UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

CONSTRUÇÃO E COMPARAÇÃO DE MODELOS DE CONSUMO DE ENERGIA PARA PLATAFORMAS ANDROID

Emiliano Carlos de Moraes Firmino

Recife, março de 2016.

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE TECNOLOGIA E GEOCIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

CONSTRUÇÃO E COMPARAÇÃO DE MODELOS DE CONSUMO DE ENERGIA PARA PLATAFORMAS ANDROID

por

EMILIANO CARLOS DE MORAES FIRMINO

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco como parte dos requisitos para a obtenção do grau de Mestre em Engenharia Elétrica.

ORIENTADOR: PROF. DR. RAFAEL DUEIRE LINS

Recife, março de 2016.

Catalogação na fonte
Bibliotecária Valdicéa Alves, CRB-4 / 1260

F525c Firmino. Emiliano Carlos de Moraes.
Construção e comparação de modelos de consumo de energia para plataformas android / Emiliano Carlos de Moraes Firmino - 2016.
154folhas, Il.; Tabs.; Eq.; e Prog.

Orientador: Prof. Dr. Rafael Dueire Lins.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Elétrica, 2016.
Inclui Referências e Apêndices.

1. Engenharia Elétrica,. 2. Modelo de consumo de energia.
3. Gerenciamento de energia. 4. Android. I. Lins, Rafael Dueire (Orientador).
II. Título.

UFPE

621.3 CDD (22. ed.) BCTG/2016 - 218

Universidade Federal de Pernambuco
Pós-Graduação em Engenharia Elétrica

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE DISSERTAÇÃO DO MESTRADO ACADÊMICO DE

EMILIANO CARLOS DE MORAES FIRMINO

TÍTULO

"CONSTRUÇÃO E COMPARAÇÃO DOS MODELOS DE CONSUMO DE ENERGIA NA PLATAFORMA ANDROID"

A comissão examinadora composta pelos professores: RAFAEL DUEIRE LINS, CIN/UFPE; CECILIO JOSÉ LINS PIMENTEL, DES/UFPE e FERNANDO JOSÉ CASTOR DE LIMA FILHO, CIN/UFPE, sob a presidência do primeiro, consideram o candidato **EMILIANO CARLOS DE MORAES FIRMINO** **APROVADO**.

Recife, 04 de março de 2016.

MARCELO CABRAL CAVALCANTI Coordenador do PPGEE	**RAFAEL DUEIRE LINS** Orientador e Membro Titular Interno
FERNANDO JOSÉ CASTOR DE LIMA FILHO Membro Titular Externo	**CECILIO JOSÉ LINS PIMENTEL** Membro Titular Interno

Resumo da Dissertação apresentada à UFPE como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

CONSTRUÇÃO E COMPARAÇÃO DE MODELOS DE CONSUMO DE ENERGIA PARA PLATAFORMAS ANDROID

Emiliano Carlos de Moraes Firmino

Março / 2016

Orientador: Prof. Dr. Rafael Dueire Lins.
Área de Concentração: Telecomunicações
Palavras-chave: modelo de consumo de energia, gerenciamento de energia. Android.
Número de Páginas: 150.

Esta dissertação de mestrado investiga o consumo e a modelagem de energia para *smartphones* que usam o sistema operacional Android. A principal reclamação dos usuários e desafio aos fabricantes é a limitada duração da carga da bateria desses dispositivos. Para melhor entender o problema, esta dissertação faz ampla revisão da literatura visando identificar os principais componentes responsáveis pelo alto consumo de energia, os modelos de consumo de energia empregados e as estratégias de otimização propostas e utilizadas. Foi construída uma ferramenta em *software* e levado a cabo um conjunto de experimentos para medir o consumo de energia dos principais componentes. A dissertação apresenta um novo modelo de consumo de energia desenvolvido com técnicas de regressão estatísticas para descrever o padrão de consumo dos componentes. O modelo produzido foi comparado a outros encontrados na revisão da literatura por meio de simulações de consumo com base em dados coletados. Por fim, os resultados foram analisados e as conclusões apresentadas.

Abstract of Dissertation presented to UFPE as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

CONSTRUCTION AND COMPARISON OF ENERGY MODELS FOR ANDROID PLATFORMS

Emiliano Carlos de Moraes Firmino

March / 2016

Supervisor: Prof. Dr. Rafael Dueire Lins.
Area of Concentration: Telecommunications
Keywords: power model, energy management, Android.
Number of Pages: 150.

This M.Sc. dissertation investigates the power consumption and modeling for Android smartphones. The most common issue reported by users and a constant challenge to manufacurers is how to handle the limited duration of the battery charge for those devices. To better understand such a problem, this dissertations presents an extensive review of the literature to identify the key components that drains most of the power and which power consumption models and optimization strategies are currently being used to handle such a problem. That information was used to develop a *software* tool and a set of experiments to measure the energy consumption of those key components. A new power consumption model is proposed here based on statistical regression to describe those key components. The model presented was compared with those found in the technical literature using simulations based on the data collected. Finally, the results obtained are analyzed and conclusions drawn.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE EQUAÇÕES

LISTA DE TABELAS

LISTA DE PROGRAMAS

1 INTRODUÇÃO

A curta duração de carga das baterias usadas em *smartphones*, *tablets* e outros dispositivos portáteis "inteligentes" é um dos grandes desafios dos fabricantes e usuários (PATHAK *et al.*, 2012) desses aparelhos. A capacidade limitada das baterias, combinada ao crescente poder de processamento, diversidade de sensores e acesso à Internet pressiona o desenvolvimento de dispositivos eficientes quanto ao uso de energia. A melhoria na capacidade da bateria depende de avanços nos processos eletroquímicos e de encapsulamento que ocorrem muito lentamente quando comparados aos observados na eletrônica (FIORE *et al.*, 2014). Fabricantes aplicam técnicas de gerenciamento de energia para reduzir o consumo e prolongar o uso do dispositivo. O sistema operacional define políticas de alocação de energia, desativação seletiva dos componentes em desuso e outras estratégias. *Smartphones*, *tablets* e outros dispositivos portáteis inteligentes aplicam políticas agressivas de economia de energia (JINDAL *et al.*, 2013), já que, para esses dispositivos, o gerenciamento de energia é crítico (DOGAR; STEENKISTE, 2010).

O modelo de consumo de energia quantifica o impacto de cada componente no consumo por métodos matemáticos (TARKOMA *et al.*, 2014), permitindo estimar o consumo de energia do aparelho, de seus componentes, saber quanto tempo resta de bateria e auxiliar o sistema operacional a determinar a melhor estratégia para prolongar a duração da bateria até a próxima recarga.

A popularização dos *smartphones* e do acesso móvel à Internet, observados na última década, reviveu o interesse dos pesquisadores na área de gerenciamento de energia nesses dispositivos (TARKOMA *et al.*, 2014). Entretanto, a construção de modelos precisos de consumo de energia continua um desafio, dada a grande diversidade de aparelhos, componentes de *hardware* e *software*, complexas interações com o ambiente e limitação de acesso a informação da plataforma pelos fabricantes. Diversos esforços foram realizados para esclarecer os padrões de consumo dos *smartphones*, evidenciar os principais "vilões" do consumo e propor estratégias de otimização para prolongar a carga da bateria.

A modelagem de consumo de energia depende do entendimento dos processos de construção e comparação dos diversos modelos disponíveis. Nesta dissertação, é apresentado um processo de construção e uma revisão da literatura sobre os componentes intensivos em energia citados na literatura para ajudar outros pesquisadores da área.

1.1 Objetivos

Esta dissertação investiga a modelagem de consumo de energia para *smartphones* da plataforma Android, escolhida por ser aberta, amplamente estudada e alcançar cerca de 80% do mercado global.

Este trabalho apresenta o estado da arte na geração de modelos e os componentes de maior consumo. Foi criado um método sistemático para construção do modelo e produzida uma ferramenta de medição em *software* ou *hardware* capaz de ser usada em experimentos de consumo que auxiliarão a modelagem do sistema. Por fim, valida-se o modelo gerado a partir de simulações de consumo.

1.1.1 Objetivos específicos

- Identificar o estado da arte na construção de modelos de energia.
- Determinar as componentes de maior impacto no consumo de energia.
- Definir um método sistemático para construção do modelo.
- Construir uma ferramenta em *software* ou *hardware* para registrar o consumo.
- Fazer experimentos de consumo de energia em aparelhos Android.
- Fazer experimentos em aparelhos de diferentes fabricantes.
- Construir o modelo considerando os componentes de maior consumo de energia.
- Validar o modelo gerado por simulações e medições experimentais.

1.2 Organização dos capítulos

A dissertação está estruturada em sete capítulos para facilitar a compreensão dos temas abordados, sendo esta introdução o primeiro deles. O capítulo 2 apresenta os fundamentos de gerenciamento de energia aplicado para *smartphones*. O capítulo 3 apresenta a revisão da literatura realizada para determinar o estado da arte da modelagem de consumo de energia. O capítulo 4 aborda brevemente as interfaces de programação e a ferramenta *Droid Energy Suite*. O capítulo 5 apresenta os celulares usados, a metodologia empregada e os resultados obtidos. No capítulo 6 são analisados os resultados e é gerado um modelo simplificado de consumo de energia. O capítulo 7 discute os objetivos e conclusões alcançadas neste trabalho e apresenta oportunidades futuras de pesquisa.

2 FUNDAMENTOS DE GERENCIAMENTO DE ENERGIA

Dispositivos móveis possuem fonte embarcada de energia, a bateria, que permite operar por tempo limitado até a próxima recarga. O objetivo do gerenciamento de energia é estender a duração da bateria, limitando o consumo por meio da alocação da energia nos recursos essenciais, sem prejudicar a experiência do usuário.

Este capítulo apresenta uma rápida introdução aos conceitos de gerenciamento de energia. Define uma visão de alto nível sobre o consumo de energia. Explica as características das baterias, dos vários componentes, a metodologia empregada para medir e analisar o consumo dos dispositivos.

2.1 Modelo de consumo de energia

O dispositivo móvel contém diversos componentes que precisam de energia elétrica para funcionar, mas, para sua operação normal, há somente uma fonte que é a bateria. A Figura 2.1 ilustra este sistema. O fluxo de energia, indicado pelas setas, e as componentes do sistema, as caixas, drenam energia da bateria, uma fonte limitada de energia. A energia armazenada na bateria somente pode ser reposta pela recarga, conectando o dispositivo a fonte de alimentação externa ao sistema.

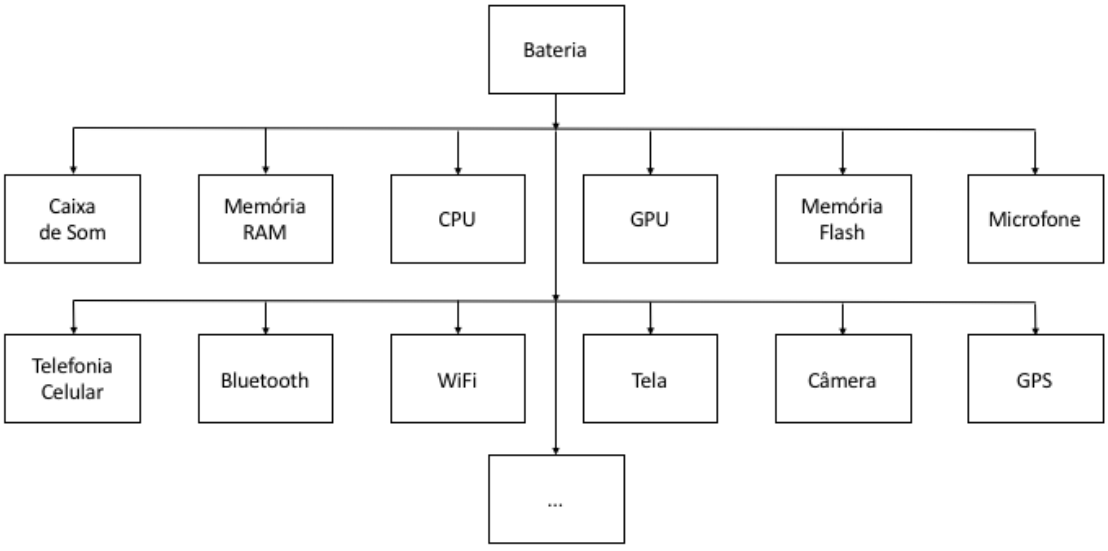

Figura 2.1: Modelo de sistema de consumo em dispositivos móveis a bateria. A bateria é a única fonte de energia a todos os outros componentes do sistema.

O tempo que o sistema consegue manter-se ativo até descarga completa da bateria depende do consumo conjunto das várias componentes. Técnicas de gerenciamento de energia permitem reduzir o consumo do sistema pela ativação seletiva das componentes,

promovendo otimização do uso e, assim, prolongar o tempo ativo do sistema. O projeto desse sistema de gerenciamento requer conhecimento sobre as características das componentes. A seguir, é apresentada breve introdução sobre tais componentes e estratégias de otimização do consumo.

2.2 Bateria

A bateria é a componente que converte energia química contida em seu material ativo em energia elétrica por meio de reações eletroquímicas de oxirredução (LINDEN; REDDY, 2001). Baterias são classificadas como *primárias* ou *secundárias*.

Baterias *primárias* são aquelas que, uma vez descarregadas, precisam ser descartadas. Elas possuem maior densidade de energia, boa retenção de carga que permite longa vida útil a taxas de descargas baixas e moderadas. São baterias de fabricação barata e de fácil uso em dispositivos eletrônicos.

Baterias *secundárias* podem de ser recarregadas pela passagem de corrente no sentido oposto à corrente de descarga. Elas possuem menor densidade de energia e retenção de carga que as baterias primárias, sendo de maior custo, elas são usadas principalmente em eletrônicos que demandam muita energia e uso frequente.

Embora o termo *bateria* seja corriqueiramente empregado, a unidade eletroquímica fundamental da bateria é a *célula* (LINDEN; REDDY, 2001). A bateria é composta de uma ou mais células conectadas em série ou paralelo. O arranjo das células é o que define a voltagem de saída e capacidade de armazenamento (LINDEN; REDDY, 2001). Os componentes principais da célula são:

1. Ânodo (eletrodo negativo): redutor que libera elétrons para o circuito e é oxidado durante o processo de oxirredução.
2. Cátodo (eletrodo positivo): oxidante que recebe elétrons do circuito e é reduzido durante o processo de oxirredução.
3. Eletrólito (meio iônico condutor): meio pelo qual ocorre transferência de íons dentro da célula, entre o ânodo e cátodo. O eletrólito pode ser líquido ou sólido. Eletrólitos líquidos são compostos de água ou outros solventes com sais, ácidos ou alcalinos dissolvidos para permitir a condução dos íons. Eletrólitos sólidos utilizam compostos sólidos não condutores para fabricação de células de baixa potência e longa vida útil.

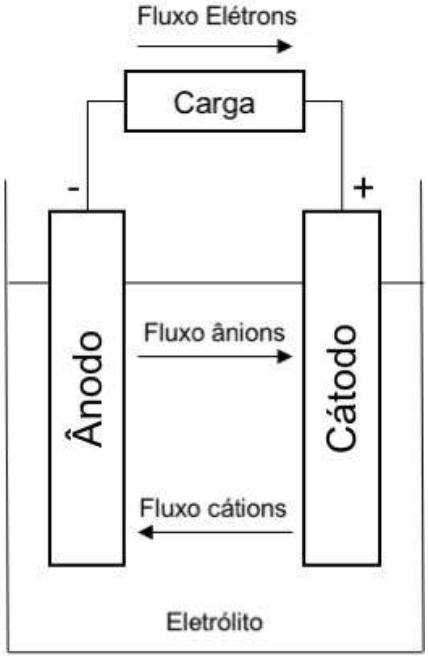

Figura 2.2: Ciclo de descarga.

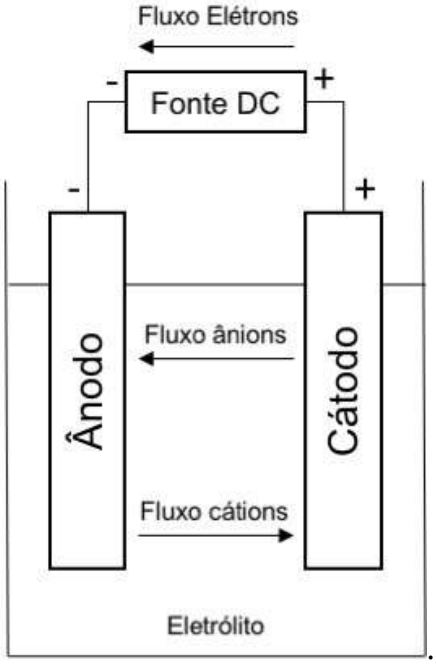

Figura 2.3: Ciclo de recarga

O ciclo de descarga da célula é apresentado na Figura 2.2. Os terminais do ânodo e cátodo, quando conectados à carga externa, permitem o fluxo de elétrons do ânodo para o cátodo. O ânodo sofre oxidação e libera elétrons, o cátodo sofre redução e recebe os elétrons. O circuito é fechado pelo eletrólito que permite a passagem de ânions (íons negativamente carregados) e cátions (íons positivamente carregados) entre os dois eletrodos.

O processo de recarga é apresentado na Figura 2.3. Os terminais do ânodo e cátodo são conectados à fonte externa de alimentação que aplica corrente contínua no sentido oposto ao sentido natural de descarga da bateria. O ânodo sofre redução e ganha elétrons, o cátodo sofre oxidação e perde elétrons.

O processo químico de descarga e recarga é exemplificado no apêndice A. A principal classe de bateria usada em *smartphones* são as baterias de lítio. Pesquisas envolvendo baterias de lítio começaram em 1912 com o trabalho pioneiro de G. N. Lewis, porém, somente na década de 1970, as primeiras baterias chegaram ao mercado (TARKOMA *et al.*, 2014). A comercialização de baterias recarregáveis de lítio-íon começou apenas em 1991 pela Sony. Hoje, a maioria dos dispositivos recarregáveis usa esse tipo de bateria (TARKOMA *et al.*, 2014).

Fabricantes podem estimar as características da bateria pelas características dos materiais empregados na fabricação do ânodo e cátodo ou determiná-las experimentalmente. Na prática, os valores reais são uma fração do valor estimado, devido

ao uso de outros materiais não reativos na fabricação, concentração do material e temperatura (LINDEN; REDDY, 2001). As características principais da bateria são a capacidade de armazenamento e a voltagem de seus terminais.

$$Energia\,(Wh) = Voltagem\ (V) \times Capacidade\,(Ah)$$
$$Energia\,(J) = Voltagem\ (V) \times Capacidade\,(C) \qquad (2\text{-}1)$$

A energia fornecida pela bateria é determinada pela Equação (2-1). A energia fornecida pela bateria em watt-hora (Wh) ou joules (J) é o produto da voltagem em volts (V) pela capacidade em ampère-hora (Ah) ou coulombs (C).

Avanços na tecnologia de baterias ocorrem lentamente quando comparados aos da microeletrônica. No período de 1940 a 2000, apresentado na Figura 2.4, é possível ver o lento aumento da capacidade de armazenamento das baterias primárias e secundárias. LINDEN e REDDY (2001) destacaram que a tecnologia de baterias não acompanhou o rápido ritmo da tecnologia de eletrônicos no mesmo período. TARKOMA *et al.* (2014) citaram que as baterias de lítio-íon e lítio-polímero atingiram a capacidade de 200 Wh/kg, um aumento de apenas 25% na capacidade quando comparado a 2000. Esse lento crescimento da capacidade é uma das principais motivações para uso de técnicas de gerenciamento de energia.

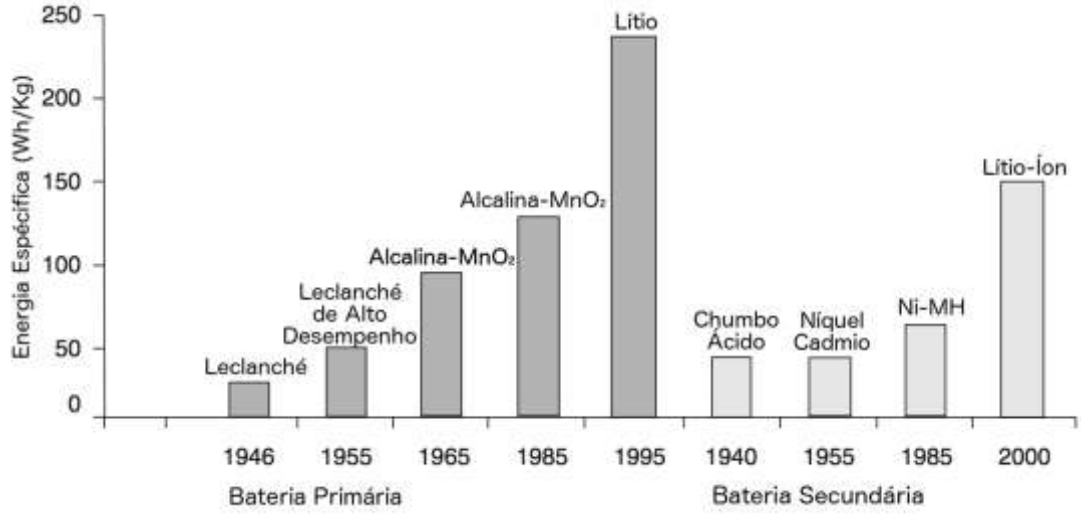

Figura 2.4: Avanço na tecnologia de baterias para aplicações portáteis (LINDEN; REDDY, 2001).

2.3 Circuitos integrados

Os circuitos integrados (CI) dominam a lista de componentes internos do dispositivo móvel. Processador central (CPU), processador gráfico (GPU), memória RAM, memória *flash* e *chipsets* de propósitos diversos são empregados na fabricação. Pode-se afirmar que dispositivos móveis só existem graças aos avanços obtidos na miniaturização desse componente em conformidade com a lei de Moore (MACK, 2011).

O elemento básico do CI é o transistor. O transistor é dispositivo semicondutor inventado no final da década de 1940 que é utilizado para uma série de aplicações, como amplificadores e circuitos digitais. Em circuitos digitais, é utilizado como interruptor para a construção de portas lógicas. Circuitos integrados implementam circuitos digitais completos em silício por meio da combinação de milhares, milhões ou bilhões de portas lógicas.

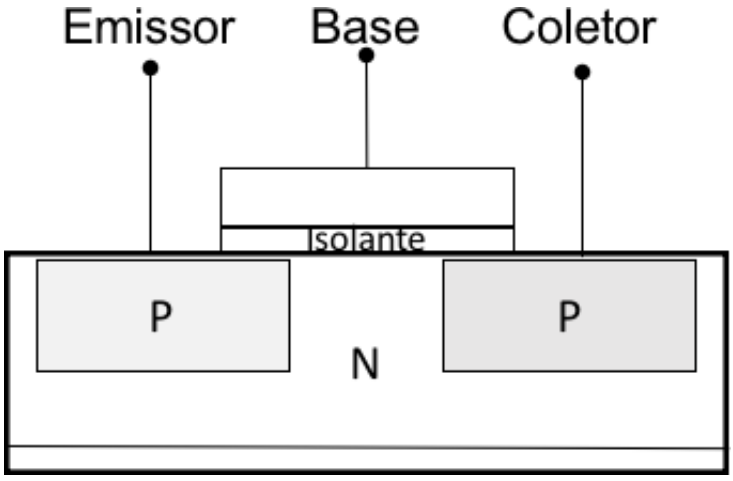

Figura 2.5: Esquema de um transistor PNP.

O transistor é componente com três terminais: coletor, emissor e base. Como exemplificado no transistor do tipo PNP da Figura 2.5. O transistor não conduz corrente até a base ser energizada por diferença de potencial. No transistor perfeito, o isolante não permite a passagem de corrente entre a base e o coletor, mas, na prática, a corrente conhecida como corrente de fuga o atravessa gerando perda constante de energia.

$$Energia_{\ total} = Energia_{\ estática} + Energia_{\ Dinâmica} \qquad (2\text{-}2)$$

A dissipação de energia do transistor é definida pela Equação (2-2). A energia dissipada é composta de duas partes. A energia estática representa a energia dissipada pela corrente de fuga. Ela tende a crescer com a miniaturização ou menor eficiência do material

isolante. A energia dinâmica é a energia útil requerida para chavear o transistor. Ela é proporcional a voltagem e frequência conforme Equação (2-3).

$$Energia_{\ dinâmica} \propto Carga\ Capacitiva \times Voltagem^2 \times Frequência \qquad (2\text{-}3)$$

A redução na voltagem tem grande impacto na redução do consumo de energia. Justificando a redução de 5 V para menos de 1 V da tensão dos processadores nos últimos 20 anos (HENNESSY; PATTERSON, 2012).

Apenas redução da frequência, por si, não permite redução na energia, porque o número de ciclos computacionais necessário permaneceria o mesmo. Porém, o uso de frequências menores permite redução da voltagem, técnica conhecida como *Dynamic Voltage Frequency Scaling* (DVFS).

A maior parte da energia é dissipada na forma de calor. Circuitos integrados são projetados para dissipação máxima do calor conhecido como projeto de força térmica ou pelo termo em inglês *thermal design power* (TDP). O TDP limita o valor máximo de dissipação e consumo de energia do circuito integrado. O TDP limita a área ativa do circuito integrado. Muitos circuitos integrados desligam áreas ou reduzem a voltagem e frequência delas para atender às restrições de TDP. Essas áreas inativas ou subutilizadas são conhecidas como *Dark Silicon*.

Embora todos esses componentes pertençam à classe de circuitos integrados, o tipo de trabalho que realizam afeta o padrão de consumo e oportunidades de otimização energética. A seguir, veremos as características e estratégias de energia de alguns deles.

2.3.1 Unidade de processamento central (CPU)

A CPU é o centro de processamento do dispositivo. Nela são executados o sistema operacional e as aplicações do usuário. Hoje, cada vez mais, as arquiteturas multinúcleo são empregadas em dispositivos móveis. *Smartphones* com CPU de 2, 4 ou 8 núcleos são encontrados no mercado.

O processamento demanda energia constante da bateria. Sempre que o usuário interage com o dispositivo, a CPU precisa estar ativa. Os processadores implementam a especificação *Advanced Configuration and Power Interface (ACPI)* que define estados chamados de *Performance State (P-states)* e *Idle Power State (C-states)* para controle da energia (ZHANG *et al.*, 2015).

P-states são configurações fixas de frequência e voltagem que permitem ao processador selecionar em tempo de operação configurações estáveis de DVFS e, assim, optar entre otimização de desempenho ou economia de energia. Por exemplo, o LG Nexus 4 possui doze diferentes configurações de frequência de 384 MHz até 1512 MHz. A seleção da frequência de operação tem impacto no consumo de energia, o consumo do núcleo a 1512 MHz é mais de três vezes maior do que a de 384 MHz (ZHANG *et al.*, 2015).

C-states são estados de espera aplicados quando não existe carga de trabalho (ZHANG *et al.*, 2015). São enumerados como C0, C1, C2 e C3. Cada estado representa um estado mais profundo de espera, com maior economia de energia, porém maior atraso para retorno as atividades. Por exemplo, o C0 desabilita o *clock*, C1 desabilita a lógica de processamento, C2 desabilita a memória *cache* L0/L1 e, por fim, C3 desabilita a memória *cache* compartilhada L2 (ZHANG *et al.*, 2015).

Processadores multinúcleos suportam um mecanismo chamado *CPU Hot-Plugging* no qual é possível desligar por completo um ou mais núcleos que não estejam ativos. A desvantagem: é processo caro e demorado comparado ao uso de *C-states* (ZHANG *et al.*, 2015).

A maioria das arquiteturas multinúcleos disponíveis é simétrica, todos os núcleos possuem a mesma capacidade de processamento e demanda de energia. Arquiteturas assimétricas como o *big.LITTLE* da ARM (LUKEFAHR *et al.*, 2014) possuem núcleos com alta capacidade de processamento lado a lado com núcleos energeticamente eficientes. Essas arquiteturas permitem reduzir o consumo por meio da alocação eficiente da carga de trabalho no processador mais adequado a tarefa (LUKEFAHR *et al.*, 2014; ZHANG *et al.*, 2015).

2.3.2 Unidade de processamento gráfico (GPU)

GPU são unidades otimizadas para processamento de imagem, geração de gráficos 3D e decodificação de vídeo. Embora a CPU possa realizar esse processamento, a GPU é mais eficiente para esse tipo de problema. Isso é consequência da diferença de paradigma e problemas que buscam resolver. Enquanto a CPU é otimizada para redução da latência de acesso a memória e processamento sequencial de instruções. A GPU é otimizada para vazão, paralelismo de dados e possui as operações comuns de processamento de imagem e vídeo em *hardware*.

Assim como a CPU, a GPU implementa interface ACPI. Técnicas de gerenciamento de energia aplicados à CPU como DVFS também são aplicáveis à GPU. NIXON *et al.* (2014) propuseram um método de controle dinâmico da resolução (DRS) e taxa de atualização da tela (DFRS), a fim de reduzir o consumo da GPU, quando exceder a capacidade de percepção humana. HONG e KIM (2010) propuseram um modelo empírico detalhado de consumo de energia para GPU. O impacto da GPU no consumo de energia é relevante em aplicações que usam recursos avançados de processamento de imagem, animações e vídeo, como em aplicações que manipulam vídeo ou jogos. Neste trabalho optou-se por não avaliar o consumo da GPU por limitações de tempo do projeto.

2.3.3 Memória RAM

Memória RAM (*Random Access Memory*) é a memória volátil que armazena informações de trabalho em uso pela CPU. Ela é um circuito integrado que contém minúsculos capacitores capazes de armazenar os bits de informação. A Figura 2.6 apresenta o esquema das células de memória RAM. Para armazenar um bit de informação é necessário o uso de um capacitor e um transistor. É forma barata de armazenamento que permite fabricar memórias de alta densidade e baixo custo. Porém, é necessária recarga periódica dos capacitores para evitar perda da informação armazenada decorrente de sua descarga natural.

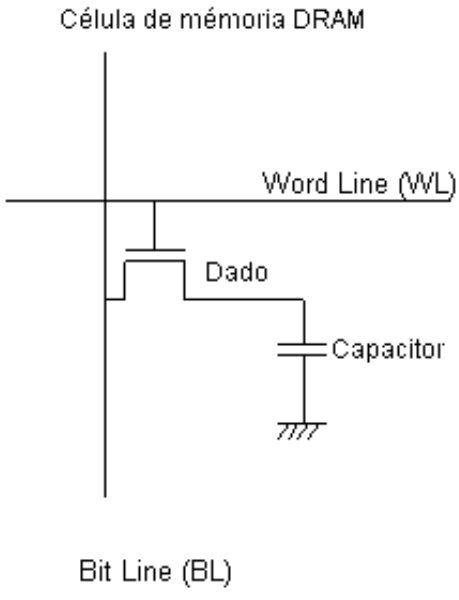

Figura 2.6: Diagrama da célula RAM.

Em dispositivos menores como *smartphones* e *tablets*, a memória RAM representa pequena parte do consumo, comparado a outras componentes, mas, nos sistemas embarcados ou computadores com grandes quantidades de memória, como *notebooks*, esse valor pode se tornar representativo.

2.3.4 Memória flash

A memória *flash* não é volátil e armazena informações por longo prazo. Nela ficam armazenados o sistema operacional, aplicações, estrutura de diretórios e arquivos. Diferente de outras tecnologias de armazenamento como disco rígido, a memória *flash* é leve, tolerante a impactos e, por isso, amplamente empregada em dispositivos móveis.

Existem dois tipos de memória *flash* no mercado: NAND e NOR. Ambas foram criadas na década de 1980 pelo Dr. Fujio Masuoka enquanto trabalhava para a Toshiba. A memória NOR foi primeiramente descrita em 1984. Internamente, é composta de células conectadas em paralelo, o que permite acesso rápido a informação, porém, possui menor densidade de armazenamento, menor número de ciclo de apagamento e são empregadas principalmente para armazenar de programas em *hardware*. As memórias NAND foram primeiramente descritas em 1987. Suas células são organizadas em série permitindo maior densidade de armazenamento, o que as torna muito mais baratas. A velocidade de escrita e apagamento é maior, porém, a leitura é lenta devido ao acesso sequencial. Elas são amplamente usadas para armazenamento secundário.

Memória NAND consome menos energia que memória NOR, consequência da maior velocidade de escrita e apagamento das memórias NAND (TOSHIBA, 2006). CARROLL e HEISER (2010) fizeram alguns *benchmarks* de escrita e leitura na memória *flash* interna e *SD card* da plataforma OpenMoko. Eles observaram que, em espera, o consumo foi muito baixo, 0,4 mW e 1,4 mW respectivamente. Leituras e escrita em memória interna tem consumo inferior a 5 mW; enquanto no *SD card* fica na faixa de 50 mW.

2.4 Tela

A tela é o dispositivo principal de entrada e saída dos *smartphones* e *tablets*. É um painel que emite luz organizado em matriz de pixels que são coloridos independentemente, permitindo a renderização da interface gráfica do sistema. Um pixel é composto de três cores primárias: verde, vermelho e azul (RGB). Eles são combinados para reproduzir

outras cores. Nesses dispositivos, a tela é coberta de sensores de toque que permitem ao usuário interagir diretamente com o que é renderizado na tela. Ela ocupa quase toda a superfície frontal do dispositivo e, sempre que o usuário interage com as aplicações, precisa estar ativa. A tela representa grande parte da energia consumida pelo sistema (VALLINA-RODRIGUEZ; CROWCROFT, 2012). Para reduzir o consumo, o sistema operacional automaticamente ajusta o brilho ao ambiente, reduz o brilho após algum tempo de inatividade ou a desliga. Os tipos de tela empregados em dispositivos móveis são LCD e AMOLED.

2.4.1 Tela de cristal líquido (LCD)

O LCD é composto de duas partes principais: uma fonte de iluminação que é chamada de *backlight* e um painel LCD que filtra a luz baseado nos valores de cor de cada pixel (ANAND *et al.*, 2011). Telas LCD tradicionais utilizam lâmpada fluorescente de cátodo frio (CCFL), enquanto telas modernas usam uma série de LEDs. Quase toda a energia consumida no LCD é do *backlight*, portanto, qualquer técnica que reduza o brilho necessário da tela tem impacto na redução da energia.

CARROLL e HEISER (2010) mediram consumo de energia do LCD do *Openmoko Neo Freerunner* variando o valor de brilho representado por valor linear inteiro de 1 a 255. Quando desligado, o consumo do LCD é desprezível. Quando ligado, o consumo variou exponencialmente como visto na Figura 2.7. Além disso, foi comparado o consumo entre uma tela completamente branca e outra completamente preta, foi observado que o consumo variou 40 mW. Portanto, embora o *backlight* domine, o conteúdo apresentado na tela também determina o consumo.

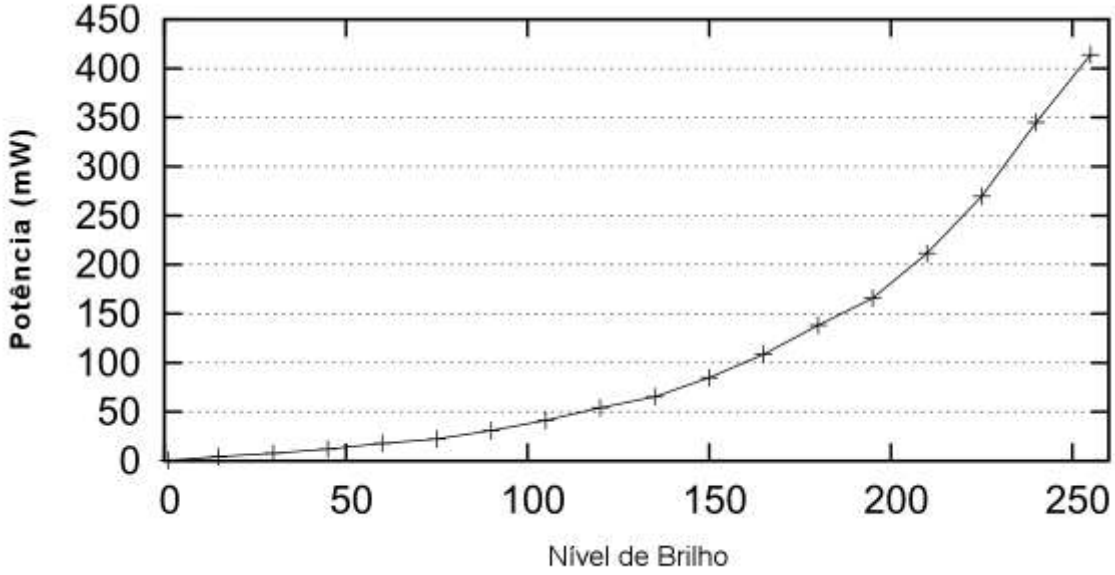

Figura 2.7: Consumo do backlight variando a intensidade de brilho (CARROLL; HEISER, 2010).

2.4.2 Matriz-ativa de diodo orgânico emissor de luz (AMOLED)

O AMOLED é tecnologia relativamente recente de tela. Diferente do LCD que precisa de um *backlight* para iluminar a tela, os componentes RGB de cada pixel emitem sua própria luz. Dessa forma, telas AMOLED podem ser muito mais finas que as telas LCD e apresentar qualidade de imagem superior (CHEN *et al.*, 2013). A energia consumida é a soma da energia de cada componente de cor de todos os pixels. Portanto, a energia depende diretamente do conteúdo apresentado. A cor é fator determinante, as componentes RGB possuem consumos diferentes, como pode ser visto na Figura 2.8. Telas AMOLED são mais eficientes que telas LCD (CHEN *et al.*, 2013), porém, continuam a dominar o consumo de energia dos dispositivos.

CHEN *et al.* (2013) analisaram o consumo de telas AMOLED de cinco dispositivos normalizados pela área. O consumo de telas AMOLED é fortemente dependente das subcomponentes de cor e da tecnologia de fabricação como visto na Figura 2.8. O consumo por pixel é determinado pela Equação *(2-4)*, onde $f(R)$, $g(B)$ *e* $h(G)$ são as curvas de consumo em função da intensidade de brilho de cada componente RGB na faixa de 0 a 255. O consumo da tela é apresentado na Equação (*2-5*), é o somatório do produto do número de pixel de certa cor pela sua energia.

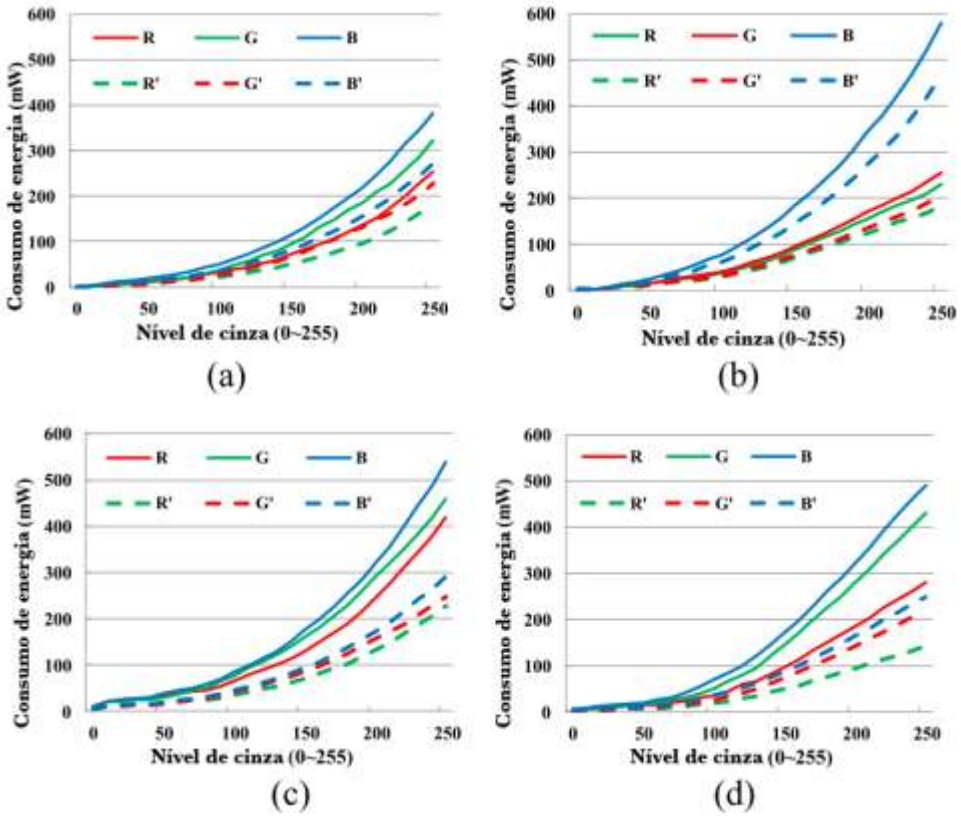

Figura 2.8: Consumo de tela AMOLED por componente de cor para (a) Nexus; (b) Galaxy S; (c) Galaxy Nexus; (d) Galaxy S3. Linha sólida representa o valor original, linha tracejada o valor normalizado para uma tela de 4in² (CHEN *et al.*, 2013).

$$Energia_{Pixel} = f(R) + h(G) + g(B) \tag{2-4}$$

$$Energia_{tela} = \sum_{i=1}^{N} (n_i \times Energia_{pixel}(R_i, G_i, B_i)) \tag{2-5}$$

Por ser tecnologia recente e em adoção, fabricantes e pesquisadores buscam novas formas de redução de consumo. SHIN *et al.* (2011) propuseram aplicar controle dinâmico de voltagem (DVS) ao painel AMOLED sem degradar a imagem percebida pelos usuários. LIU, SHENOY e CORNER (2005) propuseram a técnica chamada *Chameleon* para alterar o esquema de cores das páginas Web renderizadas pelo navegador. LI, TRAN e HALFOND (2014) propuseram que páginas Web sejam reescritas e otimizadas para telas AMOLED. CHEN *et al.* (2014) propuseram aproveitar a granularidade de controle do brilho dos pixels da telas AMOLED para escurecer áreas da tela encobertas pelos dedos do

usuário. A maioria das técnicas de otimização buscam escurecer a tela e reduzir uso de cores mais energéticas como o azul.

2.5 Redes sem fio

A conectividade oferecida pelos dispositivos móveis à telefonia e Internet em qualquer lugar somente é possível com uso de redes sem fio por rádio frequência. Esses dispositivos são estações móveis de rádio transmissão e recepção que podem comunicar entre si ou com estações fixas. Protocolos de comunicação sem fio como Bluetooth, Wi-Fi e Telefonia Celular utilizam modulação de rádio para receber e transmitir sinais digitais. Os componentes que implementam a recepção e transmissão são comuns, porém, operam em frequências e protocolos diferentes. A seguir, veremos, em mais detalhes, como funcionam esses componentes.

2.5.1 Modulação de sinais digitais

Segundo PIMENTEL (2007), um sistema de comunicação digital lida com a transmissão confiável de um conjunto finito de mensagens por um canal. A confiabilidade é obtida pela aplicação de técnicas de processamento digital de sinais. O canal é meio de natureza analógica, nesse caso, ondas eletromagnéticas, pelo qual o sinal é propagado.

A estrutura básica do sistema de comunicação digital é apresentada na Figura 2.9. O sistema de transmissão é responsável por receber símbolos da fonte e transformar em sinal analógico a ser transmitido pelo canal, essa transformação envolve compressão, criptografia e adição de código corretores de erros, dependendo do tipo do canal e eficiência desejada. O sinal é modulado por portadora para um nível de sinal ruído. O sistema de recepção realiza demodulação e realiza a transformação inversa para recuperar os símbolos originais da fonte e entregar ao destino.

O circuito interno do sistema de comunicação digital é compreendido pela parte digital, responsável por codificar e decodificar os símbolos da fonte em símbolos otimizados para o canal, e pela parte analógica, responsável por modular e transmitir, receber e demodular o sinal analógico.

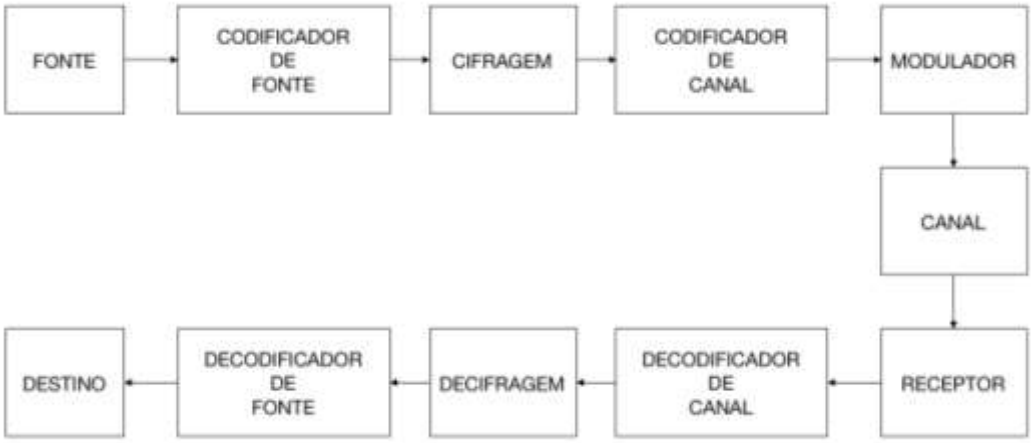

Figura 2.9: Diagrama de blocos de um sistema de comunicação digital (PIMENTEL, 2007).

A parte digital é responsável pelas transformações dos símbolos da fonte e do sinal recebido da antena. Muitas dessas transformações exigem algoritmos complexos, fluxo elevado e constante de bits que são implementados em processadores de propósito específico. O processo de recepção reverte essas transformações. Quanto mais complexas as codificações e decodificações necessárias, maior o consumo.

A parte analógica converte os símbolos a serem transmitidos no canal de sinal digital para analógico, modulados na portadora e amplificados para transmissão. Quanto maior a potência do sinal a ser transmitido maior o consumo de energia.

O que determina se a parte digital ou analógica domina o consumo de energia é a potência de transmissão. Protocolos de rede sem fio de baixo alcance, como Bluetooth, são dominados por sua parte digital, enquanto na telefonia celular, que requer longo alcance, é dominado pela sua parte analógica.

2.5.2 Características comuns de consumo e oportunidades de otimização

A comunicação sem fio é dos maiores consumidores de energia em dispositivos móveis (CARROLL; HEISER, 2010; TARKOMA *et al.*, 2014; XIAO *et al.*, 2014), senão o maior (LI *et al.*, 2014a). Porém, seu consumo é fortemente dependente do tipo de acesso à rede e carga de trabalho, isso é, os padrões de acesso causado pelas aplicações em uso (TARKOMA *et al.*, 2014). Normalmente, estratégias de otimização aplicadas a um tipo de rede não são diretamente aplicáveis a outras, por diferenças nos modos de energia e operação (VALLINA-RODRIGUEZ; CROWCROFT, 2012).

A curva típica de consumo de energia é apresentada na Figura 2.10 e Figura 2.11. Em ambas as figuras os autores utilizaram a corrente de descarga, porém, como a tensão da

bateria apresenta variação mínima, a curva da potência segue o mesmo padrão. Nela se podem identificar três estágios: espera, transmissão e cauda como fica claramente definido na Figura 2.11. O estado de espera é o de menor energia, em que o *hardware* se encontra em descanso. O estado de transmissão é ativado quando existe informação a ser transmitida ou recebida, a mudança entre o estado de espera e transmissão apresenta atraso perceptível que é dependente do *hardware* e do protocolo. Após completar a transmissão e recepção, o *hardware* continua ativo por alguns segundos no estado de cauda. Esse estado permite que nova informação possa ser transmitida ou recebida sem a penalidade de atraso da ativação entre espera e transmissão. O uso intermitente da interface de rede pode prolongar o estado de cauda e causar gastos elevados de energia ao evitar que a rede entre em espera.

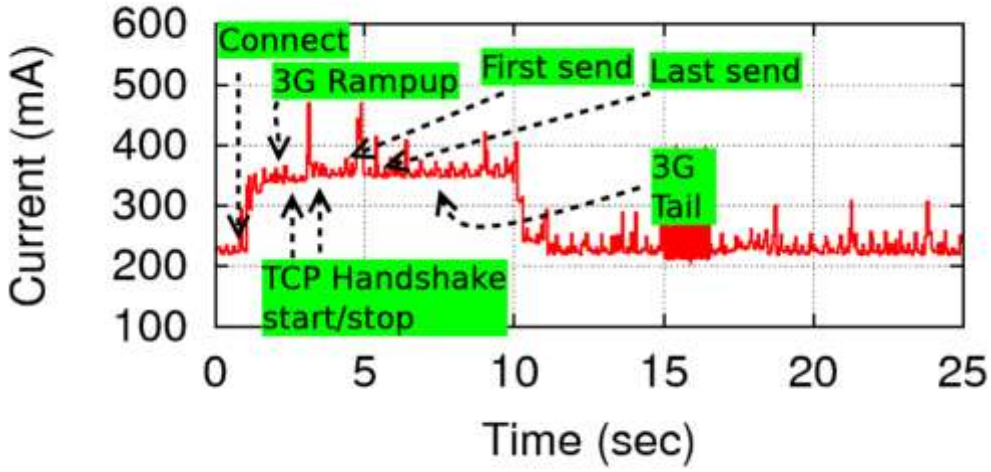

Figura 2.10: Curva típica de energia de uma transmissão sem fio.
No caso uma transmissão curta logo após estabelecer uma conexão TCP por uma rede 3G
(PATHAK; HU; ZHANG, 2012).

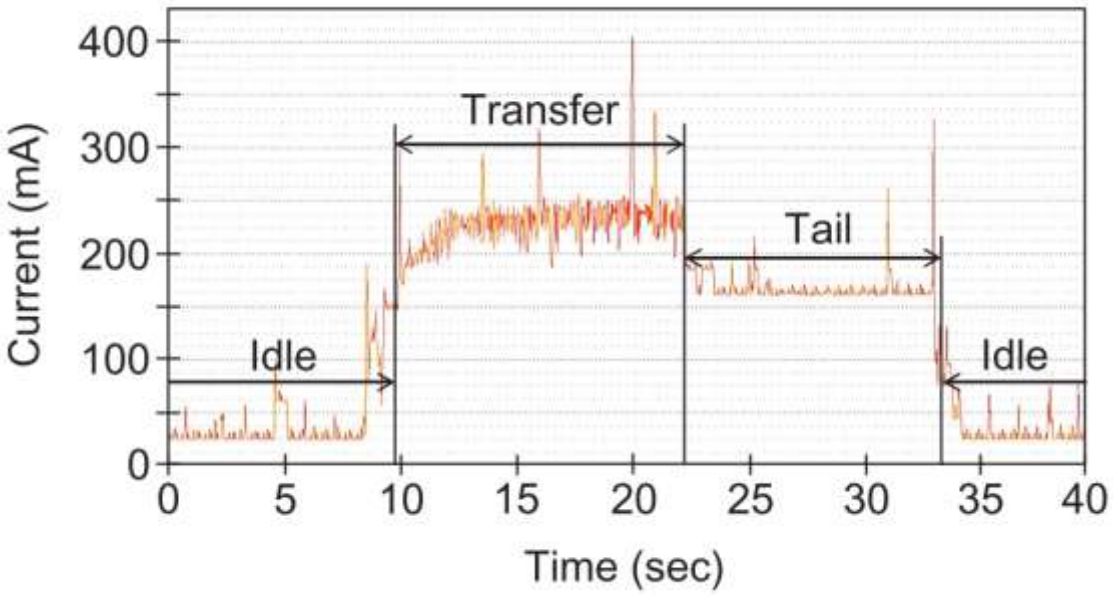

Figura 2.11: Curva de descarga de corrente observado em rede HSPA (LEE *et al.*, 2014).

Um dispositivo móvel contém diversas interfaces de redes sem fio como Bluetooth, Wi-Fi e Telefonia Celular que serão apresentadas a seguir.

2.5.3 Bluetooth

Bluetooth é um protocolo de rede sem fio para rede local pessoal (PAN) de alcance de até 100 metros. É padrão de comunicação sem fio de curta distância, baixo consumo, aberto e livre de royalties promovido pelo *Bluetooth Special Interest* Group (SIG). O protocolo opera na faixa de domínio público dos 2,4 GHz. Esta faixa é compartilhada com uma miríade de outros dispositivos de rádio frequência; para combater a interferência e atingir eficiência energética o Bluetooth emprega *Frequency-Hopping Spread Spectrum* (FHSS). FHSS subdividi o espectro em 79 canais com 1 MHz de separação, o protocolo transmite rajada de pacotes e salta entre os canais a uma taxa de 1600 ou 3200 saltos por segundo, definido por sequência pseudoaleatória acordada entre as partes.

Dispositivos Bluetooth se organizam em redes *ad-hoc* chamadas *piconets* com um nó mestre e até sete nós escravos. *Piconets* podem se agrupar em redes maiores chamadas *scatternets* por meio de algum de seus nós e conectarem a mais de uma *piconet*.

O Bluetooth define um mecanismo embarcado de descoberta de dispositivos próximos e serviços. Ela varre a proximidade com a sinalização e aguarda respostas dos dispositivos, os dispositivos que detectam a sinalização respondem e são encontrados. Localizados os dispositivos próximos, pode-se realizar a inquirição para determinar quais serviços são suportados.

PERRUCI, FITZEK e WIDMER (2011) definiram seis possíveis estados de comunicação e energia: desligado, ligado, "conectado e espera", descoberta, recebendo e transmitindo. Ele observou que, contrário à opinião popular, o aumento de consumo entre o estado de desligado e ligado é mínimo. Porém, quando em transmissão ou recepção, o valor de consumo chega a ser metade do alcançado pela Wi-Fi. O mesmo foi observado por FRIEDMAN, KOGAN e KRIVOLAPOV (2013).

O Bluetooth foi projetado para baixo consumo de energia. Tecnologias como o BLE conseguem limitar ainda mais o consumo. Muito dos trabalhos visam usar o Bluetooth para complementar outras tecnologias como Wi-Fi, que possuem taxas mais altas de transmissão, porém, maior consumo de energia. ANANTHANARAYANAN e STOICA (2009) propuseram utilizar Bluetooth para detectar disponibilidade de conectividade Wi-Fi para evitar o consumo elevado no estado de espera e escaneamento do Wi-Fi. PERING *et al.* (2006) propuseram alternar automaticamente entre múltiplas interfaces de rede como Bluetooth e Wi-Fi para reduzir o consumo de energia. (AGARWAL; SCHURGERS; GUPTA, 2005) propuseram usar Bluetooth como canal de paginação do Wi-Fi, assim prolongar a hibernação da interface Wi-Fi e usar o Bluetooth para sinalizar quando acordá-la. FRIEDMAN, KOGAN e KRIVOLAPOV (2013) realizaram estudo comparativo de energia e desempenho do Bluetooth e Wi-Fi em *smartphones*. Perceberam que, exceto quando o Wi-Fi não se encontra conectado a um ponto de acesso, não existe benefício em usar o Bluetooth.

2.5.4 Wi-Fi (IEEE 802.11)

O Wi-Fi, ou formalmente IEEE 802.11, é especificação e padrão *de facto* para redes locais sem fio (WLAN), publicado inicialmente em 1997. Hoje, Wi-Fi é encontrado em domicílios, pontos comerciais e públicos para acesso de alta velocidade à Internet. O IEEE 802.11 é conjunto de especificações para controle de acesso ao meio (MAC) e camada física (PHY) para implementação de WLAN (TEKTRONIX INC., 2013). A

camada física opera no espectro reservado ao uso industrial, científico e médico de 2,4 GHz ou 5 GHz. O MAC administra e coordena o acesso compartilhado ao canal.

Constantes melhorias na camada física permitiram obter maiores velocidades de conexão e alcance. As especificações da camada física são:

- 802.11 (1997) opera a 2,4 GHz e taxa de até 2 Mbits/s.
- 802.11a (1999) opera a 5 GHz e taxa de até 54 Mbits/s.
- 802.11b (1999) opera a 2,4 GHz e taxa de até 11 Mbits/s.
- 802.11g (2003) opera a 2,4 GHz e taxa de até 54 Mbits/s.
- 802.11n (2009) opera a 2,4 GHz ou 5 GHz e taxa de até 300 Mbits/s.
- 802.11ac (2013) opera a 5 GHz e taxa de até 6,93 Gbits/s.

Melhorias no MAC permitiram melhor uso compartilhado do canal e novas funcionalidades. MAC define duas funções de acesso ao canal: função coordenada distribuída (DCF) e função coordenada pontual (PCF). DCF é obrigatória, define a política de acesso ao canal baseado em CSMA/CA, vetor de alocação a rede (NAV) e atraso aleatório na transmissão, quando o meio está ocupado. A segunda é opcional, aguarda a inquirição da estação base para transmissão. MAC define algumas operações: escaneamento, autenticação, associação, RTS/CTS, modo de economia de energia (PSM), fragmentação, entre outros.

O consumo de energia do Wi-Fi depende do estado de conexão da interface de rede. O estado de menor eficiência energética é quando a interface Wi-Fi busca pontos de acesso para estabelecer conexão (ANANTHANARAYANAN; STOICA, 2009). Quando conectado e não transmitindo ou recebendo, o Wi-Fi possui consumo marginal, dado o uso do modo de economia de energia (PSM) (ANANTHANARAYANAN; STOICA, 2009). Quando transmitindo ou recebendo, o consumo é proporcional à quantidade de informação. Outros fatores que afetam o consumo são periodicidade de inquirição da estação base sobre disponibilidade de novos dados, tempo de descanso profundo da interface e uso do PSM.

PERING *et al.* (2006) propuseram alternar automaticamente entre Bluetooth e Wi-Fi para economizar energia. ANANTHANARAYANAN e STOICA (2009) propuseram usar o Bluetooth para predizer a disponibilidade de redes Wi-Fi e, assim, limitar o escaneamento ao mínimo necessário. KIM *et al.* (2011) propuseram usar sensores de baixo consumo para otimizar o período de busca por estações Wi-Fi. DOGAR e STEENKISTE (2010) propuseram hibernar a interface de rede durante transferências, explorando a diferença de velocidade da rede local e Internet.

2.5.5 Telefonia celular

A telefonia celular permite o acesso sem fio a voz e dados por meio de antenas distribuídas sobre grandes áreas. Cada antena é a célula que cobre uma pequena área, permitindo que transmissores móveis operem a baixa potência, sejam leves e portáteis. Um dispositivo pode alternar de forma transparente entre células, em processo chamado *handover*, sem que a ligação ou transferência de dados sejam interrompidas. Isso é utilizado para permitir que, durante deslocamentos, o dispositivo permaneça sempre conectado à rede.

As redes de telefonia celular foram projetadas originalmente para tráfego de voz, mas, com popularização da Internet na década de 1990, elas foram adaptadas ao tráfego de dados. Hoje, tecnologias como 3G e 4G permitem maior tráfego de dados para crescente demanda dos usuários. Redes de telefonia móvel operam em frequência concedidas pelo governo.

O consumo de energia da interface de rede celular é administrado por uma máquina de estados. Transições para estado de maior energia decorrem da transmissão ou recepção de dados. Transições para estado de menor energia ocorrem após um tempo de inatividade, como foi apresentado na Figura 2.10 e Figura 2.11. Isso implica em alto consumo de energia de cauda após a finalização da transferência de dados (BALASUBRAMANIAN; BALASUBRAMANIAN; VENKATARAMANI, 2009), tornando redes de telefonia móvel menos eficientes que redes Wi-Fi, que não apresentam energia de cauda. Essa característica se agrava em redes 3G, comparadas a redes GSM (VALLINA-RODRIGUEZ; CROWCROFT, 2012), e ainda é pior em redes LTE (HUANG *et al.*, 2012).

BALASUBRAMANIAN, BALASUBRAMANIAN e VENKATARAMANI (2009) propuseram um protocolo chamado *TailEnder* para reduzir a energia cumulativa de cauda. HUANG *et al.* (2012) analisaram as características energéticas de redes 4G, usando uma aplicação chamada 4GTest que atraiu mais de três mil usuários e coletou dados por cerca de dois meses, identificando que redes LTE são até 23 vezes menos eficientes energeticamente que redes Wi-Fi e até menos eficientes que redes 3G, em decorrência da longa energia de cauda.

2.6 Localização

Dispositivos móveis possuem sensores embarcados de localização que permitem determinar com precisão sua posição na superfície terrestre. Esses sensores permitem que aplicações usem a localização do dispositivo para melhorar as recomendações e oferecer novas funcionalidades ao usuário. Os principais sensores utilizados para determinar a localização são por satélite, Wi-Fi e Cell-ID (BARETH, 2012).

O sistema de navegação por satélite ou também conhecido em inglês como *Global Navigation Satellite System* (GNSS) é o sistema mais preciso e utilizado. É composto por várias constelações de satélites como a do GPS, GLONASS, Galileo e Beidou. A partir do sinal de quatro satélites determina a latitude, longitude e altitude na superfície terrestre. Consome muita energia comparado as outras alternativas e sua primeira ativação é demorada. É altamente preciso em locais abertos e zonas rurais.

O Wi-Fi, como sistema de posicionamento, identifica unicamente as estações base pelo seu endereçamento MAC a uma localização na superfície terrestre. Consome menos energia que o GPS e permite boa precisão em cidades e ambientes internos que possuem alta densidade de pontos de acesso. Porém, em zonas rurais onde existem pouco ou nenhum ponto de acesso, não podem determinar a localização. A base de dados precisa ser constantemente atualizada, posto que os pontos de acesso podem se deslocar com a mudança de endereço dos seus proprietários.

O *Cell-ID* é similar ao Wi-Fi, mas usa as antenas de telefonia celular para construir a base de dados geográfica. Como o alcance das células de telefonia celular é muito maior que Wi-Fi, ele apresenta menor precisão. Variando de uma centena de metros em centros urbanos, até quilômetros em zonas rurais. O consumo de energia é o menor dentre as alternativas.

2.7 Sistemas operacionais móveis

Sistemas operacionais móveis são, em sua maioria, baseados nas versões para computador de mesa, por exemplo, Android é baseado no núcleo do Linux, iOS é baseado no núcleo do MAC OS X e Windows Phone no núcleo do Windows NT. Entretanto, devem atender às restrições dos dispositivos móveis como capacidade de telecomunicação, localização e gerenciamento de energia decorrentes do uso de bateria, além da portabilidade necessária para conveniência de uso pelos usuários (TARKOMA *et al.*, 2014).

O Google Android e Apple iOS são os dois principais sistemas operacionais móveis em uso. Segundo IDC (2015), Android e iOS representam 96,3% de todos os *smartphones* produzidos em 2014, 81,5% e 14,8% respectivamente, seguidos por Windows Phone (2,7%), BlackBerry (0,4%) e outros (0,6%). A Figura 2.12 mostra a participação de mercado dos quatro maiores sistemas operacionais móveis entre 2010 e 2014. Nesse período, o Android foi o sistema que mais cresceu, de 23,3% em 2010 para 81,5% em 2014. Android e iOS rapidamente suplantaram outros sistemas como Symbian, BlackBerry e Windows Mobile que dominaram o mercado na década de 2000 trazendo uma série de inovações como telas sensíveis ao toque como principal interface entre usuário e dispositivo, mercado de aplicativos e distribuição por loja virtual, acesso constante à Internet e rico conjunto de sensores como de localização, câmera, Wi-Fi.

Essas inovações agravaram o problema da energia motivando novas pesquisas em área que haviam permanecido quase abandonadas (VALLINA-RODRIGUEZ; CROWCROFT, 2012). Os sistemas operacionais móveis empregam técnicas de gerenciamento de energia para monitorar e regular o consumo de energia dos componentes a fim de prolongar a vida útil até a próxima recarga. Eles expõem interfaces de programação que permitem obter informações, manipular direta ou indiretamente o consumo de energia do sistema. Mais sobre o mecanismo de gerenciamento de energia do Android e iOS nos apêndices B e C.

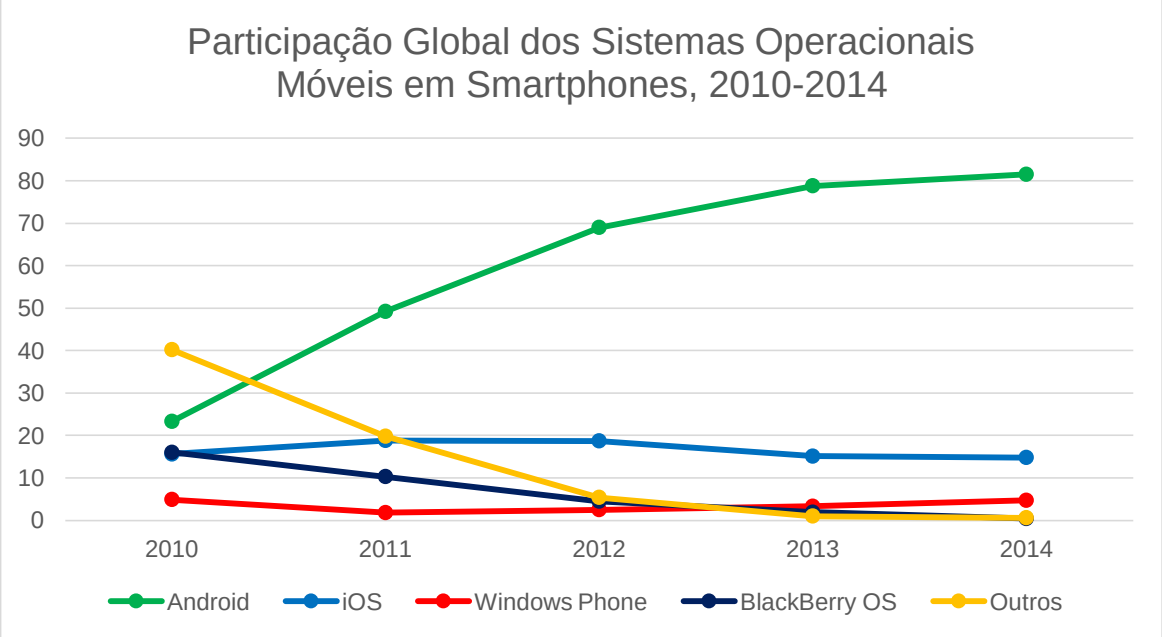

Figura 2.12: Participação global por sistema operacional móvel no mercado de smartphones no período de 2010 à 2014 (IDC, 2015).

2.8 Aplicativos

O rico ecossistema de aplicações é um dos maiores incentivos ao uso de dispositivos móveis. Fabricantes confiam a terceiros a criação de aplicativos e conteúdo para os usuários. Eles provêm arcabouços, ferramentas, documentação, suporte técnico e incentivos para os desenvolvedores criarem aplicações para sua plataforma. Hoje, mais de um milhão de aplicativos estão disponíveis no Google Play (GOOGLE, 2008) e na Apple App Store (APPLE, 2008).

Aplicações têm acesso a uma variedade de componentes para criação de rica experiência. Porém, o uso desses componentes implica em maior consumo de energia e surgimento de uma nova classe de bugs: *energy bugs*. "*Energy bug*, ou *ebug*, é um erro na aplicação, sistema operacional, hardware, firmware ou externo que causa o consumo inesperado de grandes quantidades de energia pelo sistema como um todo" (PATHAK; HU; ZHANG, 2011). Diferente de *bugs* de *software* tradicionais, o *ebug* não costuma afetar a funcionalidade do sistema, exceto pelo consumo inesperado de grandes quantidades de energia (PATHAK; HU; ZHANG, 2011).

Energy bugs são consequência da agressiva política de conservação de energia dos dispositivos móveis que obriga os fabricantes a permitir que desenvolvedores manipulem o estado de energia do sistema (PATHAK *et al.*, 2012). "A política padrão de gerenciamento de energia é que todos os componentes, incluindo a CPU, permaneçam desligados ou em estado de espera, exceto quando uma aplicação explicitamente instrua o sistema operacional para que mantenha ligado" (PATHAK *et al.*, 2012). Isso implica em novo paradigma de programação, similar ao apresentado pela programação paralela. Na programação paralela, existem situações em que é preciso que o acesso à área crítica seja sincronizado entre múltiplos processos. Isso é feito explicitamente pelo programador. Entretanto, pela facilidade de cometer erros de programação, ela ganhou a reputação de ser difícil e complexa (MCKENNY, 2015). Similarmente, existem situações nas quais o dispositivo não pode entrar em estado de menor energia, enquanto determinada tarefa não foi concluída. Porém, enquanto na maioria dos casos é perceptível quando algo de errado ocorre em programa paralelo, um erro de programação no gerenciamento de energia costuma passar despercebido. A maioria das aplicações são desenvolvidas por pequenas equipes, sem esforços dedicados a assegurar a qualidade, portanto, somente quando a aplicação está no mercado e os usuários começam a reclamar do alto consumo de energia é que o problema fica visível. Os usuários postam análises negativas sobre a aplicação que

influencia a decisão de outros usuários. O número de reclamações muito grande pode inviabilizar comercialmente a aplicação (ABOUSALEH *et al.*, 2014).

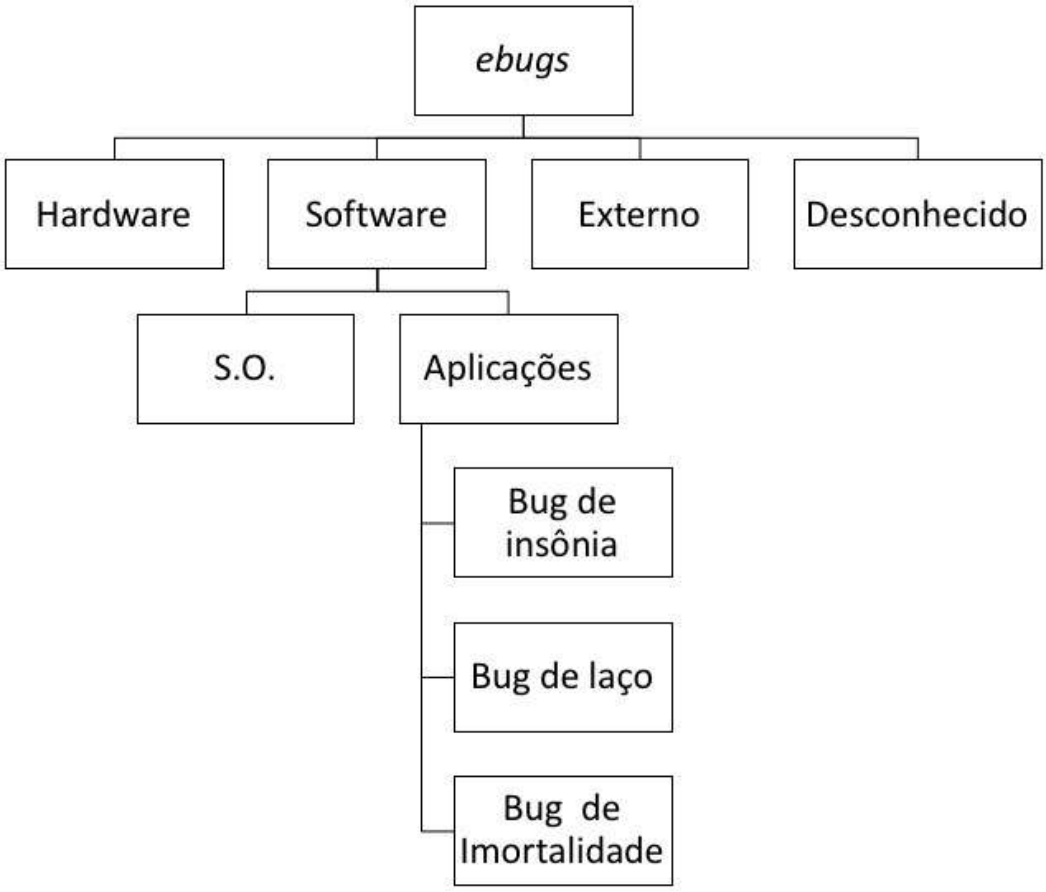

Figura 2.13: Taxonomia de ebugs proposta por (PATHAK; HU; ZHANG, 2011).

Uma taxonomia foi proposta para ebug por (PATHAK; HU; ZHANG, 2011) e é apresentada na Figura 2.13. Os ebugs de aplicação são divididos em três grupos principais: insônia, laço e imortalidade. Os bugs de insônia são os mais frequentes. Neles, a aplicação impede, incorretamente, um componente do dispositivo de dormir. Ocorre quando a aplicação não libera o componente mesmo após ele ter concluído sua atividade. Os bugs de laço ocorrem quando a aplicação realiza atividade periódica desnecessária que consome muita energia. Os bugs de imortalidade ocorrem quando a aplicação, mesmo após explicitamente terminada pelo usuário, é reaberta automaticamente pelo sistema para continuar consumindo muita energia erroneamente.

Segundo LI *et al.* (2014a), desenvolvedores carecem de informações quantitativas e objetivas sobre o comportamento das aplicações a respeito do consumo de energia. O autor identificou algumas importantes conclusões em seu estudo sobre consumo de energia no código com 405 aplicações comerciais para plataforma Android. Primeiro, a maioria

das aplicações consomem 60% da energia em estado de espera, portanto, apenas otimizar o código é insuficiente para reduzir o consumo geral da aplicação. Segundo, a interface de rede é a componente que mais consome energia na maioria das aplicações, em particular requisições HTTP, que chegam a representar 89% da energia consumida pela interface de rede de 75% das aplicações estudadas. Terceiro, o consumo de energia é dominado pelas interfaces de programação do sistema, portanto, desenvolvedores devem otimizar o uso delas. Quarto, apenas pequeno número de interfaces de programação do sistema é responsável pelo consumo significativo de energia. Portanto, desenvolvedores podem limitar a otimização de uso a esse pequeno conjunto. Quinto, instruções em laço são responsáveis por 40% da energia consumida pelo aplicativo quando ativo. Por último, operações de manipulação são as que mais consomem energia em nível de *byte-code*.

PATHAK, HU e ZHANG (2012) observaram que 65% a 75% da energia em aplicativos gratuitos é originada de módulos de propaganda de terceiros usados para monetização. Apenas pequena fração da energia é consumida pela funcionalidade proposta. A maior parte da energia é consumida em operações de entrada e saída em componentes como 3G, Wi-Fi e GPS. Esses componentes apresentam um desafio único, pois o consumo apresenta comportamento assíncrono que persiste muito depois de a operação ter concluído assim como foi mostrado nas seções 2.5 e 2.6.

PROCHKOVA, SINGH e NURMINEN (2012) estudaram o impacto da propaganda em jogos gratuitos para plataforma Android. Observaram que o intervalo de atualização dos anúncios é importante fator no consumo de energia, quanto maior a frequência de atualização maior o consumo. Sugerem que aplicações que não requerem essa conexão constante à Internet devem preferir intervalos maiores.

LIU *et al.* (2014) estudaram as causas da ineficiência energética em aplicações Android. Analisaram 173 aplicações de código aberto e 229 aplicações comerciais e descobriram duas causas principais: a não liberação do sensor ou dispositivo necessário para que ele entre em estado de menor energia e custo ineficaz dos dados obtidos pelo sensor.

3 MODELO DE CONSUMO DE ENERGIA

O modelo de consumo de energia é necessário para entender como os diferentes componentes de *hardware* e *software* interagem ao consumir. O modelo de energia é a representação matemática que estima o dispêndio de energia do dispositivo no tempo pelo estado interno de suas componentes. A construção desse modelo não é tarefa trivial, dado que nem todos os componentes se comportam de maneira linear, postas as relações de dependência entre eles que não são observáveis quando usados individualmente.

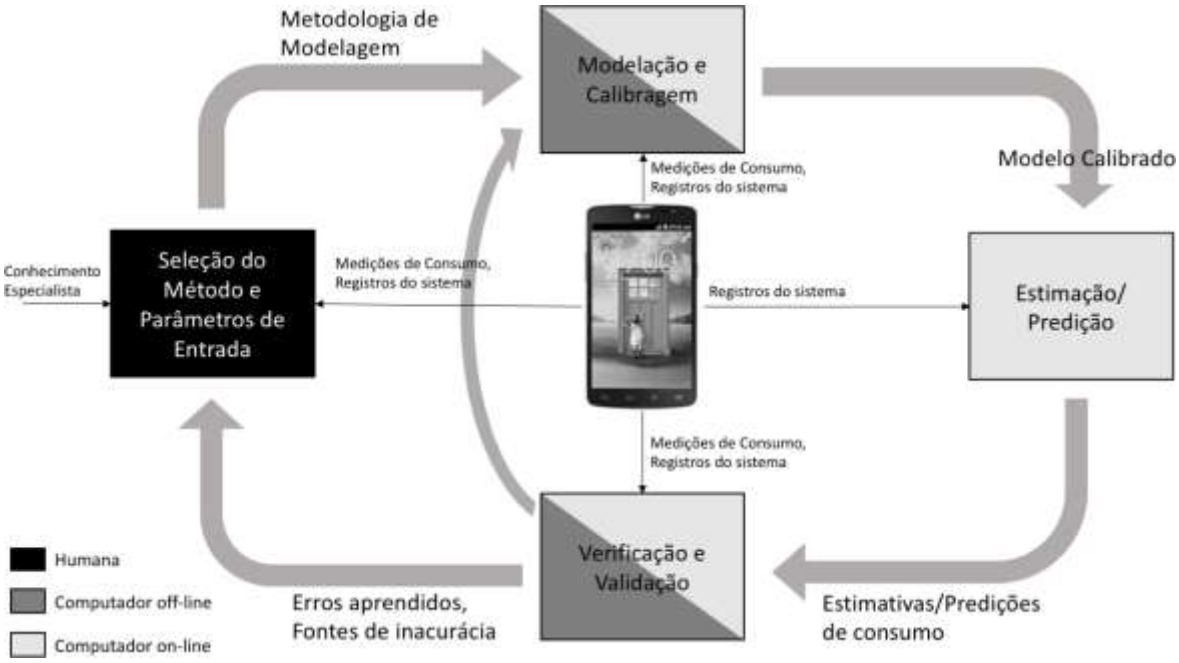

Figura 3.1: Procedimento de construção do modelo.

Segundo TARKOMA *et al.* (2014), o processo de construção do modelo de consumo de energia é interativo, uma sequência de tarefas em quatro etapas, podendo ser realizado por uma pessoa ou computador, como apresentado na Figura 3.1. O computador pode analisar os dados *on-line* ou *off-line*. Na análise *off-line*, os registros são previamente coletados. Na análise *on-line*, os registros são recebidos diretamente sem armazenamento prévio.

As quatro etapas na construção e validação do modelo são:

1. Seleção de método e parâmetros de entrada: um especialista define os parâmetros a serem observados, as ferramentas, os experimentos, as configurações e as técnicas que serão empregadas para construção do modelo.
2. Modelagem e calibração: fazem-se medições e experimentos com finalidade de identificar a correlação entre os diversos estados do sistema e o consumo.

3. Estimação ou predição: usa-se o modelo para gerar estimativas sobre o consumo do aparelho.
4. Verificação e validação: compara-se a estimativa aos dados empíricos de consumo, a fim de determinar a margem de erro e possíveis fontes de incerteza. Os resultados podem ser usados na próxima iteração, para aprimorar o modelo.

3.1 Perfil de consumo de energia

Segundo TARKOMA *et al.* (2014), a ferramenta de perfil de energia é o componente de *software* ou *hardware* que monitora e caracteriza o consumo de energia do dispositivo. A medição direta do consumo de energia por sensores costuma ser limitada ou indisponível. Para contornar o problema, a caracterização do consumo de energia é auxiliada pelo modelo de consumo de energia que descreve como um ou mais subsistemas de *hardware* e *software* interagem. Assim, estima-se o consumo das componentes de *hardware* e *software*. O modelo pode ser produzido em laboratório ou dinamicamente, durante o uso do aparelho.

Perfis de energia podem ser classificado por diferentes atributos, como o local de operação, o método de medição de consumo, método de atribuição de consumo, a granularidade da atribuição, o estado do sistema, entre outros.

Quanto ao local da operação, o perfil pode ser *on-line* ou *off-line*. O perfil *on-line* é produzido no próprio aparelho. Enquanto o perfil *off-line* é produzido em computador, usando as informações registradas em arquivo ou de equipamentos externos.

Quanto à medição, o perfil de energia pode ser classificado como interno ou externo. O interno usa sensores do próprio aparelho, como os da unidade de gerenciamento da bateria expostos por interfaces de programação para medir o consumo de energia. Porém, sensores internos apresentam limitações de precisão, taxa de atualização e necessitam de *software* embarcado que adiciona carga ao sistema e impacta no consumo registrado. O externo usa equipamento dedicado que fica ligado aos terminais da bateria ou a substitui por um simulador, permitindo registrar com alta taxa de amostragem e precisão os valores de tensão e corrente. Entretanto, o processo requer modificações físicas do aparelho que impedem sua mobilidade.

Um dos simuladores de bateria mais usados para medição externa é o Monsoon Power Monitor (MONSOON SOLUTIONS, 2015), apresentado na Figura 3.2. É ferramenta externa para análise de consumo de dispositivos móveis movido a bateria, ela

alimenta e registra o consumo do dispositivo a taxas de 300 kHz e resolução de 286 μA. Diversas publicações como ZHANG *et al* (2010),. XU *et al.* (2014) , LI e HALFOND (2014) referem-se aos autores terem feito uso dessa ferramenta.

Figura 3.2: Monsoon Power Monitor conectado a um celular
(CUERVO; BALASUBRAMANIAN, 2010).

Uma alternativa para resolver o problema de mobilidade decorrente da medição externa do consumo é a construção de módulos embarcados que fiquem diretamente ligados ao aparelho. SCHULMAN *et al.* (2011) utilizaram essa abordagem para criar um monitor portátil chamado BattOr. BROUWERS, ZUNIGA e LANGENDOEN (2014) projetaram uma ferramenta chamada NEAT para monitoramento externo e portátil de consumo que não interfere na portabilidade do aparelho. O BattOr e NEAT são apresentados na Figura 3.3.

(a) BattOr *(b) NEAT*

Figura 3.3: Ferramenta de monitoramento externo portátil.

A estimação do consumo por *software* é a técnica menos invasiva usada por pesquisadores e desenvolvedores para observar o consumo. Além da facilidade de configuração, instalação e uso, essa técnica permite aos pesquisadores recrutar voluntários distribuídos pelo mundo para levantar dados e identificar padrões de uso. Algumas das pesquisas que fizeram uso desse paradigma foram as de OLINER *et al*. (2013), BÖHMER *et al*. (2011), OLIVER e KESHAV (2010) e WANG *et al*. (2013). Entretanto, a medição por *software* interfere no consumo de energia. A fim de reduzir esse impacto, o *software* opera a baixa taxas de amostragem e prioridade de execução.

ALAWNAH e SAGAHYROON (2013) desenvolveram uma ferramenta de *log* para registrar o consumo e as atividades do sistema que seria usado na geração de modelos de consumo usando técnicas de regressão linear. KIM, KONG e CHUNG (2012a) desenvolveram uma técnica avançada de estimação de consumo baseada em técnicas de regressão linear e captura por *software* de consumo.

Uma solução para reduzir a energia desse *software* é embarcá-lo no próprio sistema operacional. Isso permite maior controle, acesso às interfaces e precisão nas medições. Porém, requer modificações no sistema operacional. PATHAK *et al.* (2011) utilizaram essa abordagem para criar modelos de consumo baseados em chamadas de sistema para Windows Mobile e Android que requerem modificações no núcleo. YOON *et al.* (2012) desenvolveram o *AppScope* com essa mesma premissa. Ele é um módulo do núcleo que monitora eventos e estima o consumo. JUNG, KIM e CHA (2013) construíram o *UserScope*, um módulo do núcleo que captura eventos com baixo custo de operação.

O modo de atribuição do consumo pode ser classificado como por tempo, por uso ou por máquina de estado finito. Atribuição por tempo particiona o tempo em segmentos e atribui o consumo aos componentes em operação capturados durante cada segmento. A atribuição por uso utiliza contadores e um modelo de consumo para estimar o consumo de energia de cada componente, esse é o modelo usado pelo Android para monitorar a duração da bateria (TARKOMA *et al.*, 2014). A atribuição por máquina de estado finito possui modelo detalhado de funcionamento do sistema baseado em eventos, estados e transições. Ele monitora mudanças no sistema usando, por exemplo, chamadas de sistema para estimar o consumo de energia de cada componente e do sistema em geral.

Quanto à granularidade da atribuição, o perfil de energia pode ser classificado como geral, por componente, por processo, por *thread*, por instrução, entre outras. Na granularidade geral é avaliado o consumo total do sistema sem diferenciação. Na

granularidade por componente os elementos principais de consumo como processador, Wi-Fi, rádio celular são destacados. Na granularidade por processo, o consumo de cada aplicação em execução e seu histórico de carga são destacados. Na granularidade por *thread* o consumo de cada *thread* é identificado. Na por instrução, o custo de cada instrução, e assim por diante.

Quanto ao estado do sistema, o perfil de energia pode ser classificado como implícito ou explicito. No implícito, não é óbvio quem é responsável por qual parte do consumo; sendo necessário estimar, isso ocorre na atribuição por tempo, por exemplo. No explicito, a contribuição é diretamente determinada, isso ocorre, por exemplo, na atribuição por uso. O consumo de cada componente é claro, o consumo geral do sistema é a soma do consumo de cada componente.

Na Tabela 3-1 são listadas alguma das ferramentas de perfil de energia encontradas na literatura.

Tabela 3-1: Ferramentas de perfil de energia encontradas na literatura.

Nome/Autor	**Ano**	**Propósito**
PowerScope (FLINN; SATYANARAYANAN, 1999)	1999	Perfil de energia de dispositivos e processos.
Joule Watcher (BELLOSA, 2000)	2000	Perfil de consumo em *thread*.
Nokia Energy Profiler (CREUS; KUULUSA, 2007)	2006	Registro e analisador de consumo para celulares Nokia.
SHYE, *SCHOLBROCK e MEMIK*.(2009)	2009	Perfil de energia e aplicação de log.
PowerBooter (ZHANG *et al.*, 2010)	2009	Modelo de consumo.
PowerTutor (ZHANG *et al.*, 2010)	2009	Ferramenta de profile híbrido baseado no PowerBooter.
SEMO (DING *et al.*, 2011)	2011	Ferramenta de monitoramento de energia.
Sesame (DONG; ZHONG, 2011)	2011	Modelo autoconstruído de consumo.
PowerProf (KJÆRGAARD; BLUNCK, 2012)	2011	Modelo autoconstruído de consumo na aplicação.
MobiBug (AGARWAL *et al.*, 2010)	2011	Ferramenta de diagnóstico automático de *crashes*.
Carat (OLINER *et al.*, 2013)	2012	Perfil de energia e depuração colaborativo.
eProf (PATHAK; HU; ZHANG, 2012)	2012	Modelo preciso de consumo para dispositivos, componentes e aplicações.
DevScope (JUNG *et al.*, 2012)	2012	Modelo autoconstruído para dispositivos e componentes.

AppScope (YOON *et al.*, 2012)	2012	Perfil de energia de aplicações baseado no DevScope.
e-Doctor (MA *et al.*, 2013)	2012	Ferramenta de diagnóstico automático de problema de consumo de bateria.
V-edge (XU *et al.*, 2013)	2013	Modelo autoconstruído para dispositivos e componentes.

3.2 Técnicas de modelagem de consumo

A construção de modelos utiliza métodos sistemáticos desenvolvidos na computação e matemática para determinar relações de causalidade entre as entradas, as saídas e os estados do sistema para estimar o consumo de energia. Algumas das principais técnicas usadas são apresentadas a seguir.

3.2.1 Modelagem por análise de regressão

Um dos métodos mais usados para determinar a relação entre as entradas e as saídas é por análise por regressão. A regressão busca ajustar a função de distribuição às entradas e saídas observadas para o conjunto de dados. Modelos de regressão podem ser classificados pelo número de variáveis e linearidade, conforme exemplificado na Figura 3.4.

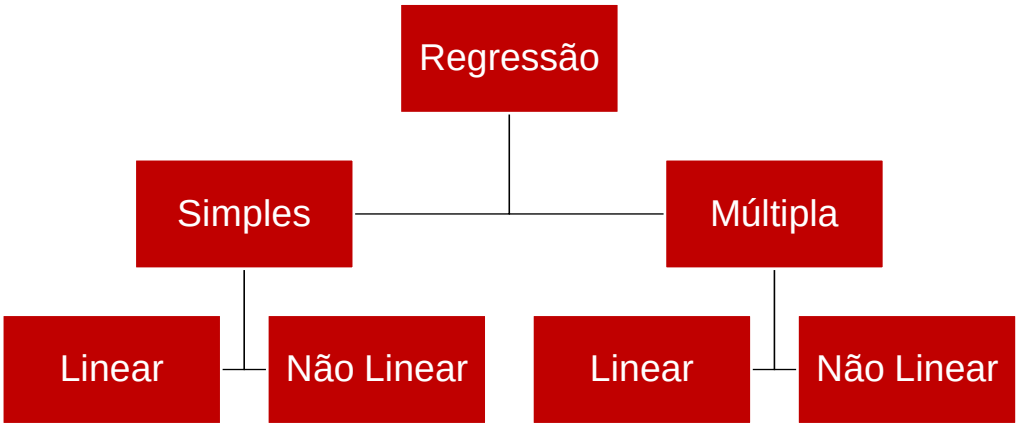

Figura 3.4: Classificação dos modelos de regressão.

O modelo é simples quando possui variável única independente. Caso contrário, ele é múltiplo, com duas ou mais variáveis independentes.

O modelo é linear quando a função de ajuste é linear do tipo apresentado na Equação (*3-1*) (ROUSSEEUW; LEROY, 1987). Nela n representa o número de amostras. As variáveis x_{i1} até x_{ip} são as variáveis explanatórias. A variável y_i é a variável de

resposta. O valor e_p representa o termo de erro, assume-se que o termo é normalmente distribuído, isso é, sua média é zero e o desvio padrão desconhecido. O modelo não linear é aproximado por equações não lineares, por exemplo, exponenciais, logarítmicas.

$$y_i = x_{i1}\theta_1 + \cdots + x_{ip}\theta_p + e_p; para\ i = 1, \ldots, n \quad (3\text{-}1)$$

Um exemplo de regressão linear é apresentado na Figura 3.5. O conjunto de amostras foi aproximado pela Equação (*3-4*) que é linear. Os coeficientes α e β foram obtidos pela função *Fit* da ferramenta Gnuplot (MERRITT, 2007) que usa o método de Lavenberg-Marquardt.

$$y(x) = \alpha x + \beta \quad (3\text{-}2)$$

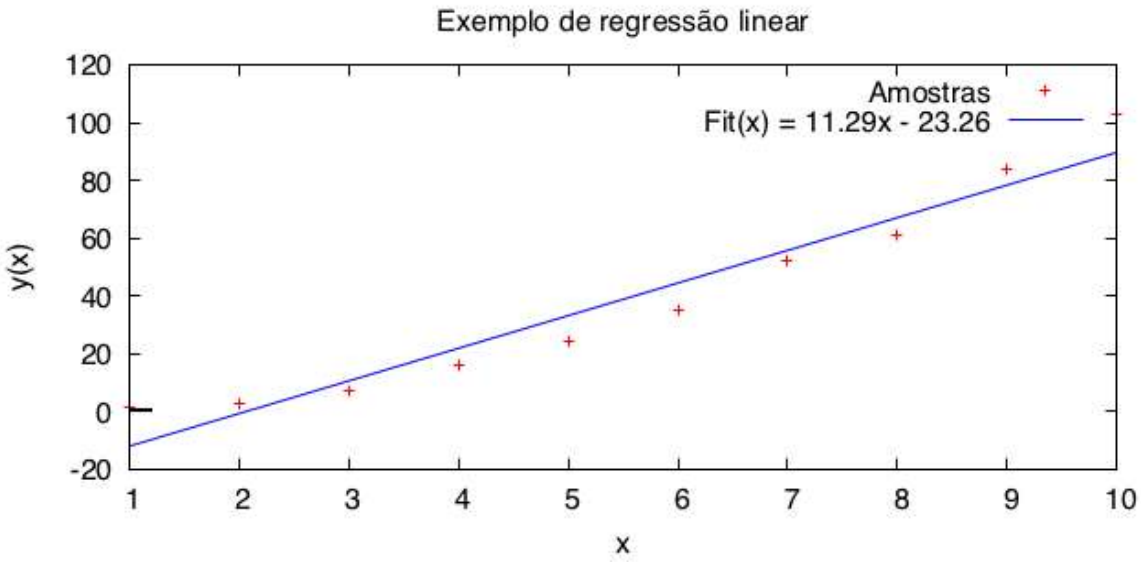

Figura 3.5: Exemplo de regressão linear.

No caso da regressão não linear, para o mesmo conjunto de dados, a função de aproximação foi função cúbica, como apresentado na Equação (*3-3*) e gráfico apresentado na Figura 3.6. Os coeficientes α e β foram obtidos pela função *Fit* do Gnuplot (MERRITT, 2007). Neste exemplo, a equação não linear se ajusta melhor ao conjunto de amostras apresentado. Entretanto a melhor aproximação varia para cada conjunto de dados.

$$y(x) = \alpha x^3 + \beta \quad (3\text{-}3)$$

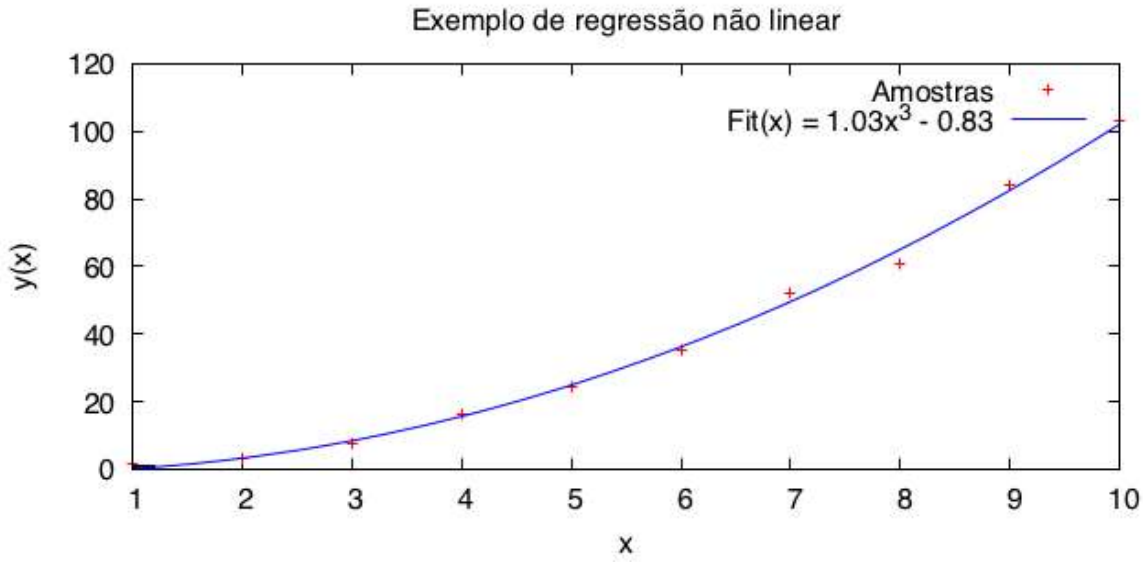

Figura 3.6: Exemplo de regressão não linear.

O método de regressão é amplamente suportado em ferramentas de análise estatísticas como *Matlab* (MATHWORKS, 2016), R(FARAWAY, 2002) e *Octave* (GNU, [s.d.]).

3.2.2 Modelagem por máquina de estados

O comportamento de vários dos componentes do sistema é melhor descrito em representação por máquina de estados finito, como é o caso das interfaces de rede como Wi-Fi e 3G. A máquina de estado é modelo de computação que consiste em conjunto de estados, um estado inicial, um alfabeto de entradas e uma função de transição que mapeia entradas e o estado atual para o próximo estado (NIST, [s.d.]).

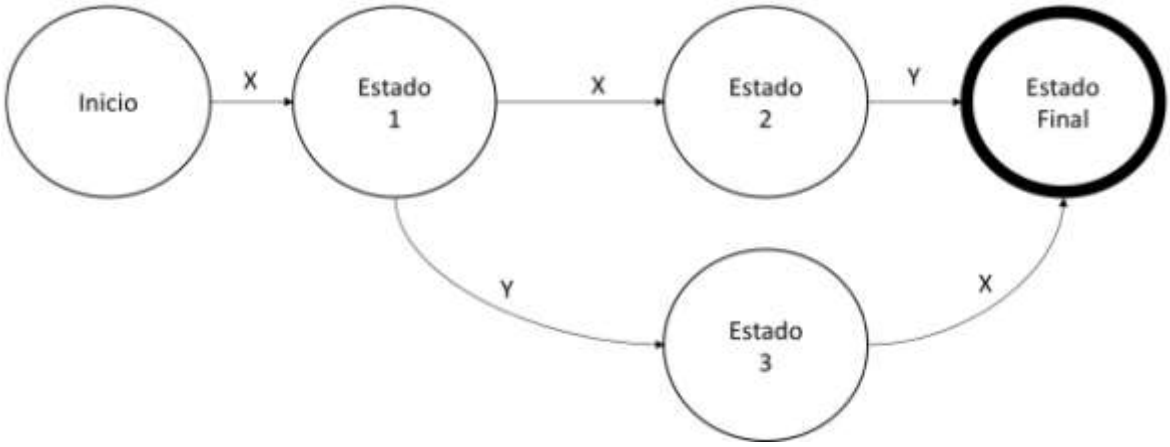

Figura 3.7: Exemplo simples de máquina de estados finito.

A Figura 3.7 apresenta a representação simples de a máquina de estados finito. No exemplo, uma máquina simples de cinco estados é apresentada. Transições entre os estados são determinada pelo símbolo de entrada e o estado atual. Por exemplo, se o estado atual for o estado 1 e o símbolo de entrada recebido X, o próximo estado será o estado 2. Caso o símbolo de entrada recebido seja Y, o próximo estado será o estado 3. Máquinas de estados finito permitem identificar todos os possíveis estados válidos do sistema e sequências de símbolos válidos. Por exemplo, as sequências válidas de símbolos para a máquina apresentada na Figura 3.7 são XXY e XYX. Qualquer outra sequência é ilegal e não atinge o estado final do sistema. Os símbolos de entrada também são conhecidos por eventos. Os estados também podem ser chamados de nós.

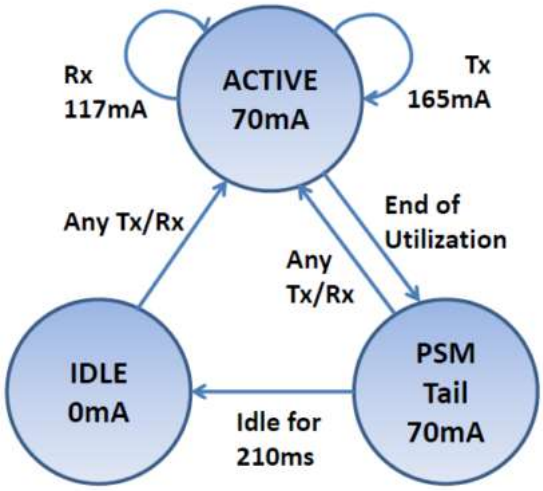

Figura 3.8: Exemplo de modelagem de consumo por máquina de estado
(DING *et al.*, 2013).

Na modelagem por máquina de estados, cada estado possui uma característica de consumo diferente que pode ser estimada aplicando regressão linear. Eventos como transmissão e recebimento de pacotes, término do tempo de espera e interações do usuário provocam transição entre os estados da máquina virtual como exemplificado na Figura 3.8.

3.2.3 Outros métodos de modelagem

Técnicas de aprendizagem de máquina como algoritmos genéticos, redes neurais, e etc. Além de métodos estatísticos como redes Bayesianas e cadeia de Markov podem ser aplicados para modelar o consumo do sistema.

KJÆRGAARD e BLUNCK (2012) construíram o PowerProf, ferramenta de perfil de energia que aplica aprendizado não supervisionado baseado em algoritmos genéticos para construção de modelos de consumo de energia.

ALAWNAH e SAGAHYROON (2013) aplicaram redes neurais com regressão linear para construção de modelos de consumo de energia.

LEE e CHO (2012) produziram um modelo probabilístico baseado em redes bayesianas para gerenciamento e redução do consumo de energia.

3.3 Modelo de consumo do processador

Na seção 2.3.1 foram apresentadas as características de consumo de energia do processador. Nesta seção veremos alguns modelos encontrados na literatura para estimar seu consumo.

3.3.1 Processador de núcleo único

O modelo mais simples de consumo do processador é a de um processador de um núcleo com frequência de operação fixa. O modelo é definido na Equação (*3-4*) (TARKOMA *et al.*, 2014). A potência do processador P_{cpu} é diretamente proporcional a carga U_{cpu}. Os coeficientes α e β são obtidos a partir de técnicas de regressão linear.

$$P_{CPU} = \alpha U_{CPU} + \beta \tag{3-4}$$

3.3.2 Processador de núcleo único com DVFS

O modelo anterior precisa ser estendido para suportar DVFS. DVFS é uma técnica empregada para redução do consumo de energia por meio da redução da frequência e voltagem de operação. Diferente da Equação (*3-4*), na Equação (*3-5*) os coeficientes são dependentes da frequência f de operação do processador.

$$P(f)_{CPU} = \alpha(f) U_{CPU} + \beta(f) \tag{3-5}$$

3.3.3 Processador de múltiplos núcleos com DVFS e CPU Hot-Plugging

Processadores modernos como os utilizados em dispositivos móveis aplicam uma série de otimizações de energia como DVFS e Hot-Plugging para reduzir o consumo.

CARROLL e HEISER (2014) investigaram a interação entre o DVFS e *CPU Hot-Plugging* e propuseram nova política de escalonamento que leva em conta a interação entre esses dois mecanismos. Observaram que a componente estática de consumo, discutida na seção 2.3, varia bastante entre os diversos fabricantes de processador ARM. Definiram um

modelo simplificado de processadores de múltiplos núcleos, onde cada processador pode estar num de três estados: ativo, espera ou *off-line*. O estado ativo é quando o processador está trabalhando, decodificando instruções e as executando. O estado de espera representa os vários *C-states* quando o processador suspende vários de seus componentes para conservar energia. O estado *off-line* é quando o núcleo é completamente desligado, entretanto, entrar ou sair desse estado é caro.

O consumo do processador multinúcleos para n-núcleos a frequência f é definido pela Equação (*3-6*).

$$P_{cpu} = P_{uncore} + n(P_{dynamic} + P_{static}) \tag{3-6}$$

$$P_{dynamic} = C_{eff} V^2 f \tag{3-7}$$

Onde P_{static} representa o consumo quando ativo, porém em espera, é independe da carga de trabalho, mas varia em função da voltagem de operação. O $P_{dynamic}$ é o consumo adicional do processador quando ativo e é em função da carga de trabalho, frequência e voltagem de operação. O P_{uncore} é a potência restante que independe da quantidade de núcleos ativos, necessário enquanto tiver pelo menos um núcleo ativo. O $P_{dynamic}$ é um valor conhecido apresentado na Equação (*3-7*), onde C_{eff} é a capacitância efetiva, V a voltagem de operação e f a frequência de operação.

ZHANG *et al.* (2013) (2015) estudaram o consumo de processadores de múltiplos núcleos em dispositivos móveis e descobriram que os modelos atuais de consumo não representam corretamente o comportamento de consumo. Eles propuseram novo modelo que leva em conta os *C-states*, a frequência e carga de uso do processador. Primeiro, é estimado o consumo individual de cada núcleo, conforme apresentado na Equação (*3-8*).

$$P_{Core} = \sum_{i}^{N} \beta_{C_i} \times WED_{C_i} + \beta_U \times U + c \tag{3-8}$$

Onde WED_{C_i} representa a duração média de transição para cada *C-state*, β_{C_i} e β_u são coeficientes de WED_{C_i}, U é a taxa de uso e c é uma constante (TARKOMA *et al.*, 2014). Esses coeficientes são determinados pela regressão linear a partir dos resultados da suíte de treinamento.

O consumo de um núcleo único é estendido para múltiplos núcleos aplicando a Equação (*3-9*). Onde N_c é o número de núcleos ativos, P_{BL,N_c} é o consumo de base do

processador para N_c núcleos ativos, e $P_{\Delta,core,U_i,f_i}$ é o consumo adicional do processador a frequência f_i e taxa de utilização U_i. Para cada frequência f_i, $P_{\Delta,core,U_i,f_i}$ é estimado usando o modelo de consumo de núcleo único apresentado na Equação (*3-8*). P_{BL,N_c} é constante que pode ser medida previamente.

$$P_{cpu} = P_{BL,N_c} + \sum_{i}^{N_c} P_{\Delta,core,U_i,f_i} \tag{3-9}$$

3.4 Modelo de consumo da tela LCD

Na seção 2.4.1, foram apresentadas as características das telas LCD. A tela LCD tem o consumo predominantemente determinado pelo nível de brilho do *backlight*. O consumo da tela é aproximado na maioria das publicações analisadas por regressão linear na forma da equação de reta, como apresentado na (*3-10*). Entretanto, como observado por CARROLL e HEISER (2010), o conteúdo apresentado tem efeito no consumo geral da tela e este nem sempre é linear.

$$P_{lcd\_on} = \alpha x_{brilho} + \beta \tag{3-10}$$

A potência da tela ligada P_{lcd_on} é proporcional a intensidade do brilho x_{brilho}. As constantes α e β são as determinadas por regressão linear. O valor α representa o consumo adicional por unidade de brilho, x_{brilho} representa um valor inteiro de 0 a 255. β o consumo mínimo da tela ligada. Esse modelo foi usado por vários autores (SHYE; SCHOLBROCK; MEMIK, 2009), (XIAO *et al.*, 2010), (ZHANG *et al.*, 2010), (JUNG *et al.*, 2012), (YOON *et al.*, 2012) e (KAMIYAMA; INAMURA; OHTA, 2014).

FALAKI, GOVINDAN e ESTRIN (2009) propuseram outro modelo de consumo de energia baseado em máquina de estados finito para telas LCD e um escalonador otimizado para elas. Eles observaram que existe custo na transição entre os estados que deve ser considerado.

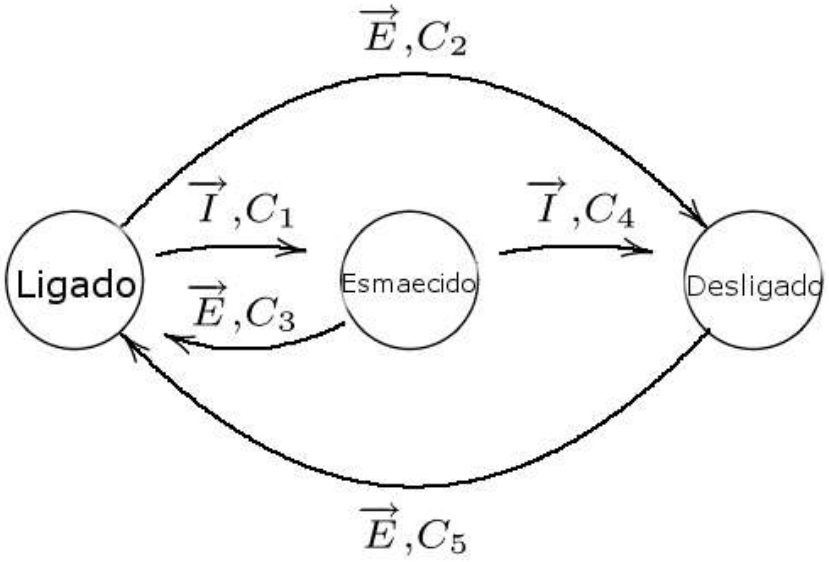

Figura 3.9: Máquina de estados finitos da tela LCD.

A Figura 3.9 apresenta a máquina de estados finitos proposta. Os círculos representam os estados. As setas as transições. $\vec{E}$ indica um evento, por exemplo, o desbloqueio da tela pelo usuário e $\vec{I}$ indica o término do período de espera. Cada transição possui um Custo (C) associado. A mesma informação é apresentada na Tabela 3-2. Nela é possível ver que existe custo de execução associado a cada estado.

Tabela 3-2: Lista de possíveis transições de tela.

Estado Atual	**Custo de Execução**	**Próximo Estado**	**Custo de Transição**
Ligado	R_1	Esmaecido Desligado	C_1 C_2
Esmaecido	R_2	Ligado Desligado	C_3 C_4
Desligado	R_3	Ligado Esmaecido	C_5 C_6

KIM, KONG e CHUNG (2012b), com objetivo de acomodar a não linearidade das telas LCD, optaram por particionar o modelo em segmentos com coeficientes de inclinação da reta diferentes para cada faixa de brilho. Por exemplo, o coeficiente α da Equação (*3-10*) pode ser determinado conforme a Equação (*3-11*).

$$(x_{brilho}) = \begin{cases} \alpha_1, se\ 0 \leq x_{brilho} < 64 \\ \alpha_2, se\ 64 \leq\ x_{brilho} < 128 \\ \alpha_3, se\ 128 \leq\ x_{brilho} < 192 \\ \alpha_4, se\ x \geq 192 \end{cases} \quad (3\text{-}11)$$

3.5 Modelo de consumo para redes Wi-Fi

Na seção 2.5.4 foram apresentadas brevemente as características de energia das redes Wi-Fi. A interface Wi-Fi opera em quatro estados principais (FRIEDMAN; KOGAN; KRIVOLAPOV, 2013): desligado, buscando, conectado a um ponto de acesso ou conectado *ad-hoc*. A busca por pontos de acesso é a operação de maior consumo (ANANTHANARAYANAN; STOICA, 2009). Trabalhos como o de KIM *et al.* (2011) e ANANTHANARAYANAN e STOICA (2009) propuseram o uso de outros sensores como o Bluetooth para reduzir o número de buscas. O modo *ad-hoc* é usado para conectar dispositivos sem necessidade de ponto de acesso, uma forma de conexão pouco difundida; não é suportada em dispositivos Android (FRIEDMAN; KOGAN; KRIVOLAPOV, 2013). Segundo FRIEDMAN, KOGAN e KRIVOLAPOV (2013), em testes usando Windows Mobile, o consumo de energia é muito maior em modo de espera que quando usando ponto de acesso, devido à ausência de política de economia de energia.

O modo mais usado de conexão Wi-Fi é por ponto de acesso. O ponto de acesso é um nó central responsável por receber e transmitir os pacotes entre os dispositivos. O Wi-Fi possui definido um modo de economia de energia chamado PSM. O Wi-Fi é capaz de economizar até 90% da energia durante transmissões em rajada quando PSM está habilitado (ANASTASI *et al.*, 2008). Essa economia é possível pela ativação da interface somente durante a troca de dados, sempre que a interface em espera é posta para hibernar.

Para utilizar o PSM, a interface precisa informar o ponto de acesso que o suporta. O ponto de acesso passa a armazenar os quadros recebidos em memória e a notificar a disponibilidade deles pelo quadro de *Beacon*, um tipo de quadro periódico de controle. No *Beacon* fica armazenado um mapa de indicação de tráfego (TIM) que indica aos dispositivos a disponibilidade de novos dados. O dispositivo acorda e envia um frame de controle chamado *ps-poll* para indicar que está pronto para receber os dados. Cada pacote enviado pelo ponto de acesso possui um indicador da existência de mais dados. O dispositivo repete o procedimento enviando outro *ps-poll* até que todos os dados tenham sido transferidos.

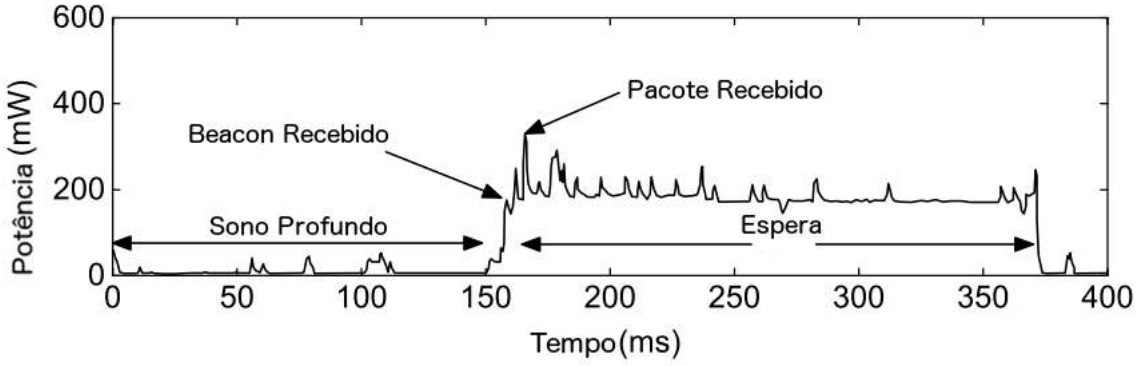

Figura 3.10: Potência observada para o recebimento de um pacote (LI *et al.*, 2014b).

A Figura 3.10 mostra o consumo do Wi-Fi com PSM habilitado. A interface fica em estado de sono profundo até a notificação do *Beacon* da disponibilidade de novo pacote. A interface desperta para receber o pacote. Após recepção ser concluída, existe um período de espera antes de o dispositivo entrar novamente em sono profundo.

Na literatura, a abordagem mais comum para modelar o consumo da interface Wi-Fi é por máquina de estados combinada a regressão linear para cada estado. O método de consumo de energia considera principalmente as taxas de transferências e recepção dos dados para estimar o consumo da interface.

BALASUBRAMANIAN, BALASUBRAMANIAN, e VENKATARAMANI (2009) produziram um modelo de consumo para redes Wi-Fi para o download de *x* bytes. Ele considera apenas a taxa de transferência e energia mínima necessária à manutenção da interface, o modelo é apresentado na Tabela 3-5. Método similar foi usado por KALIC, BOJIC e KUSEK (2012) para modelar o consumo durante *download* ou *upload* de dados.

XIAO *et al.* (2010) produziram um modelo de consumo de sistema que representa o consumo da interface Wi-Fi como função da taxa de *download*, taxa de *upload* e do modo operação.

ZHANG *et al.* (2010) produziram um detalhado modelo de consumo de sistema que utiliza uma máquina de estados finito para descrever o consumo da interface Wi-Fi. O consumo é função de quatro variáveis e quatro estados. As quatro variáveis são o número de pacotes enviados por segundo, pacotes recebidos por segundo, taxa do canal de transmissão e a taxa de transferência de dados do canal de transmissão. Os quatro estados são baixa potência, alta potência, transmissão baixa e transmissão alta. Transmissão baixa e alta são estados breves em que ocorre a transferência de dados.

HUANG *et al.* (2012) realizaram detalhada análise das redes 4G comparando-as a redes Wi-Fi e 3G. Eles produziram um modelo de consumo para redes Wi-Fi que consiste de cinco estados: promoção, transmissão, cauda, *beacon* e espera. Para cada estado, existem energia e duração associados. A transmissão é função linear da taxa de transmissão e recepção.

YOON *et al.* (2012), JUNG *et al.* (2012) bem como LEE, JOE e KIM (2014) usaram um modelo de consumo de energia para interface Wi-Fi que consiste de função linear com limiar de pacotes. Quando a taxa ultrapassa esse limiar, a potência da interface atinge seu valor máximo e permanece constante.

DING *et al.* (2013) realizaram extenso estudo sobre o impacto da força do sinal em redes sem fio no consumo de energia em *smartphones*. Um modelo de consumo foi construído para redes Wi-Fi que utiliza máquina de estados finitos. Esse modelo de consumo para o HTC Nexus One é apresentado na Figura 3.11. A interface Wi-Fi apresentada considera que o PSM está em uso.

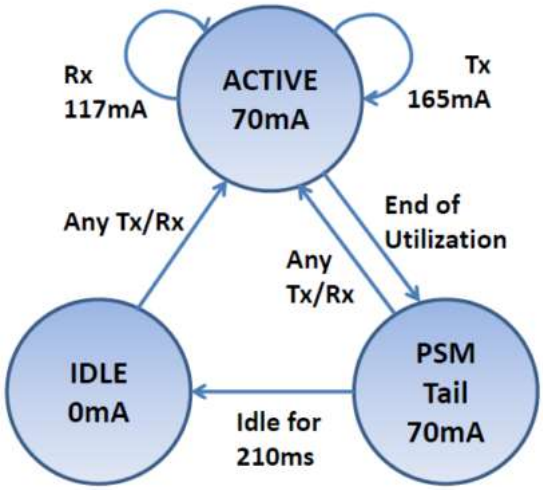

Figura 3.11: Máquina de estados de energia do HTC Nexus One (DING *et al.*, 2013).

LI *et al.* (2014b) projetaram detalhada máquina de estados finito e algoritmo de estimação para descrever o consumo de energia da interface Wi-Fi. Essa interface, segundo a pesquisa, apresentou 86% de precisão quando comparada a medições por *hardware*, resultado superior ao de outros modelos disponíveis na literatura.

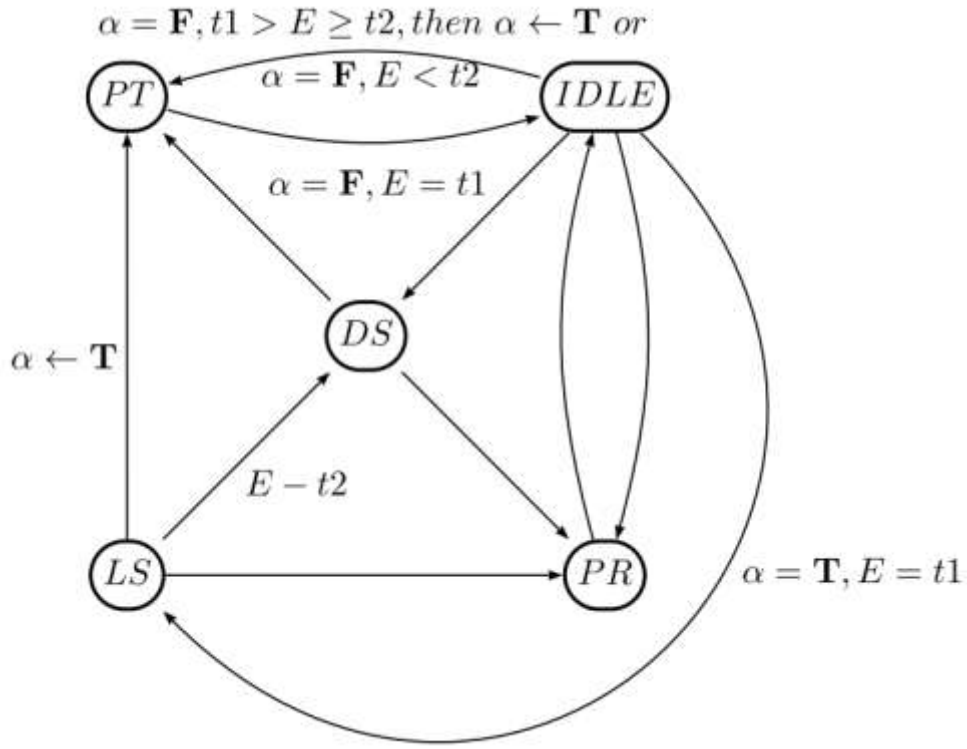

Figura 3.12: Modelo de energia proposto por (LI *et al.*, 2014b).

A máquina de estados finito é apresentada na Figura 3.12, ela define cinco estados para interface de Wi-Fi com diferentes níveis de energia. O acrônimo usado para cada estado é apresentado na Tabela 3-3 e o consumo para cada estado na Tabela 3-4. O algoritmo de estimação é apresentado em (LI *et al.*, 2014b).

Tabela 3-3: Lista de estados do modelo de energia proposto por (LI *et al.*, 2014b).

Estado	**Descrição**
IDLE	Estado de espera por novos pacotes.
PT	Estado de transmissão de pacotes.
PR	Estado de recebimento de pacotes.
LS	Estado de sono leve.
DS	Estado de sono profundo.

Tabela 3-4: Consumo de energia por estado da interface (LI *et al.*, 2014b).

Estado	**Potência (mW)**	**Duração (ms)**
IDLE	200	205
PT	$0{,}069 \times pacote_{tamanho} + 286$	1
PR	240	1
LS	80	$T_{intervalo\_entre\_pacotes} - 140$
DS	10	-

3.6 Modelo de consumo para redes 3G

Na sessão 2.5.5 foram abordadas brevemente as características das redes de telefonia celular, principalmente as redes 3G. O modelo de consumo de redes 3G encontrado na literatura é representado por máquina de estados finitos, regressão linear para cada estado e tempo de promoção e rebaixamento associados. Destaca-se o elevado tempo de rebaixamento associado a cauda que é responsável por alto consumo. Segundo BALASUBRAMANIAN, BALASUBRAMANIAN e VENKATARAMANI (2009), cerca de 60% da energia gasta pelo 3G ocorre no período de cauda.

Em redes de telefonia móvel, o protocolo de comunicação entre torre e aparelho é definido por máquina de estados administrada pelo controle de recurso de rádio, ou em inglês *radio resource control* (RRC). O RRC possui três estados possíveis na transmissão de dados. O estado de espera (IDLE), canal dedicado (CELL_DCH) e canal de acesso direto (CELL_FACH).

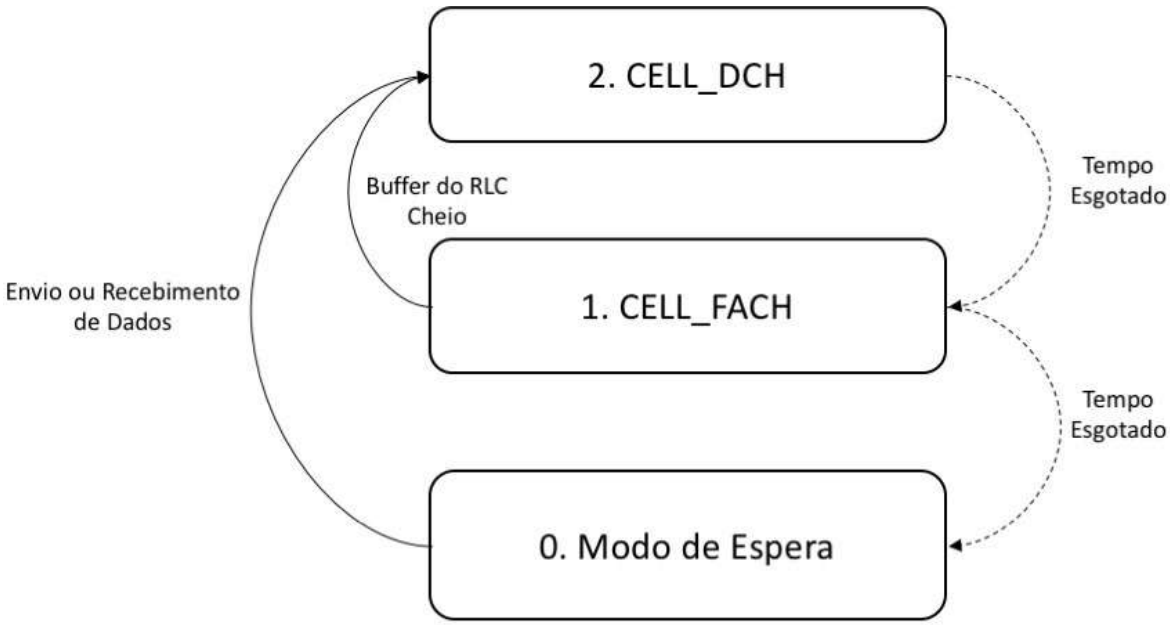

Figura 3.13: Máquina de estado do RRC para comunicação de dados do 3G.

A máquina de estados pode ser descrita conforme a Figura 3.13. O envio ou recebimento de pacotes força a transição do RRC para o CELL_DCH, ao término da comunicação ocorre um período de espera antes de ir para o estado CELL_FACH. Caso nova transmissão ou recepção seja necessária, o estado pode rapidamente retornar para CELL_DCH. Caso nenhuma comunicação ocorra, o tempo de espera no CELL_FACH se esgota e o estado retorna ao modo de espera.

Exemplo desse comportamento de consumo de energia pode ser visto na fFigura 3.14, a curva de consumo causada na transmissão de um pacote UDP único.

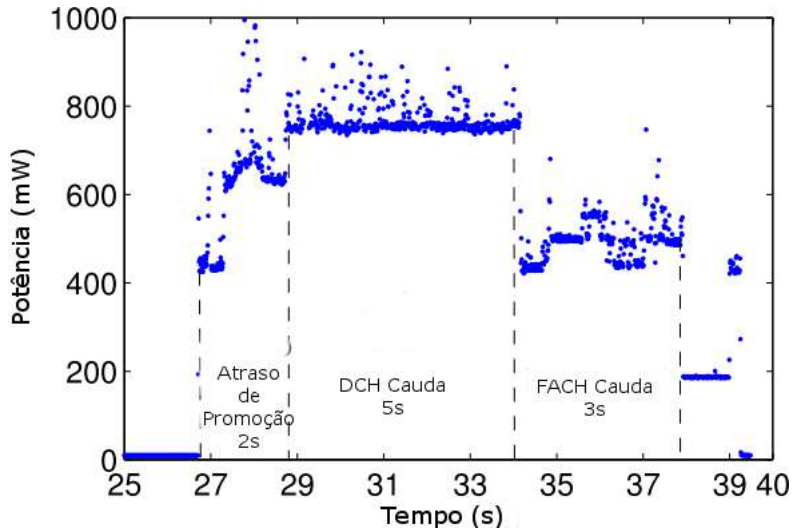

Figura 3.14: Consumo da transmissão de um único pacote UDP enviado em t = 26,8 s (QIAN *et al.*, 2011).

BALASUBRAMANIAN, BALASUBRAMANIAN e VENKATARAMANI (2009) produziram um modelo de consumo redes 3G para download de *x* bytes que considera a energia de transferência função linear entre a energia de cauda e a energia de manutenção. O modelo é apresentado na Tabela 3-5.

Tabela 3-5: Modelo de consumo usado por (BALASUBRAMANIAN; BALASUBRAMANIAN; VENKATARAMANI, 2009).

Estado	**3G**	**Wi-Fi**
Energia de Transferência	$0{,}025x + 3{,}5$	$0{,}007x + 5{,}9$
Energia de Cauda	$0{,}62\ J/s$	-
Manutenção	$0{,}02\ J/s$	$0{,}05\ J/s$
Duração da Cauda	$12{,}5\ s$	-

ZHANG *et al.* (2010) produziram um modelo de energia detalhado para o HTC Dream que considera o consumo individual de cada estado da interface 3G. Os estados definidos são IDLE, CELL_FACH e CELL_DCH conforme apresentado na Figura 3.15.

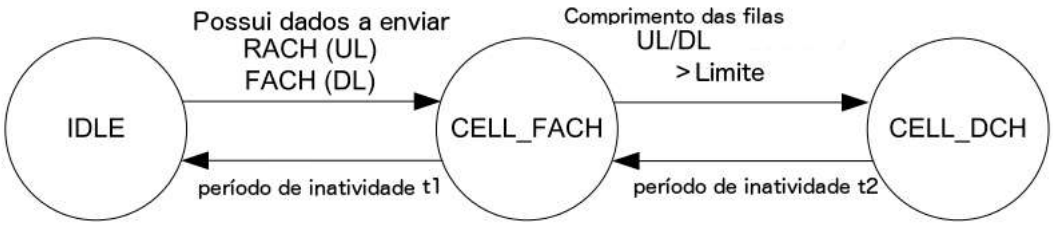

Figura 3.15: Estados de energia da interface 3G (ZHANG *et al.*, 2010).

$$P_{3g} = \beta_{3G\_idle} \times 3G_{idle} + \beta_{3G\_FACH} \times 3G_{FACH} + \beta_{3G\_DCH} \times 3G_{DCH} \quad (3\text{-}12)$$

A Equação (*3-12*) apresenta a componente 3G do modelo de consumo produzido por ZHANG *et al.* (2010). Os coeficientes são apresentados na Tabela 3-6. Neste modelo não é considerada a taxa de transferência, supõe-se que o consumo de energia de cada estado seja constante.

Tabela 3-6: Modelo de Energia do 3G do HTC Dream (ZHANG *et al.*, 2010).

Variável	**Intervalo**	**Coeficiente**
Taxa de Transferência	$0 - \infty$	-
Fila de transmissão	$0 - \infty$	-
Fila de recepção	$0 - \infty$	-
$3G_{idle}$	$0\ ou\ 1$	$\beta_{3G\_idle} = 10$
$3G_{FACH}$	$0\ ou\ 1$	$\beta_{3G\_FACH} = 401$
$3G_{DCH}$	$0\ ou\ 1$	$\beta_{3G\_DCH} = 570$

JUNG *et al.* (2012) propuseram um modelo de consumo similar baseado nos três estados da interface 3G conforme apresentado na Equação (*3-13*). O consumo do estado IDLE e CELL_DCH foi inferido durante os testes. Enquanto o consumo de CELL_FACH é estimado pela Equação (*3-14*).

$$P_{cellular} = \beta_{rrc};\ rrc \in \{IDLE, FACH\ e\ DCH\} \quad (3\text{-}13)$$

$$\beta_{FACH} = \frac{P_{component} \times R - \beta_{DCH} \times \Delta t_{DCH}}{\Delta t_{FACH}} \quad (3\text{-}14)$$

4 CONSTRUÇÃO DO MONITOR DE ENERGIA

Para a geração do modelo de consumo de energia, foi necessário coletar as informações de consumo de energia do sistema e de suas componentes a fim de identificar suas características. Essa coleta de informações pode ser realizada por um equipamento externo ou por um programa que acessa as interfaces de programação providas pelo sistema operacional.

A captura por programa é simples de realizar, não requer modificações físicas do aparelho. Entretanto, a taxa de amostragem é baixa para não afetar o desempenho do sistema, a precisão depende das interfaces de programação e dos sensores disponíveis. A captura por equipamento externo apresenta alta precisão, alta taxa de amostragem e baixa interferência no desempenho. Entretanto, mobilidade do aparelho é prejudicada por requerer acesso físico aos terminais da bateria ou de outros componentes.

Neste trabalho, as medições foram realizadas via interfaces de programação. Uma aplicação Android chamada de *Droid Energy Suite* (DES) foi criada. Ela é composta de um monitor de energia do sistema e um conjunto de testes. O monitor de energia foi responsável por coletar periodicamente o estado do sistema e persistir a informação para análise posterior. A suíte de testes permitiu destacar a relação entre o estado do sistema e de suas principais componentes no consumo de energia.

4.1 Interfaces de monitoramento

O Android é sistema operacional construído sobre o núcleo do Linux. Ele oferece um conjunto misto de interfaces que permitem obter informações do sistema. As interfaces expostas pelo núcleo do Linux e as interfaces providas pelo arcabouço do Android.

As interfaces do núcleo oferecem informações detalhadas sobre o sistema, porém sofrem com a fragmentação gerada pela variedade de fabricantes. A localização e disponibilidade de informações varia entre os aparelhos.

No Android, duas interfaces principais permitem obter informações sobre o sistema. A primeira é pelo acesso a objetos que representam os serviços do sistema, a segunda é pelo registro para recebimento de notificações assíncrona do sistema.

4.2 Interfaces do núcleo do Linux

O núcleo do Linux disponibiliza informações sobre o sistema por meio de dois sistemas de arquivos virtuais em memória que representam objetos do núcleo na forma de

arquivos, diretórios e referências simbólicas. Os dois sistemas virtuais são o *SysFS e ProcFs*.

O *SysFs* foi adicionado ao núcleo a partir da versão 2.6 com objetivo de armazenar de forma estruturada as informações do núcleo e substituir o *ProcFs* (MOCHEL, 2005). Esse sistema virtual de arquivos é montado no caminho **/sys**. As estruturas do núcleo são representadas conforme a Tabela 4-1.

Tabela 4-1: Mapeamento do núcleo em espaço de usuário (MOCHEL, 2005).

Interna	**Externa**
Objetos do Núcleo	Diretório
Atributos do Objeto	Arquivos
Relação entre Objetos	Referências Simbólicas

O *ProcFs* é montado no caminho **/proc**. O diretório é originário do Unix, proposto e publicado em 1984 (KILLIAN, 1984). É encontrado em praticamente todas as versões atuais dos sistemas operacionais Unix e Linux. Neles, os processos em execução são representados por um diretório correspondente ao número de identificação do processo. Nos sistemas Linux, outras informações são armazenadas nesse mesmo diretório. As interfaces do Núcleo do Linux monitoradas pelo DES são apresentados no APÊNDICE D.

4.3 Serviços de sistema

O Android implementa várias das funcionalidades por meios de serviços de sistema acessados via comunicação entre processos representado como instâncias de objetos Java. Uma aplicação Android consegue instanciar esses objetos usando a interface de programação *getSystemService* e o identificador do serviço. Obtido o objeto, basta utilizar os métodos definidos para obter as informações. Mais sobre os serviços no APÊNDICE E.

4.4 Eventos de sistema

O Android implementa interface de programação para o recebimento de notificações assíncronas do sistema e de outras aplicações. Objetos chamado *Intent* são usados representar a mensagem. *Intents* podem ser enviados e recebidos. Eles são a principal forma de comunicação de alto nível entre processos da plataforma Android.

Intents são diferenciados pelo atributo *action*, uma cadeia de caracteres que identifica o tipo do evento, opcionalmente um conjunto extra de parâmetros pode ser agregado a mensagem. Mais sobre os eventos monitorados no APÊNDICE F.

4.5 Droid Energy Suíte (DES)

O *Droid Energy Suite* é aplicação Android escrita em Java e C. Ela é composta de duas partes: serviço de monitoramento de energia e conjunto automatizado de testes. As interfaces monitoradas são descritas em detalhes no APÊNDICE G. O serviço de monitoramento de energia é de plano de fundo e controla quatro monitores para a bateria, processador, rede e eventos do sistema. Monitoramento da bateria, processador e rede são realizados pela interface do Linux (APÊNDICE D). Foi usado código em linguagem C consumido via chamadas JNI para reduzir o impacto no processamento e memória. Cada monitor produz um arquivo binário contendo os registros capturados. Os registros dos eventos de sistema utilizam a interface de programação do Android. Uma classe chamada *SystemEventReceiver* é responsável por registrar os eventos notificados pelo sistema (APÊNDICE F) em um arquivo.

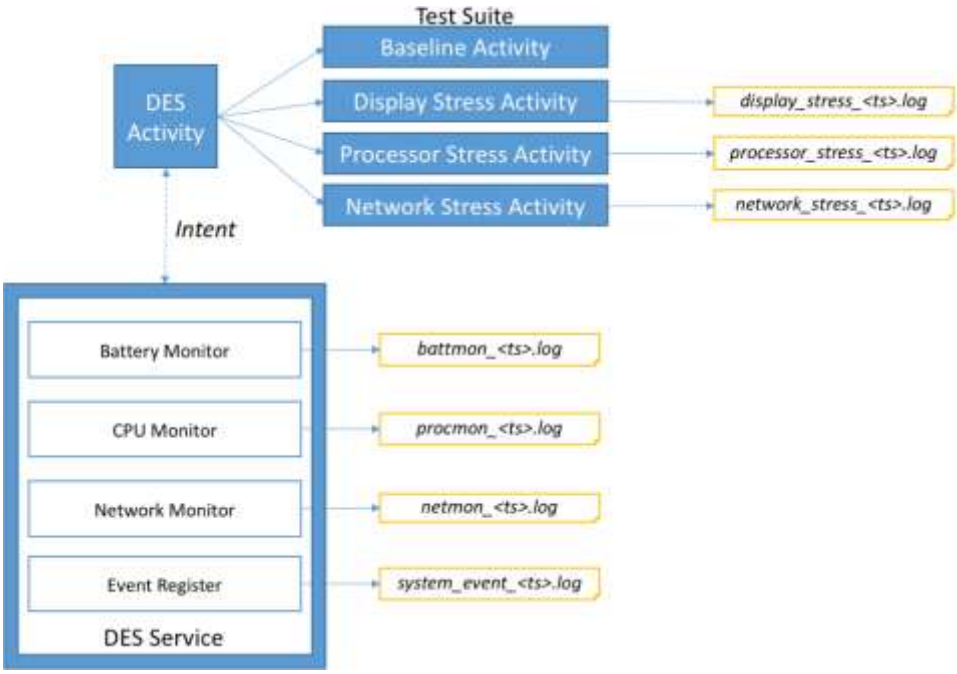

Figura 4.1: Estrutura da aplicação DES.

A arquitetura da aplicação é apresentada na Figura 4.1. Ela possui conjunto automatizado de testes para exercitar as componentes de tela, processador, rede Wi-Fi e

rede móvel. Testes foram criados para realçar o impacto das diversas variáveis no consumo da componente. Por exemplo, o teste de brilho busca identificar a relação da intensidade do brilho no consumo da tela. Outros testes visam estabelecer relações similares entre aumentar ou diminuir o valor da única variável no consumo da componente em observação. Os testes são descritos em mais detalhes no APÊNDICE H.

4.6 Procedimento de teste

Os testes de consumo seguiram o seguinte passo a passo:

1. **Preparação do aparelho e ambiente:** nessa etapa, o aparelho foi recarregado e desconectado de fontes de alimentação externas, as interfaces de rede são habilitadas ou desabilitadas, aplicativos em segundo plano são encerrados, é instalado o aplicativo e outras configurações adicionais são realizadas.
2. **Início da captura e teste de energia:** nessa etapa, o aplicativo *Droid Energy Suite* é aberto, selecionada a configuração de captura e iniciada a captura. Em seguida, é realizado o teste seguindo as especificações. Aguarda-se o tempo necessário para a conclusão do experimento. O aparelho é deixando isolado e não recebe interferência humana até a conclusão de teste. Após a conclusão, é interrompida a captura e encerrado o aplicativo.
3. **Recolhimento dos logs:** o aparelho é conectado a um computador via interface USB, usando o visualizador de arquivos ou ADB. Os logs são copiados para o computador.
4. **Conversão dos logs para texto:** Os logs gerados pelo aplicativo no formato binário são convertidos para texto. Um simples programa em C é utilizado para converte o arquivo binário para texto. Outros campos são gerados baseados nas informações coletadas para facilitar a plotagem e análise dos valores.
5. **Processamento dos Resultados:** Os logs em formato textual são passados para um programa em Python que usa R para regressão linear. Os gráficos são plotados usando Gnuplot.

4.7 Avaliação técnica

Os resultados são avaliados para identificar padrões entre parâmetros de entrada e a energia consumida. Modelos de consumo ou gráficos são gerados para destacar a relação

usando Python e R. Caso o teste apresente problemas na execução, ele é descartado e novos experimentos são realizados.

5 EXPERIMENTO DE CONSUMO

Para o sucesso da construção do modelo de consumo de energia é necessária a correta caracterização do consumo das componentes do sistema. Foram criados e executados cenários de teste de consumo que facilitem a identificação e caracterização dessas componentes. As componentes avaliadas foram escolhidas entre as mais citadas no estudo da literatura visto no capítulo 3 e disponíveis nos aparelhos de teste: CPU, tela LCD, Wi-Fi e 3G.

Primeiro, foi medido o consumo de base do dispositivo no estado de hibernação e espera, a fim de identificar a energia mínima para o funcionamento dos aparelhos. Em seguida foram realizados vários experimentos de consumo de energia para caracterizar o consumo de cada componente. O consumo de energia foi calculado pela Equação (*5-1*).

$$Potência_{dispositivo} = Voltagem_{\ bateria} * Corrente_{bateria} \qquad (5\text{-}1)$$

Os cenários de teste de consumo de energia foram projetados para destacar a influência dos parâmetros chave no consumo de cada componente, esses parâmetros foram identificados na literatura e apresentados no capítulo 3. A captura e registro dos valores de consumo de energia, o estado geral do sistema e a suíte de testes foram implementados na ferramenta de monitoramento de energia vista no capítulo 4.

Nas próximas seções, são apresentados os aparelhos usados nos testes, os experimentos e procedimentos, os resultados obtidos e a análise geral do que foi observado. Por fim uma conclusão geral é apresentada. No capítulo 6, essas informações foram usadas para construir um modelo simples de consumo de energia para os dispositivos.

5.1 Dispositivos de teste

Os experimentos foram realizados em aparelhos Android fornecidos pelo antigo Instituto Nokia de Tecnologia (INdT). Os aparelhos usados foram um LG Nexus 4 E960, um Xiaomi Redmi 2 e um LG L80. As especificações desses aparelhos são apresentadas na Tabela 5-1, Tabela 5-2 e Tabela 5-3, respectivamente. No Xiaomi Redmi 2 foi observado que as informações da bateria são atualizadas a cada 1,2 segundos. Situação que não foi observada nos outros aparelhos. Esse efeito é atenuado em experimentos de longa duração. Mas prejudica a observação de picos curtos de consumo.

Alguns outros aparelhos como um Motorola Moto E, um Sony Xperia Z1 foram considerados para realização dos experimentos. Porém tiveram que ser descartados. O Motorola Moto E não permitiu acesso ao sistema virtual de arquivos necessário para capturar as informações da bateria, processador e rede. O Sony Xperia Z1 apresentou leituras fora de escala e, portanto, pouco confiáveis.

Tabela 5-1: Especificações do LG Nexus 4 E960.

Nome	LG Nexus 4, LG Mako, LG E960
Data de lançamento	Outubro de 2012
Sistema operacional	Android 5.1.1 (Lollipop)
Chipset	Qualcomm APQ8064 Snapdragon S4 Pro
CPU	Quad-core 1,5 GHz Krait
GPU	Adreno 320
Memória RAM	2 GB
Memória Flash	8/16 GB
Câmera	8 MP (Traseira), 1,3 MP (Frontal)
Resolução da tela	768x1280, ~318 ppi, 4,7”
Tecnologia da tela	True HD IPS Plus
Dimensões físicas	133,9 x 68,8 x 9,1 mm
Interface sem fio	Bluetooth NFC (Android Beam) Wi-Fi (802.11 b/g/n)
Rede celular	GSM / HSPA
Bateria	Li-Po 2.100 mAh (Não removível)
Peso	139 g

Tabela 5-2: Especificação do Xiaomi Redmi 2.

Nome	Xiaomi Redmi 2, Xiaomi Hongmi 2
Data de lançamento	Agosto de 2015
Sistema operacional	Android 4.4.4 (KitKat)
Chipset	Qualcomm MSM8916 Snapdragon 410
CPU	Quad-core 1,2 GHz Cortex-A53
GPU	Adreno 306
Memória RAM	1/2 GB
Memória Flash	8/16 GB (Extensível com microSD)
Câmera	8 MP (Traseira), 2 MP (Frontal)
Resolução da tela	768x1280, ~318 ppi, 4,7”
Tecnologia da tela	IPS LCD
Dimensões físicas	134 x 67,2 x 9,4 mm
Interface sem fio	Bluetooth Wi-Fi (802.11 b/g/n)
Rede celular	GSM / HSPA / LTE (Dual SIM)
Bateria	Li-Po 2.200 mAh
Peso	133 g

Tabela 5-3: Especificação do LG L80.

Nome	LG L80 Dual SIM, LG D380
Data de lançamento	Abril de 2014
Sistema operacional	Android 4.4.2 (KitKat)
Chipset	Qualcomm MSM8210 Snapdragon 200
CPU	Dual-core 1,2 GHz Cortex-A7
GPU	Adreno 302
Memória RAM	1 GB
Memória Flash	4 GB (Extensível com microSD)
Câmera	5 MP (Traseira)
Resolução da tela	480x800, ~187 ppi, 5”
Tecnologia da tela	IPS LCD
Dimensões físicas	138,2 x 74,3 x 9,7 mm
Interface sem fio	Bluetooth Wi-Fi (802.11 b/g/n)
Rede celular	GSM / HSPA (Dual SIM)
Bateria	Li-Ion 2.540 mAh
Peso	-

5.2 Consumo de base em espera e hibernação

O consumo de base permite determinar o consumo mínimo do aparelho. São considerados dois estados de consumo mínimo: hibernação e espera.

O estado de hibernação ocorre quando os núcleos do processador são suspensos e nenhuma conectividade de rede habilitada (modo avião), como nenhum núcleo está ativo, não é registrado o consumo de energia, portanto os valores foram estimados pelos limites que antecedem e sucedem a hibernação. Medições são observadas nas curtas ativações periódicas do sistema.

O estado de espera representa o consumo mínimo do aparelho despertado. Nessa condição, um núcleo do processador se encontra ativo a frequência mínima de operação, carga mínima de processamento, tela desligada e interfaces de rede desligadas (modo avião). Um *wakelock* é adquirido pelo monitor para prevenir a hibernação do sistema.

5.2.1 Procedimento experimental

Pré-condições:

- Aplicação *Droid Energy Suite* instalada;
- Aparelho em modo avião;
- Aparelho desconectado de fontes externas de energia;

- Bateria com carga acima de 70%;
- Brilho de Tela no modo manual e no valor mínimo.

Procedimento:

1. Iniciar o monitor de energia;
2. Habilitar ou não a opção prevenir hibernação:
    a. Marcar para medir o consumo em estado de espera;
    b. Não marcar para medir o consumo em hibernação;
3. Pressionar o botão para iniciar a captura;
4. Pressionar o botão de bloqueio do aparelho para desligar a tela;
5. Deixar o aparelho durante 1 hora para o teste de espera ou 6 horas para o teste de hibernação sem interação externa;
6. Desbloquear o aparelho;
7. Abrir o monitor de energia e pressionar o botão para interromper a captura;
8. Coletar os registros gerados pelo monitor de energia para um computador.

5.2.2 Resultado LG Nexus 4

Consumo em hibernação:

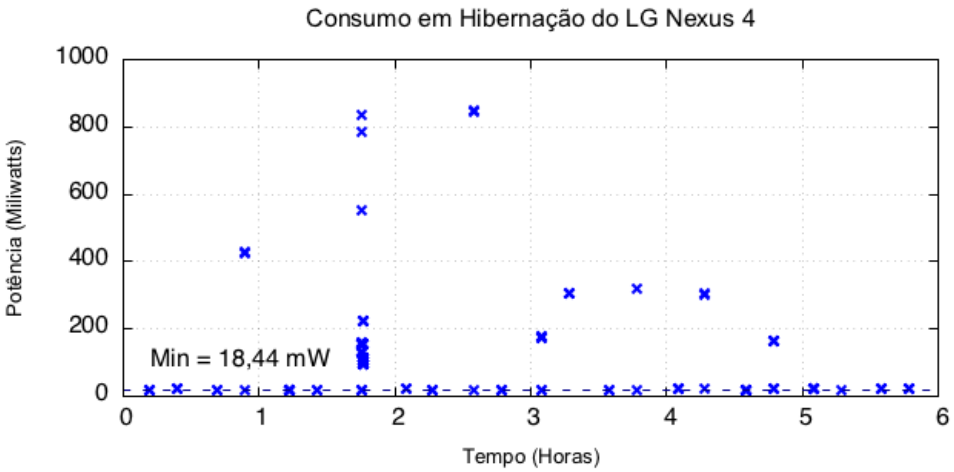

Figura 5.1: Consumo em hibernação do LG Nexus 4.

A Figura 5.1 apresenta o consumo registrado em hibernação do LG Nexus 4, a amostragem é espaçada no tempo devido a inatividade dos núcleos, apenas nas periódicas e curtas ativações é registrado o consumo. O menor valor registrado fica na faixa de 18 mW indicado pela linha tracejada.

Consumo em espera:

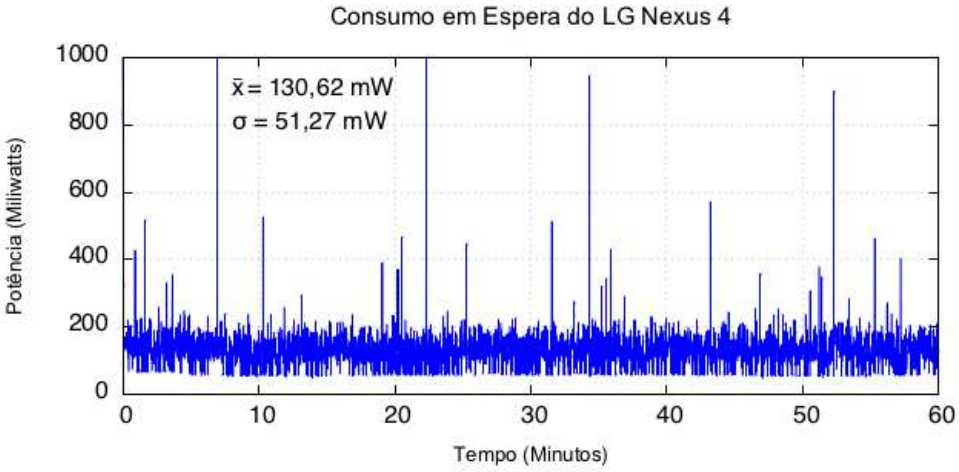

Figura 5.2: Consumo em espera do LG Nexus 4.

A Figura 5.2 apresenta o consumo em espera obtido por meio de um *Wakelock*. O consumo fica na média de 130 mW muito superior ao registrado em hibernação.

5.2.3 Resultado Xiaomi Redmi 2

Consumo em hibernação:

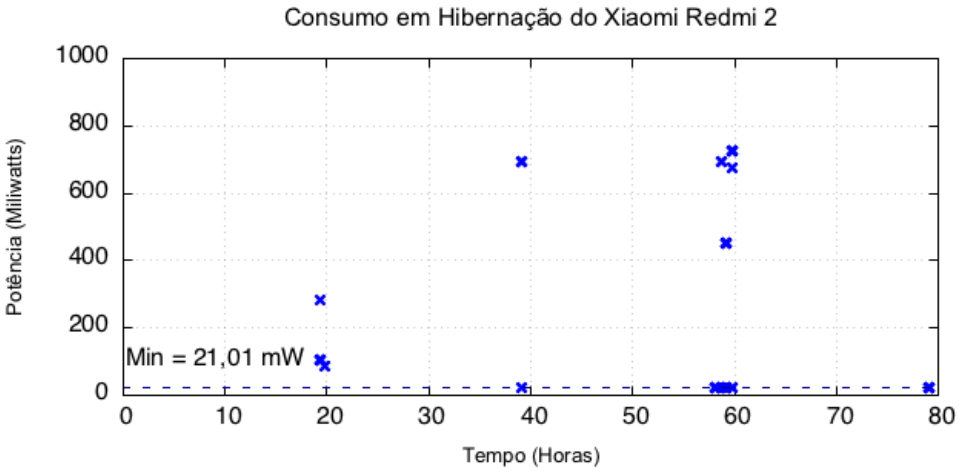

Figura 5.3: Consumo em hibernação do Xiaomi Redmi 2.

A Figura 5.3 apresenta o consumo em hibernação do Xiaomi Redmi 2, diferente do LG Nexus 4 é possível notar que o espaçamento dos despertares é muito longo. Foi necessário que o teste se prolongasse por mais de oitenta horas para coleta de algum resultado representativo. Testes anteriores, com período de seis horas, só foram capazes de

registrar no máximo um despertar. Essa grande diferença, quando comparada ao Nexus 4, pode ser atribuída a diferenças na personalização do fabricante e aplicações instaladas.

Consumo em espera:

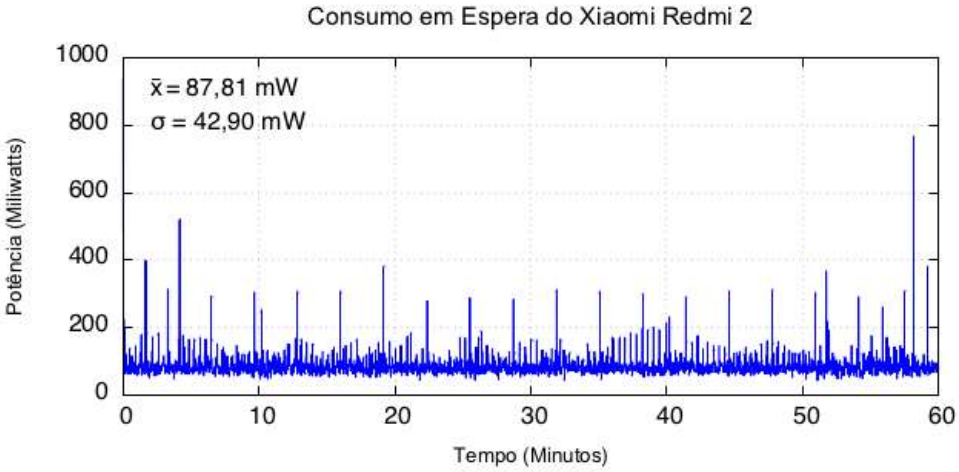

Figura 5.4: Consumo em espera do Xiaomi Redmi 2.

A Figura 5.4 apresenta o consumo em espera do Xiaomi Redmi 2. É possível observar menor variação nos valores observado quando comparado ao LG Nexus 4 devido à baixa taxa de atualização descrita na seção 5.1. O consumo ficou na faixa de 90 mW.

5.2.4 Resultado LG L80

Consumo em hibernação:

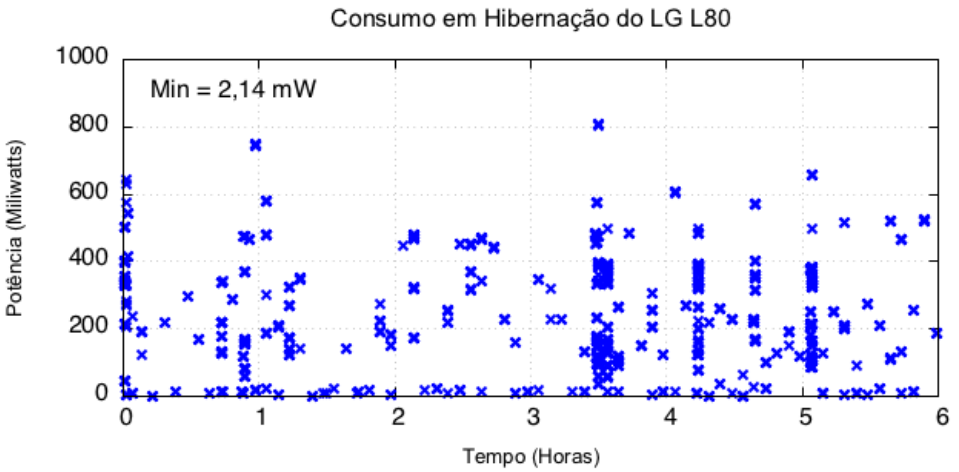

Figura 5.5: Consumo em hibernação do LG L80.

A Figura 5.5 apresenta o consumo em hibernação do LG L80. É possível notar alta taxa de ativações no período do experimento. O menor valor registrado ficou na faixa de 2 mW. Outra característica observada foram leituras negativas ou zero de descarga de corrente, indicando baixa confiabilidade dos sensores de bateria em alguns momentos.

Consumo em espera:

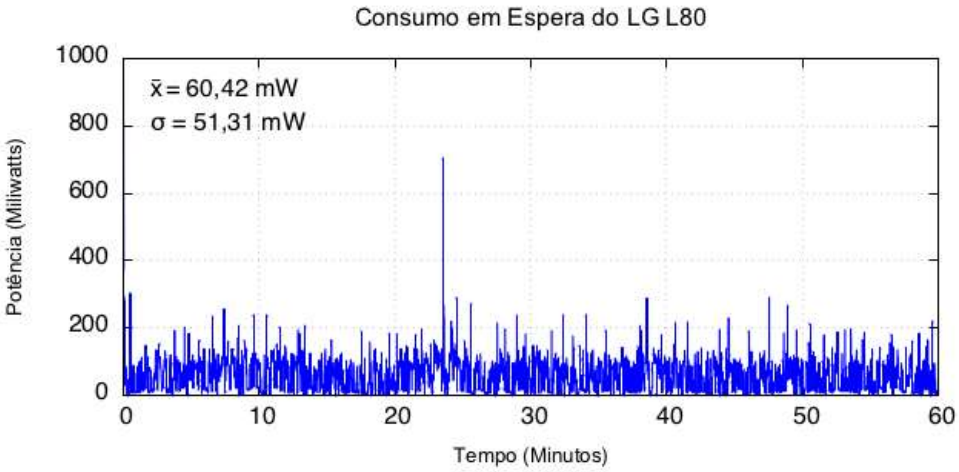

Figura 5.6: Consumo em espera do LG L80.

A Figura 5.6 apresenta o consumo em espera do LG L80. O consumo ficou na faixa de 60 mW.

5.3 Consumo da tela LCD

O consumo de tela permite determinar o consumo adicional pelo uso da tela. Esse experimento utilizou a suíte de teste para exercitar o consumo da tela em diferentes níveis de brilho e colorações de tela.

O nível de brilho é um valor de 0 a 255, onde 0 é o nível mínimo de brilho com a tela ligada e 255 o valor máximo. Durante o teste a intensidade de brilho foi incrementada uma unidade a cada 5 segundos. Tempo suficiente para estabilizar as medições. Levou-se cerca 21 minutos e meio para completar o teste de brilho para uma coloração de fundo. Foram usadas cinco colorações de fundo diferentes: preto, branco, vermelho, verde e azul. Em valores RGB: {0, 0, 0}, {255, 255, 255}, {255, 0, 0}, {0, 255, 0} e {0, 0, 255}. No total foi necessária cerca de 1 hora e 45 minutos para concluir uma rodada do experimento.

5.3.1 Procedimento experimental

Pré-condições:

- Aplicação *Droid Energy Suite* instalada;
- Aparelho em modo avião;
- Aparelho desconectado de fontes externas de energia;
- Bateria com carga acima de 70%;
- Brilho de tela no mínimo.

Procedimento:

1. Iniciar o monitor de energia;
2. Pressionar o botão para iniciar a captura;
3. Abrir a suíte de teste de tela;
4. Deixar o aparelho sem interação externa até conclusão do teste;
5. Pressionar o botão para interromper a captura pelo monitor de energia;
6. Coletar os registros gerados pelo monitor de energia para um computador.

5.3.2 Resultado LG Nexus 4

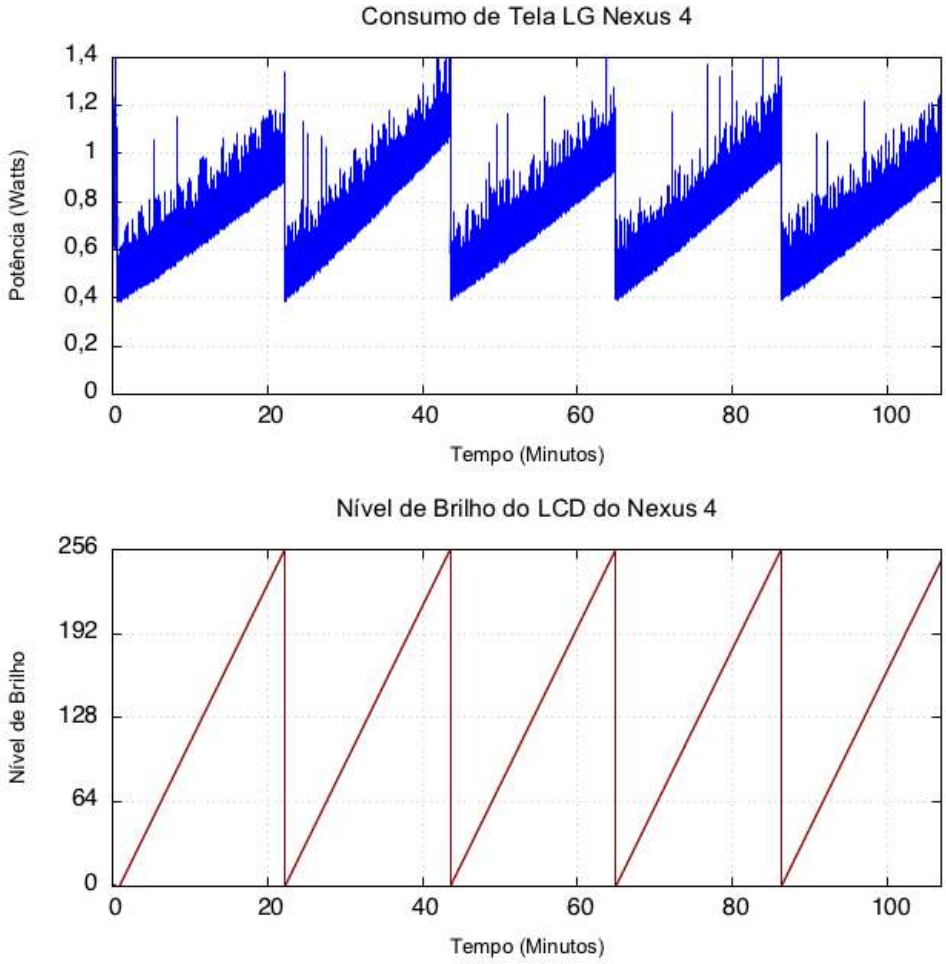

Figura 5.7: Consumo de energia observado durante o teste da tela LCD do LG Nexus 4.

Na Figura 5.7 é apresentado o consumo da tela do LG Nexus 4. É possível notar que a curva de consumo é similar à reta. As cinco inclinações representam o consumo para a tela preta, branca, vermelho, verde e azul para a curva de nível de brilho apresentados no gráfico de baixo. Pequena variação nos valores máximos e mínimos pode ser observada.

5.3.3 Resultado Xiaomi Redmi 2

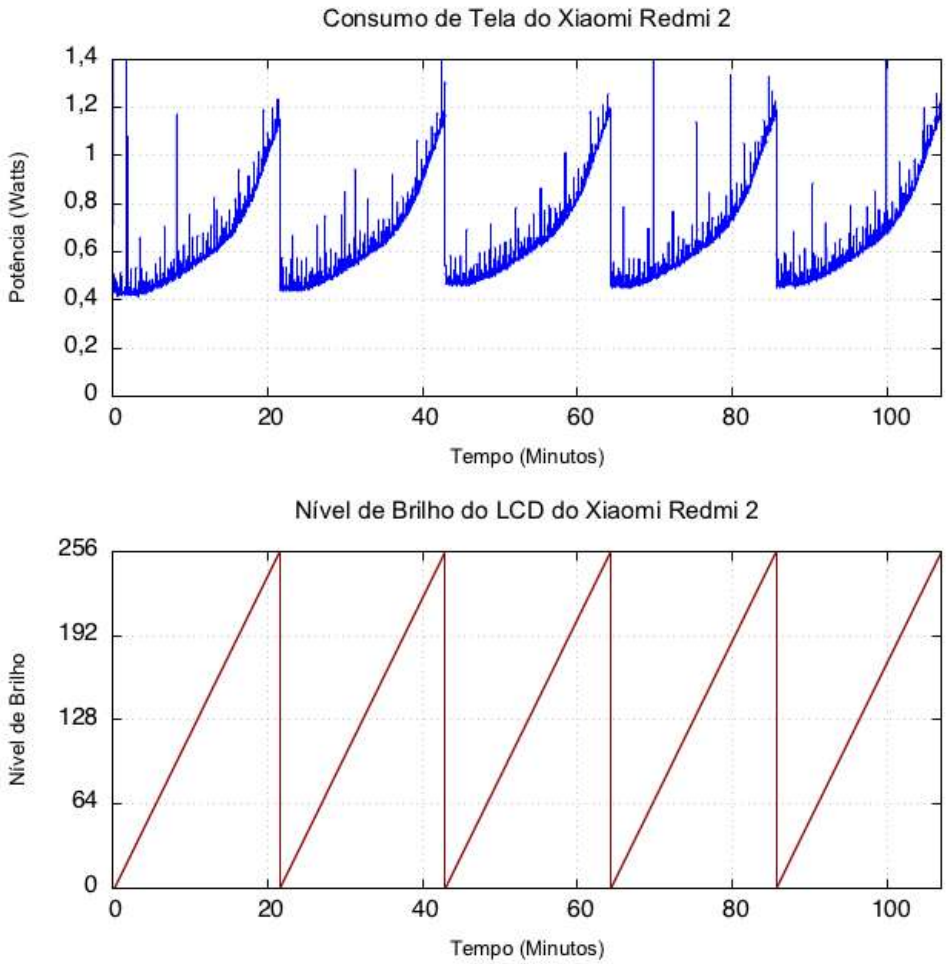

Figura 5.8: Consumo de energia observado para o teste de tela LCD do Xiaomi Redmi 2.

Na Figura 5.8 é apresentado o consumo da tela do Xiaomi Redmi 2. Diferente do que foi observado no LG Nexus 4, a curva de consumo não segue uma reta. A variação dos valores máximos e mínimos foi menor que a observada no LG Nexus 4.

5.3.4 Resultado LG L80

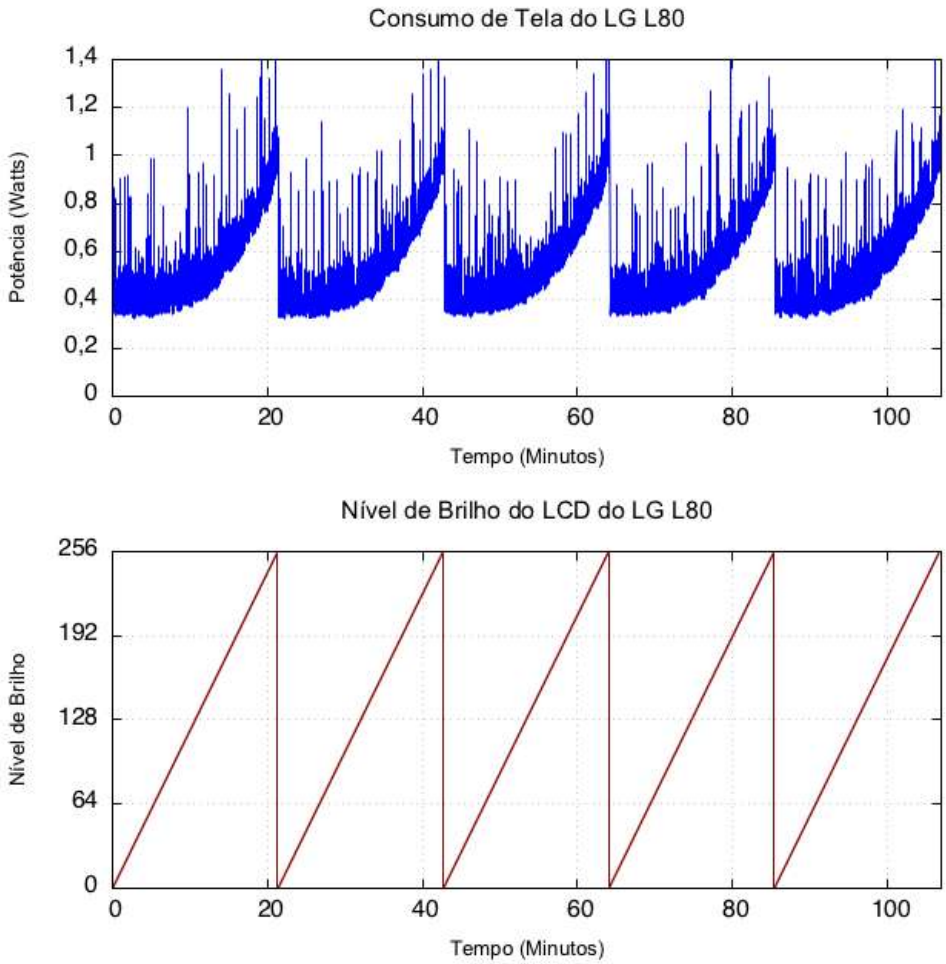

Figura 5.9: Consumo de energia observado para o teste de tela LCD do LG L80.

Na Figura 5.9 é apresentado o consumo de tela do LG L80. Assim como no Xiaomi Redmi 2, a curva de consumo foi não linear.

5.4 Consumo do processador por frequência e carga

O consumo de CPU mede a relação de carga de trabalho, frequência de operação e número de núcleos ativados no consumo de energia do processador. Esse experimento é importante para determinar o impacto de aplicações intensiva de CPU no consumo de energia.

Esse teste requer configuração dos parâmetros de operação da plataforma, portanto o aparelho precisa suportar o modo superusuário para permitir a configuração manual da frequência de operação do processador e do número de núcleos em operação. O teste avalia o consumo da CPU em espera e com carga de 100%. A tela e todas as interfaces de rede são desligadas durante o teste.

O procedimento de configuração foi escrito em linguagem em *shell script* e é apresentado no Programa 5-1. Primeiro foi desabilitado o serviço automático de ativação e desativação de núcleos da Qualcomm, o *mpdecision*. A política de escalonamento foi modificada para manual. Em seguida os núcleos foram ativados ou desativados e configurado para frequência de operação escolhida.

```
#!/bin/bash

ENABLE_CPU1= 0 # 0 – desabilitado, 1 – habilitado
ENABLE_CPU2= 0 # 0 – desabilitado, 1 – habilitado
ENABLE_CPU3= 0 # 0 – desabilitado, 1 – habilitado
CPU_FREQUENCY= 384000 # Frequência em Mhz

# Entrar em modo superusuário
su

# Interromper Qualcomm Hotplugging Driver
stop mpdecision

# Habilitar ou Desabilitar os núcleos secundários
# CPU0 não pode ser desabilitada
echo ${ENABLE_CPU1} > /sys/devices/system/cpu/cpu1/online
echo ${ENABLE_CPU2} > /sys/devices/system/cpu/cpu2/online
echo ${ENABLE_CPU3} > /sys/devices/system/cpu/cpu3/online

# Para cada cpu
for CPU_N in cpu0 cpu1 cpu2 cpu3; do
    # Ajuste da frequência manual
    echo userspace /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

    # Configurar frequência de operação
    echo ${CPU_FREQUENCY} > \
        /sys/devices/system/cpu/${CPU_N}/cpufreq/scaling_min_freq
    echo ${CPU_FREQUENCY} > \
        /sys/devices/system/cpu/${CPU_N}/cpufreq/scaling_max_freq
    echo ${CPU_FREQUENCY} > \
        /sys/devices/system/cpu/${CPU_N}/cpufreq/scaling_setspeed
```

```
done

# Sair do modo superusuário
exit
```

Programa 5-1: Procedimento de configuração de CPU em shell script.

5.4.1 Procedimento experimental

Pré-condições:

- Aplicação *Droid Energy Suite* instalada;
- Aparelho em modo avião;
- Aparelho desconectado de fontes externas de energia;
- Bateria com carga acima de 70%;
- Brilho de tela no mínimo.

Procedimento:

1. Colocar o aparelho em modo avião;
2. Configurar a frequência de operação e número de núcleos ativos usando o *shell script* visto no Programa 5-1;
3. Iniciar o monitor de energia com a opção adquirir *Wakelock* marcada;
4. Iniciar o stress de carga de processamento;
    a. Configurar para 1 minuto de atraso;
    b. Configurar para 5 minutos a 100%;
    c. Configurar para 5 minutos de descanso;
    d. Configurar para realizar 10x;
    e. Clicar no botão executar.
5. Bloquear a tela do aparelho para desligar a tela;
6. Deixar o aparelho sem interação externa durante o tempo de execução do teste;
7. Desbloquear o aparelho;
8. Interromper o monitor de energia;
9. Coletar os registros gerados pelo monitor de energia e suíte de carga.

5.4.2 Resultado LG Nexus 4

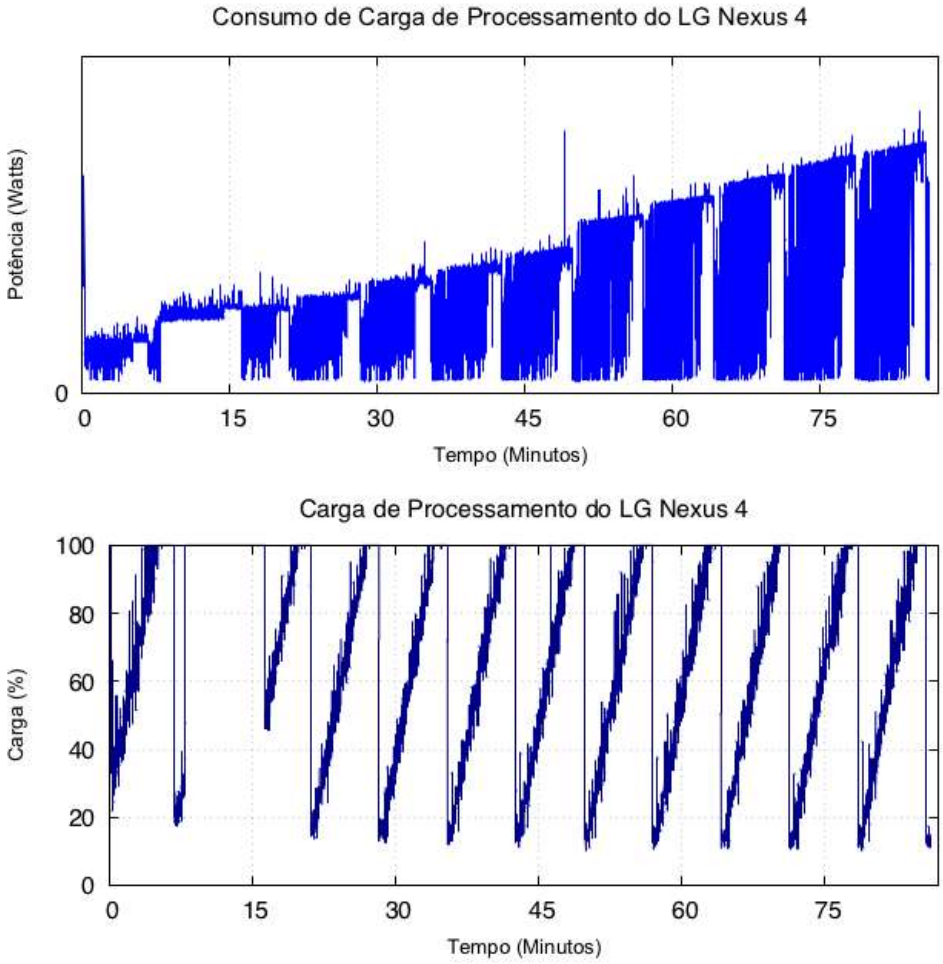

Figura 5.10: Consumo de energia observado para o teste de carga de processamento do LG Nexus 4.

Na Figura 5.10 é apresentado o consumo de carga de processamento do LG Nexus 4. O gráfico de cima representa o consumo de energia. O gráfico de baixo a carga normalizada para um segundo para melhor visualização. Cada inclinação é um teste de carga para uma frequência fixa do processador. O consumo cresceu proporcional a frequência e carga de processamento.

5.4.3 Resultado Xiaomi Redmi 2

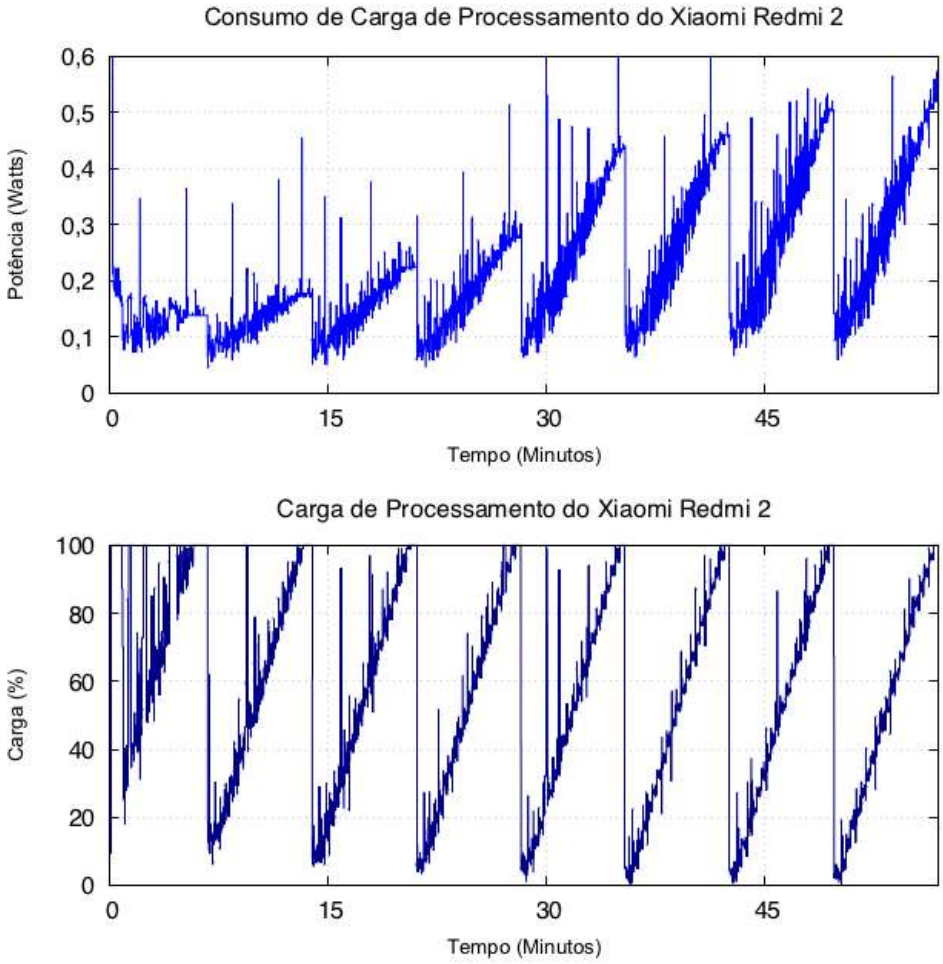

Figura 5.11: Consumo de energia observado para o teste de carga de processamento do Xiaomi Redmi 2.

Na Figura 5.11 é apresentado o consumo de carga de processamento do Xiaomi Redmi 2. Assim como o LG Nexus 4 o consumo aumentou em função da frequência e carga.

5.4.4 Resultado LG L80

O experimento não foi realizado no LG L80 devido à falta de suporte ao modo superusuário no aparelho necessário para controle da carga de processamento.

5.5 Consumo do processador por núcleo

O consumo de CPU por núcleo visa identificar o custo adicional na ativação de cada núcleo. Esse teste não precisa que o aparelho suporte o modo superusuário. O experimento consiste na criação de uma *thread* limitada por CPU a cada cinco minutos, até que todos os núcleos estejam saturados. Essa *thread* usa 100% da capacidade de processamento do processador. O número de *threads* criadas é limitada pelo número de núcleos do processador.

5.5.1 Procedimento experimental

Pré-condições:

- Aplicação *Droid Energy Suite* instalada;
- Aparelho em modo avião, mas com Wi-Fi habilitada;
- Aparelho desconectado de fontes externas de energia;
- Bateria com carga acima de 70%;
- Brilho de tela no mínimo.

Procedimento:

1. Colocar o aparelho em modo avião
2. Iniciar o monitor de energia com a opção adquirir *Wakelock* marcada.
3. Iniciar o stress de carga de processamento.
    a. Configurar para 1 minuto de atraso;
    b. Configurar para 5 minutos a 100%;
    c. Configurar para 5 minutos de descanso;
    d. Configurar para realizar 10x;
    e. Clicar no botão executar.
4. Bloquear a tela do aparelho para desligar a tela.
5. Deixar o aparelho sem interação externa durante o tempo de execução do teste.
6. Desbloquear o aparelho.
7. Interromper o monitor de energia.
8. Coletar os registros gerados pelo monitor de energia e suíte de carga.

5.5.2 Resultado LG Nexus 4

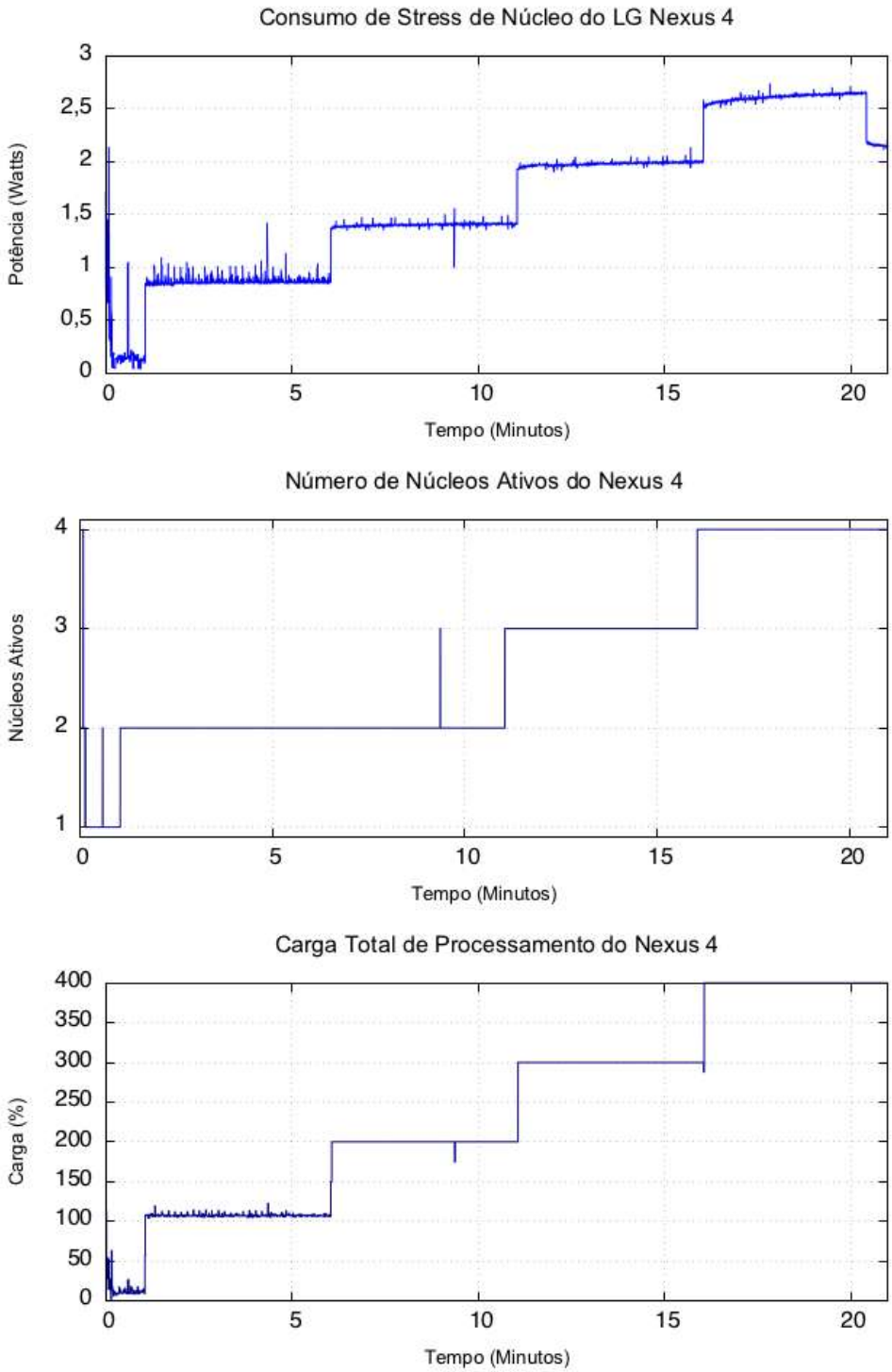

Figura 5.12: Consumo de energia durante o teste de estresse de núcleo do LG Nexus 4.

Na Figura 5.12 é apresenta o consumo dos núcleos em estresse do LG Nexus 4. A cada núcleo adicionado o consumo total cresceu proporcionalmente.

5.5.3 Resultado Xiaomi Redmi 2

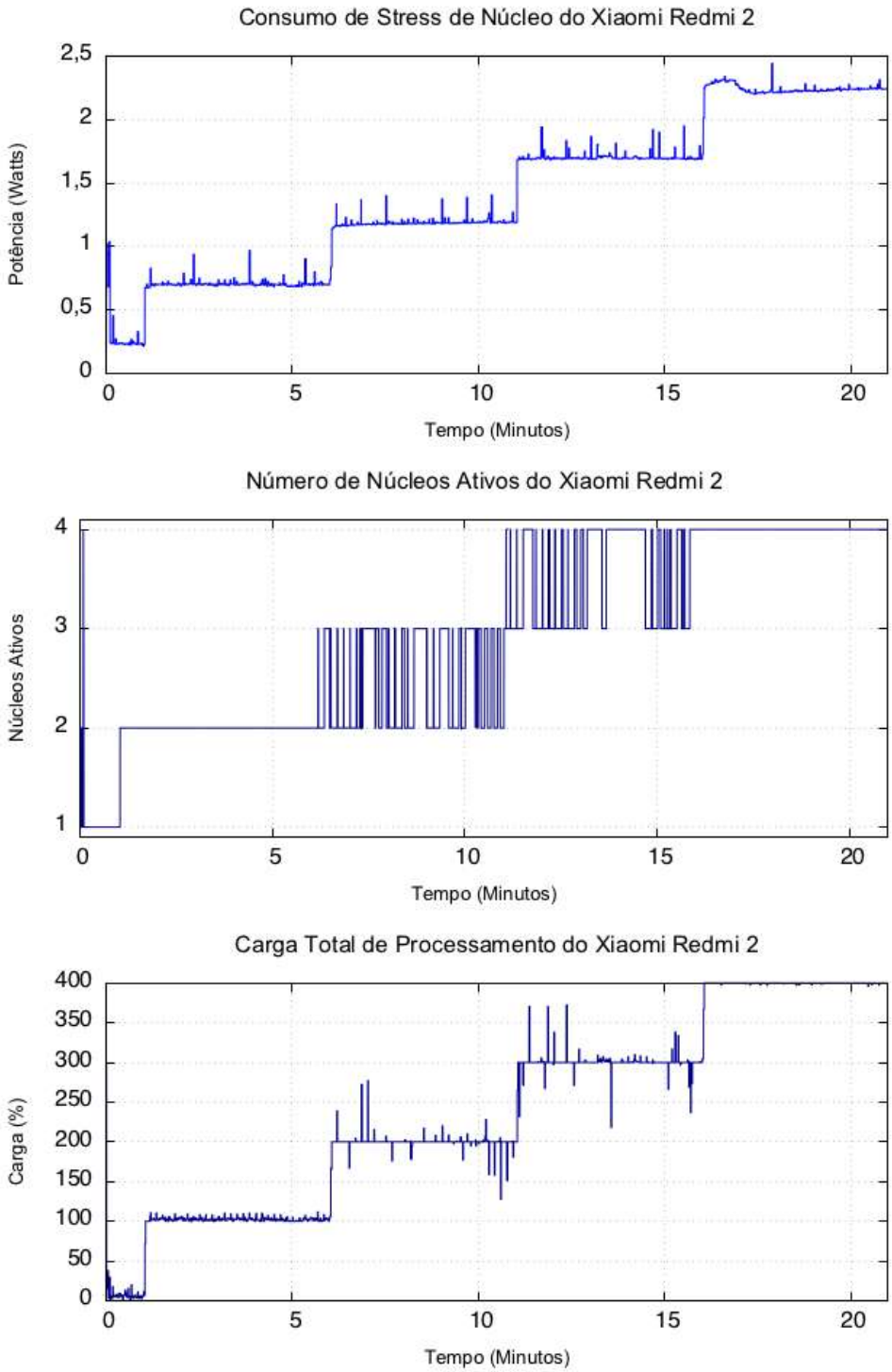

Figura 5.13: Consumo de energia durante o teste de estresse de núcleo do Redmi 2.

Na Figura 5.13 é apresentado o consumo dos núcleos em estresse do Xiaomi Redmi 2. Assim como no LG Nexus 4, o consumo cresceu proporcional ao número de núcleos.

5.5.4 Resultado LG L80

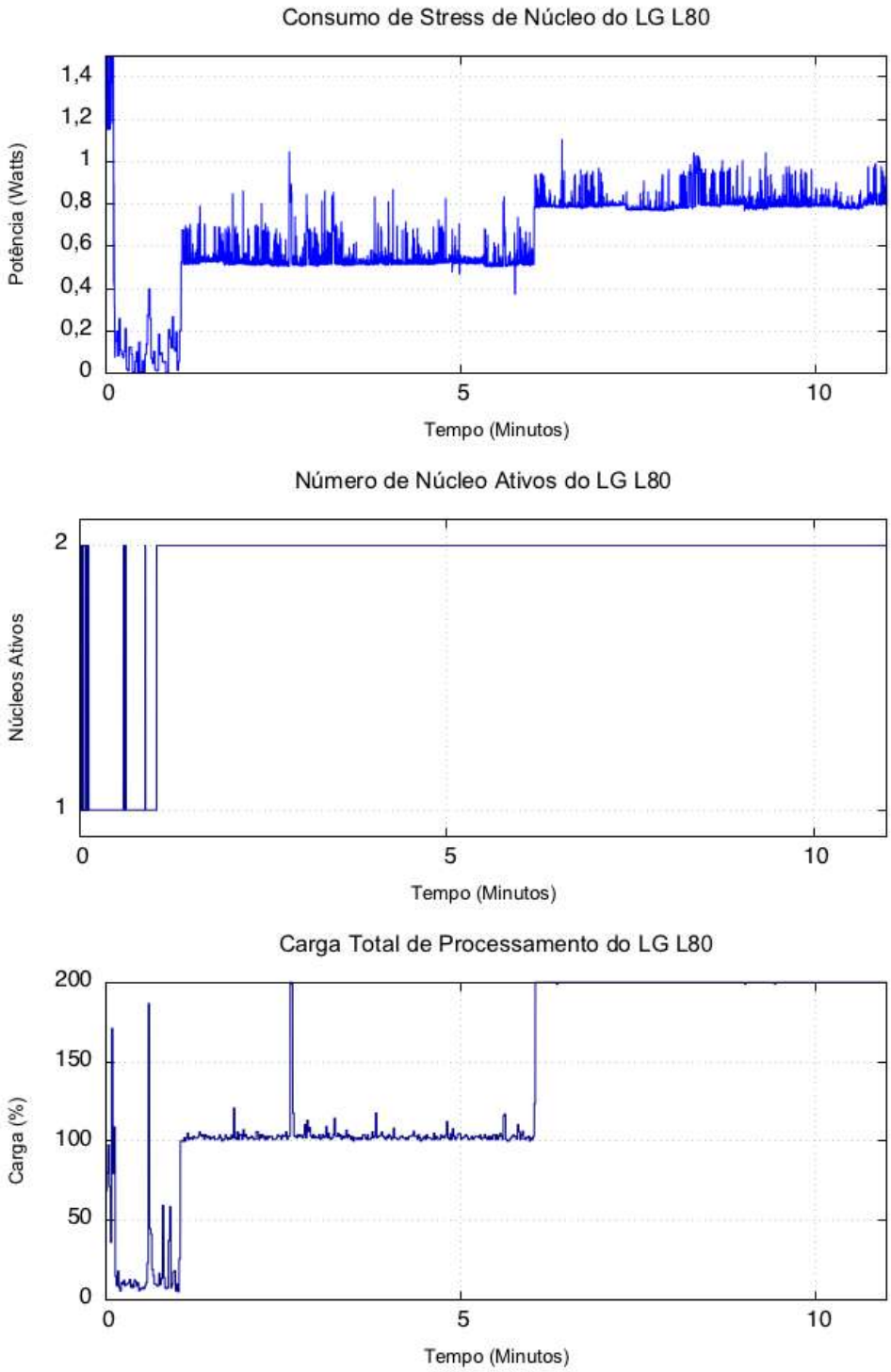

Figura 5.14: Consumo de energia durante o teste de estresse de núcleo do LG L80.

A Figura 5.14 apresenta o consumo dos núcleos em estresse do LG L80. Segue o mesmo comportamento que nos dois anteriores. Entretanto o LG L80 possui apenas 2 núcleos.

5.6 Consumo de rede Wi-Fi

Este teste permite determinar o comportamento de consumo de energia da interface Wi-Fi. Ele usa um servidor TCP escrito em Erlang para mensurar o consumo da transmissão e recepção de dados.

O programa de teste é executado para transmissão ou recepção de 1 KB, 100 KB, 1 MB e 100 MB e 1 GB. O teste é executado dez vezes intervalados por trinta segundos de espera.

5.6.1 Procedimento experimental

Pré-condições:

- Aplicação *Droid Energy Suite* instalada;
- Aparelho em modo avião, mas com Wi-Fi habilitada;
- Aparelho desconectado de fontes externas de energia;
- Bateria com carga acima de 70%;
- Brilho de Tela no mínimo.

Procedimento:

1. Iniciar o monitor de energia;
2. Iniciar a suíte de stress de rede:
    a. Configurar a Taxa de Transmissão em Bytes;
    b. Configurar para 1 minuto de atraso;
    c. Configurar para intervalo de conexão para 30 segundos;
    d. Configurar para realizar o teste 10 vezes;
    e. Clicar no botão executar.
3. Deixar a tela ligada para evitar desligamento automático da interface de rede;
4. Deixar o aparelho sem interação externa durante o tempo de execução do teste;
5. Interromper o monitor de energia;
6. Coletar os registros gerados pelo monitor de energia e da suíte de rede.

5.6.2 Resultado LG Nexus 4

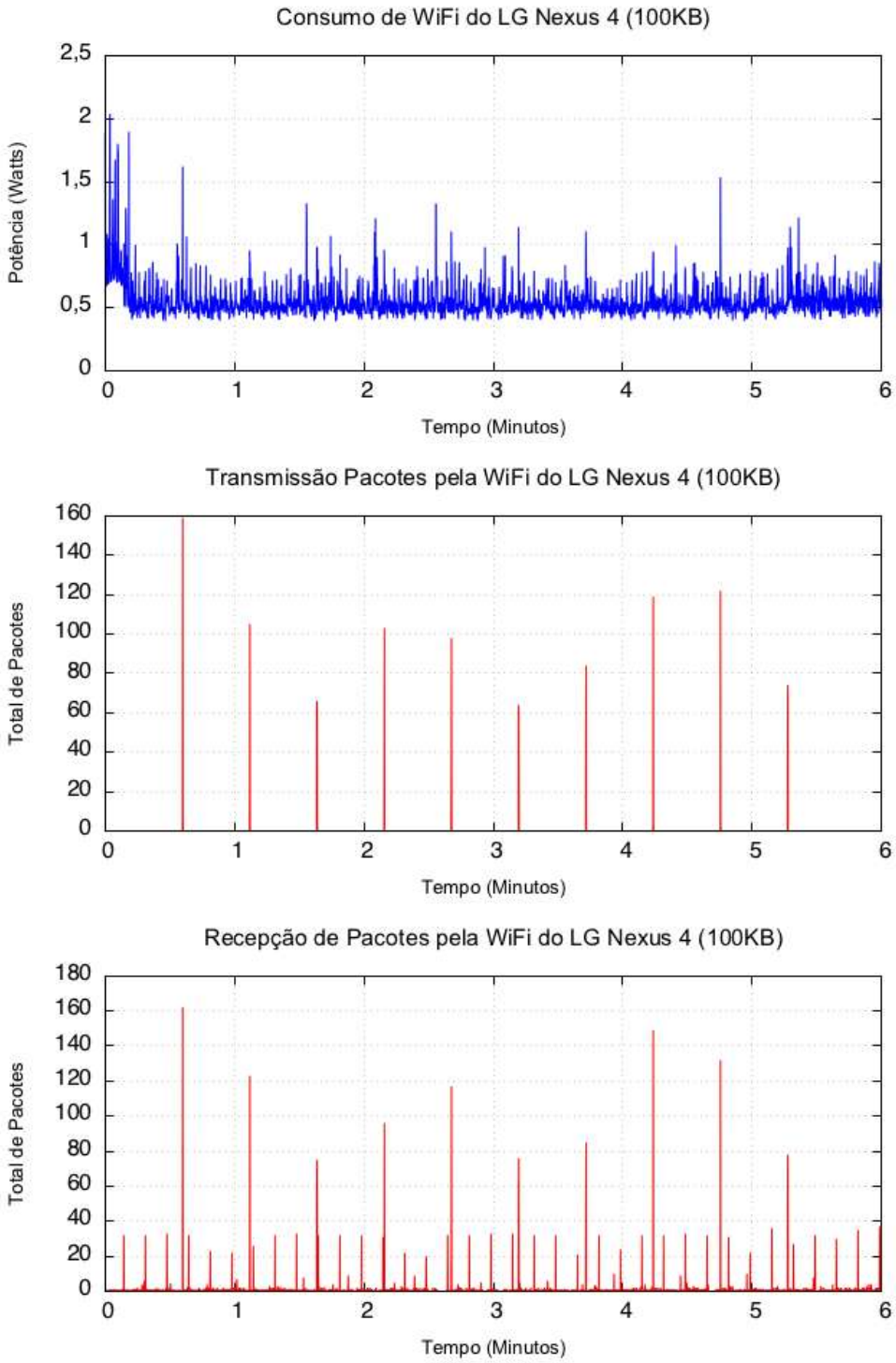

Figura 5.15: Consumo de energia durante o teste de Wi-Fi para 100 KB do LG Nexus 4.

Na Figura 5.15, é possível observar picos de consumo nos instantes de transmissão e recepção de pacotes.

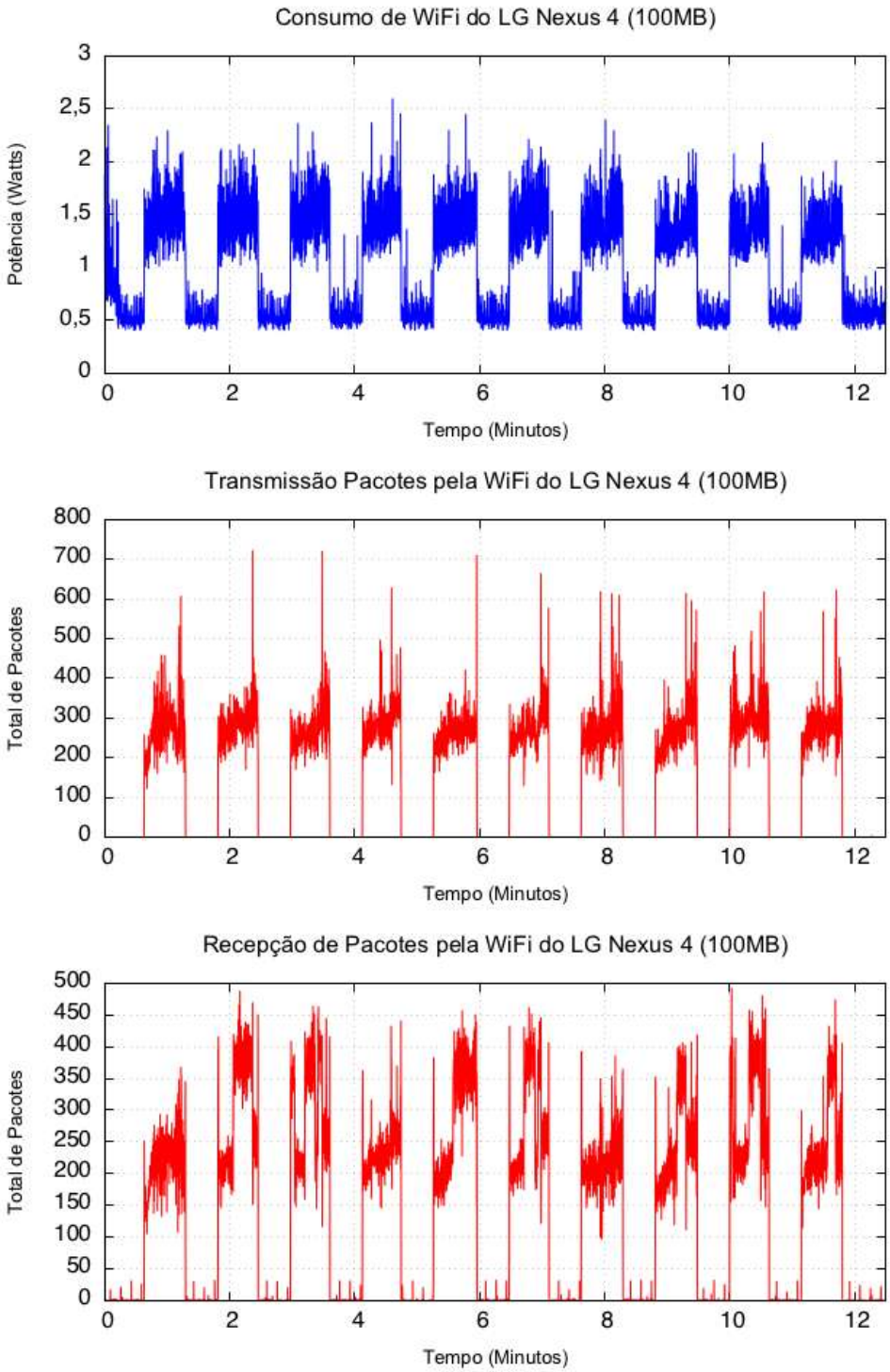

Figura 5.16: Consumo de energia durante o teste de Wi-Fi para 100 MB do LG Nexus 4.

A Figura 5.16 apresenta transmissão mais longa. É possível observar que consumo coincide com a transmissão e recepção. Não existe energia de cauda associada.

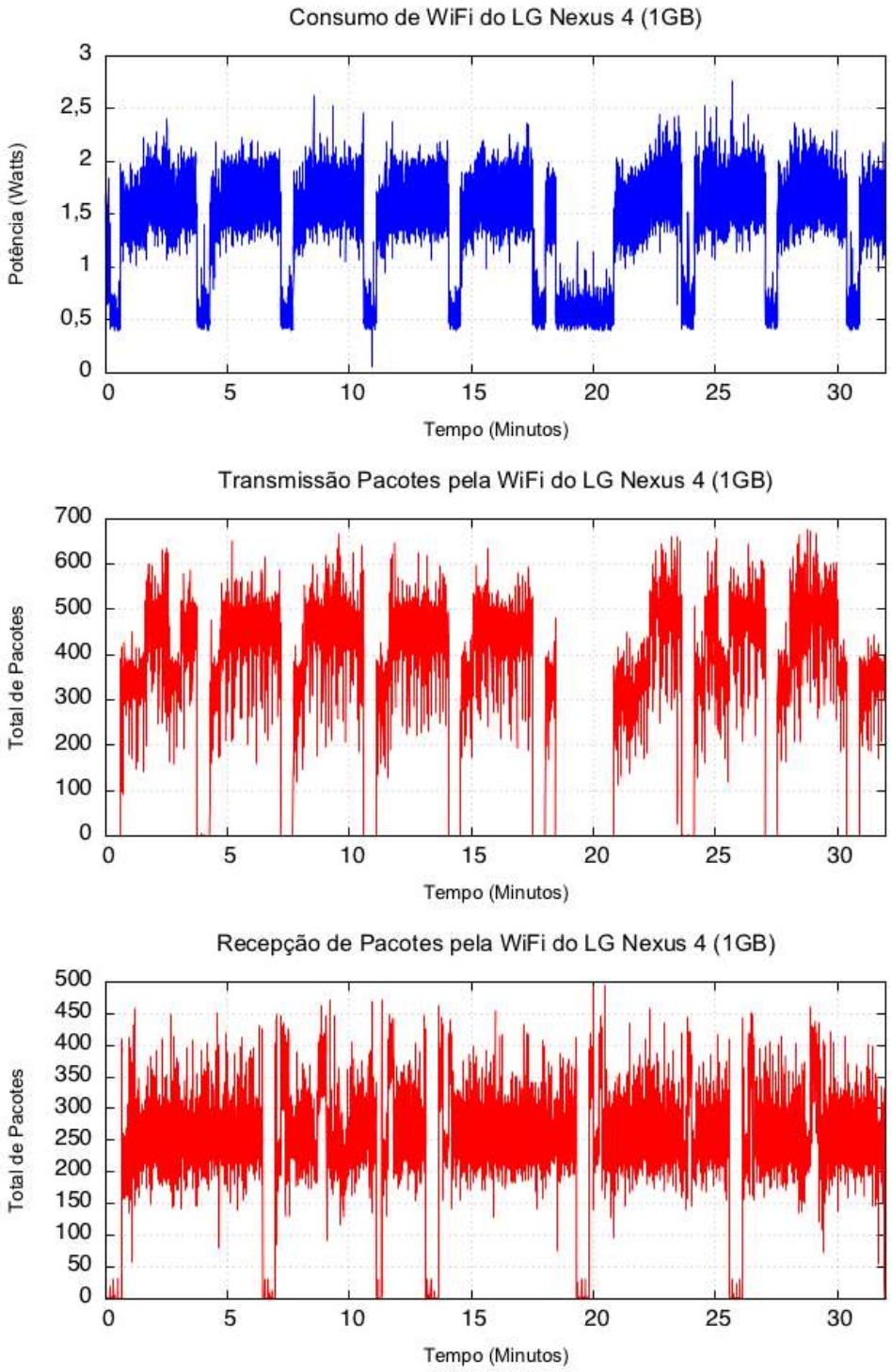

Figura 5.17: Consumo de energia durante o teste de Wi-Fi para 1 GB do LG Nexus 4.

Na Figura 5.17, é apresentada uma sequência de transmissões de 1 GB. O consumo segue o mesmo padrão visto na Figura 5.16.

5.6.3 Resultado Xiaomi Redmi 2

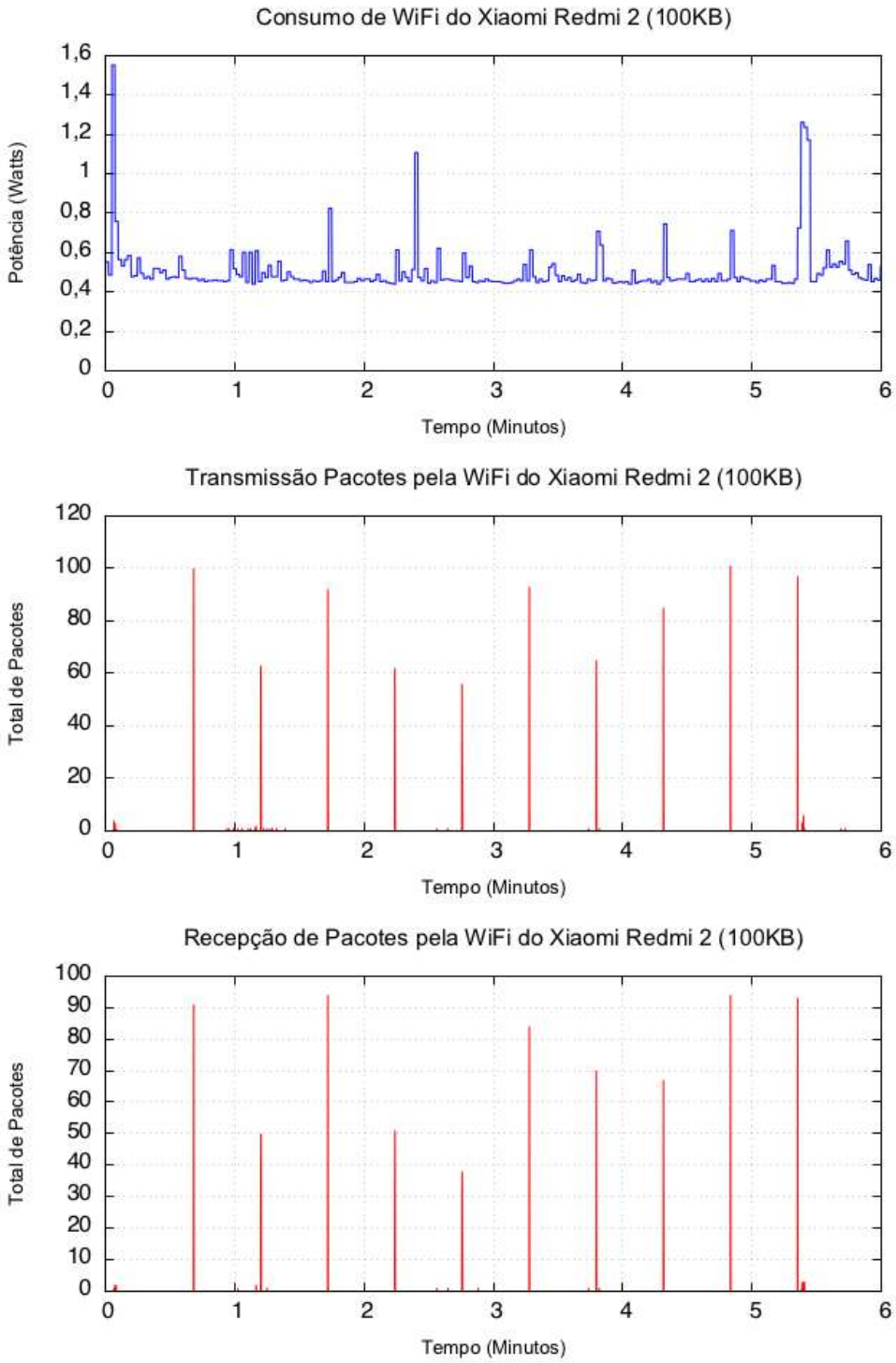

Figura 5.18: Consumo de energia durante o teste de Wi-Fi para 100 KB do Redmi 2.

Na Figura 5.18, é apresentado o experimento no Xiaomi Redmi 2. É difícil visualizar o valor máximo dos picos de consumo devido a limitações da interface da bateria.

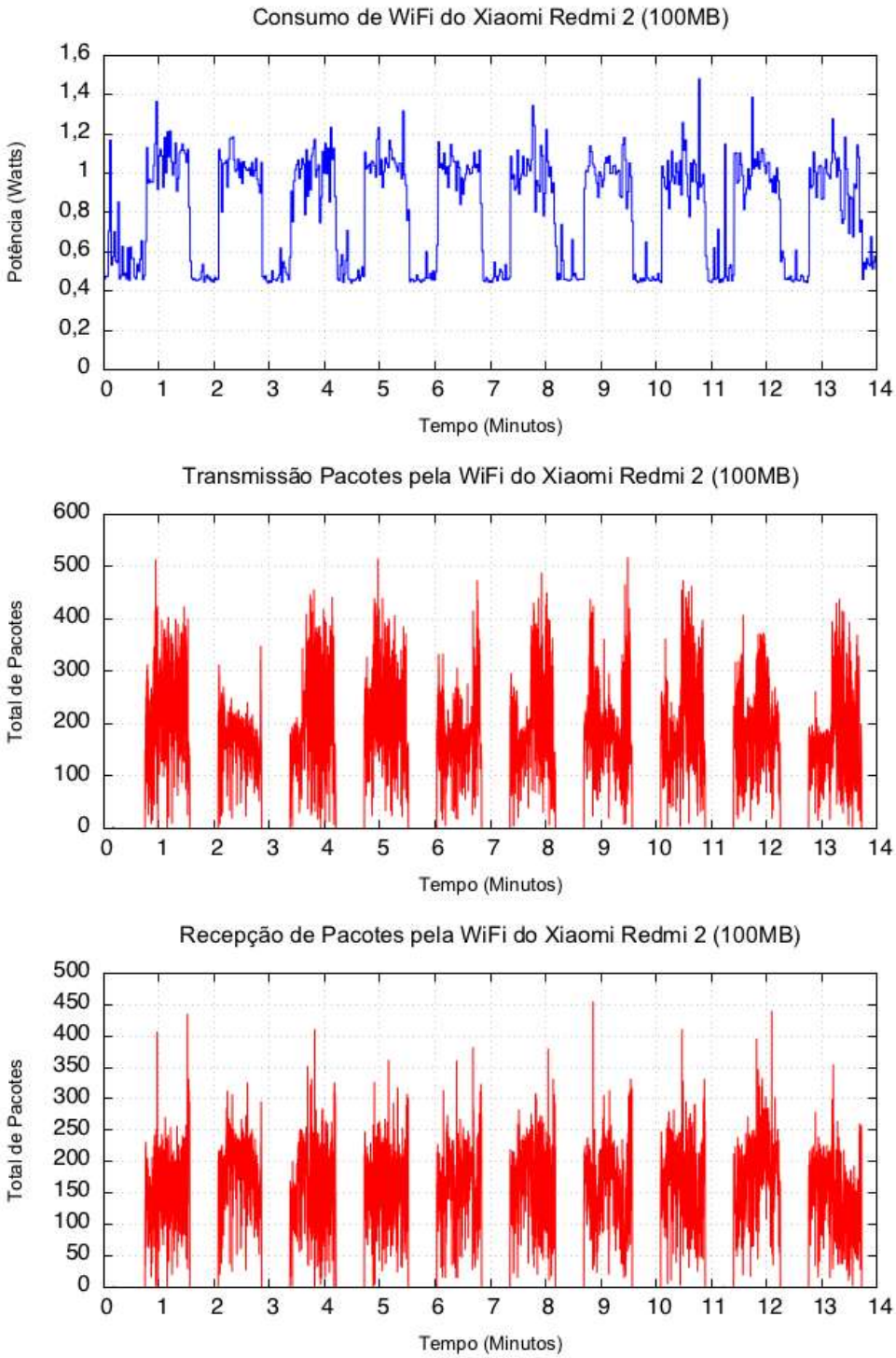

Figura 5.19: Consumo de energia durante o teste de Wi-Fi para 100 MB do Redmi 2.

Na Figura 5.19, é possível visualizar melhor o consumo de energia devido à duração maior do teste. O consumo foi diretamente proporcional à transmissão e recepção de pacotes.

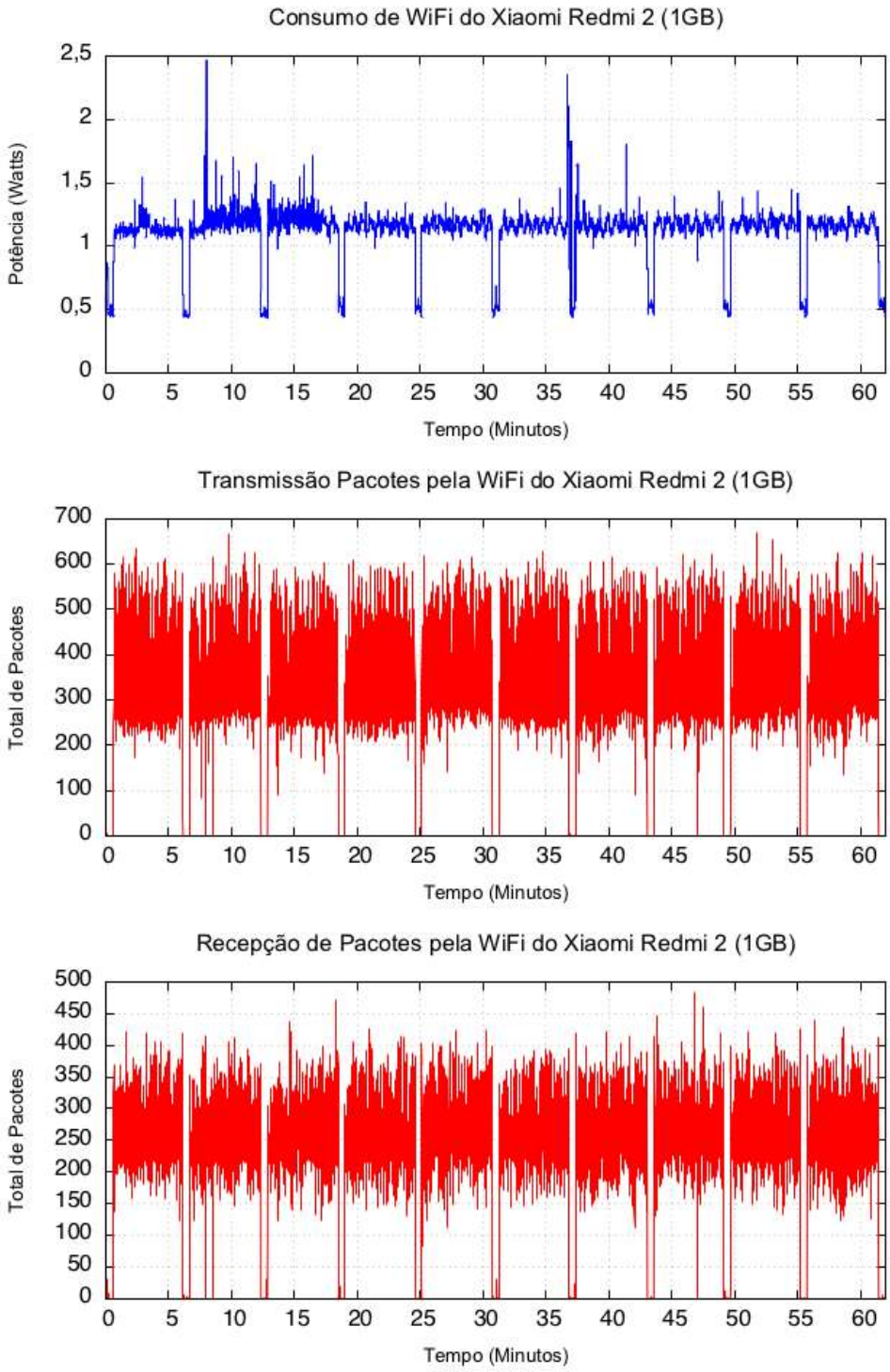

Figura 5.20: Consumo de energia durante o teste de Wi-Fi para 1 GB do Redmi 2.

Na Figura 5.20, o consumo para a transmissão e recepção de 1 GB. O consumo e transmissão são saturados em valor máximo.

5.6.4 Resultado LG L80

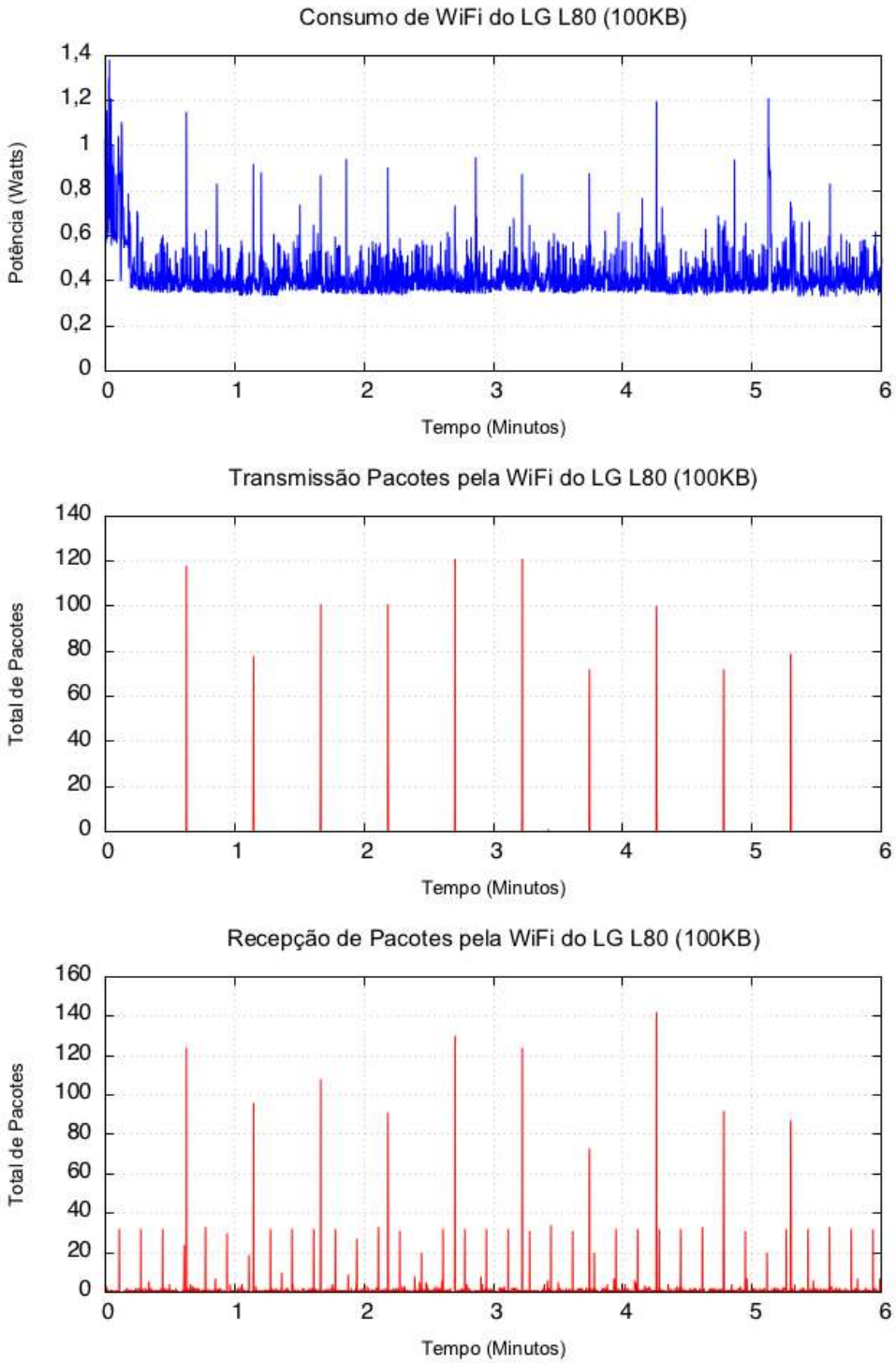

Figura 5.21: Consumo de energia durante o teste de Wi-Fi para 100 KB do LG L80.

Na Figura 5.21, é apresentado o consumo da interface Wi-Fi no LG L80. Similar ao LG Nexus 4 e Xiaomi Redmi 2, picos de consumo são observados nos instantes de transmissão e recepção de pacotes.

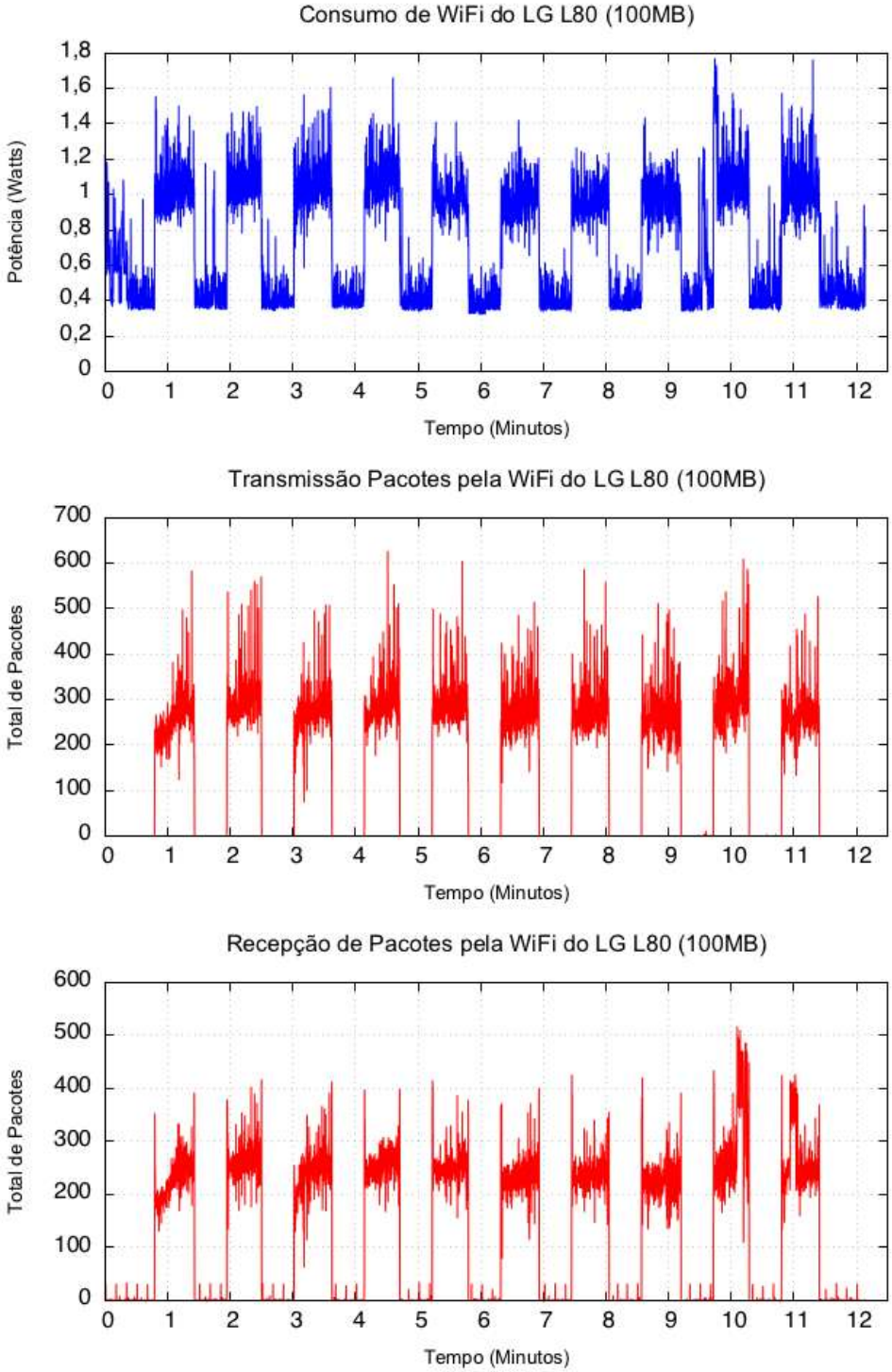

Figura 5.22: Consumo de energia durante o teste de Wi-Fi para 100 MB do LG L80.

Na Figura 5.22, é apresentado o consumo para 100 MB de transmissão. O consumo coincide com os momentos de transmissão e recepção. Indicando que o mecanismo de economia de energia (PSM) do Wi-Fi é bastante eficiente.

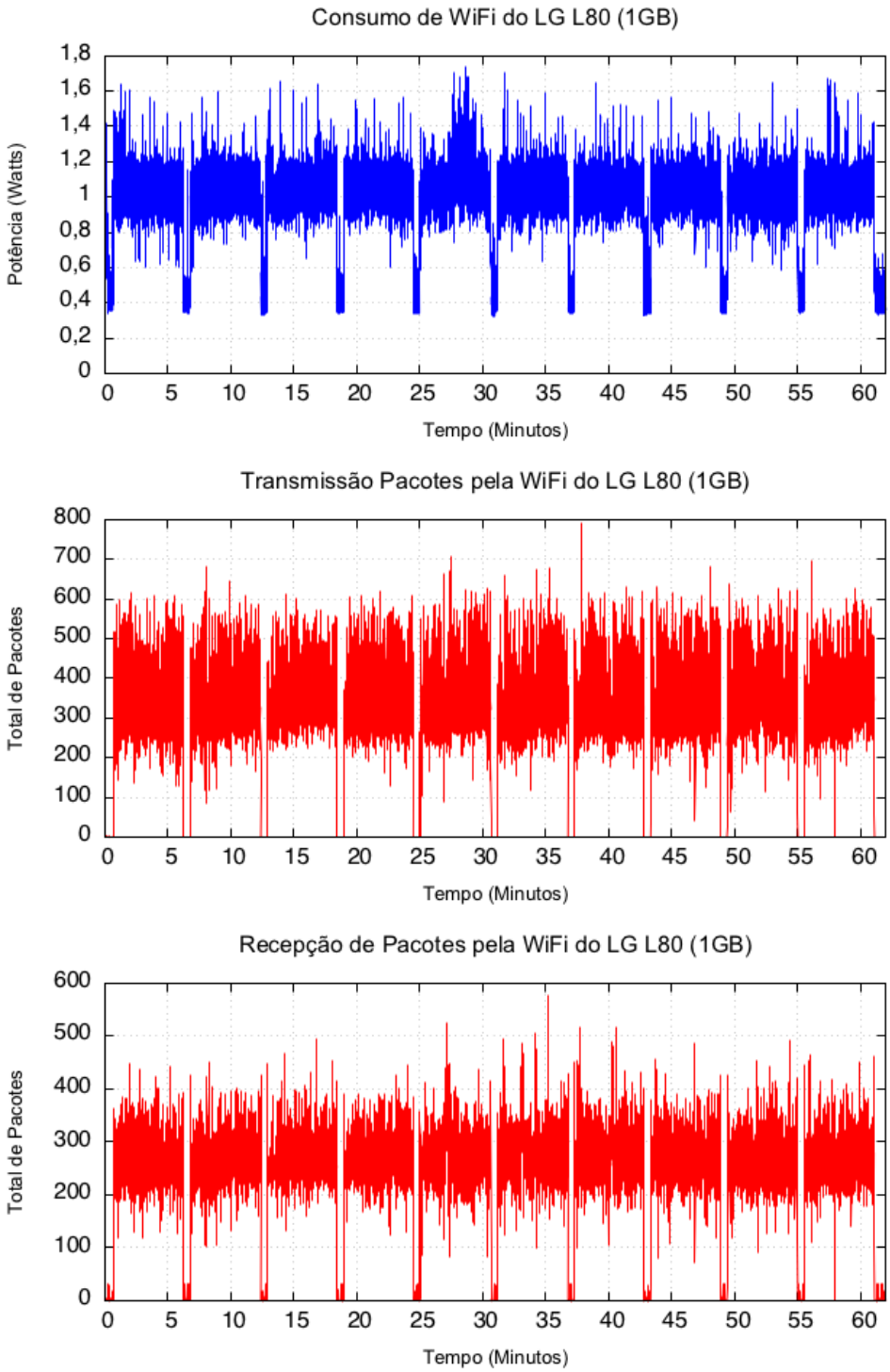

Figura 5.23: Consumo de energia durante o teste de Wi-Fi para 1 GB do LG 80.

A Figura 5.23 apresenta o teste para 1 GB de transmissão. Observa-se que característica é a mesma da vista na Figura 5.22.

5.7 Consumo de Rede 3G

Este teste é similar ao da interface Wi-Fi. O programa de teste é executado para transmissão ou recepção de 100 KB, 1 MB. O teste é executado dez vezes intervalados por trinta segundos de espera.

5.7.1 Procedimento experimental

Pré-condições:

- Aplicação *Droid Energy Suite* instalada;
- Aparelho com interface Wi-Fi desabilitada;
- Aparelho desconectado de fontes externas de energia;
- Bateria com carga acima de 70%;
- Brilho de tela no mínimo.

Procedimento:

1. Iniciar o monitor de energia;
2. Iniciar a suíte de stress de rede:
    a. Configurar a Taxa de Transmissão em Bytes;
    b. Configurar para 1 minuto de atraso;
    c. Configurar para intervalo de conexão para 1 minuto;
    d. Configurar para realizar o 10 vezes;
    e. Clicar no botão executar.
3. Deixar a tela ligada para evitar desligamento automático da interface de rede;
4. Deixar o aparelho sem interação externa durante o tempo de execução do teste;
5. Interromper o monitor de energia;
6. Coletar os registros gerados pelo monitor de energia e da suíte de rede.

5.7.2 Resultado LG Nexus 4

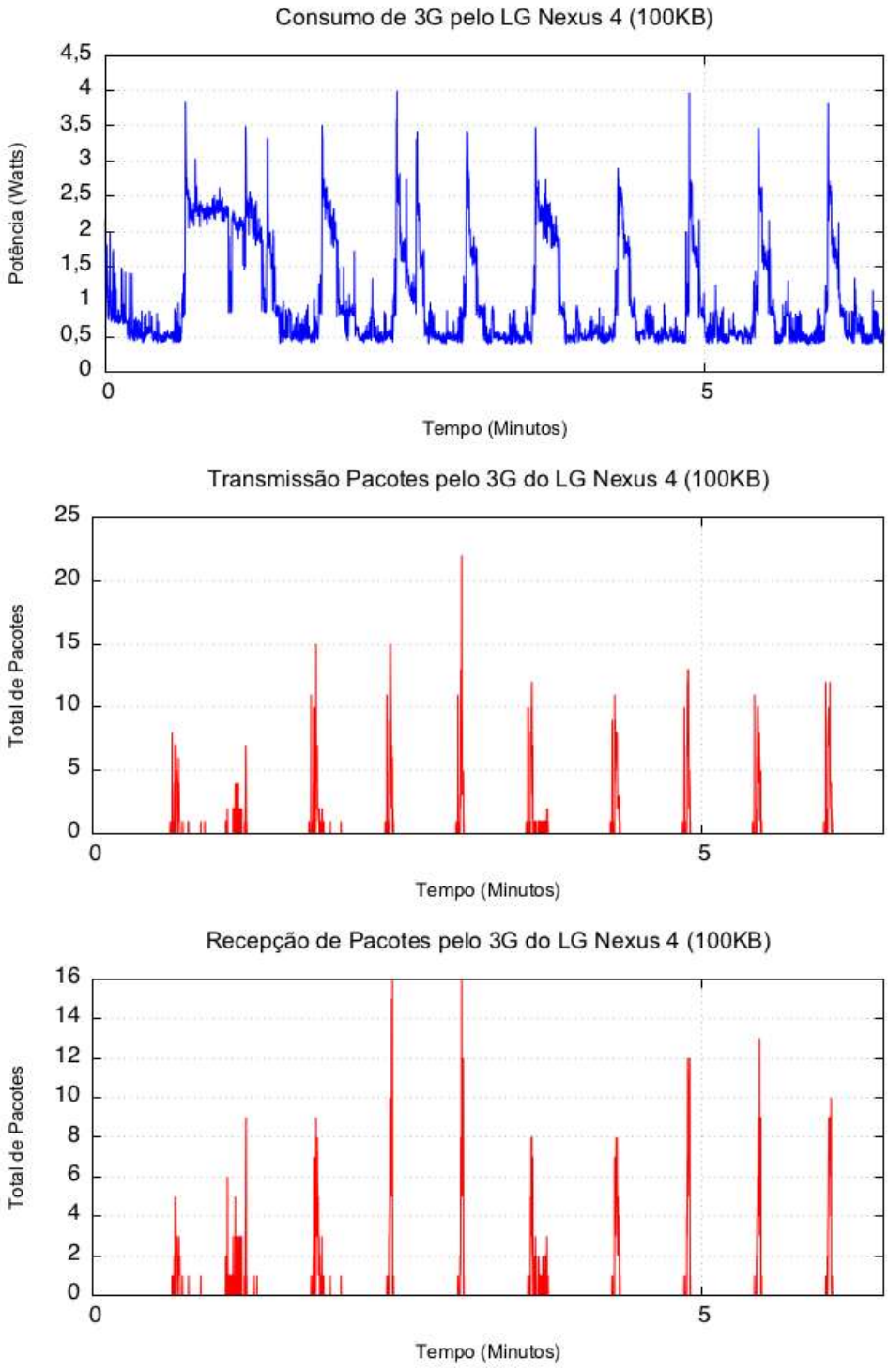

Figura 5.24: Consumo de energia durante o teste de 3G para 100 KB do LG Nexus 4.

Na Figura 5.24, é apresentado o consumo usando a interface 3G. Diferente do Wi-Fi, existe energia de cauda após o término da transmissão.

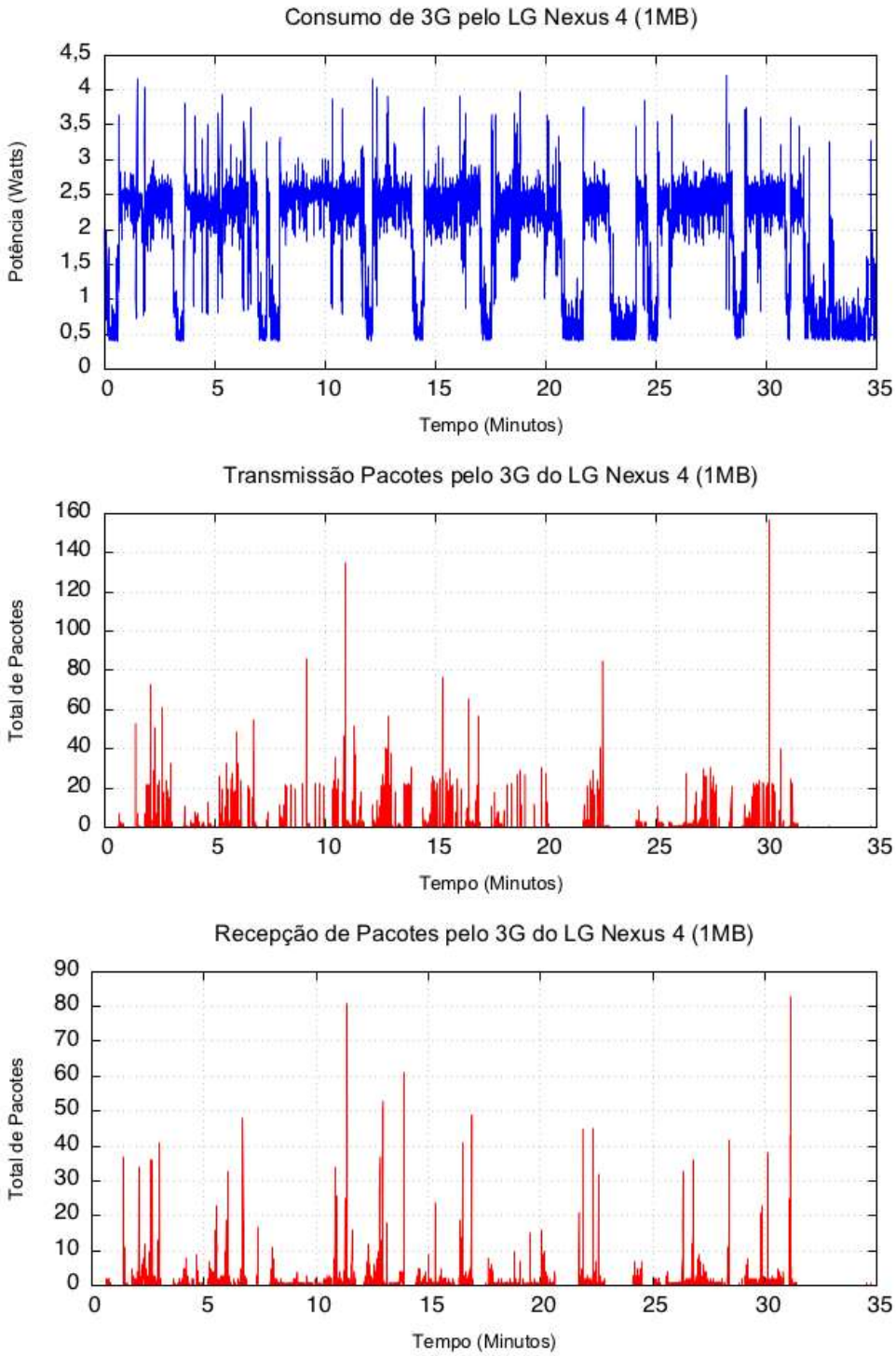

Figura 5.25: Consumo de energia durante o teste de 3G para 1 MB do LG Nexus 4.

Na Figura 5.25, é apresentado o consumo para a transmissão de 1 MB. Nesse teste, a qualidade do sinal 3G e grau de congestionamento afetam o tempo de transmissão e, consequentemente, a energia consumida.

5.7.3 Resultado Xiaomi Redmi 2

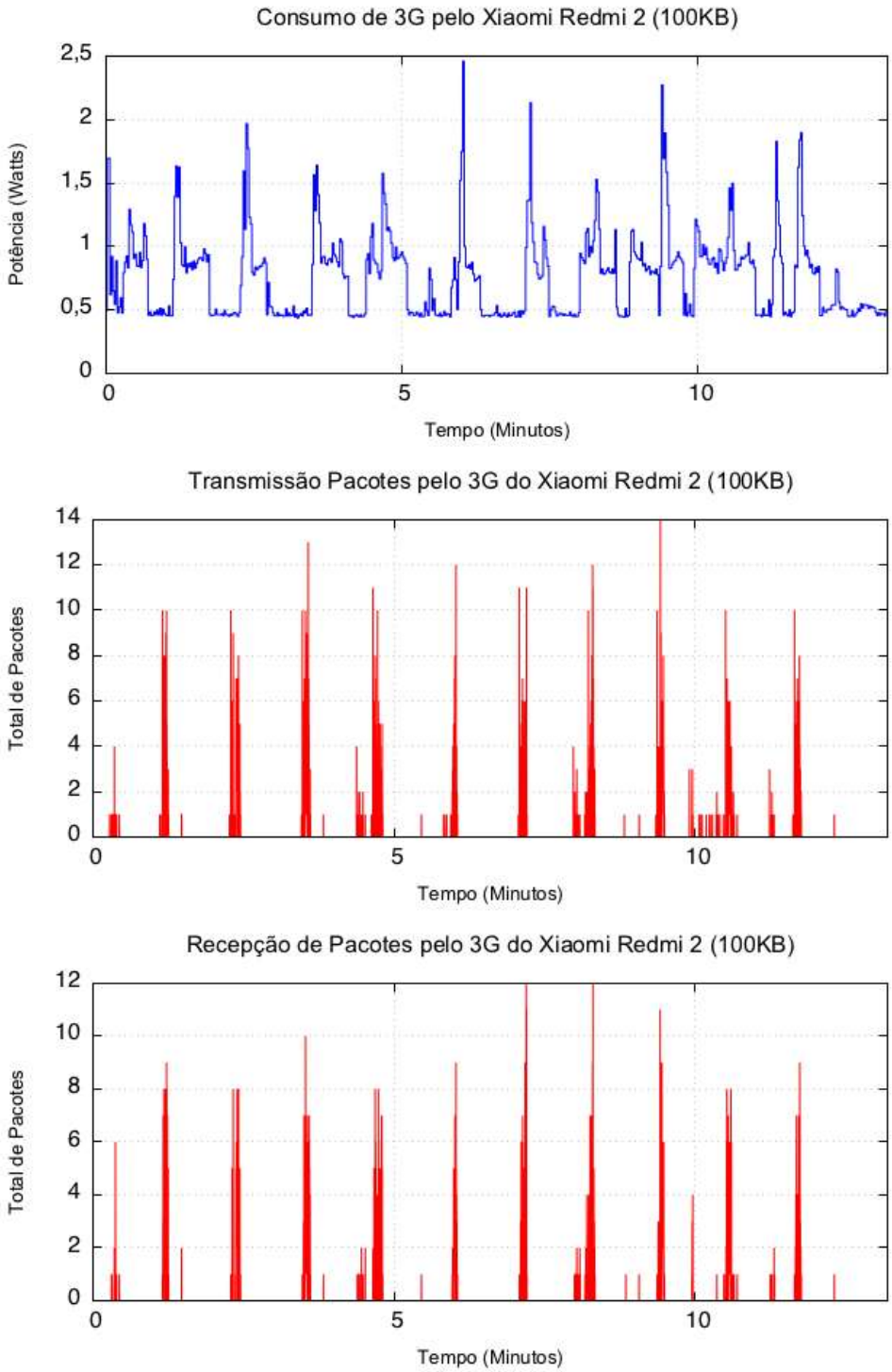

Figura 5.26: Consumo de energia durante o teste de 3G para 100 KB do Xiaomi Redmi 2.

Na Figura 5.26, é apresentado o consumo do Xiaomi Redmi 2. É possível notar três fases de consumo. Um pico durante a transmissão, uma cauda e espera.

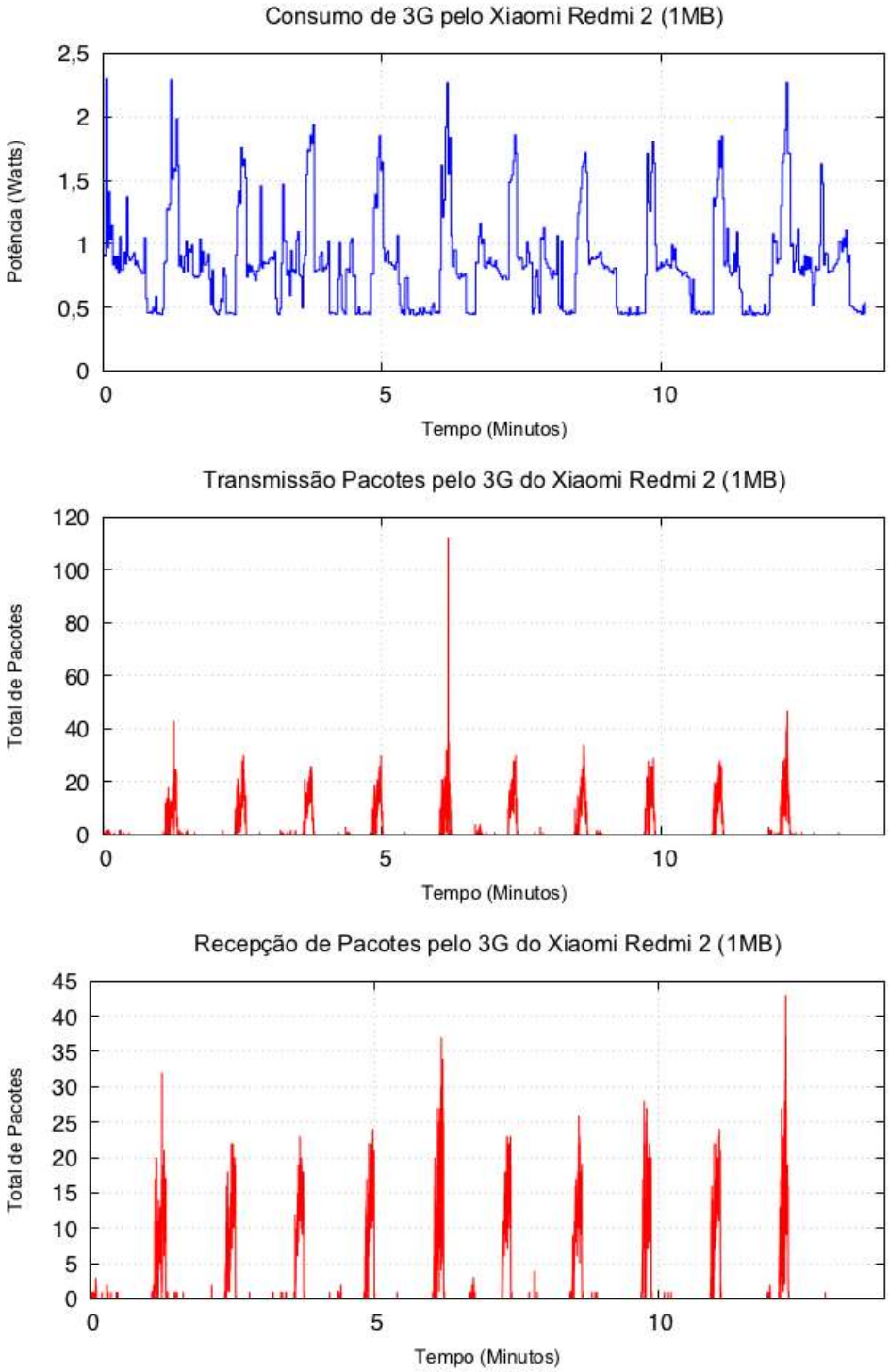

Figura 5.27: Consumo de energia durante o teste de 3G para 1 MB do Xiaomi Redmi 2.

Na Figura 5.27, é apresentado o consumo para 1 MB. Assim como na Figura 5.26, a curva segue a mesma característica de consumo.

5.7.4 Resultado LG L80

O teste da interface 3G não foi realizado no LG L80 devido à indisponibilidade de cartão SIM para realização do experimento.

6 CONSTRUÇÃO DO MODELO DE ENERGIA

A partir dos resultados experimentais apresentados no capítulo 5 foi construído um modelo de consumo de energia seguindo como base o estado da arte que foi apresentado no capítulo 3.

O modelo foi construído seguindo um processo incremental. Primeiro foi determinado o consumo de base da plataforma, isto é, o consumo constante necessário para manter o sistema ativo. Em seguida, foi determinada a energia consumida pelo processador em função da sua frequência de operação e nível de atividade. O consumo da tela LCD foi isolado removendo o consumo estimado do processador e base aplicando a Equação (*6-1*). O mesmo foi feito para determinar o consumo da interface Wi-Fi e 3G. Para a interface 3G foi necessário considerar a duração da energia de cauda.

$$P_{componente} = P_{mensurado} - P_{estimado} \quad (6\text{-}1)$$

Obtido o consumo de cada componente, foi então aplicada regressão linear usando a ferramenta R para determinar os coeficientes. Veremos, a seguir, os modelos produzidos e uma discussão sobre os resultados observados.

6.1 Construção do modelo de base

O modelo de base consiste do consumo em hibernação e consumo em espera. A Tabela 6-1 contém os resultados dos experimentos apresentados na seção 5.2.

Tabela 6-1: Resumo dos resultados experimentais do consumo de base.

Aparelhos	**Espera**		**Hibernação**
	Média	*Desvio Padrão*	*Mínimo*
LG Nexus 4	130 mW	51 mW	18 mW
Xiaomi Redmi 2	87 mW	43 mW	21 mW
LG L80	60 mW	51 mW	2 mW

A potência base P_{base} consiste de dois possíveis estados: hibernação ou espera, Equação (*6-2*), ambos constantes. O coeficiente β_{espera} é estimado pelo valor médio da potência mensurada durante o teste de consumo de energia, Equação (*6-3*). O coeficiente $\beta_{hibernação}$ é estimado indiretamente pelo valor mínimo do consumo mensurado durante a hibernação, Equação (*6-4*).

$$P_{base} = \begin{cases} \beta_{espera}, caso\ aparelho\ em\ espera \\ \beta_{hibernação}, caso\ aparelho\ em\ hibernaçõa \end{cases} \quad (6\text{-}2)$$

$$\beta_{espera} = Consumo\ Médio\ em\ Espera \quad (6\text{-}3)$$

$$\beta_{hibernação} = Consumo\ Mínimo\ em\ Hibernação \quad (6\text{-}4)$$

6.2 Construção do modelo do processador

O modelo de processador foi definido conforme estado da arte da seção 3.3 e os resultados vistos nas seções 5.4 e 5.5. O modelo empregado, Equação (*6-5*), considera a potência do processador como o somatório das potências de cada um dos núcleos.

A potência de um núcleo é composta de duas partes, Equação (*6-6*). A parte dinâmica que depende da frequência e carga de uso. E parte estática que depende apenas da frequência de operação. Quanto maior a frequência e carga de processamento, maior é o consumo.

$$P_{processador} = \sum_{i}^{N} P_{núcleo_i} \quad (6\text{-}5)$$

$$P_{núcleo} = P_{dinâmico} + P_{estático} \quad (6\text{-}6)$$

$$P_{dinâmico} = \alpha_{processador}(f) U_{CPU} \quad (6\text{-}7)$$

$$P_{estático} = \beta_{processador}(f) \quad (6\text{-}8)$$

A análise por regressão linear do consumo em função da frequência e carga de processamento foi aplicada nos resultados dos experimentos vistos na seção 5.4. No modelo linear, Equação *(6-9)*, U_{cpu} é a carga de processamento de 0 a 100; $\alpha_{processador}$ e $\beta_{processador}$ o coeficiente angular e linear da reta. Para cada frequência exercitada no experimento, foi calculado cada coeficiente. Os resultados para o LG Nexus 4 são apresentados na

Tabela 6-2. Os resultados para o Xiaomi Redmi 2 na Tabela 6-3.

$$y(x) = \alpha x + \beta \quad (6\text{-}9)$$

Tabela 6-2: Coeficientes lineares obtidos por regressão do teste de carga do LG Nexus 4.

Frequência (MHz)	**Coeficiente Angular (α)**	**Coeficiente Linear (β)**
384	1,064	124,138
486	1,921	138,611
594	2,281	124,1765
7021	2,515	134,624
810	3,286	129,454
918	3,767	123,753
1026	4,326	134,007
1134	5,245	176,884
1242	6,167	154,518
1350	7,238	145,409
1458	7,421	179,073
1512	8,117	165,311

Tabela 6-3: Resumo dos coeficientes lineares obtidos por regressão do teste de carga do Xiaomi Redmi 2.

Frequência (MHz)	**Coeficiente Angular (α)**	**Coeficiente Linear (β)**
200	1,049	63,927
400	1,030	76,351
533	1,421	79,526
800	2,061	73,706
998	3,430	87,026
1094	3,741	82,569
1152	3,930	100,259
1209	3,978	103,332

É possível observar nos resultados que os valores dos coeficientes angulares e lineares são dependentes da frequência de operação do núcleo. Quanto maior a frequência de operação, maiores os coeficientes.

O teste de stress de núcleo da seção 5.5 permitiu identificar o consumo máximo do sistema por número de núcleos ativos. O resultado da média e desvio padrão na Tabela 6-4 e Tabela 6-5, respectivamente.

Tabela 6-4: Média de consumo observado durante o teste de stress de núcleo.

Núcleos	**1**	**2**	**3**	**4**
LG Nexus 4	865,62 mW	1403,518 mW	1979,99 mW	2610,884 mW
Xiaomi Redmi 2	707,479 mW	1190,414 mW	1704,56 mW	2242,127 mW
LG L80	543,782 mW	806,7756 mW	-	-

Tabela 6-5: Desvio padrão observado durante o teste de stress de núcleo.

Núcleos	**1**	**2**	**3**	**4**
LG Nexus 4	27,017 mW	14,865 mW	16,138 mW	30,618 mW
Xiaomi Redmi 2	33,851 mW	36, 621 mW	40,051 mW	36,524 mW
LG L80	59,687 mW	47,690 mW	-	-

6.3 Construção do modelo da tela LCD

O modelo de consumo da tela foi definido conforme os resultados obtidos nos experimentos vistos na seção 3.4 e o estado da arte apresentado na seção 3.4. Foram escolhidos dois modelos diferentes de consumo de energia. Um modelo linear para o LG Nexus 4, Equação (*6-10*). Um modelo não linear para o Xiaomi Redmi 2 e LG L80, Equação (*6-11*).

$$y(x) = \alpha x + \beta \quad (6\text{-}10)$$

$$y(x) = \alpha x^3 + \beta \quad (6\text{-}11)$$

A motivação pelo uso de dois modelos pode ser vista na Figura 6.1 e Figura 6.2. O comportamento de consumo da tela do LG Nexus 4 é melhor representado usando o modelo linear, enquanto nos outros dois aparelhos usados o modelo não linear é mais apropriado.

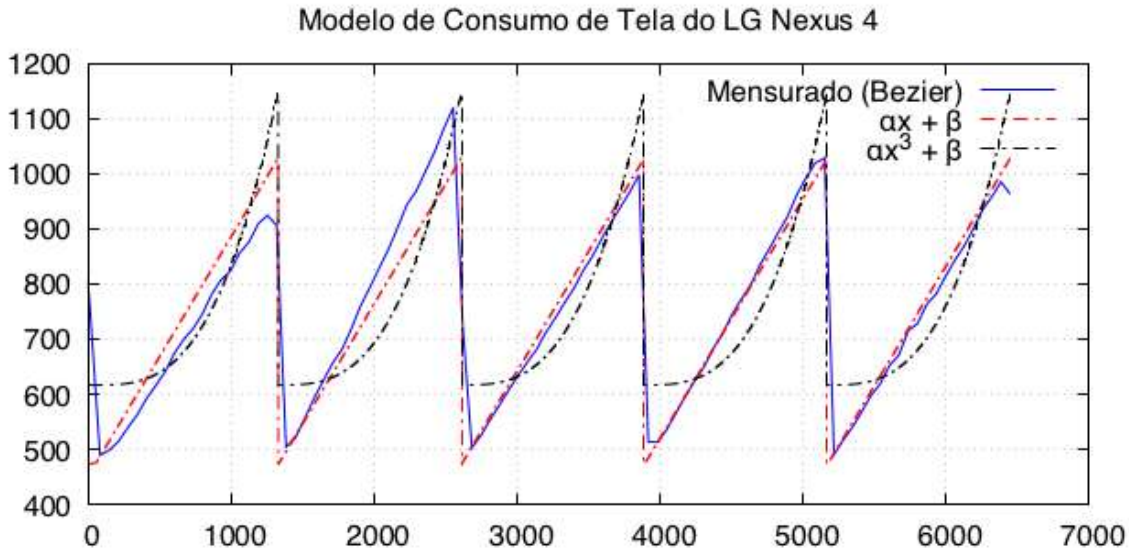

Figura 6.1: Comparação entre consumo medido (amortizado por curva de Bezier), modelo linear (6-10) *e não linear* (6-11) *do LG Nexus 4.*

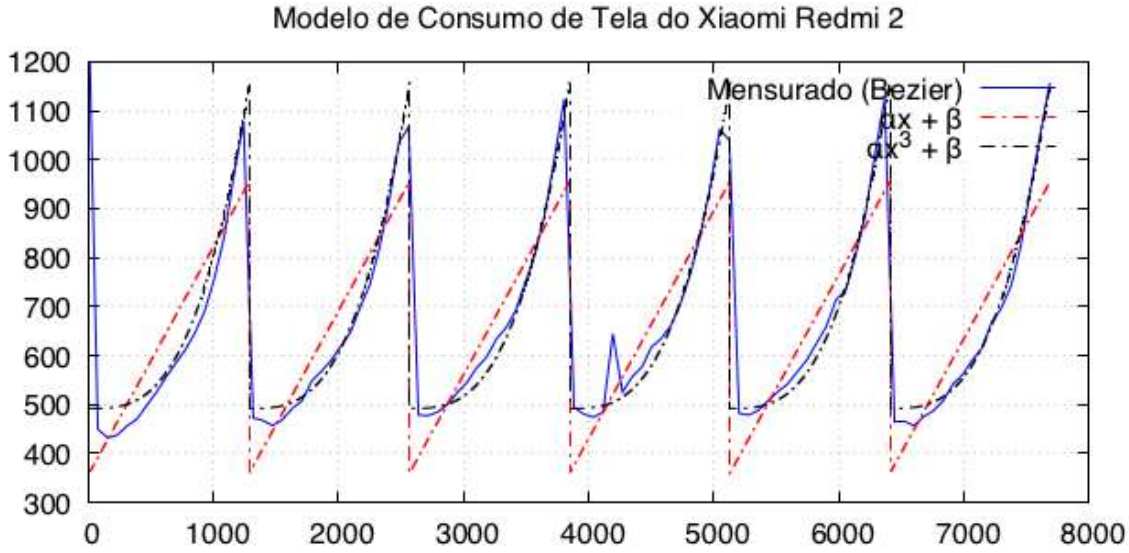

Figura 6.2: Comparação entre consumo medido (amortizado por curva de Bezier), modelo linear (6-10) *e não linear* (6-11) *do Xiaomi Redmi 2.*

Os coeficientes calculados por regressão dos dois modelos são apresentados na Tabela 6-6 e Tabela 6-7.

Tabela 6-6: Coeficiente da regressão linear pela Equação(6-10).

	α	β
LG Nexus 4	2,174	473,1

Tabela 6-7: Coeficientes da regressão não linear pela Equação (6-11).

	α	β
Xiaomi Redmi 2	4,006e-05	491,9
LG L80	3.470e-05	381,3

6.4 Construção do modelo de rede Wi-Fi

O modelo de consumo da interface Wi-Fi foi definido conforme o estado da arte apresentado na seção 3.5 e os resultados da seção 5.6. É uma máquina de estados com dois estados, Equação (*6-12*). A descrição dos símbolos é apresentada na Tabela 6-8.

$$P_{wifi} = \begin{cases} Pcts_{env_{wifi}}\,\beta_{env_{wifi}} + Pcts_{rec_{wifi}}\,\beta_{rec_{wifi}} + \beta_{base_{wifi_{ativo}}} & \textit{,se ativo} \\ \beta_{base_{wifi_{espera}}} & \textit{,se espera} \end{cases} \quad (6\text{-}12)$$

Tabela 6-8: Descrição dos símbolos da Equação (6-12).

Símbolo	**Descrição**
P_{wifi}	Potência da interface Wi-Fi.
$Pcts_{env_{wifi}}$	Número de pacotes enviados pela Wi-Fi no intervalo de amostragem.
$Pcts_{rec_{wifi}}$	Número de pacotes recebidos pela Wi-Fi no intervalo da amostragem.
$\beta_{env_{wifi}}$	Coeficiente angular associado ao envio de pacotes.
$\beta_{rec_{wifi}}$	Coeficiente angular associado a recepção de pacotes.
$\beta_{base_{wifi_{ativo}}}$	Coeficiente linear associado ao estado ativo da interface Wi-Fi.
$\beta_{base_{wifi_{espera}}}$	Coeficiente linear associado ao estado de espera da interface Wi-Fi.

O consumo durante o estado ativo é determinado pelo número de pacotes enviados e recebidos. Quando a transmissão e recepção são concluídos, a interface Wi-Fi alterna para o estado de espera. Nesse estado, o consumo é mínimo e constante.

6.5 Construção do modelo de rede 3G

O modelo de consumo da interface 3G foi definido conforme o estado da arte da seção 3.6 e resultados do experimento da seção 5.7. O mesmo procedimento que foi aplicado na seção 6.4 foi aplicado para rede 3G. Porém, diferentemente da interface Wi-Fi, foi necessário adotar um estado a mais para representar o consumo de cauda.

O modelo adotado, Equação (*6-13*), define o consumo durante a transmissão, a partir dos pacotes enviados ou transmitidos. Após o término da comunicação, o consumo de cauda se segue durante um período de alguns segundos. Caso novo pacote seja enviado ou transmitido, o consumo retorna para o primeiro estado. Caso contrário, o consumo é reduzido para o valor de espera. Na Tabela 6-9 são discriminados os significados dos símbolos da Equação (*6-13*).

$$P_{3G} = \begin{cases} Pcts_{env_{3G}}\,\beta_{env_{3G}} + Pcts_{rec_{3G}}\beta_{rec_{3G}} + \beta_{base_{3G_{ativo}}} & , se\ ativo \\ \beta_{base_{3G_{cauda}}} & , se\ cauda \\ \beta_{base_{3G_{espera}}} & , se\ espera \end{cases} \quad (6\text{-}13)$$

Tabela 6-9: Descrição dos símbolos da Equação (6-13).

Símbolo	**Descrição**
P_{3G}	Potência da interface 3G.
$Pcts_{env_{3G}}$	Número de pacotes enviados pelo 3G durante o intervalo de amostragem.
$Pcts_{rec_{3G}}$	Número de pacotes recebidos pelo 3G durante o intervalo da amostragem.
$\beta_{env_{3G}}$	Coeficiente angular associado ao envio de pacotes pela interface 3G.
$\beta_{rec_{3G}}$	Coeficiente angular associado a recepção de pacotes pela interface 3G.
$\beta_{base_{3G_{ativo}}}$	Coeficiente linear associado ao estado ativo da interface 3G.
$\beta_{base_{3G_{cauda}}}$	Coeficiente linear associado ao estado de cauda da interface 3G.
$\beta_{base_{3G_{espera}}}$	Coeficiente linear associado ao estado de espera da interface 3G.

6.6 Modelo completo

O consumo do sistema, Equação (*6-14*), apresenta dois estados: ativo e inativo. O sistema está ativo quando está processando informação, enquanto, no inativo, o sistema está suspenso para economizar energia.

$$P_{sistema} = \begin{cases} P_{ativo} \\ P_{inativo} \end{cases} \quad (6\text{-}14)$$

O consumo do sistema inativo, Equação (*6-15*), é constante. Esse é o estado de menor energia possível pelo sistema. Dispositivos móveis são projetados para permanecer nesse estado o máximo de tempo possível, para prologar a duração da carga da bateria.

$$P_{inativo} = \beta_{hibernação} \quad (6\text{-}15)$$

O consumo do sistema ativo, Equação (*6-16*), é em função dos componentes ativos. O modelo considera o processador, tela, Wi-Fi e 3G. Outros componentes do sistema não foram considerados. Além disso, existe energia mínima para operação β_{espera}.

$$P_{ativo} = P_{cpu} + P_{tela} + P_{wifi} + P_{3G} + \beta_{espera} \quad (6\text{-}16)$$

O consumo do processador, Equação (*6-17*), é a soma do consumo dos núcleos determinado pela carga de processamento e frequência de cada núcleo.

$$P_{cpu} = \sum_{i}^{N} \left(\beta_{carga_{cpu_{f_i}}} U_{CPU} + \beta_{base_{cpu_{f_i}}} \right) \quad (6\text{-}17)$$

O modelo de consumo da tela pode ser linear, Equação (*6-18*), ou não linear, Equação (*6-19*). O modelo não linear escolhido foi exponencial, por melhor se adequar ao consumo observado para os aparelhos Xiaomi Redmi 2 e LG L80.

$$P_{tela} = \beta_{brilho}(brilho) + \beta_{base_{tela}} \quad (6\text{-}18)$$

$$P_{tela} = \beta_{brilho}(brilho)^3 + \beta_{base_{tela}} \quad (6\text{-}19)$$

O consumo da interface Wi-Fi, Equação (6-20), é composto de dois estados. No estado de transmissão e recepção de pacotes, a interface está ativa enquanto envia e recebe pacotes. O consumo é linear em função do número de pacotes transmitidos e recebido. O estado de espera é de baixo consumo e valor constante.

$$P_{wifi} = \begin{cases} Pcts_{env_{wifi}} \beta_{env_{wifi}} + Pcts_{rec_{wifi}} \beta_{rec_{wifi}} + \beta_{base_{wifi_{ativo}}} & \text{, se ativo} \\ \beta_{base_{wifi_{espera}}} & \text{, se espera} \end{cases} \quad (6\text{-}20)$$

O consumo da rede 3G, Equação (6-21), é definido por três estados: transmissão, cauda e espera. Durante a transmissão, o consumo é linear em função do número de pacotes enviados e recebidos assim como no Wi-Fi. O consumo de cauda se segue após o término do envio de pacotes por alguns segundos. O estado de espera de menor energia se segue após o termino da cauda.

$$P_{3G} = \begin{cases} Pcts_{env_{3G}} \beta_{env_{3G}} + Pcts_{rec_{3G}} \beta_{rec_{3G}} + \beta_{base_{3G_{ativo}}} & \text{, se ativo} \\ \beta_{base_{3G_{cauda}}} & \text{, se cauda} \\ \beta_{base_{3G_{espera}}} & \text{, se espera} \end{cases} \quad (6\text{-}21)$$

6.7 Validação do modelo

O modelo de consumo que foi produzido foi verificado usando simulação simples. O modelo foi usado para estimar o consumo de energia a partir dos registros produzidos nos vários experimentos de consumo que foram feitos durante este projeto.

Um programa em linguagem de programação Python foi criado para agregar os dados dos quatro arquivos de saída produzidos pela ferramenta DES em um arquivo único. Esse arquivo é um simples arquivo de texto no qual as colunas representam os campos registrados e as linhas uma amostra no tempo. Esse arquivo então foi usado como entrada em outro programa Python para processar os campos de interesse para o modelo e assim estimar o consumo instantâneo de energia. Outro arquivo foi gerado contendo o consumo estimado e o consumo mensurado de energia. Esse arquivo foi usado como entrada em programa de plotagem gráfica como Gnuplot para geração de gráfico comparativo entre o consumo estimado e o mensurado.

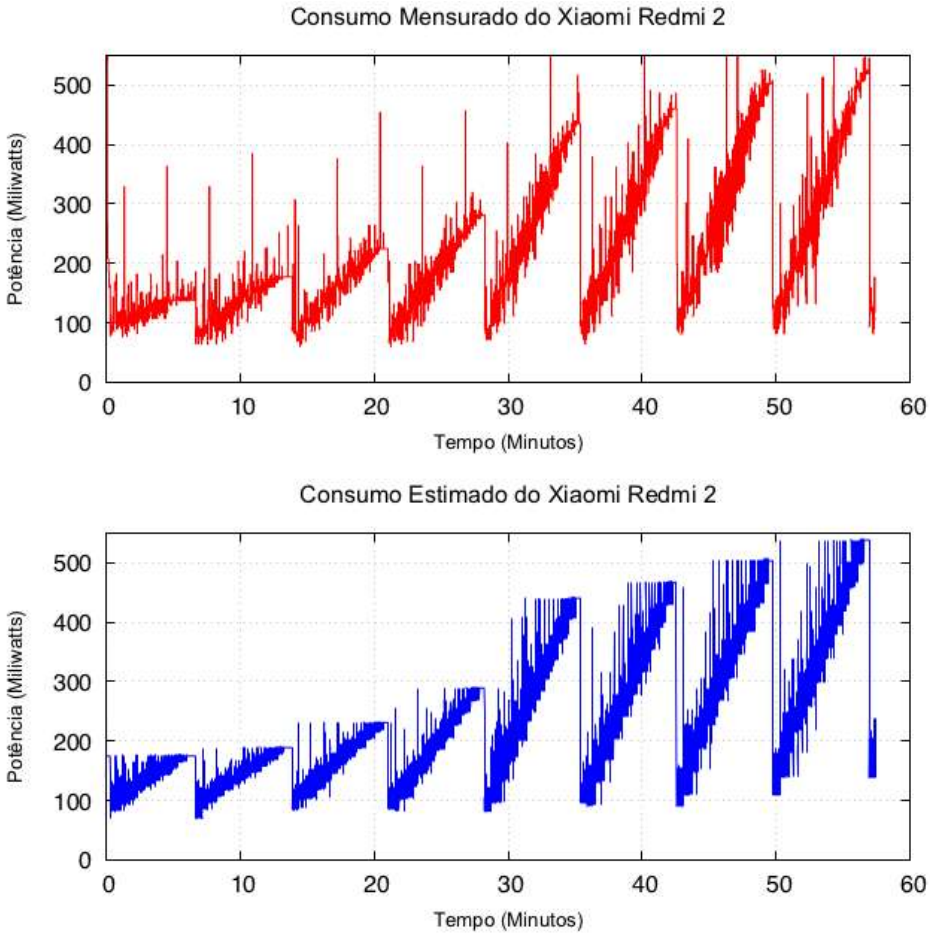

Figura 6.3: Exemplo comparativo do consumo e estimação no Xiaomi Redmi 2.

A Figura 6.3 apresenta exemplo de gráfico comparativo produzido pela ferramenta Gnuplot a partir dos resultados do programa em Python. Foi observado que o modelo produzido permitiu estimar o consumo com variado grau de sucesso.

A partir dos resultados analisados, foi possível extrair as seguintes conclusões. O modelo foi capaz de estimar com boa precisão o consumo do dispositivo em hibernação, espera e quando um núcleo único está ativo. No caso da tela e processamento com múltiplos núcleos, foi observada visível margem de erro. Ela indica que outros fatores precisam ser considerados no modelo para atingir melhor precisão. No caso da tela, é necessário determinar a contribuição da pigmentação no consumo de energia; entretanto, isso é limitado pelo alto custo computacional de analisar a pigmentação quadro a quadro. Uma alternativa possível é utilizar amostragem de baixa periodicidade para atenuar o custo computacional da operação. No caso do consumo por processadores multinúcleos, é provável que essa diferença seja causada por fatores que não foram considerados no modelo, por exemplo, o consumo dos *C-states* e das transições de estado do processador.

O consumo da interface 3G e Wi-Fi apresentou qualidade de estimação razoável. O aspecto ruidoso da curva de consumo, atraso na propagação do pacote da camada física, além da potência do sinal e retransmissão podem ser alguns dos fatores que limitaram a qualidade da estimação. Melhorias nos modelos das interfaces de rede podem ser realizadas para aumentar sua qualidade.

6.8 Discussão

O modelo construído foi obtido por medições por *software* e por um conjunto de experimentos realizados com objetivo de exercitar os componentes de interesse para destacar a relação entre o estado do componente e o consumo do sistema. Os resultados foram utilizados em regressões linear e não linear para determinar a curva de consumo de cada componente e do sistema em geral.

Devido a indisponibilidade de equipamento de medição, não foi possível utilizar medição por *hardware* que, segundo a literatura, é mais precisa e confiável que a utilizada neste projeto. Outro desafio ao desenvolvimento do modelo foram as limitações das interfaces de programação usadas na amostragem. A fragmentação do sistema operacional dificultou a realização de vários dos experimentos, alguns aparelhos tiveram de ser descartados devido à baixa confiabilidade dos sensores ou porque as interfaces estavam

inacessíveis. Testes precisaram ser realizados múltiplas vezes para reduzir a influência de eventos esporádicos do sistema e falhas ocasionais de rede Wi-Fi e 3G.

O LG L80 não pode ser completamente testado devido à indisponibilidade de configuração do modo superusuário necessário ao teste de processador e indisponibilidade de cartão SIM. O Xiaomi Redmi 2 apresentou limitações na interface da bateria que somente era atualizada a cada 1,25 segundos. O melhor dispositivo de teste foi o LG Nexus 4 que pertence à família Nexus de modelos de referência produzido pela Google e outras fabricantes como a Motorola.

Este método de construção do modelo é exaustivo, porém necessário para compreender o problema existente de energia decorrente das limitações da bateria e da grande diversidade de aparelhos. Uma próxima etapa seria o desenvolvimento ou melhoria de técnicas de autoconstrução de modelos de consumo que, por meio de aprendizagem de máquina, ativamente otimizam o modelo até atingir boa margem de precisão.

7 CONCLUSÃO

Nesta dissertação, foi apresentado o desenvolvimento de modelos de consumo de energia para dispositivos móveis com foco na plataforma Android. Um modelo foi produzido experimentalmente a partir dos resultados coletados pela ferramenta de monitoramento *Droid Energy Suíte* (DES). A ferramenta foi desenvolvida do zero para realização desses testes. O modelo foi validado a partir dos dados experimentais coletados nos experimentos de consumo. O modelo apresentou resultado satisfatórios, dadas as limitações do projeto. Abaixo um resumo do que foi visto em cada capítulo:

No capítulo 2, foi apresentada a introdução ao gerenciamento de energia em dispositivos móveis, o problema da limitação de carga das baterias atuais e as características gerais dos componentes que, segundo a literatura, são responsáveis pela maior parte do consumo de energia.

No capítulo 3, foram vistas algumas das técnicas empregadas por engenheiros e pesquisadores para analisar e construir modelos de consumo de energia. Foi apresentado o estado da arte de algum dos componentes.

No capítulo 4, foram apresentadas as interfaces de programação disponíveis na plataforma Android para realização do monitoramento de consumo. O DES foi construído baseado nas interfaces apresentadas.

No capítulo 5, foram apresentados os experimentos e resultados de consumo de energia utilizados como base para a construção do modelo de consumo de energia para os aparelhos LG Nexus 4, Xiaomi Redmi 2 e LG L80.

No capítulo 6, foi construído um modelo de consumo de energia simples, a partir dos resultados apresentados no capítulo anterior. Por meio de técnicas de regressão, foram identificadas e modeladas as curvas de energia do processador, tela LCD, interface Wi-Fi e interface 3G. O modelo produzido apresentou boa estimação para estado de hibernação, espera e processador com apenas um núcleo ativo. Estimação com pequeno erro para tela LCD e múltiplos núcleos ativos. Estimação razoável, porém, não exata, para o caso das interfaces Wi-Fi e 3G. Indicando oportunidades de melhorias em projetos futuros.

7.1 Trabalhos futuros

Durante os experimentos práticos de consumo de energia, foram detectadas limitações nas interfaces de monitoramento da bateria que limitaram o número de aparelhos disponíveis para a execução dos experimentos. Um trabalho futuro seria repetir

os experimentos usando monitoramento externo e comparar os resultados. Tal trabalho propiciaria melhoria dos coeficientes produzidos e adição de outros componentes de interesse ao modelo que não puderam ser avaliados nessa dissertação.

Alguns outros trabalhos futuros que podem ser desenvolvidos a partir dos resultados dessa dissertação são:

- Desenvolvimento de simulador de consumo para o emulador do Android visando estimar o impacto energético da aplicação durante seu desenvolvimento.
- Desenvolvimento de aplicação para monitoramento e otimização do consumo de energia para Android.
- Melhoria do modelo de consumo que foi produzido a partir do monitoramento do sistema via instrumentação do núcleo do sistema operacional.
- Desenvolvimento de técnicas de otimização de uso das interfaces 3G e Wi-Fi, que são, segundo a literatura, responsáveis por alto consumo de energia. Este trabalho pode ser utilizado como um ponto inicial com objetivo de redução de consumo de energia.
- Desenvolvimento de algoritmos de agendamento otimizado para processadores multinúcleos para dispositivos móveis. A popularização desse tipo de processador é recente. O agendamento incorreto das tarefas pode provocar desperdícios desnecessários de energia. O mesmo pode ser dito de processadores de núcleos assimétricos como a arquitetura ARM big.LITTLE.

APÊNDICE A – EXEMPLO DE OXIRREDUÇÃO DA BATERIA

As baterias lítio-íon, usadas em dispositivos como *smartphones*, possuem uma série de vantagem como a alta densidade energética (até 150 Wh/kg), alta voltagem (até 4 V/célula) e longa retenção de carga e vida útil (5-10 anos) (LINDEN; REDDY, 2001). O processo de químico de descarga e recarga é exemplificado a seguir:

Processo de Descarga

O ciclo de descarga de uma célula de cátodo de óxido de lítio cobalto ($LiCoO_2$) e ânodo de grafite (C) é apresentado nas equações a seguir:

Reação de redução do Cátodo: Perda de íons de Lítio (Li^+) e ganho de elétrons (e^-).

$$LiCoO_2 \rightarrow Li_{1-x}CoO_2 + xLi^+ + xe^- \quad \text{(A-1)}$$

Reação oxidação do Ânodo: Ganho de íons de Lítio (Li^+) e perda de elétrons (e^-).

$$C + xLi^+ + xe^- \rightarrow C^+Li_x^- \quad \text{(A-2)}$$

Reação completa de descarga da célula: Íons de Lítio migram do cátodo para o ânodo, os elétrons do ânodo para o cátodo.

$$LiCoO_2 + C \rightarrow Li_{1-x}CoO_2 + C^+Li_x^- \quad \text{(A-3)}$$

Processo de Recarga

O ciclo de recarga de uma célula com cátodo de óxido de lítio cobalto ($LiCoO_2$) e ânodo de grafite (C) é apresentado nas equações a seguir.

Reação de oxidação do Cátodo: Ganho de íons de Lítio (Li^+) e perda de elétrons (e^-), o estado original do cátodo é restaurado.

$$Li_{1-x}CoO_2 + xLi + xe^{-1} \rightarrow LiCoO_2 \quad \text{(A-4)}$$

Reação de redução do Ânodo: Perda de íons de Lítio (Li^+) e ganho de elétrons (e^-), restaurando o estado original do ânodo.

$$C^+Li_x^- \rightarrow C + xLi + xe^- \quad \text{(A-5)}$$

Reação completa de recarga da célula: O cátodo e ânodo retornam ao estado original.

$$Li_{1-x}CoO_2 + C^+Li_x^- \rightarrow LiCoO_2 + C \quad \text{(A-6)}$$

APÊNDICE B – GOOGLE ANDROID

O Android é um sistema operacional baseado no núcleo do Linux que começou a ser desenvolvido em 2005 e publicamente anunciado em 2007 pelo Google e *Open Handset Alliance* para *smartphones* e *tablets*. *Open Handset Alliance* é um consórcio entre empresas de tecnologia, fabricantes de dispositivos, operadoras de telefonia móvel e fabricantes de *chipset* com objetivo de desenvolver um padrão aberto de dispositivos móveis. A principal diferença do Android para seus concorrentes foi o rápido ciclo de desenvolvimento. No primeiro ano de lançamento no mercado, uma nova versão do sistema operacional era lançada a cada dois meses e meio (AMADEO, 2014). Hoje, mais de 20 versões depois, o ciclo de lançamento é semestral, com a versão mais recente anunciada no final de agosto de 2015 (EASON, 2015).

O rápido ciclo de desenvolvimento do Android pelo Google conflita com o ciclo de lançamento e suporte dos fabricantes de hardware. Isto levou ao problema de fragmentação do sistema operacional. A fragmentação significa que num dado momento diferentes versões do Android estão em uso pelos usuários, porque os fabricantes demoram a disponibilizar atualizações ou não pretendem fazer para investir em novos dispositivos. A fragmentação afeta os desenvolvedores que precisam fazer mais esforço para suportar a grande variedade de versões e hardwares disponíveis. A Tabela B-1, Figura B.1 e Figura B.2 apresentam a fragmentação do sistema operacional como vista em Outubro de 2015 (DEVELOPERS, 2015). Segundo a OPENSIGNAL (2015), considerando o período de janeiro a agosto de 2015, cerca de 24.093 dispositivos distintos em uso, comparado a 18.796 em 2014, cerca de 37,8% destes dispositivos pertence a Samsung e cerca de 1.294 marcas usando Android.

Tabela B-1: Distribuição das diferentes versões do Android em uso segundo dados coletados durante 7 dias até 5 de outubro de 2015 (DEVELOPERS, 2015) .

Versão	**Codinome**	**Versão da API**	**Distribuição**
2.2	*Froyo*	8	0,2%
2.3.3-2.3.7	*Gingerbread*	10	3,8%
4.0.3-4.0.4	*Ice Cream Sandwich*	15	3,4%
4.1.x	*Jelly Bean*	16	11,4%
4.2.x		17	14,5%
4.3		18	4,3%
4.4	*KitKat*	19	38,9%
5.0	*Lollipop*	21	15,6%
5.1		22	7.9%

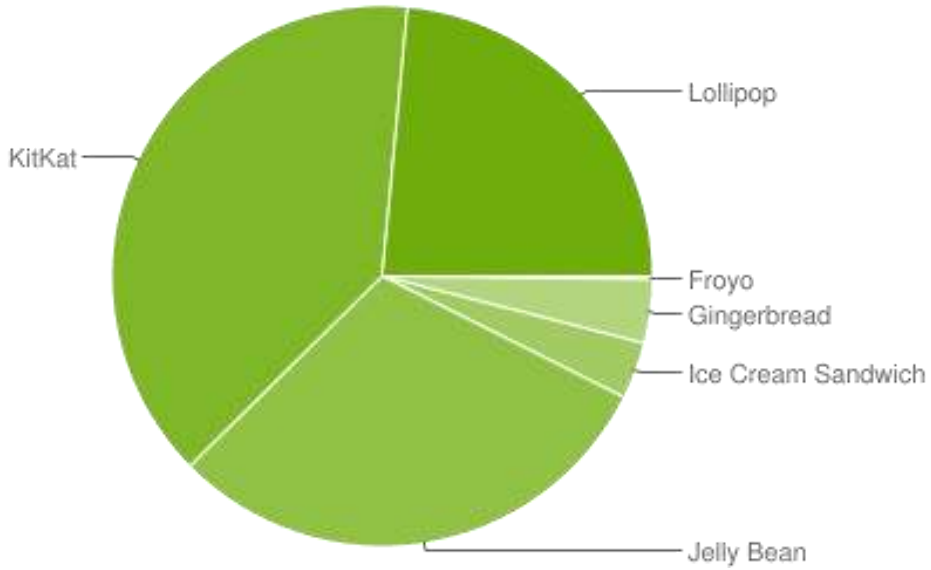

Figura B.1: Gráfico da distribuição relativa das versões principais do Android em uso, segundo dados coletados durante 7 dias até 5 de outubro de 2015 (DEVELOPERS, 2015).

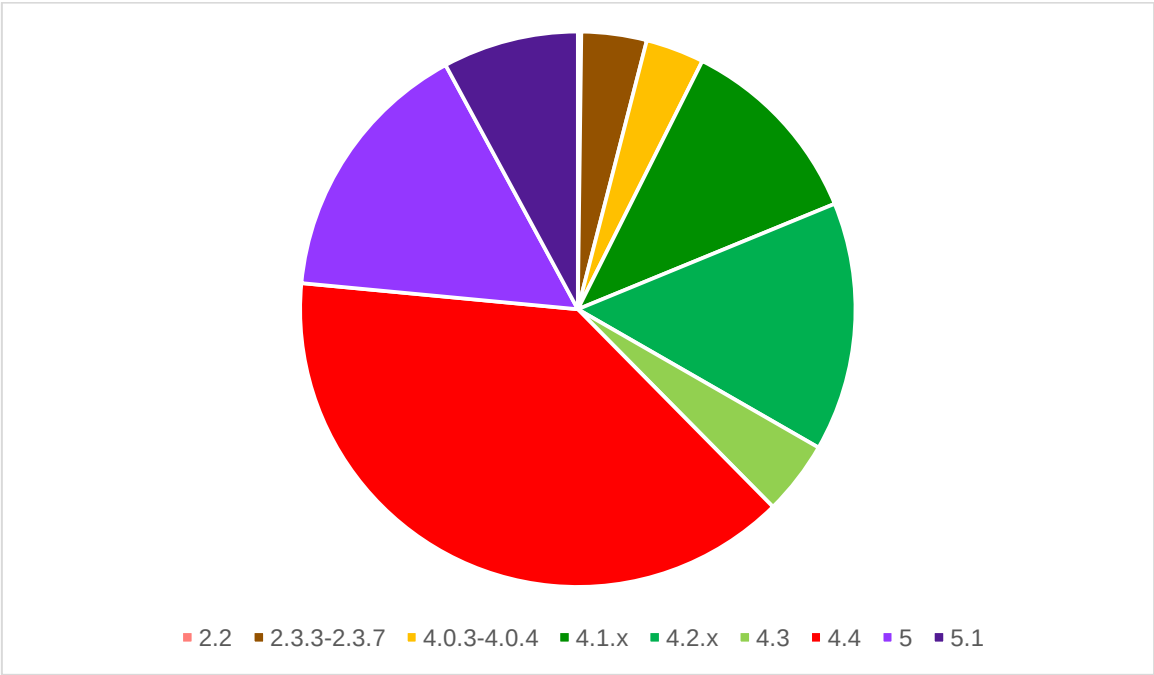

Figura B.2: Gráfico da distribuição relativa por versão Android em uso, segundo dados coletados durante 7 dias até 5 de outubro de 2015 (DEVELOPERS, 2015).

O Android usa o núcleo do Linux que foi criado originalmente para computadores de mesa e servidores. Adaptações foram necessárias para atender os novos requisitos de memória e energia como:

- Sistema de alarme para despertar dispositivos em hibernação;
- Novo gerenciador de memória;
- Novo mecanismo de comunicação entre processos;
- Melhorias no mecanismo de gerenciamento de energia do Linux;
- Terminador de processos em situações de pouca memória;
- Depurador de núcleo;
- Log de sistema.

Diferente de outros sistemas operacionais GNU/Linux, Android não é um sistema do tipo Unix. As interfaces de programação e bibliotecas do sistema para o desenvolvimento de aplicações não estão disponíveis. Ao invés disso, aplicações são desenvolvidas na linguagem de programação Java, compiladas para um formato próprio de *byte-code* chamado *dex* e executadas numa máquina virtual otimizada para dispositivos móveis, a Dalvik ou ART. Como pode ser visto na Figura B.3, a arquitetura do Android é organizada para suportar esse modelo de aplicações. Grande maioria das aplicações do sistema ou de terceiros é escrita em Java e executa na máquina virtual.

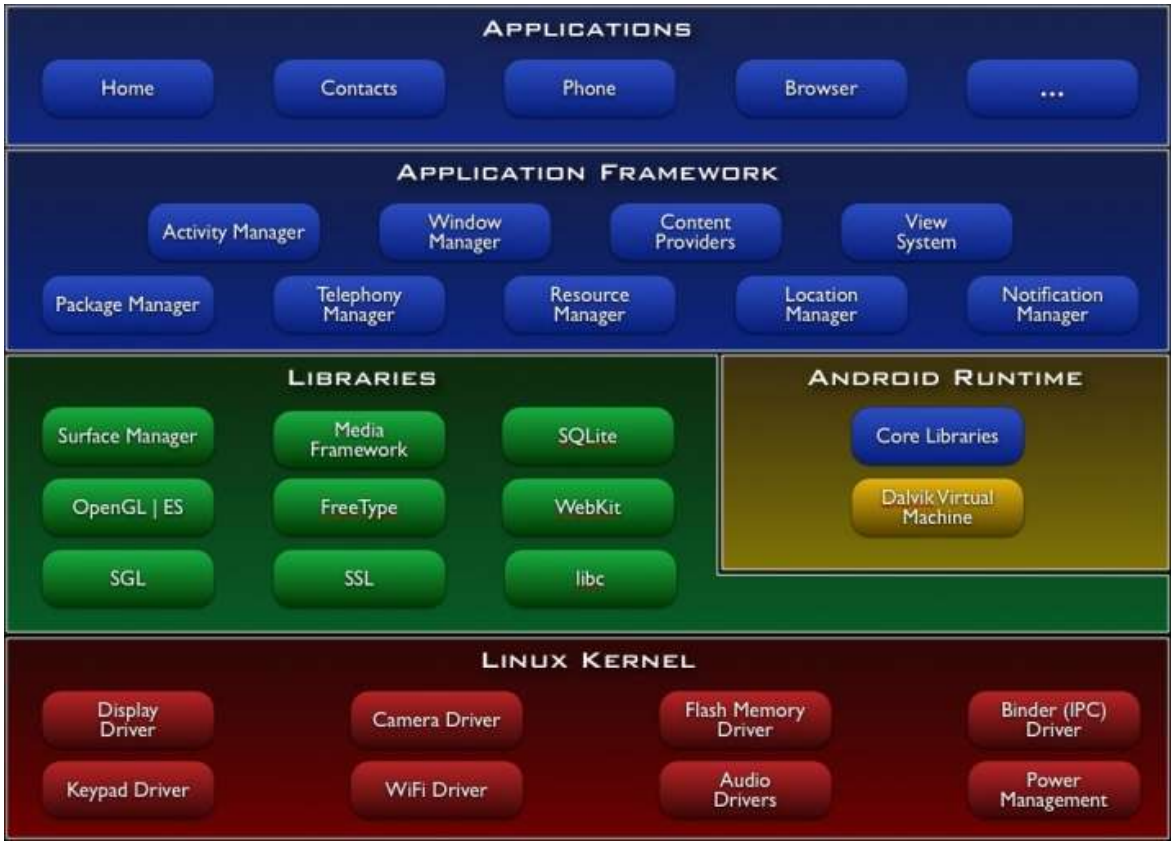

Figura B.3: Diagrama da arquitetura interna do Android.

As interfaces de gerenciamento de energia usadas pelo núcleo do Linux em computadores tradicionais baseada em APM e ACPI não são suficientes para conservar energia em dispositivos móveis. Ao invés disso, o Android usa uma extensão própria a nível de núcleo chamada *PowerManager* (PM) para conservar energia. O PM agressivamente busca suspender por completo o sistema para economizar a maior quantidade de energia possível (WYSOCKI, 2010).

Nessa abordagem o estado natural do sistema é hibernação (WYSOCKI, 2010) *apud* (BROWN; WYSOCKI, 2008), ou seja, a energia é usada somente para manter a memória e alguns componentes capazes de acordar o sistema (WYSOCKI, 2010). O estado de trabalho somente é executado em resposta a um evento de despertar, permanecendo ativo apenas o necessário para que o trabalho requerido pelo usuário se complete e, em seguida, automaticamente retorna ao estado de hibernação (WYSOCKI, 2010).

O PM implementa um mecanismo chamado *wakelock* para administrar o estado de energia do sistema. *Wakelock* é um objeto que apresenta dois estados, ativo ou inativo. Ele previne que o sistema seja suspenso enquanto houver ao menos um *wakelock* ativo. A camada de usuário, onde residem as aplicações, pode manipular *wakelocks* para prevenir que o sistema seja suspenso durante uma atividade crítica.

O arcabouço de gerenciamento de energia do Android é exposto para a camada de aplicação como uma classe Java com nome `android.os.PowerManager` (DEVELOPERS, [s.d.]) que permite acessar o *wakelock* como uma instância da classe `android.os.PowerManager.WakeLock` (DEVELOPERS, [s.d.]), como apresentado no Programa B-1.

```
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
WakeLock wl = pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "TAG");
wl.acquire();
// ... tela permanecerá ligada durante esse período
wl.release();
```

Programa B-1: Exemplo de utilização de wakelock.

A Figura B.4 mostra o fluxo de chamadas pro PM ao invocar um método do `PowerManager` ou `WakeLock`, por meio de comunicação entre processos. O serviço da camada de arcabouço de aplicação invoca a implementação nativa para acessar os serviços do núcleo onde é implementado o PM.

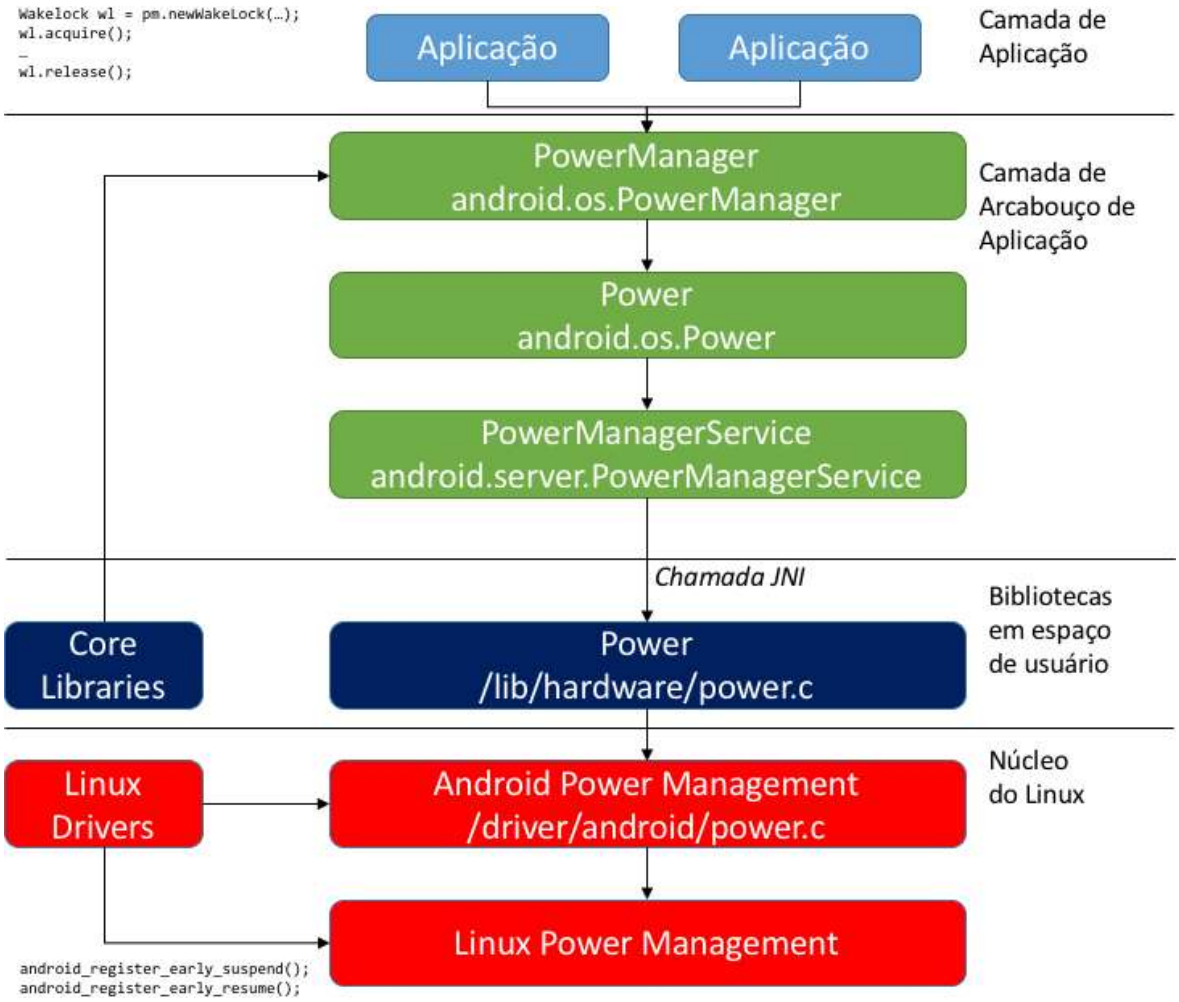

Figura B.4: Sistema de gerenciamento de energia do Android (TARKOMA et al., 2014).

Quatro tipos de *Wakelock* são acessíveis para camada de aplicação, conforme listado na Tabela B-2. Eles permitem impedir que o processador, tela ou iluminação do teclado sejam suspensos enquanto o *Wakelock* estiver adquirido. Entretanto a manipulação incorreta dos *Wakelocks* provocou a geração de uma nova classe de problemas chamados ebugs que foram descritos na seção 2.8.

Tabela B-2: Tipos de Wakelock.

Tipos de *Wakelock*	**CPU**	**Brilho da Tela**	**Iluminação do Teclado**
`PARTIAL`	Ligado	Desligado	Desligado
`SCREEN_DIM`	Ligado	Escurecido	Desligado
`SCREEN_BRIGHT`	Ligado	Claro	Desligado
`FULL`	Ligado	Claro	Claro

O uso mandatório de bloco *try-finally* é sugerido por ALAM *et al.* (2014) como uma forma de garantir que o *Wakelocks* sejam liberados mesmo em caso de exceção ou outros desvios de fluxo de controle, como mostrado no Programa B-2. Diferente do uso típico apresentado no Programa B-1.

```
try {
    wl.acquire();
    // stuff
} finally {
    wl.release();
}
```

Programa B-2: Exemplo de uso seguro de wakelock.

PATHAK *et al.* (2012) afirmaram que o acesso direto a interfaces de programação para controle do gerenciamento de energia, como *Wakelocks* inevitavelmente originou uma nova classe de bugs causados pelo uso incorreto dessas interfaces de programação. Ele chamou esse novo paradigma de programação onerada de energia. Este paradigma onera os desenvolvedores a explicitamente manipular as interfaces de controle de energia a fim de garantir o funcionamento correto de suas aplicações. ALAM *et al.* (2014) sugeriram aplicar análise de fluxo de dados para otimizar a locação de *wakelocks* assim limitando o consumo de energia e garantindo o uso correto dessas interfaces.

JINDAL *et al.* (2013) estudaram essa nova classe de bugs de energia gerados pelo uso incorreto das interfaces de controle de energia que chamou de conflitos de sono. Conflitos de sono ocorrem quando um componente de alto consumo de energia é incapaz de retornar a um estado de menor energia porque o driver responsável por acionar esta transição não consegue progredir enquanto aguarda por um evento.

Assim como em outros sistemas operacionais Linux, o Android aplica políticas DVFS conforme governante em uso pelo driver *cpufreq*, responsável por controlar o processador. Por padrão é aplicada a política *ondemand* (CARROLL; HEISER, 2013) que ajusta a frequência do processador conforme o carga atual de uso (BRODOWSKI; GOLDE, [s.d.]). Além disso um serviço em espaço de usuário chamado *mpdecision*, de código fechado e não documentado, controla a ativação e desativação dos núcleos da CPU em uso pelo sistema (CARROLL; HEISER, 2013).

O Android provê um modelo de energia embarcado e serviço estimação estatística chamado *BatteryStats* para estimar o consumo de energia a nível de componente (TARKOMA et al., 2014). O serviço registra estatísticas básicas de uso dos componentes como processador, tela, telefonia celular, Wi-Fi, GPS e etc. Todas as transições de energia como ligado/desligado, esperando/executando, claro/esmaecido são registrados pelo serviço. O serviço não monitora diretamente o consumo de energia, mas os momentos em que a energia é consumida pelo serviço para estimar o consumo dos componentes. Além disso o serviço é capaz atribui o consumo as aplicações, ou parte dela quando um componente é compartilhado entre varias aplicações.

APÊNDICE C – APPLE IOS

O iOS é o sistema operacional da Apple para seus dispositivos móveis e portáteis como *smartphones*, *tablets* e tocadores de multimídia. Primeiramente anunciado simultaneamente com o iPhone em 9 de janeiro de 2007. O iOS tem suas origens no núcleo do sistema operacional Mac OS X, porém com personalizações especificas para arquitetura ARM.

O iOS é organizado em quatro camadas como apresentado na Figura C.1. O *Core OS* engloba o núcleo do sistema operacional, pilha de rede, gerenciador de energia, sistema de arquivos e gerenciador de segurança. O *Core Services* prover as funcionalidades usadas pelas duas camadas seguintes como acesso a rede, localização e banco de dados SQLite. A camada de mídia é responsável pelo processamento de áudio, vídeo e aceleração gráfica. O *Cocoa Touch* disponibiliza as interfaces de programação necessárias para o suporte de aplicações iOS. Por último a camada de aplicação onde executa aplicações da Apple ou de terceiros.

Diferente do Android, o iOS não sofre do problema de fragmentação devido ao forte controle da plataforma pela Apple. Atualizações de software são rapidamente propagada para os dispositivos. A Figura C.2 mostra a distribuição do iOS em uso medida cerca de um mês após o lançamento do iOS 9, a maioria dos dispositivos recebeu e aplicou a atualização. A pouca variedade de dispositivos e facilidade de atualização reduz o trabalho dos desenvolvedores que podem alcançar um grande número de usuários com baixo custo de desenvolvimento e manutenção.

Segundo TARKOMA *et al.* (2014), o iOS utiliza padrões de conservação de energia combinado com mecanismo de gerenciamento de energia do lado do núcleo para economizar energia. Os padrões de conservação determinam políticas de agregação de notificações, atualizações, execução de operações assíncronas e aglutinação de tarefas para estender o período de hibernação do aparelho ao máximo. O arcabouço de *drivers* de dispositivo, chamado *I/O kit framework*, possui definido políticas de gerenciamento de energia que administram a energia da hierarquia de dispositivos do sistema.

SON *et al.* (2014) realizaram uma extensa análise de consumo entre Android e iOS. Eles identificaram que o iOS apresentou eficiência energética, em média, 19.7% maior que a do Android. Isto pode ser atribuído a melhores otimizações de *hardware* e *software* do iOS em comparação ao Android decorrente do maior controle sobre as especificações dos dispositivos.

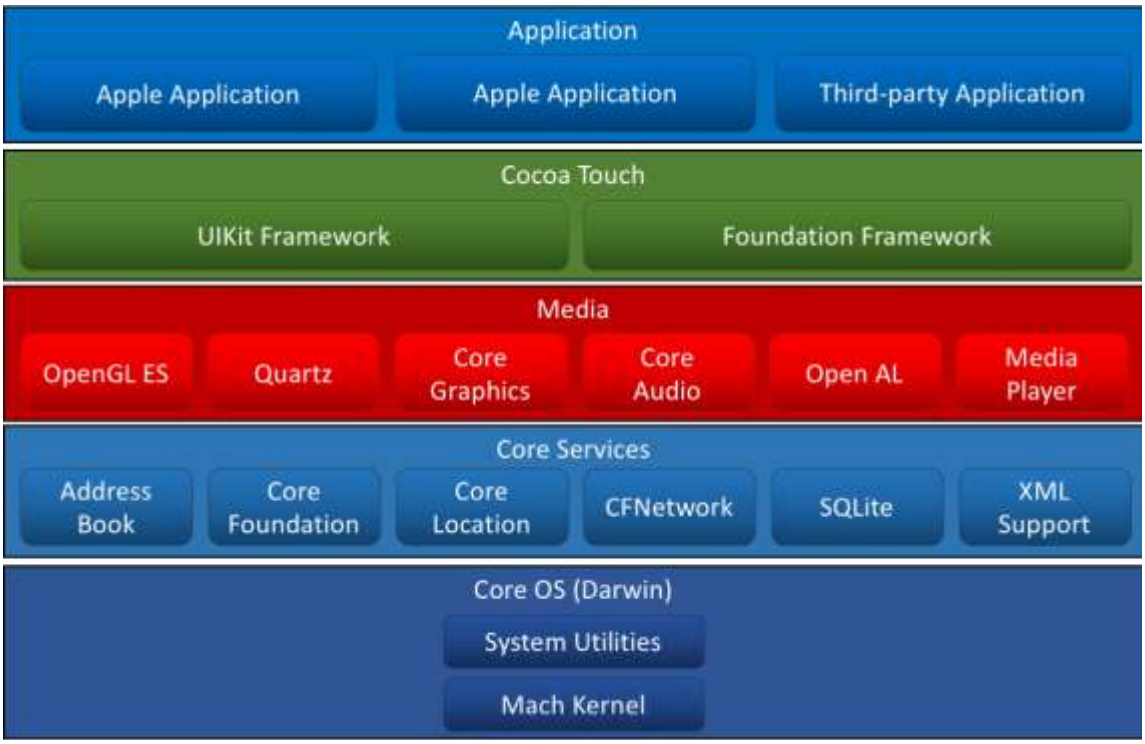

Figura C.1: Diagrama de camadas do sistema operacional iOS.

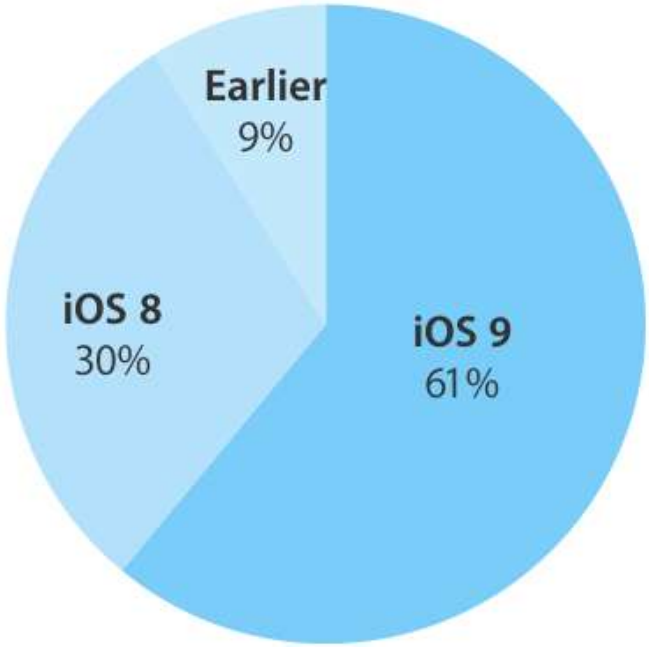

Figura C.2: Distribuição relativa da versão iOS em uso medida pela App Store em 19 de outubro de 2015 (DEVELOPERS, [s.d.]).

APÊNDICE D – ESTRUTURAS DO NÚCLEO DO LINUX

O diretório **/sys** contém informações sobre os principais componentes do sistema. Informações sobre a bateria são encontradas em **/sys/class/power_supply/battery**. Informações sobre o processador no diretório e subdiretórios do **/sys/devices/system/cpu**. Mais detalhes sobre a operação do processador são encontrados no **/proc/stat**.

A carga de processamento pode ser calculada pelo conteúdo do **/proc/stat**, este arquivo codifica a informação de uso do processador conforme apresentado no Programa D-1. Um único registro informa o tempo total de uso desde o inicio do sistema, usando duas amostras podemos identificar a carga de processamento no período. Para calcular a carga do sistema basta aplicar a Equação (D-*1*).

```
cpu<#> <user_time> <nice_time> <system_time> <idle_time> …
```

Programa D-1: Formato de um registro de cpu do /proc/stat.

$$Processador_{carga\%} = \left(1 - \frac{Idle}{User + Nice + System + Idle}\right) \times 100\% \qquad \text{(D-1)}$$

Informações sobre a interface de rede se encontram em **/sys/class/net**. A interface de loop local é chamada *lo*, a interface Wi-Fi de *wlan0* e celular como *rmnet0* ou *rmnet_usb0*. Caso o diretório não exista, a interface se encontra indisponível no momento. O conteúdo do diretório de uma interface descreve as informações de tráfego da interface.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/
|-- block/     # representa blocos de dispositivos
|-- bus/       # informações sobre os vários barramentos
|-- class/     # informações por classe de dispositivo
|-- dev/       # contém links para acesso direto ao dispositivo
|-- devices/   # contém links para os dispositivos
|-- firmware/  # contém informações sobre o firmware do sistema
|-- fs/        # informações sobre as partições montadas
|-- kernel/    # configurações do núcleo e política de segurança
|-- module/    # módulos carregados pelo sistema
|-- power/     # contém informações sobre o estado de energia e estatíscas
```

Programa D-2: Estrutura típica do diretório /sys.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/class/power_supply/battery/
|-- capacity           # carga atual da bateria de 0 a 100.
|-- charge_type        # método de carregamento.
|-- current_now        # corrente atual de microampere.
|-- device@            # link para a controladora
```

```
|-- energy_full        # capacidade total da bateria em microampere-hora
|-- health             # a saúde da bateria
|-- power/             # detalhes sobre consumo de energia
|-- present            # se bateria está conectada
|-- status             # estado da bateria
|-- subsystem@         # link para o subsistema
|-- technology         # tecnologia de fabricação da bateria
|-- temp               # temperatura em centigrado-celsius
|-- type               # tipo de fonte de energia
|-- uevent             # objeto usado para notificar alterações
|-- voltage_max_design # voltagem máxima de projeto em microvolts
|-- voltage_min_design # voltagem mínima de projeto em microvolts
|-- voltage_now        # voltagem atual em microvolts
```

Programa D-3: Listagem do conteúdo do diretório /sys/class/power_supply/battery no Nexus 4.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
# /sys/class/power_supply/battery/uevent
POWER_SUPPLY_NAME=battery
POWER_SUPPLY_STATUS=Charging
POWER_SUPPLY_CHARGE_TYPE=Fast
POWER_SUPPLY_HEALTH=Good
POWER_SUPPLY_PRESENT=1
POWER_SUPPLY_TECHNOLOGY=Li-ion
POWER_SUPPLY_VOLTAGE_MAX_DESIGN=4360000
POWER_SUPPLY_VOLTAGE_MIN_DESIGN=3200000
POWER_SUPPLY_VOLTAGE_NOW=4364178
POWER_SUPPLY_CAPACITY=99
POWER_SUPPLY_CURRENT_NOW=-154500
POWER_SUPPLY_TEMP=336
POWER_SUPPLY_ENERGY_FULL=2087000000
```

Programa D-4: Exemplo do conteúdo do arquivo /sys/class/power_supply/battery/uevent no Nexus 4.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/devices/system/cpu/
|-- cpu0/       # informações do núcleo 0
|-- cpu1/       # informações do núcleo 1
|-- cpu2/       # informações do núcleo 2
|-- cpu3/       # informações do núcleo 3
|-- cpuctl      # objeto para controlar cpu
|-- cpufreq/    # informações sobre escalonamento de frequência
|-- cpuidle/    # informações sobre espera
|-- kernel_max  # número máximo de indexação de núcleos suportados
|-- offline     # núcleos desligados
|-- online      # núcleos ativado
|-- possible    # núcleos possíveis de ativação
|-- power/      # informações sobre energia
|-- present     # núcleos presentes
|--  uevent         # objeto usado para notificar espaço de usuário sobre
mudanças
```

Programa D-5: Listagem do /sys/devices/system/cpu no Nexus 4.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
# CPU Offline
/sys/devices/system/cpu/cpu#/
|-- cpuidle    # informações sobre espera
|-- online     # se núcleo está ativo (1) ou não (0).
|-- power/     # informações sobre energia
|-- subsystem@
|-- uevent
```

Programa D-6: Listagem do /sys/devices/system/cpu/cpu# no Nexus 4 quando processador está desligado.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
# CPU Online
/sys/devices/system/cpu/cpu#/
|-- cpufreq/    # informações sobre escalonamento de frequência
|-- cpuidle/    # informações sobre espera
|-- online      # se núcleo está ativo (1) ou não (0).
|-- power/      # informações sobre energia
|-- rq-stats    # informações sobre a filas de trabalho
|-- subsystem@  #
|-- topology/   # topologia
|--  uevent         # objeto usado para notificar espaço de usuário sobre
mudanças
```

Programa D-7: Listagem do /sys/devices/system/cpu/cpu# no Nexus 4 quando processador está ligado.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/devices/system/cpu/cpu#/cpufreq/
|-- affected_cpus                  #
|-- cpu_utilization                # carga atual da cpu de 0 a 100
|-- cpuinfo_cur_freq               # frequência atual de operação em Hz
|-- cpuinfo_max_freq               # frequência máxima operação em Hz
|-- cpuinfo_min_freq               # frequência mínima de operação em Hz
|-- cpuinfo_transition_latency     # latência de transição
|-- related_cpus                   #
|-- scaling_available_frequencies  # lista de frequências disponível
|--  scaling_available_governors   # lista de políticas de escalonamento de
frequência
|-- scaling_cur_freq               # frequência atual de operação em Hz
|-- scaling_driver                 #
|-- scaling_governor               # política de escalonamento
|-- scaling_max_freq               # frequência máxima do escalonador
|-- scaling_min_freq               # frequência mínima do escalonador
|-- scaling_setspeed               # frequência em uso pelo escalonador
|-- stats/                         # statísticas do escalonador de frequência
```

Programa D-8: Listagem do /sys/devices/system/cpu/cpu#/cpufreq no Nexus 4.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/devices/system/cpu/cpu#/cpufreq/stats/
|-- time_in_state # tempo em mili-segundo gasto cada frequência de operação
|-- total_trans   # total de transições entre frequências
```

Programa D-9: Listagem do /sys/devices/system/cpu/cpu#/cpufreq/stats no Nexus 4.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
# /proc/stat
cpu  118200 2876 122112 8481328 14276 8 288 0 0 0
cpu0 101922 1359 110762 1917589 11098 8 282 0 0 0
cpu1 11091 715 7423 2176006 2327 0 5 0 0 0
cpu2 3245 452 2481 2192560 519 0 1 0 0 0
cpu3 1942 350 1446 2195173 332 0 0 0 0 0
intr 4590275 0 … # conteúdo completo omitido
ctxt 24728692
btime 1448189762
processes 29263
procs_running 1
procs_blocked 0
softirq 6812840 1662 1137002 3403 39990 1662 1662 21311 90905 4372805 1142438
```

Programa D-10: Exemplo do conteúdo do /proc/stat no Nexus 4.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/class/net/
|-- dummy0
|-- lo           # Loop local
|-- rev_rmnet0
|-- rev_rmnet1
|-- rev_rmnet2
|-- rev_rmnet3
|-- rev_rmnet4
|-- rev_rmnet5
|-- rev_rmnet6
|-- rev_rmnet7
|-- rev_rmnet8
|-- rmnet0       # 3G/4G
|-- rmnet1
|-- rmnet2
|-- rmnet3
|-- rmnet4
|-- rmnet5
|-- rmnet6
|-- rmnet7
|-- rmnet_smux0
|-- rmnet_usb0   # 3G/4G (Alternativo)
|-- rmnet_usb1
|-- rmnet_usb2
|-- rmnet_usb3
|-- sit0
|-- wlan0        # Wi-Fi
```

Programa D-11: Listagem de interfaces do /sys/class/net/ no Nexus 4.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/class/net/<interface>/
|-- addr_assign_type # tipo de atribuição de endereço de hardware.
|-- addr_len         # comprimento de bytes do endereço.
|-- address          # endereço de hardware.
|-- broadcast        # endereço de difusão.
|-- carrier          # indica se existe portadora.
```

```
|-- dev_id           # identificador do dispositivo.
|-- device@          #
|-- dormant          # indica se a interface está dormente.
|-- duplex           # indica se a interface é half-duplex ou full-duplex.
|-- flags            # flags como definido em uapi/linux/if.h
|-- ifalias          # apelido da interface
|-- ifindex          # indice interface
|-- iflink           # indice interface
|-- link_mode        #
|-- mtu              # unidade máxima de transferência em bytes.
|-- netdev_group     # grupo ao qual a interface pertence.
|-- operstate        # Indica se está ativa (up) ou inativa (down).
|-- power/           #
|-- queues/          # informações sobre as filas rx e tx.
|-- speed            # velocidade da interface.
|-- statistics/      # informações estatísticas da interface.
|-- subsystem@       #
|-- tx_queue_len     # comprimento da fila de transmissão em pacotes.
|-- type             # tipo da interface como definido em uapi/linux/if_arp.h
|-- uevent           #
```

Programa D-12: Listagem da estrutura típica do /sys/class/net/<interface>.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
/sys/class/net/<interface>/statistics/
collisions           # total de colisões
multicast            # total do multicasts
rx_bytes             # bytes recebidos
rx_compressed        # pacotes comprimidos recebidos
rx_crc_errors        # pacotes recebidos com erro de crc
rx_dropped           # pacotes recebidos descartados
rx_errors            # pacotes recebidos com erro
rx_fifo_errors       # pacotes com erro de fifo
rx_frame_errors      # pacotes com erro de quadro
rx_length_errors     # pacotes com erro de comprimento
rx_missed_errors     # pacotes perdidos
rx_over_errors       #
rx_packets           # total de pacotes recebidos
tx_aborted_errors    # pacotes com transmitição abortada
tx_bytes             # bytes transmitidos
tx_carrier_errors    # pacotes transmitidos com erro na portadora
tx_compressed        # pacotes transmitidos comprimidos
tx_dropped           # pacotes transmissão descartados
tx_errors            # pacotes transmitidos com erro
tx_fifo_errors       # pacotes com erro de fifo
tx_heartbeat_errors  # erros de heartbeat
tx_packets           # total de pacotes transmitidos
tx_window_errors       # pacotes com erro de janela|-- statistics/        #
informações estatísticas da interface.
|-- subsystem@       #
|-- tx_queue_len     # comprimento da fila de transmissão em pacotes.
|-- type             # tipo da interface como definido em uapi/linux/if_arp.h
|-- uevent           #
```

Programa D-13: Listagem da estrutura típica do /sys/class/net/<interface>/statistics.

```
# Nexus 4 - Android 5.1.1 (LMY48T)
# cat /sys/class/net/wlan0/statistics/*
0       # collisions
0       # multicast
794865  # rx_bytes
0       # rx_compressed
0       # rx_crc_errors
0       # rx_dropped
0       # rx_errors
0       # rx_fifo_errors
0       # rx_frame_errors
0       # rx_length_errors
0       # rx_missed_errors
0       # rx_over_errors
4055    # rx_packets
0       # tx_aborted_errors
53104   # tx_bytes
0       # tx_carrier_errors
0       # tx_compressed
0       # tx_dropped
0       # tx_errors
0       # tx_fifo_errors
0       # tx_heartbeat_errors
450     # tx_packets
0       # tx_window_errors
```

Programa D-14: Exemplo do conteúdo dos vários arquivos de estatísticas /sys/class/net/wlan0/statistics/ no Nexus 4.

APÊNDICE E – INTERFACE DE SERVIÇOS DO ANDROID

Serviços no Android podem ser acessados pela aplicação através de objetos obtidos pelo método *getSystemService* encontrado nas classes que derivam da classe abstrata *Context*. Eles são identificados por uma cadeia de caracteres, muitos deles se encontram nessa mesma classe como constantes no formato *<NAME>_SERVICE*.

A chamada típica de acesso a um serviço da plataforma é apresentado no Programa E-1. Por meio desses objetos podemos obter informações variadas do sistema como apresentado no exemplo do Programa E-2. Na versão mais atual do Android, existem mais de cinquenta serviços diferentes acessíveis usando *getSystemService*. Alguns desses serviços são apresentados na Tabela E-1.

```
<Class> service = (<Class>)
    <Context>.getSystemService(Context.<NAME>_SERVICE);
```

Programa E-1: Padrão de acesso a serviços da plataforma no Android.

```
DisplayManager displayManager = (DisplayManager)
    context.getSystemService(Context.DISPLAY_SERVICE);
int numberOfAvailableDisplays = displayManager.getDisplays().length;

PowerManager powerManager = (PowerManager)
    context.getSystemService(Context.POWER_SERVICE);
boolean isDeviceIdle = powerManager.isDeviceIdleMode();

Wi-FiManager Wi-FiManager = (Wi-FiManager)
    context.getSystemService(Context.WI-FI_SERVICE);
boolean isWi-FiEnable = Wi-FiManager.isWi-FiEnabled();
```

Programa E-2: Exemplo de acesso e uso de serviços da Plataforma.

O Programa E-3 demonstra o uso da interface de recebimento de eventos. A aplicação ou serviço deve cadastrar um objeto que deriva da classe *BroadcastReceiver* e identificar os tipos de eventos que tem interesse usando um *IntentFilter*. Para o cadastro é necessário acesso a uma instância da classe abstrata *Context* que representa uma interface global de acesso a informação e serviços do sistema, algumas das subclasses são as classes *Activity* e *Service* que representam respectivamente uma tela da aplicação e um serviço de segundo plano.

```
BroadcastReceiver broadcastReceiver = new BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        switch (action) {
            // identificação e tratamento do evento
        }
    }
};

IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(<ACTION>);
intentFilter.addAction(<ACTION>);
//... outras actions se necessário

// cadastro
<Context>.register(broadcastReceiver, intentFilter);

// remoção
<Context>.unregister(broadcastReceiver);
```

Programa E-3: Exemplo de padrão de uso da interface de registro para notificação de eventos do Android.

Tabela E-1: Exemplo dos serviços de sistema disponíveis.

Classe	**Identificador**	**Descrição**
AlarmManager	ALARM_SERVICE	Permite agendar execuções para um momento no futuro.
AudioManager	AUDIO_SERVICE	Permite acessar o volume e toque.
ClipboardManager	CLIPBOARD_SERVICE	Permite acesso ao *clipboard* global para colocar e recuperar textos.
ConnectivityManager	CONNECTIVITY_SERVICE	Permite inquirir o estado da conectividade de rede.
DisplayManager	DISPLAY_SERVICE	Permite gerenciar propriedades das telas conectadas ao sistema.
InputMethodManager	INPUT_METHOD_SERVICE	Ponto de acesso central ao arcabouço de métodos de entrada.
LocationManager	LOCATION_SERVICE	Prover acesso ao serviço de localização.
NetworkStatsManager	NETWORK_STATS_SERVICE	Permite acessar informações estatísticas e histórico de uso da rede.
NotificationManager	NOTIFICATION_SERVICE	Notifica o usuário de eventos que ocorreram em plano de fundo.

PowerManager	POWER_SERVICE	Permite controlar o estado de energia do dispositivo.
SensorManager	SENSOR_SERVICE	Acesso as sensores como acelerômetro, proximidade entre outros.
TelephonyManager	TELEPHONY_SERVICE	Acesso a informações de telefonia e registrar para receber notificações na mudança de estado.
Wi-FiManager	WI-FI_SERVICE	Acesso primário ao gerenciamento da conectividade Wi-Fi.

APÊNDICE F – EVENTOS MONITORADOS

Os eventos relacionados ao monitoramento de energia são listados na Tabela F-1. Eles permitem saber mudanças no estado da bateria, estado da tela e conectividade.

Tabela F-1: Principais eventos de interesse para monitoramento de energia.

Nome da Action	**Descrição**
`Intent .ACTION_AIRPLANE_MODE_CHANGED`	Notifica ativação ou desativação do modo avião.
`Intent.ACTION_BATTERY_CHANGED`	Notifica a cada 1 segundo o estado da bateria.
`Intent.ACTION_BATTERY_LOW`	Notifica quando a carga da bateria está baixa.
`Intent.ACTION_BATTERY_OKAY`	Notifica quando a carga da bateria volta ao normal.
`Intent.ACTION_POWER_CONNECTED`	Notifica quando conecta a uma fonte externa de energia.
`Intent.ACTION_POWER_DISCONNECTED`	Notifica quando desconecta da fonte externa.
`Intent.ACTION_SCREEN_OFF`	Notifica quando a tela foi desligada.
`Intent.ACTION_SCREEN_ON`	Notifica quando a tela foi ligada.
`Intent.ACTION_SHUTDOWN`	Notifica o desligamento do dispositivo.
`ConnectivityManager .CONNECTIVITY_ACTION`	Notifica alterações na interface de rede.
`Wi-FiManager .WI-FI_STATE_CHANGED_ACTION`	Notifica alterações na interface Wi-Fi.

O evento de mudança do estado da bateria, *ACTION_BATTERY_CHANGED*, possui uma série de informações adicionais que são detalhados na Tabela F-2.

Tabela F-2: Conteúdo adicional do evento de mudança do estado da bateria.

Extras do `ACTION_BATTERY_CHANGED`		
Extras	**Tipo**	**Descrição**
`BatteryManager .EXTRA_HEATH`	Inteiro	Saúde como definido em `BatteryManager.BATTERY_HEATH_*`
`BatteryManager .EXTRA_ICON_SMALL`	Inteiro	*Resource id* para o ícone da bateria.
`BatteryManager .EXTRA_LEVEL`	Inteiro	Valor atual do nível de bateria.
`BatteryManager .PLUGGED`	Inteiro	Valor definido como definido em `BatteryManager.PLUGGED_*` ou 0 caso

		não haja nenhuma.
`BatteryManager .EXTRA_PRESENT`	Booleano	**True** se bateria estiver presente.
`BatteryManager .EXTRA_SCALE`	Inteiro	Valor máximo do nível da bateria.
`BatteryManager .EXTRA_STATUS`	Inteiro	Status conforme definido em `BatteryManager.BATTERY_STATUS_*`
`BatteryManager .EXTRA_TECHNOLOGY`	Texto	Tipo de tecnologia da bateria.
`BatteryManager .EXTRA_TEMPERATURE`	Inteiro	Valor atual da temperatura.
`BatteryManager .EXTRA_VOLTAGE`	Inteiro	Valor atual da diferença de tensão.

O evento de entrada e saída do modo avião, *ACTION_AIRPLANE_MODE_CHANGED*, possui apenas um parâmetro extra que identifica se o modo está ativado, Tabela F-3.

Tabela F-3: Conteúdo adicional do evento de mudança do modo avião.

Extras do `ACTION_AIRPLANE_MODE_CHANGED`		
Extras	**Tipo**	**Descrição**
`"state"`	Booleano	Indica se está no modo avião.

O evento de mudança de conectividade, *CONNECTIVITY_ACTION*, identifica a interface de rede está em uso no momento, se é Wi-Fi ou móvel. O estado da rede é um parâmetro adicional, *EXTRA_NETWORK_INFO*, detalhado na Tabela F-4.

Tabela F-4: Conteúdo adicional do evento de mudança na conectividade.

Extras do `CONNECTIVITY_ACTION`		
Extras	**Tipo**	**Descrição**
`ConnectivityManager .FAILOVER_CONNECTION`	Booleano	Se a mudança de rede foi em decorrência de falha na anterior.
`ConnectivityManager .EXTRA_NO_CONNECTIVITY`	Booleano	Se existe conectividade a Internet.
`ConnectivityManager .EXTRA_NETWORK_INFO`	Obj. Serializado (`NetworkInfo`)	Detalhes da Interface de Rede.
`ConnectivityManager .EXTRA_REASON`	Texto	Razão da falha (opcional).

O evento de mudança na interface Wi-Fi, *WI-FI_STATE_CHANGED_ACTION*, permite saber o estado atual da rede Wi-Fi. Este é apresentado na Tabela F-5 e os estados possíveis na Tabela F-6.

Tabela F-5: Conteúdo adicional do evento de mudança na interface Wi-Fi.

Extras do `WI-FI_STATE_CHANGED_ACTION`		
Extras	**Tipo**	**Descrição**
`Wi-FiManager .EXTRA_WI-FI_STATE`	Inteiro	Estado atual conforme definido em `Wi-FiManager.WI-FI_STATE_*.`
`Wi-FiManager .EXTRA_PREVIOUS_WI-FI_STATE`	Inteiro	Estado anterior conforme definido em `Wi-FiManager.WI-FI_STATE_*`.

Tabela F-6: Lista de possíveis Wi-Fi states.

Estado	**Descrição**
`WI-FI_STATE_DISABLED`	Wi-Fi está desabilitado.
`WI-FI_STATE_DISABLING`	Wi-Fi está sendo desabilitado.
`WI-FI_STATE_ENABLED`	Wi-Fi esta habilitado.
`WI-FI_STATE_ENABLING`	Wi-Fi esta sendo habilitado.
`WI-FI_STATE_UNKNOWN`	O estado do Wi-Fi é desconhecido.

APÊNDICE G – INFORMAÇÕES MONITORADAS

MONITOR DE BATERIA

O monitoramento da bateria registra a 10 Hz as informações localizadas em **/sys/class/power_supply/battery**. Uma *thread* é responsável pela leitura dos arquivos e escrita do log. O arquivo de log recebe o nome *battmon_<ts>.log*, onde *ts* é a data de inicio do registro por exemplo: battmon_2015-9-20_13-15-10, representa uma captura que começou as 13:15:10 de 20/09/2015. São registrados os seguintes campos:

Horário da Amostra:

O horário de registro da amostra em microssegundos desde o inicio da era Unix, 1 de janeiro de 1970. É armazenado em duas partes, a primeira é um valor Inteiro de 32 bits representando os segundos desde o inicio da era Unix, a segunda um valor Inteiro de 32 bits representando o número de microssegundos.

Carga da Bateria:

Capacidade atual da bateria em %, por limitações da interface sempre é um valor inteiro de 0 a 100. O valor é extraído de **/sys/class/power_supply/battery/capacity**.

Corrente de Descarga:

A corrente de descarga da bateria em miliAmpère representada por um ponto flutuante de precisão simples. O valor é extraído como μAmpère e divido por 10^3. O arquivo fonte é **/sys/class/power_supply/battery/current_now**.

Voltagem da Bateria:

O valor da voltagem instantânea da bateria em Volts representado por um ponto flutuante de precisão simples. O valor é extraído como μVolts e dividido por 10^6. O arquivo fonte é o **/sys/class/power_supply/battery/voltage_now**.

Temperatura da Bateria:

O valor da temperatura instantânea da bateria em Celsius representado como ponto flutuante de precisão simples. O valor é extraído como deciCelsius e dividido por 10. O valor é extraído de **/sys/class/power_supply/battery/temp**.

MONITOR DE CPU

O monitoramento da CPU captura amostras a 10 Hz por uma *thread* dedicada. As informações são extraídas dos arquivos localizados em **/sys/devices/system/cpu/** e do **/proc/stat**. Os seguintes campos são registrados:

Horário da Amostra:

O horário de registro da amostra em microssegundos desde o inicio da era Unix. Assim como na amostra do monitor da bateria, são dois números inteiros de 32 bits que representa o número de segundos desde o inicio da era Unix e o número de microssegundo. O arquivo de log recebe o nome de *procmon_<ts>.log*, onde *ts* é a data e hora do horário de inicio do registro.

Estado do Núcleo:

Um número inteiro de 32 bits para cada núcleo do processador. Por simplicidade foi fixado em quatro, representando respectivamente núcleo 0, 1, 2 e 3. Quando o núcleo está ligado o valor é um, caso contrário é zero. Os valores são extraídos de **/sys/devices/system/cpu/cpu<n>/online** onde *n* é o indice do núcleo.

Frequência de operação de Núcleo:

Um número inteiro de 32 bits para cada núcleo do processador. Por simplicidade foi fixado em quatro, representando a frequência de operação do núcleo 0, 1, 2 e 3. O valor representa a frequência em Hz do processador extraído de **/sys/devices/system/cpu/cpu<n>/cpufreq/scaling_cur_freq** onde *n* é o indice do núcleo. No caso o núcleo desabilitado o valor registrado é zero.

Tempo Absoluto do Processador:

Quatro números inteiros de 32 bits para cada núcleo do processador, fixado em quatro totalizando dezesseis números. Os valores são lidos do arquivo **/proc/stat**. Representa respectivamente o tempo em *jiffies*, unidade arbitrária de tempo que varia de plataforma para plataforma, que mede o tempo gasto pela CPU em operações no modo usuário, *nice*, sistema e espera.

MONITOR DE REDE

O monitor de rede, assim como de bateria e processador realiza amostragem a 10 Hz. Duas interfaces são monitoradas, a interface Wi-Fi e interface de dados móveis. As informações registradas são obtidas de **/sys/class/net/wlan0/statistics/** para interface Wi-Fi. E **/sys/class/net/rmnet0/statistics/** ou **/sys/class/net/rmnet_usb0/statistics/** para interface de dados celular dependendo do modelo do aparelho. O arquivo de log recebe o nome de *netmon_<ts>.log*, onde *ts* é a data e hora do horário de inicio do registro. Os campos armazenados são os seguintes:

Horário da Amostra:

O horário de registro da amostra em microssegundos desde o inicio da era Unix. Assim como na amostra do monitor da bateria e processador são dois números inteiros de 32 bits representando o número de segundos desde o inicio da era Unix e o número de microssegundos.

Número de Bytes Recebidos e Transferidos:

O número de bytes como número inteiros de 64 bits. O número de bytes recebidos localizado no arquivo **rx_bytes**. O número de bytes enviados localizado no arquivo **tx_bytes**.

Número de Pacotes Recebidos e Transmitidos:

O número de pacotes como números inteiro de 64 bits. O número de pacotes recebidos localizados no arquivo **rx_packets**. O número de pacotes enviados localizado no arquivo **tx_packets**.

Número de Erros Registrados:

O número de erros como número inteiro de 64 bits. O número de erros de recepção localizados no arquivo **rx_errors**. O número de erros de transmissão localizados no arquivo **tx_errors**.

Número de Pacotes Descartados:

O número de pacotes descartados na transmissão e recepção com inteiro de 64 bits. O número de descarte na recepção localizado no arquivo **rx_dropped**. O número de descarte na transmissão localizado no arquivo **tx_dropped**.

REGISTRADOR DE EVENTOS DE SISTEMA

O registrador de evento, diferente dos monitores, não opera por amostragem (exceto o brilha da tela). Ele aguarda notificações assíncronas do Android. Consiste de uma classe chamada *SystemEventReceiver* que é herda de *BroadcastReceiver*. Os eventos apresentados na Tabela F-1 são registrados assim como o horário do evento em milissegundos desde o inicio da era Unix. Para a mudança de brilho da tela, enquanto a tela estiver ligada, a cada segundo é verificado a intensidade de brilho da tela e registrado um evento caso ocorra mudança, quando a tela é desligada essa verificação é suspensa. O tempo de um segundo foi escolhido pois o brilho da tela é raramente alterado. O arquivo de log recebe o nome de *system_event_<ts>.log*, onde *ts* é a data e hora do horário de inicio do registro.

Tabela G-1: Lista de eventos de interesse do DES.

Eventos	**Descrição**
AIRPLANE_MODE_ON	Aparelho habilitou o modo avião.
AIRPLANE_MODE_OFF	Aparelho desabilitou o modo avião.
BATTERY_CHARGING <MODE>	Bateria carregando via alimentação externa identificada por *mode*. *Mode* pode ser AC, USB ou Wireless.
BATTERY_DISCHARGING	Bateria descarregando.
BATTERY_CHARGE_FULL	Bateria completamente carregada.
BATTERY_NOT_CHARGING	Bateria conecta a fonte externa, mas não carregando.
BATTERY_LOW	Carga da bateria criticamente baixa.
BATTERY_OK	Carga da bateria restaurada a um nível aceitável.
BATTERY_LEVEL	Nível de carga da bateria de 0 a 100, notificado a cada mudança de nível.
DISPLAY_ON	A tela foi ligada. Dispara o monitoramento de brilho da tela.
DISPLAY_OFF	A tela foi desligada. Suspende o monitoramento de brilho da tela.
DISPLAY_BRIGHTNESS <0-255>	O nível de brilho foi alterado. O valor atual é informado em seguida como um valor entre 0 e 255. 0 o menor brilho possível, 255 o maior brilho possível.

DISPLAY_BRIGH_MODE_AUTO	O modo de seleção foi alterado para automático, o sistema operacional vai escolher o valor dependendo das condições de iluminação do local.
DISPLAY_BRIGH_MODE_MANUAL	O modo de seleção foi alterado para manual, o usuário ou aplicação vai escolher a intensidade que considerar mais adequada.
POWER_CONNECTED	O aparelho foi ligado a um fonte externa de energia.
POWER_DISCONNECTED	O aparelho foi desligado da fonte externa de energia.
<NETWORK> <DETAILED_STATE>	Mudança no estado interno da interface de rede reportadas por CONNECTIVITY_ACTION. DETAILED_STATE representa o enumerador NetworkInfo.DetailedState listado na Tabela G-3
<PREV_WI-FI_STATE> <NEXT_WI-FI_STATE>	Mudança de estado na interface Wi-Fi reportadas por WI-FI_STATE_CHANGED_ACTION. Os valores possíveis são apresentados na Tabela F-6

Tabela G-2: Algumas das interfaces de rede reportadas CONNECTIVITY_ACTION.

Interface de Rede	**Descrição**
MOBILE	Refere-se a interface de dados de telefonia celular 2G, 3G ou 4G.
WI-FI	Refere-se a interface Wi-Fi.
BLUETOOTH	Refere-se a interface Bluetooth.
ETHERNET	Refere-se a interface cabeada Ethernet, dificilmente encontrada em celulares.

Tabela G-3: Valores possíveis de NetworkInfo.DetailedState.

Detalhamento do estado da interface	**Descrição**
AUTHENTICATING	Conexão estabelecida, autenticação em progresso.
BLOCKED	O acesso a esta rede foi bloqueado.
CAPTIVE_PORTAL_CHECK	Verificando se rede precisa de autenticação via Web.
CONNECTED	Conectado. Tráfego de dados deve está disponível.
CONNECTING	Configurando conexão de dados.
DISCONNECTED	Desconectado, tráfego indisponível.
DISCONNECTING	Desconexão em progresso.
FAILED	Tentativa de conexão falhou.
IDLE	Pronto para estabelecer conexão.
OBTAINING_IPADDR	Aguardando resposta do servidor DHCP para atribuir endereço IP da conexão.

SCANNING	Buscando pontos de acesso.
SUSPENDED	Tráfego de dados suspenso.
VERIFYING_POOR_LINK	Problemas de conectividade na conexão.

APÊNDICE H - SUÍTE DE TESTE DE ENERGIA

A suíte automatizada consiste numa série de telas da aplicação DES projetadas para executar com ou sem interação humana um comportamento predeterminado a fim executar um cenário de teste no dispositivo. A seguir é apresentado os quatro cenários principais.

Teste de Base

O teste de base visa identificar o consumo mínimo necessário para o funcionamento do aparelho. É usado como um negativo para isolar o custo das componentes. Consiste em dois testes principais.

1. Consumo em estado de espera: consiste em medir o consumo do aparelho com todas as interfaces de rede e tela desligadas. Porém sem permitir que o aparelho entre em hibernação, isso é realizado adquirindo um *Wakelock*. É necessário fechar todas as aplicações em segundo plano para reduzir a variação no consumo.
2. Consumo em estado de hibernação: consiste em medir o consumo do aparelho em hibernação profunda, segue o mesmo procedimento do teste anterior, exceto que o *Wakelock* nunca é adquirido. O problema desse teste é que em hibernação não existe registro do consumo, portanto isso é inferido pelos limites que antecedem e sucedem a hibernação.

Teste de Tela

O teste de tela visa estabelecer a relação entre o conteúdo apresentado na tela com o consumo observado. Consiste no seguinte teste:

1. Consumo de brilho: A literatura destaca que a principal fonte de consumo nas telas LCD é a intensidade do brilho do *backlight*. Esse teste visa variar o brilho de 0 a 255 e mensurar o consumo, são usados um fundo preto, branco, vermelho, verde e azul para verificar se existe alguma relação entre a colação da tela e o consumo medido. Outro fato de interesse e definir se existe uma relação linear entre brilho e consumo.

Teste de Processamento

O teste de processamento busca avaliar o impacto das arquiteturas multinúcleos e práticas de DVFS no consumo de energia do processador.

Consiste de um teste de carga de processamento, onde a configuração da frequência e número de núcleos ativos e mantida constante através da desativação manual do controlador de DVFS. É construído uma matriz de consumo que relaciona o consumo do aparelho por frequência, número de núcleos e carga de processamento.

O segundo teste visa identificar verificar o consumo do processador adicionar para cada núcleo ativo, uma *thread* é criada para utilizar 100% do processamento do núcleo, a intervalos regulares uma nova *thread* é criada para forçar a ativação do próximo núcleo até todos os núcleos terem sido testados.

Teste de Rede

O teste de rede consiste em identificar os padrões de consumo e diferenças entre as redes Wi-Fi e redes de telefonia celular. O mesmo teste é aplicado em ambas as interfaces, porém com diferente quantidade de tráfego.

1. Ativação e Desativação da Interface: Este teste visa identificar o custo de ativação e desativação da interface incluindo o estabelecimento de conexão.
2. Transmissão e Recepção: Este teste visa identificar o custo de transmitir e receber dados pela interface. São usados diferentes magnitudes de dados para destacar a relação.
3. Consumo de Cauda: Esse é um problema citado pela literatura como principal responsável pelo alto consumo de energia da interface de dados móvel. Não existente na interface Wi-Fi. Consiste em transmitir uma rajada de dados e observar a duração da cauda. A rajada é enviada em intervalos variados para observar o impacto no consumo.

Procedimento de Teste

O procedimento geral de teste é composto de seis etapas:

1. Configuração: A aplicação DES é instalada no aparelho alvo, pré-condições de teste como bateria e conectividade são preparadas.
2. Execução: O monitoramento é iniciado, seguido pelo procedimento de teste escolhido.
3. Coleta: Os registros produzidos são copiados do celular para o computador.
4. Transformação: Os registros são convertidos do formato binário para textual usando programa em C e Python que decodificam as estruturas para texto,

cada linha é uma entrada registrada, as colunas são os valores registrados para aquela entrada no registro.

5. Plotagem e Processamento: As informações tabuladas são usadas por programa em R e Python para geração de gráficos e modelos lineares e não lineares de consumo para cada componente observado.
6. Análise Técnica: Os resultados são analisados com cuidado para identificar erros ou inconsistências nos testes. Caso observado um problema, os dados experimentais serão descartados. Caso correto, é feita uma analise cuidadosa do que foi observado, novos modelos ou gráficos são gerados para destacar as observações e conclusões são extraídas.

Gráficos Gerados

Para melhorar a visualização dos dados, os seguintes gráficos são automaticamente gerados pelo programa de plotagem descrito na seção anterior para as componentes:

Bateria

- Descarga de corrente no tempo decorrido em miliAmpère.
- Carga da bateria no tempo decorrido em por cento.
- Voltagem da bateria no tempo decorrido em Volts.
- Potência consumida no tempo decorrido em miliWatts.
- Histograma da potência.

CPU

- Número de núcleos do processador ativos no tempo decorrido.
- Carga de processamento instantânea por núcleo e total.
- Carga de processamento normalizada para um segundo por núcleo e total.
- Carga de processamento de usuário instantânea por núcleo e total.
- Carga de processamento de usuário normalizado para um segundo por núcleo e total.
- Carga de processamento de sistema instantânea por núcleo e total.

- Carga de processamento de sistema normalizado para um segundo por núcleo e total.
- Frequência instantânea por núcleo em MHz.
- Histograma da frequência instantânea por núcleo.
- Histograma da carga de processamento instantânea por núcleo e geral.
- Histograma da carga de processamento normalizado para um segundo por núcleo e geral.
- Histograma da carga de processamento de usuário instantânea por núcleo e geral.
- Histograma da carga de processamento de usuário normalizado para um segundo por núcleo e geral.
- Histograma da carga de processamento de sistema instantânea por núcleo e geral.
- Histograma da carga de processamento de sistema normalizado para um segundo por núcleo e geral.

Tela

- Nível de Brilho da Tela no tempo decorrido.
- Tela ativa ou inativa no tempo decorrido.

Rede (Wi-Fi e 3G)

- Transmissão instantânea no tempo decorrido.
- Recepção instantânea no tempo decorrido.
- Transmissão acumulada no tempo decorrido.
- Recepção acumulada no tempo decorrido.

REFERÊNCIAS BIBLIOGRÁFICAS

ABOUSALEH, M. et al. Determining Per-Mode Battery Usage within Non-trivial Mobile Device Apps. *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, p. 202–209, 2014.

AGARWAL, S. et al. Diagnosing mobile applications in the wild. *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks Hotnets 10*, p. 1–6, 2010.

AGARWAL, Y.; SCHURGERS, C.; GUPTA, R. Dynamic power management using on demand paging for networked embedded systems. *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, v. 2, p. 755–759, 2005.

ALAM, F. et al. Energy Optimization in Android Applications through Wakelock Placement. *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, p. 2–5, 2014.

ALAWNAH, S.; SAGAHYROON, A. Modeling smartphones power. *IEEE EuroCon 2013*, n. July, p. 369–374, 2013.

AMADEO, R. *The history of Android*. Disponível em: <http://arstechnica.com/gadgets/2014/06/building-android-a-40000-word-history-of-googles-mobile-os/>. Acesso em: 25 out. 2015.

ANAND, B. et al. Adaptive display power management for mobile games. *Proceedings of the 9th international conference on Mobile systems, applications, and services (MobiSys'10)*, p. 57–70, 2011.

ANANTHANARAYANAN, G.; STOICA, I. Blue-Fi: enhancing Wi-Fi performance using bluetooth signals. *Proceedings of the 7th international conference on Mobile systems, applications, and services*, p. 249–262, 2009.

ANASTASI, G. et al. 802.11 power-saving mode for mobile computing in Wi-Fi hotspots: Limitations, enhancements and open issues. *Wireless Networks*, v. 14, n. 6, p. 745–768, 2008.

APPLE. *App Store*. Disponível em: <https://www.apple.com/appstore>. Acesso em: 25 out. 2015.

BALASUBRAMANIAN, N.; BALASUBRAMANIAN, A.; VENKATARAMANI, A. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. *ACM Internet Meausurement Conference*, p.

280–293, 2009.

BARETH, U. *Simulating power consumption of location tracking algorithms to improve energy-efficiency of smartphonesProceedings - International Computer Software and Applications Conference*. *Anais*...Izmir: IEEE, 2012

BELLOSA, F. The benefits of event: driven energy accounting in power-sensitive systems. *Proceedings of the 9th workshop on ACM SIGOPS European workshop beyond the PC: new challenges for the operating system - EW 9*, p. 37, 2000.

BÖHMER, M. et al. Falling asleep with Angry Birds, Facebook and Kindle – A Large Scale Study on Mobile Application Usage. *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services - MobileHCI '11*, p. 47, 2011.

BRODOWSKI, D.; GOLDE, N. *Linux CPUFreq, CPU Freq Governors: information for users and developers*. Disponível em: <https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt>. Acesso em: 25 out. 2015.

BROUWERS, N.; ZUNIGA, M.; LANGENDOEN, K. *NEAT: a novel energy analysis toolkit for free-roaming smartphones. Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems - SenSys '14*. *Anais*...New York, New York, USA: ACM Press, 2014Disponível em: <http://dl.acm.org/citation.cfm?doid=2668332.2668337>

BROWN, A. L.; WYSOCKI, R. J. Suspend-to-RAM in Linux. *Linux Symposium*, v. 1, p. 39, 2008.

CARROLL, A.; HEISER, G. An analysis of power consumption in a smartphone. *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, p. 21–21, 2010.

CARROLL, A.; HEISER, G. Mobile Multicores : Use Them or Waste Them. *Proceedings of the Workshop on Power-Aware Computing and Systems - HotPower '13*, p. 1–5, 2013.

CARROLL, A.; HEISER, G. *Unifying DVFS and offlining in mobile multicores2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. *Anais*... Berlin: IEEE, 2014. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6926010>

CHEN, X. et al. How is Energy Consumed in Smartphone Display Applications? *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, v. 1, n.

412, p. 3:1–3:6, 2013.

CHEN, X. et al. FingerShadow: an OLED power optimization based on smartphone touch interactions. *Proceedings of the 6th Workshop on Power-Aware Computing and Systems (HotPower '14)*, 2014.

CREUS, G. B.; KUULUSA, M. Optimizing Mobile Software with Built-in Power Profiling. *Mobile Phone Programming*, p. 449–462, 2007.

CUERVO, E.; BALASUBRAMANIAN, A. MAUI: making smartphones last longer with code offload. *MobiSys '10 Proceedings of the 8th international conference on Mobile systems, applications, and services*, v. 17, p. 49–62, 2010.

DEVELOPERS, A. *PowerManager*. Disponível em: <http://developer.android.com/reference/android/os/PowerManager.html>. Acesso em: 25 out. 2015a.

DEVELOPERS, A. *PowerManager.WakeLock*. Disponível em: <http://developer.android.com/reference/android/os/PowerManager.WakeLock.html>. Acesso em: 25 out. 2015b.

DEVELOPERS, A. *Support App Store*. Disponível em: <https://developer.apple.com/support/app-store/>. Acesso em: 25 out. 2015c.

DEVELOPERS, A. *Dashboards*. Disponível em: <https://developer.android.com/about/dashboards/index.html>. Acesso em: 25 out. 2015.

DING, F. et al. Monitoring energy consumption of smartphones. *Proceedings - 2011 IEEE International Conferences on Internet of Things and Cyber, Physical and Social Computing, iThings/CPSCom 2011*, p. 610–613, 2011.

DING, N. et al. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. *ACM SIGMETRICS Performance Evaluation Review*, v. 41, p. 29, 2013.

DOGAR, F. R.; STEENKISTE, P. Catnap : Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices. *Energy*, p. 107–122, 2010.

DONG, M.; ZHONG, L. Sesame: Self-Constructive System Energy Modeling for Battery-Powered Mobile Systems. *MobiSys '11 Proceedings of the 9th international conference on Mobile systems, applications, and services*, p. 335–348, 2011.

EASON, J. *Develop a sweet spot for Marshmallow: Official Android 6.0 SDK & Final M Preview*. Disponível em: <http://android-developers.blogspot.com.br/2015/08/m-developer-preview-3-final-sdk.html>. Acesso em: 25 out. 2015.

FALAKI, H.; GOVINDAN, R.; ESTRIN, D. Smart screen management on

mobile phones. *Energy*, 2009.

FARAWAY, J. J. *Practical Regression and Anova using R*. [s.l: s.n.]. v. 21

FIORE, U. et al. Multimedia-Based Battery Drain Attacks for Android Devices. *Consumer Communications and Networking Conf. (CCNC)*, n. Ccnc, p. 347–352, 2014.

FLINN, J.; SATYANARAYANAN, M. *PowerScope: A tool for profiling the energy usage of mobile applicationsProceedings - WMCSA'99: 2nd IEEE Workshop on Mobile Computing Systems and Applications*. *Anais*...IEEE, 1999

FRIEDMAN, R.; KOGAN, A.; KRIVOLAPOV, Y. On power and throughput tradeoffs of WiFi and bluetooth in smartphones. *IEEE Transactions on Mobile Computing*, v. 12, p. 1363–1376, 2013.

GNU. *Octave Documentation: Linear Least Squares*. Disponível em: <https://www.gnu.org/software/octave/doc/interpreter/Linear-Least-Squares.html>. Acesso em: 25 jan. 2016.

GOOGLE. *Google Play*. Disponível em: <https://play.google.com/store>. Acesso em: 25 out. 2015.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*. 5th. ed. [s.l.] Morgan Kaufmann, 2012.

HONG, S.; KIM, H. An integrated GPU power and performance model. *ACM SIGARCH Computer Architecture News*, v. 38, n. 3, p. 280, 2010.

HUANG, J. et al. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. *MobiSys*, p. 225–238, 2012.

IDC. *Android and iOS Squeeze the Competition, Swelling to 96.3% of the Smartphone Operating System Market for Both 4Q14 and CY14, According to IDC.* Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS25450615>. Acesso em: 25 out. 2015.

JINDAL, A. et al. *Hypnos: understanding and treating sleep conflicts in smartphones. Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13*. *Anais*...New York, New York, USA: ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2465351.2465377>

JUNG, W. et al. DevScope: a nonintrusive and online power analysis tool for smartphone hardware components. *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES+ISSS '12*, p. 353, 2012.

JUNG, W.; KIM, K.; CHA, H. UserScope: A fine-grained framework for collecting energy-related smartphone user contexts. *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, p. 158–165, 2013.

KALIC, G.; BOJIC, I.; KUSEK, M. Energy consumption in android phones when using wireless communication technologies. *MIPRO, 2012 Proceedings of the 35th …*, p. 754–759, 2012.

KAMIYAMA, T.; INAMURA, H.; OHTA, K. A model-based energy profiler using online logging for Android applications. *2014 7th International Conference on Mobile Computing and Ubiquitous Networking, ICMU 2014*, p. 7–13, 2014.

KILLIAN, T. J. Processes as Files. *Summer Usenix Conference*, 1984.

KIM, K. H. et al. Improving energy efficiency of Wi-Fi sensing on smartphones. *Proceedings - IEEE INFOCOM*, p. 2930–2938, 2011.

KIM, M.; KONG, J.; CHUNG, S. W. Enhancing online power estimation accuracy for smartphones. *IEEE Transactions on Consumer Electronics*, v. 58, p. 333–339, 2012a.

KIM, M.; KONG, J.; CHUNG, S. W. An online power estimation technique for multi-core smartphones with advanced display components. *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*, p. 666–667, 2012b.

KJÆRGAARD, M. B.; BLUNCK, H. Unsupervised power profiling for mobile devices. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, v. 104 LNICST, p. 138–149, 2012.

LEE, J. et al. *PhonePool: On energy-efficient mobile network collaboration with provider aggregation2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. *Anais*...Singapore: IEEE, 2014Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6990396>

LEE, J.; JOE, H.; KIM, H. Automated Power Model Generation Method for Smartphones. *IEEE Transactions on Consumer Electronics*, v. 60, n. 2, p. 190–197, 2014.

LEE, Y.-S.; CHO, S.-B. An Efficient Energy Management System for Android Phone Using Bayesian Networks. *2012 32nd International Conference on Distributed Computing Systems Workshops*, p. 102–107, 2012.

LI, D. et al. *An Empirical Study of the Energy Consumption of Android Applications. 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. *Anais*...Victoria, BC: IEEE, 2014aDisponível em:

<http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6976078&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6976078>

LI, D.; HALFOND, W. G. J. An investigation into energy-saving programming practices for Android smartphone app development. *Proceedings of the 3rd International Workshop on Green and Sustainable Software - GREENS 2014*, p. 46–53, 2014.

LI, D.; TRAN, A. H.; HALFOND, W. G. J. Making Web Applications More Energy Efficient for OLED Smartphones. *Icse'14*, 2014.

LI, J. et al. *PowerGuide: Accurate Wi-Fi power estimator for smartphones. The 16th Asia-Pacific Network Operations and Management Symposium*. *Anais*...Hsinchu: IEEE, 2014bDisponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6996567>

LINDEN, D.; REDDY, T. B. *Handbook Of Batteries*. 3rd. ed. [s.l.] McGraw-Hill Professional, 2001.

LIU, X.; SHENOY, P.; CORNER, M. *Chameleon: Application Level Power Management with Performance Isolation*. *Proceedings of the 13th annual ACM international conference on Multimedia - MULTIMEDIA '05*. *Anais*...ACM New York, 2005Disponível em: <http://dl.acm.org/citation.cfm?id=1101332>

LIU, Y. et al. GreenDroid: Automated Diagnosis of Energy Inefficiency for Smartphone Applications. *IEEE Transactions on Software Engineering*, v. 40, n. 9, p. 1–1, 2014.

LUKEFAHR, A. et al. Heterogeneous Microarchitectures Trump Voltage Scaling for Low-Power Cores Categories and Subject Descriptors. *Proceedings of the 23rd international conference on Parallel architectures and compilation*, p. 237–250, 2014.

MA, X. et al. eDoctor: automatically diagnosing abnormal battery drain issues on smartphones. *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, p. 57–70, 2013.

MACK, C. A. Fifty years of Moore's law. *IEEE Transactions on Semiconductor Manufacturing*, v. 24, n. 2, p. 202–207, 2011.

MATHWORKS. *Matlab Documentation: Regression Analysis*. Disponível em: <http://www.mathworks.com/help/symbolic/regression-analysis.html?>. Acesso em: 25 jan. 2016.

MCKENNY, P. E. *Is Parallel Programming Hard, And, If So, What Can You Do About It?* [s.l: s.n.].

MERRITT, E. *Gnuplot Documentation: Fit*. Disponível em: <http://gnuplot.sourceforge.net/docs_4.2/node82.html>. Acesso em: 27 jan. 2016.

MOCHEL, P. The sysfs Filesystem. *Linux Symposium*, p. 313, 2005.

MONSOON SOLUTIONS, I. *Monsoon Power Monitor*. Disponível em: <https://www.msoon.com/LabEquipment/PowerMonitor/>. Acesso em: 20 out. 2015.

NIST. *Finite state machine*. Disponível em: <https://xlinux.nist.gov/dads//HTML/finiteStateMachine.html>. Acesso em: 25 jan. 2016.

NIXON, K. W. et al. Mobile GPU Power Consumption Reduction via Dynamic Resolution and Frame Rate Scaling. *HotPower'14 Proceedings of the 6th USENIX conference on Power-Aware Computing and Systems*, p. 5–10, 2014.

OLINER, A. J. et al. Carat: Collaborative Energy Diagnosis for Mobile Devices. *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, p. 10:1–10:14, 2013.

OLIVER, E.; KESHAV, S. *Data Driven Smartphone Energy Level Prediction*. Ontario, Canada: [s.n.]. Disponível em: <https://cs.uwaterloo.ca/research/tr/2010/CS-2010-06.pdf>.

OPENSIGNAL. *Android Fragmentation Visualized (August 2015)*. Disponível em: <http://opensignal.com/reports/2015/08/android-fragmentation/>. Acesso em: 25 out. 2015.

PATHAK, A. et al. Fine-Grained Power Modeling for Smartphones Using System Call Tracing. *Proceedings of the sixth conference on Computer systems EuroSys 11*, p. 153, 2011.

PATHAK, A. et al. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. *Proceedings of the 10th international conference on Mobile systems, applications, and services - MobiSys '12*, p. 267, 2012.

PATHAK, A.; HU, Y. C.; ZHANG, M. Bootstrapping Energy Debugging on Smartphones: A First Look at Energy Bugs in Mobile Devices. *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, p. 5:1–5:6, 2011.

PATHAK, A.; HU, Y. C.; ZHANG, M. Fine Grained Energy Accounting on Smartphones with Eprof. *Power*, p. 29–42, 2012.

PERING, T. et al. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. *MobiSys '06 Proceedings of the 4th international conference on Mobile systems, applications, and services*, p. 220–232, 2006.

PERRUCCI, G. P.; FITZEK, F. H. P.; WIDMER, J. *Survey on energy consumption entities on the smartphone platformIEEE Vehicular Technology Conference*. *Anais*...Yokohama: IEEE, 2011

PIMENTEL, C. J. L. *Comunicação Digital*. Rio de Janeiro: Brasport, 2007.

PROCHKOVA, I.; SINGH, V.; NURMINEN, J. K. Energy cost of advertisements in mobile games on the android platform. *Proceedings - 6th International Conference on Next Generation Mobile Applications, Services, and Technologies, NGMAST 2012*, p. 147–152, 2012.

QIAN, F. et al. Profiling resource usage for mobile applications. *Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11*, p. 321, 2011.

ROUSSEEUW, P. J.; LEROY, A. M. Robust Regression and Outlier Detection. *Time*, v. 3, p. 329, 1987.

SCHULMAN, A et al. Demo: Phone power monitoring with BattOr. *MobiCom 2011*, 2011.

SHIN, D. et al. Dynamic voltage scaling of OLED displays. *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, p. 53–58, 2011.

SHYE, A.; SCHOLBROCK, B.; MEMIK, G. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, p. 168–178, 2009.

SON, D. O. et al. *Analysis on the Power Efficiency of Mobile Systems Varying Device Parameters2014 International Conference on IT Convergence and Security (ICITCS)*. *Anais*...Beijing: IEEE, 2014

TARKOMA, S. et al. *Smartphone energy consumption: modeling and optimization*. [s.l.] Cambridge University Press, 2014.

TEKTRONIX INC. *Wi-Fi : Overview of the 802.11 Physical Layer and Transmitter Measurements*. [s.l: s.n.]. Disponível em: <http://www.cnrood.com/public/docs/WiFi_Physical_Layer_and_Transm_Meas.pdf>.

TOSHIBA. *NAND vs NOR flash Memory - Technology OverviewToshiba*. Irvine, CA: [s.n.]. Disponível em: <http://aturing.umcs.maine.edu/~meadow/courses/cos335/Toshiba NAND_vs_NOR_Flash_Memory_Technology_Overviewt.pdf>.

VALLINA-RODRIGUEZ, N.; CROWCROFT, J. Energy Management

Techniques in Modern Mobile Handsets. *IEEE Communications Surveys & Tutorials*, p. 1–20, 2012.

WANG, C. et al. Power estimation for mobile applications with profile-driven battery traces. *Proceedings of the International Symposium on Low Power Electronics and Design*, p. 120–125, 2013.

WYSOCKI, R. J. Technical Background of the Android Suspend Blockers Controversy. p. 1–18, 2010.

XIAO, Y. et al. *A system-level model for runtime power estimation on mobile devicesProceedings - 2010 IEEE/ACM International Conference on Green Computing and Communications, GreenCom 2010, 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, CPSCom 2010*. *Anais*...Washington DC: IEEE, 2010

XIAO, Y. et al. Modeling Energy Consumption of Data Transmission Over Wi-Fi. *IEEE Transactions on Mobile Computing*, v. 13, n. 8, p. 1760–1773, 2014.

XU, C. et al. Context-aware Global Power Management for Mobile Devices Balancing Battery Outage and User Experience. *HotMobile 2014*, p. 2012, 2014.

XU, F. et al. V-edge: fast self-constructive power modeling of smartphones based on battery voltage dynamics. *nsdi'13 Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, p. 43–55, 2013.

YOON, C. et al. AppScope: application energy metering framework for android smartphones using kernel activity monitoring. *Usenix Atc'12*, p. 36, 2012.

ZHANG, L. et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, p. 105–114, 2010.

ZHANG, Y. et al. *Towards better CPU power management on multicore smartphonesProceedings of the Workshop on Power-Aware Computing and Systems - HotPower '13*. *Anais*...New York, New York, USA: ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2525526.2525849>

ZHANG, Y. et al. *Accurate CPU Power Modeling for Multicore Smartphones*. [s.l: s.n.]. Disponível em: <http://research.microsoft.com/apps/pubs/default.aspx?id=238914>.