumentação consiste na inserção de chamadas à API do ECODroid responsável pela aquisição do consumo de energia. Essa API é uma versão adaptada do *framework* desenvolvido em [Couto et al. 2014]. A API possui a classe `Estimator` que provê a implementação dos métodos cujas chamadas são inseridas no código-fonte do aplicativo via instrumentação. Os métodos utilizados na instrumentação do aplicativo são: `traceMethod()`, responsável por rastrear o consumo de energia dos

[4] *Sunburst* é um desenho ou figura constituída por raios ou "vigas" que irradiam para fora a partir de um disco central em forma de raios de sol.

métodos da aplicação; `saveResults()`, responsável por salvar os resultados do perfil de energia em um arquivo; `start()`, responsável por iniciar a *thread* de monitoramento de energia; e `stop()` responsável por parar a *thread* de monitoramento de energia.

No processo de instrumentação, todos os métodos das classes do aplicativo, assim como as classes destinadas aos testes, terão novas linhas de código adicionadas. Essas linhas de código são inseridas no começo (logo após a declaração do método) e no final (antes da instrução `return` ou na última instrução para métodos que retornam `void`) de cada método. A Figura 1 apresenta um exemplo de código instrumentado. As linhas de código 4 e 6 fazem chamadas à API do ECODroid para inicializar e finalizar, respectivamente, o rastreamento e medição do consumo de energia do método `initialize()` da classe `Controller`.

```
1. public class Controller {
2.  ...
3.  public void initialize(){
4.     Estimator.traceMethod("initialize", "Controller", Estimator.BEGIN);
5.     ...
6.     Estimator.traceMethod("initialize", "Controller", Estimator.END);
7.  }
8.  ...
9. }
```

Figura 1. Exemplo de método instrumentado.

O ECODroid utiliza o *framework* de testes do *Android* para executar automaticamente a aplicação a ser analisada, assim como aplicar os diferentes casos de teste. A API de testes do *Android* é baseada no *JUnit*, a qual é parte integrante do ambiente de desenvolvimento, fornecendo uma arquitetura e ferramentas que ajudam o desenvolvedor a testar seu aplicativo. O processo de instrumentação também é aplicado na classe que possui os casos a serem testados. Os métodos `setUp()` e `tearDown()` são adicionados para serem executados no começo e no final de cada caso de teste, respectivamente, como exemplificado no código da Figura 2.

```
1. public class MyTestCase extends TestCase {
2.  ...
3.  public void setUp(){
4.     Estimator.start(uid);
5.     ...
6.  }
7.  ...
8.  public void tearDown(){
9.     Estimator.stop();
10.    ...
11. }
12. ...
13.}
```

Figura 2. Exemplo de código de teste instrumentado.

Com esta abordagem, o método `Estimator.start(uid)` (linha 4, Figura 2) é chamado toda vez que o caso de teste é iniciado. Nesse momento, o ECODroid inicia a *thread* que coleta as informações do sistema operacional para o cálculo do consumo de energia e em seguida aplica o modelo analítico para estimar a demanda energética. O argumento `uid` do método `start()` é o *Android* `UID` (*Unique Identification*) do aplicativo em teste, o qual é necessário para coletar as informações corretamente. Dentro

do método `tearDown()` é feita uma chamada ao método `Estimator.stop()` (linha 9, Figura 2) que finaliza a execução da *thread* de monitoramento, salvando os resultados obtidos em um arquivo.

3.3. Execução e Teste

Nesta etapa, todas as classes do aplicativo e os seus casos de teste já se encontram instrumentados. Desse modo, um arquivo contendo um conjunto de comandos executados em lote é criado para automatizar o processo. O primeiro comando a ser executado é o `android update project`, o qual é utilizado para atualizar configurações do projeto, como versão do SDK, ambiente de desenvolvimento, modificar o *target* e o nome do projeto. Após sua execução será gerado todos os arquivos e diretórios que estão faltando ou estão desatualizados. O comando tem a seguinte sintaxe:

```
> android update project --name <project_name> --target <target_id> --path <project_path
    >
```

Em seguida são executados os comando `ant clean` e `ant debug`, utilizados para limpar o projeto e criar módulos de depuração, respectivamente.

```
> ant <path_to_your_project> build.xml clean
> ant <path_to_your_project> build.xml debug
```

Para instalar a aplicação e os casos de teste no dispositivo é utilizado o comando `adb install`. O ADB faz parte do pacote do SDK do *Android* e possibilita a comunicação entre o computador e o dispositivo móvel, tornando possível a instalação de aplicações, a troca de dados e também a execução comandos de `shell`.

```
> adb install <path_to_application_apk>
> adb install <path_to_test_apk>
```

Os testes são executados por meio do comando `adb shell am instrument`. Cada caso de teste é executado duas vezes, utilizando opções e `flags` diferentes. No comando abaixo, `<test_package>` representa o nome do pacote de teste atribuído no arquivo do `AndroidManifest.xml`, enquanto `<runner_class>` representa o nome da classe do executor de testes que está sendo utilizado, no caso do ECODroid, o `InstrumentationTestRunner`.

```
> adb shell am instrument [flags] <test_options> <test_package>/<runner_class>
```

Por fim, o comando `adb pull` é executado para que os dados de consumo de energia obtidos sejam copiados do dispositivo móvel (`<remote_folder_path>`) para o computador que está sendo utilizado (`<local_folder_path>`) pelo desenvolvedor.

```
> adb pull <remote_folder_path> <local_folder_path>
```

3.4. Visualização dos Resultados

A Figura 3 apresenta a interface gráfica do ECODroid. Nela é possível identificar duas *Tool Windows* distintas: (i) a **janela principal** (assinalada com o número 1), que apresenta um gráfico *Sunburst* que permite visualizar a distribuição do consumo de energia sobre o código do aplicativo analisado; e (ii) a **janela secundária** (assinalada com o número 2), que traz detalhes sobre o consumo de energia do aplicativo analisado.

O gráfico *Sunburst* apresentado na janela principal do *plugin* retrata dados hierárquicos a partir de discos radiais. No ECODroid, o disco central representa o projeto analisado e o disco radial seguinte refere-se aos pacotes do projeto. As classes são mapeadas no disco radial seguinte e, por fim, os métodos são mapeados no disco radial mais

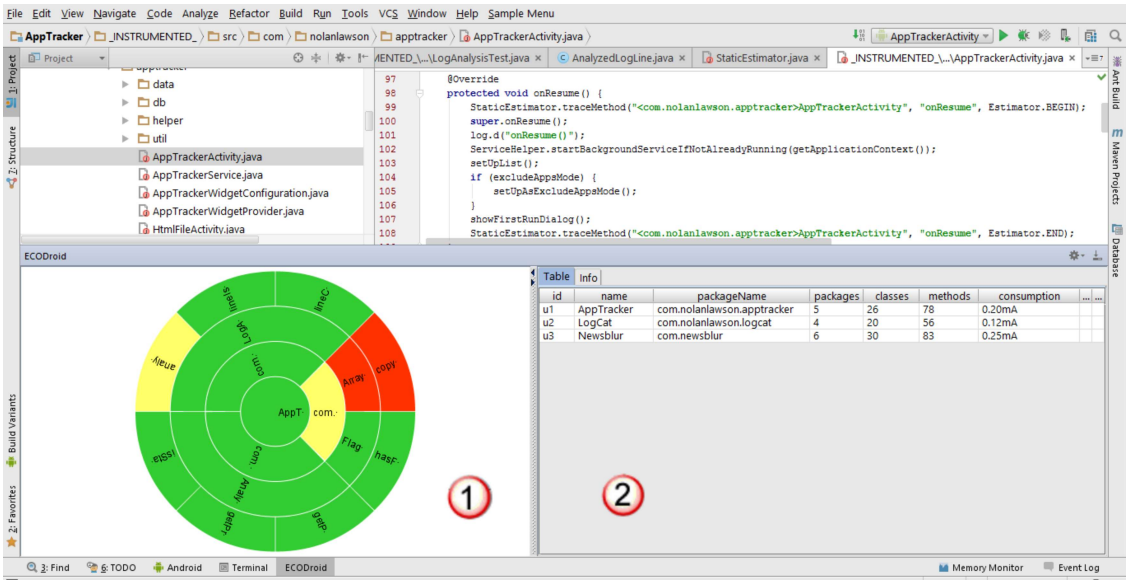

Figura 3. Interface gráfica do ECODroid

externo. Além disso, de acordo com a influência no consumo de energia, o método pode ser classificado como "verde", "amarelo" ou "vermelho". Os 'métodos verdes" são métodos que não influenciam no consumo anormal de energia, dificilmente são chamados quando o aplicativo consome energia acima da média. Já os "métodos vermelhos" influenciam significativamente para o consumo anormal de energia. Com base no modelo analítico adotado, uma aplicação para ser classificada como de baixo consumo de energia deve possuir no máximo 30% dos seus métodos classificados como " métodos vermelhos". Por fim, os "métodos amarelos" possuem influência intermediária no consumo de energia. Essa representação de métodos é extensível para classes, pacotes e projetos. Assim, se uma classe possui mais de 50% dos seus métodos classificados como "vermelhos", então ela também é considerada uma "classe vermelha". Por outro lado, se possui mais de 50% de seus métodos classificados como "verdes", trata-se de uma "classe verde".

A janela secundária está subdividida em duas abas, `Table` e `Info`. A aba `Table` é responsável por mostrar o histórico de informações de projetos que foram analisados ao longo do tempo e possui os seguintes campos: nome do projeto, nome do pacote principal do projeto, quantidade de pacotes, classes e métodos do projeto, além de fornecer o consumo de energia obtido com a análise da ferramenta. Já a aba `Info` permite ao desenvolvedor visualizar as informações sobre o componente selecionado no gráfico *Sunburst*. Quando o desenvolvedor "clica" em alguma parte do gráfico, as seguintes informações sobre o elemento selecionado são apresentadas: o nome do elemento, o seu tipo (projeto, pacote, classe ou método), os nomes dos elementos que estão na hierarquia acima (se houver), o impacto no consumo da aplicação e o consumo medido em joules.

4. Avaliação Inicial

Até o presente momento nenhuma avaliação rigorosa (e.g., estudo de caso ou experimento controlado) foi conduzida para avaliar a real eficácia do ECODroid. Entretanto, um conjunto de aplicativos móveis, desenvolvidos em parcerias do nosso

grupo de pesquisa[5] com empresas de grande porte do setor de mobilidade, vêm sendo alvo de avaliações de consumo de energia por meio do ECODroid. Essas avaliações têm sido úteis para identificar áreas do código-fonte desses aplicativos que possuem problemas relacionados ao consumo anormal de energia. De posse da localização dos trechos de código mais críticos, têm sido possível concentrar esforços para reduzir esses problemas. Durante essas avaliações, foi possível identificar que existe um método específico da API padrão do *Android* que, quando invocado dentro de um aplicativo, faz aumentar consideravelmente o consumo médio de energia. Trata-se do método `dispatchMessage(android.os.Message)` da classe `android.os.Handler`, dedicado a tratar mensagens de sistema.

Com base nessa descoberta, foi selecionado um aplicativo exemplo para evidenciar a identificação dessa anomalia de consumo de energia com o uso do ECODroid. O aplicativo escolhido como alvo dessa avaliação foi o `App Tracker`, que possui código fonte aberto e disponível para *download* no `GitHub`[6]. Esse aplicativo registra as estatísticas de uso de outros aplicativos por meio de uma tarefa que executa em *background* e exibe essas informações em uma interface gráfica (*widget* ou na *activity* principal). Dessa forma, foram feitas duas avaliações do `App Tracker` utilizando o ECODroid. Na primeira, o aplicativo foi avaliado sem que nenhuma alteração tenha sido feita no seu código fonte. Já na segunda avaliação, uma chamada ao método `dispatchMessage()` da API padrão do *Android* foi inserida no código do `App Tracker`. As Figuras 4 e 5 apresentam, respectivamente, o gráfico *Sunburst* para a primeira e para a segunda avaliação.

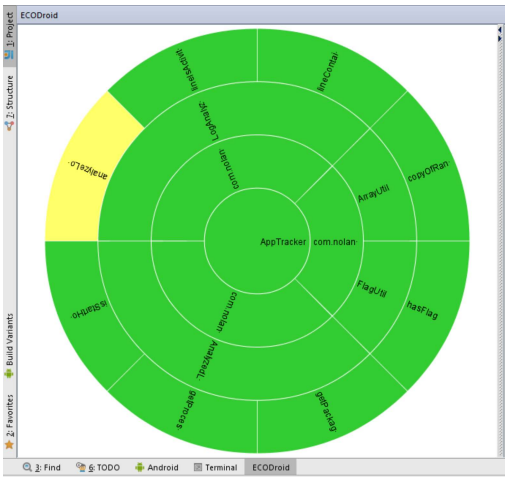

Figura 4. Consumo de energia do `App Tracker` original.

Figura 5. Consumo de energia do `App Tracker` após alteração.

Observe que para a primeira avaliação (Figura 4) apenas o método `analyzeLogLine` obteve consumo de energia um pouco superior a média e foi classificado como sendo um "método amarelo". Já na segunda avaliação (Figura 5), o método `copyOfRange`, que inicialmente é classificado como "método verde" na primeira avaliação (Figura 4), foi modificado para fazer chamadas ao método `dispatchMessage` da API padrão do *Android*. Com essa alteração, o método passou a ser classificado como "método vermelho", evidenciando assim o aumento no consumo de energia.

[5]http://www.great.ufc.br/

[6]https://github.com/nolanlawson/AppTracker

5. Trabalhos Relacionados

O trabalho descrito em [Couto et al. 2014] é fortemente relacionado ao ECODroid. Tanto o modelo analítico de consumo de energia, quanto a forma de apresentar os resultados no formato de gráfico de *Sunburst*, serviram de inspiração para este trabalho. Entretanto, diferente de [Couto et al. 2014], o ECODroid utiliza valores de referência, providos pelos fabricantes, sobre o consumo energético dos componentes de hardware do dispositivo, tornando as suas estimativas mais precisas. Além disso, o ECODroid provê ao desenvolvedor um suporte ferramental automatizado e integrado ao ambiente de desenvolvimento.

Já o trabalho descrito em [Liu et al. 2014] apesenta um estudo sobre a identificação das causas de *bugs* relacionados ao consumo anormal de energia em aplicativos *Android*. Naquele trabalho, os autores utilizam uma técnica de instrumentação de código para gerar *traces* de execução que serão analisados por meio de um verificador de modelos. Essa análise ajuda a identificar padrões de *bugs* previamente catalogados. Enquanto o trabalho de [Liu et al. 2014] está preocupado em encontrar a causa dos problemas de consumo de energia, o ECODroid está preocupado em identificar as partes do código-fonte que apresentam consumo anormal de energia. Portanto, o trabalho de [Liu et al. 2014] pode ser visto como uma abordagem complementar do presente trabalho.

6. Considerações Finais

Este trabalho apresentou o ECODroid, uma ferramenta para análise e visualização do consumo de energia em aplicativos Android. Uma avaliação inicial do ECODroid foi realizada, apresentando indícios de sua viabilidade como ferramenta de auxílio na localização de trechos de código-fonte que apresentam consumo anormal de energia. Embora o ECODroid ajude a localizar o trechos de código-fonte problemáticos, ela não oferece suporte à resolução dos problemas de consumo de energia identificados, sendo esse um dos trabalhos futuros a ser realizado. Além disso, uma avaliação mais rigorosa do ECODroid deve ser conduzida a fim de evidenciar a sua efetividade.

Referências

Carroll, A. and Heiser, G. (2010). An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, Berkeley, CA, USA. USENIX Association.

Couto, M., Carção, T., Cunha, J., Fernandes, J., and Saraiva, J. (2014). Detecting anomalous energy consumption in android applications. In Quintão Pereira, F. M., editor, *Programming Languages*, volume 8771 of *Lecture Notes in Computer Science*, pages 77–91. Springer International Publishing.

Liu, Y., Xu, C., Cheung, S., and Lu, J. (2014). Greendroid: Automated diagnosis of energy inefficiency for smartphone applications. *Software Engineering, IEEE Transactions on*, 40(9):911–940.

Pathak, A., Hu, Y. C., and Zhang, M. (2012). Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA. ACM.

Tarkoma, S., Siekkinen, M., Lagerspetz, E., and Xiao, Y. (2014). *Smartphone Energy Consumption: Modeling and Optimization*. Cambridge University Press.