
Chanmin Yoon*, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, Hojung Cha

Dept. of Computer Science
Yonsei University, Korea

USENIX ATC’ 12
2012.06.15
Motivation

Q) I find it is $100mW$ when I just run my app, and it is $20mW$ when I do nothing. I think $80mW$ is consumed by my app. But it is $200mW$ I run another app B and my app also run, and it is $160mW$ when I just run app B, so my app also consume $40mW$? Which one is correct?

..., So I want to know how to estimate power consumption correctly?

A) …only use the radios when necessary. …
Motivation

• Why application/component energy information is valuable?

Application/Component Power Metering
Challenge

How can we estimate Application Energy?

Power Models

• Linear regression models
  a. MANTIS
  b. Lasso regression
  c. Others

• Non-linear regression models
  a. Exponential
  b. SVM
  c. Others

• Finite-state machine models
  • System call-based

Utilization-based Model

\[
E^\text{App} = \sum_{i=0}^{\# \text{of Comp}} (\beta_i \times x_i^\text{App}) \times d_i^\text{App}
\]

\[\beta_i \rightarrow \text{Power coefficient value}\]

\[x_i^\text{App} \rightarrow \text{Utilization}\]

\[d_i^\text{App} \rightarrow \text{Activated duration}\]

Hardware Component Usage
How can we estimate Application Energy?

Conventional methods to get Hardware component usage

- Reading hardware performance counter
  - Very accurate
  - Dependency on processor architecture

- Reading `/proc`, `/sys` file system
  - Update rate problem
    - e.g., CPU utilizations/frequencies
  - GPS, display, cellular?

- Using `BatteryStat` class
  - Update rate problem
  - Information granularity problem
  - It’s a Java class

Limitations
- Accuracy
- Granularity
- Real-time
Objectives

AppScope | Application Energy Metering Framework for Android Smartphone

- Online and autonomous estimation in real-time
  - No external measurement device
- Fine-grained energy consumption information
  - Process & hardware component-level granularity
- System portability
  - No modification in system software
The AppScope Framework

1. **Kernel Activity Monitoring**
   - **Event Detector**
     - Binder IPC data
   - Request to use H/W by system call

2. **Hardware Component Usage Analyzer**
   - $x_i, d_i$
   - binder_ioctl, ioctl, socket, read, write, ...

3. **Application Energy Estimator**
   - **Component Power Models**
     - DevScope
     - Fuel-gauge IC
     - Others
     - External Devices

**Estimation Result**

**AppScope employs the DevScope’s power model**
**Online modeling**
- Android application
- Assume built-in fuel-gauge IC

**Non-intrusive power modeling**
- Probe OS, H/W component
- Monitor fuel-gauge IC
- Component-specific
- Training set generation
  - Workload
  - Control scenario

---

**Component Power Model**

<table>
<thead>
<tr>
<th>Component</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>$p_{CPU} = p_{CPU}^{freq} \times u + p_{CPU}^{idle}$, u: utilization, 0 ≤ u ≤ 100 freq: frequency index, $j_{freq}=0,1,2,\cdots,n$</td>
</tr>
<tr>
<td>LCD</td>
<td>$p_{LCD} = \beta_{LCD}^{b}$, b: brightness level, MIN(level) ≤ b ≤ MAX(level)</td>
</tr>
<tr>
<td>WiFi</td>
<td>$p_{WIFI} = \begin{cases} p_{WIFI}^{idle} + p_{WIFI}^{base} &amp; \text{if } p \leq t \ p_{WIFI}^{idle} + p_{WIFI}^{base} &amp; \text{if } p &gt; t \end{cases}$, p: packet rate, t: threshold</td>
</tr>
<tr>
<td>Cellular(3G)</td>
<td>$p^{3G} = \begin{cases} p^{idle,3G} &amp; \text{if } \text{RRC state is IDLE} \ p^{FACH,3G} &amp; \text{if } \text{RRC state is FACH} \ p^{DCH,3G} &amp; \text{if } \text{RRC state is DCH} \end{cases}$</td>
</tr>
<tr>
<td>GPS</td>
<td>$p^{GPS} = \beta_{on}^{GPS}$, if GPS is on</td>
</tr>
</tbody>
</table>
Kernel Activity Monitoring

• How to detect hardware component operation?
  – Event-driven approach

Event of component operation

Duration/Utilization of operation

<table>
<thead>
<tr>
<th>Component</th>
<th>Event</th>
<th>Duration/Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>245000 Hz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>576000 Hz</td>
<td></td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>recv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xmit</td>
<td></td>
</tr>
<tr>
<td>Cellular</td>
<td>IDLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FACH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DCH</td>
<td></td>
</tr>
<tr>
<td>Display</td>
<td>On</td>
<td>Launcher, Browser</td>
</tr>
<tr>
<td>GPS</td>
<td>On</td>
<td>Launcher, SNS, LBS</td>
</tr>
</tbody>
</table>
AppScope Implementation with “Kprobes”

Kernel path

- System calls
- Kernel functions
- Binder calls

Event detector
- binder_ioctl()
- binder_transaction()
- cpufreq_cpu_put()
- sched_switch()
- dev_queue_xmit()
- netif_rx()
- ...

Instrumentation routine
- 0x81808c144 INT03 Break
- Probe Handler

Hardware Component Usage Analyzer
- Parameters
- IPC data
- CPU frequency/utilization
- WiFi transmission packet rate
- LCD display duration
- GPS activated duration
- 3G network connection type
- ...

Return to original path

End
Hardware Component Usage Analyzer

Linux Kernel

Event

H/W Component Usage Analyzer

Governor Interface

Network Interface

Binder Driver

CPU
- Utilization/Frequency

Wi-Fi
- Transferred Packet Rates

Cellular
- Power States Transition

Display
- Foreground Duration

GPS
- Activation Duration

Analyze kernel functions

Analyze binder IPC data
Why analyzing the Binder IPC?

User space
- 3G Radio connection type
- Not available in kernel

Binder driver
- Binder driver works in kernel
- Can probe IPC data

Hooking by Kprobes

Analyze RPC data to obtain radio connection type
Inspecting Binder IPC data

1. Hook binder_transaction()
2. Extract RPC code/data
3. Check RPC Code
4. Read RPC Data

Hardware Operation

Binder IPC Data
- handle
- RPC Data
- RPC Code
- Protocol

Service Server
- Service
  - foo()

Binder Driver (/dev/binder)

# define BC_TRANSACTION
# define BC_REPLY
# define BR_TRANSACTION
# define BR_REPLY

struct binder_transaction_data {  
  union {  
    size_t handle;
    void *ptr;
  } target;
  void *cookie;
  code;  
  flags;
  sender_pid;
  sender_euid;
  data_size;
  offset_size;
  union {  
    struct {  
      const void *buffer;
      const void *offsets;
    } ptr;
    uint8_t buf[8];
  } data;
};
Hardware Component Usage Analysis (1)

WiFi
- DAI layer

\[ p_{WIFI} = \begin{cases} 
\beta_i^{WIFI} \times p + \beta_{i,\text{base}}, & \text{if } p \leq t \\
\beta_h^{WIFI} \times p + \beta_{h,\text{base}}, & \text{if } p > t 
\end{cases} \]

where \( p \) is the packet rate and \( t \) is the threshold.

CPU
- DFVS governor

\[ p_{CPU} = \beta_{\text{freq},u} + \beta_{\text{idle}} \]

where \( u \) is the utilization, \( 0 \leq u \leq 100 \), \( freq \) is the frequency index, and \( freq = 0, 1, 2, \ldots, n \).
Hardware Component Usage Analysis (2)

3G
- Network connection type → RRC state transition

\[ p^{3G} = \begin{cases} 
\beta^{3G}_{\text{IDLE}}, & \text{if RRC state is IDLE} \\
\beta^{3G}_{\text{FACH}}, & \text{if RRC state is FACH} \\
\beta^{3G}_{\text{DCH}}, & \text{if RRC state is DCH} 
\end{cases} \]

Display
- Activity Manager IPC data

\[ p^{\text{LCD}} = \beta^{\text{LCD}}_b \]
\[ b: \text{brightness level}, \quad \text{MIN}(\text{level}) \leq b \leq \text{MAX}(\text{level}) \]

GPS
- \text{loc_api()}
- LocationManager IPC data

\[ p^{\text{GPS}} = \beta^{\text{GPS}}_{\text{on}}, \text{if GPS is on} \]
## Evaluation (1)

<table>
<thead>
<tr>
<th>Component Usage Monitoring</th>
<th>Energy Metering Validation</th>
<th>Overhead Analysis</th>
<th>Real Application Energy Metering</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Hardware event detection • Hardware usage statistics</td>
<td>• Granularity of information • Accuracy of power metering</td>
<td>• CPU overhead • Power overhead</td>
<td>• Case Study • Error analysis</td>
</tr>
<tr>
<td>• 6 test apps • Pre-defined workload • Workload scheduling</td>
<td>• DevScope power model • Per UID • Per Component • Vs. Monsoon</td>
<td>• Loaded case • Unloaded case</td>
<td>• Angry Birds • Skype (WiFi) • Browser (WiFi) • Browser (3G) • Google Maps</td>
</tr>
</tbody>
</table>
Evaluation (2)

• Development environments
  – Linux kernel 2.6.35.7
  – SystemTap 1.3 (also uses Kprobes)
  – Android platform 2.3

• Device
  – HTC Google Nexus One
    • Qualcomm QSD 8250 Snapdragon 1GHz
    • 3.7-inch Super LCD display
    • MAXIM DS2784 Fuel-gauge-IC
  – External measurement device
    • The Monsoon Power Monitor
Test Scenario

- 1 “Master” and 5 “Slave” applications

Set pre-defined workloads, ready to control the Slaves

- Master
- Slave 1
- Slave 2

Change foreground activity

- Transmit 2,000 packets via WiFi
- LCD
- CPU

Transmit data via 3G interface

- Start CPU job
- Slave 4
- 3G

Turn on GPS interface

- Transmit data via 3G interface
- Slave 5
- GPS
Component Usage Monitoring (1)

Medium frequency (766/576 Mhz) due to OnDemand policy
Component Usage Monitoring (2)

Packet rate is 100 pps for 20 secs

"System" uses GPS for 2 ~ 4.5 sec
Component Usage Monitoring (3)

Data transferring during 20 seconds by slave4

1000 ms

RRC state transition

Activity change: master to slave2

Activity change: slave2 to master

Activity change occurs in ActivityManager

Turned on

“System” transmits some data
## Power Model for Google Nexus One (N1)

### Component-specific DevScope Power Models

<table>
<thead>
<tr>
<th>Component</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>[P_{CPU} = \beta_{freq} \times u + \beta_{idle}]</td>
</tr>
<tr>
<td></td>
<td>(u: ) utilization, (0 \leq u \leq 100)</td>
</tr>
<tr>
<td></td>
<td>(freq: ) frequency index, (freq = 0,1,2 \ldots, n)</td>
</tr>
<tr>
<td>LCD</td>
<td>[P_{LCD}^{b} = \beta_{b}^{LCD}]</td>
</tr>
<tr>
<td></td>
<td>(b: ) brightness level, (MIN(\text{level}) \leq b \leq MAX(\text{level}))</td>
</tr>
<tr>
<td>WiFi</td>
<td>[P_{WIFI}^{p,t} = \begin{cases} \beta_{WIFI}^{t} \times p + \beta_{base}\text{, if } p \leq t \ \beta_{WIFI}^{t} \times p + \beta_{base}\text{, if } p &gt; t \end{cases}]</td>
</tr>
<tr>
<td></td>
<td>(p: ) packet rate, (t: ) threshold</td>
</tr>
<tr>
<td>cellular(3G)</td>
<td>[P_{3G}^{RRC} = \begin{cases} \beta_{3G}^{IDLE}\text{, if RRC state is IDLE} \ \beta_{3G}^{FACH}\text{, if RRC state is FACH} \ \beta_{3G}^{DCH}\text{, if RRC state is DCH} \end{cases}]</td>
</tr>
<tr>
<td>GPS</td>
<td>[P_{GPS} = \beta_{on}\text{, if GPS is on}]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comp.</th>
<th>Index</th>
<th>Coefficient</th>
<th>Comp.</th>
<th>Index</th>
<th>Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(freq)</td>
<td>(\beta_{freq})</td>
<td>(\beta_{idle})</td>
<td>LCD</td>
<td>(b)</td>
</tr>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>245.0</td>
<td>201.0</td>
<td>35.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>384.0</td>
<td>257.2</td>
<td>39.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>460.8</td>
<td>286.0</td>
<td>35.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>499.2</td>
<td>303.7</td>
<td>36.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>576.0</td>
<td>332.7</td>
<td>39.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WiFi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>652.8</td>
<td>378.4</td>
<td>36.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>860.4</td>
<td>470.7</td>
<td>38.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>844.8</td>
<td>493.1</td>
<td>43.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPS</td>
<td>ON</td>
<td>(\beta_{gps})</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>354.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WiFi</td>
<td>Transmit</td>
<td>1.2</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Base</td>
<td>238.7</td>
<td>247.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Threshold</td>
<td>25</td>
<td>25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Source:** Mobile Embedded System Lab., Yonsei University
System automatically activated the 3G interface after WiFi off. But, AppScope cannot detect this.

Limitation of CPU power model in AppScope (cache memory, I/O operation, ... ?)
Overhead Analysis

- **8.4J energy overhead for 240 seconds**
- **About 2.1% CPU overhead on average**
Real Application Energy Metering (1)

Error is caused by timer bug in SystemTap
Real Application Energy Metering (2)

- GPU error

Angry Birds

GPU mismodeling

StabilityTest for GPU
Limitations

- **Processor power modeling**
  - No consideration on GPU
  - Do not cope with multi-core processor architecture
  - No consideration on memory component
    - CPU-bound job vs. Memory-bound job

- **Tail-state energy estimation**
  - Limitation of linear power model (c.f. FSM power model)

- **Hardware components**
  - OLED display
  - Sensors: INS, MIC, Camera, ...
## Related Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android Built-in Battery Info</td>
<td>• This does not provide fine-grained power profile</td>
</tr>
<tr>
<td>PowerTop</td>
<td>• This is not available for smartphones</td>
</tr>
<tr>
<td>Trepn Profiler [Qualcomm]</td>
<td>• Hardware sensor-based power profiler</td>
</tr>
<tr>
<td></td>
<td>• This is only available on Snapdragon MDP</td>
</tr>
<tr>
<td>Energy Profiler [Nokia]</td>
<td>• Device power consumption</td>
</tr>
<tr>
<td></td>
<td>• External APIs for testing a application</td>
</tr>
<tr>
<td></td>
<td>• Developer’s solution</td>
</tr>
<tr>
<td>PowerTutor</td>
<td>• State of the ART</td>
</tr>
</tbody>
</table>
## AppScope Vs. PowerTutor

<table>
<thead>
<tr>
<th>PowerTutor</th>
<th>AppScope</th>
</tr>
</thead>
<tbody>
<tr>
<td>An Android application</td>
<td>Linux kernel module (+ External power profilers)</td>
</tr>
<tr>
<td>Polling using Android <em>BatteryStat</em></td>
<td>Event-driven using Linux <em>Kprobes</em></td>
</tr>
<tr>
<td>Application (UID) level</td>
<td>Process level</td>
</tr>
<tr>
<td>Reading <code>/proc</code> and <code>/sys</code>, Using Android API, Using modified Android framework</td>
<td>Monitoring kernel function call</td>
</tr>
<tr>
<td>CPU, LCD/OLED, WIFI, 3G data, GPS, AUDIO</td>
<td>CPU, LCD, WIFI, 3G data + voice call, GPS</td>
</tr>
</tbody>
</table>
The AppScope Project

App Developer

System Software Developer

End User

Energy Bug Report / Energy Consumption Statistics

Applications

AppScope

AppScope Library

AppScopeViewer I/F

CPU
Display
Wi-Fi
Cell
GPS

Simple On/Off States Component X

Power Model DB (DevScope/Vendor)

AppScopeViewer

Single Core
Multi Core
LCD
OLED
USPA
LTE

Applications

AppScope

AppScope Viewer

Power Model DB (DevScope/Vendor)
AppScope Suite

• AppScopeViewer: Real-time Android power profiler
  – An Java application providing device’s power profile graphically
  – Interacts with AppScope in target device
  – Easy to use without any external measurement device

• Visit our project homepage
  – http://mobed.yonsei.ac.kr/~appscape
  – Our release includes binaries of AppScope, kernel image, and AppScopeViewer.
  – Currently, AppScope supports Google Nexus One
    • CPU, 3G, WiFi, LCD, GPS.
Demo
Conclusion

• Contributions
  – Provide energy consumption of Android application, being customized to the underlying system software and hardware components in device
  – Accurately estimates in real-time (with AppScopeViewer)
  – Implemented using module programming to improve portability

• Future work (in progress)
  – Supporting diverse hardware components:
    • OLED display, various sensors, ...
  – Supporting multi-core processor architecture
  – GPU power modeling
Thank You

http://mobed.yonsei.ac.kr/~appscope