Fine Grained Power Modeling For Smartphones Using System Call Tracing

Abhinav Pathak
Y. Charlie Hu

Ming Zhang
Paramvir Bahl
Yi-Min Wang

PURDUE UNIVERSITY
Microsoft Research
Smartphone is Energy Constrained

• Energy: One of the most critical issues in smartphones
  – Limited battery lifetime

• Battery energy density only doubled in last 15 yrs

• Smartphone capability has increased drastically
  – Multiple Components: GPS, 3G, retina display, ....
Towards Understanding Energy Drain

- **Key Question**: Where is energy being spent?
  - Which component/process/thread/function(?)

- **Approach 1: Use Power Meter**
  - Buy an expensive equipment ($770)
  - Problems:
    - Only reports entire device energy consumption

- **Approach 2: Develop Online Power Models**
Generic Power Modeling

Actual Power Consumption

Power meter

Training Phase

Model

Prediction Phase

Triggers

Predicted Power Consumption
Smartphone Power Modeling
State-of-Art: Utilization Based (1/2)

Actual Power Consumption

Power meter

Training Phase

Model

Prediction Phase

Predicted Power Consumption

Triggers Utilization

Triggers Utilization

Model = \((\text{Util}_{\text{Net}}) \times E_{\text{Net}} + (\text{Util}_{\text{CPU}}) \times E_{\text{CPU}} + (\text{Util}_{\text{Disk}}) \times E_{\text{Disk}}\)
Smartphone Power Modeling
State-of-Art: Utilization Based (2/2)

Fundamental (yet intuitive) assumption

(Only active) Utilization => power consumption

Second assumption

Energy scales linearly with amount of work

Third assumption

Components power consumption add linearly

Desired Feature

Which process/thread/function? Hard to correlate
(Only active) Utilization => Power Consumption

File open/delete/close/create change power state

Several components have tail states (3G, disk, wifi, gps)
Energy scales linearly with amount of work

(1) Send packets @ < 50pkts/s

(2) Send packets @ > 50pkts/s

WM6.5 on Tytn II
Components power consumption add linearly

WM6.5 on HTC Touch

Send start   Send done   Socket close

(1) Send(10mb); sleep(); Socket.close();

Spin_CPU(2M) (i = 1)
Spin_CPU(2M) (i = 5)
Send(2mb) (i = 5)

Spin_cpu(2M) (i = 5)

Send(2mb) (i = 1)

(2) Spin_CPU(10M);

Network tail

Socket close

Sleep(); Socket.close();
What have we learnt so far?
Simple (state-of-art) energy modeling assumptions are wrong
There exits a notion of power states

What have we hinted so far?
Device drivers have intelligent power control rules
System calls play a role in power consumption

Challenges in fine-grained power modeling?
Device drivers are closed source (no code/no information)
Key observation: System call is the interface through which an application communicates with the underlying system (hardware) and outside world (Internet, GPS, etc.)

Key Idea: Use System Calls as triggers in power modeling

Advantages:

– Encapsulates utilization based triggers
  • Parameters of system calls
– Captures power behavior of ones that do not necessarily imply utilization
– Can be traced back to process, thread, function
  • Eases energy accounting
Finite-State-Machine (FSM) as Power Model Representation

We Use Finite-State-Machine (FSM)

• **Nodes:** Power states
  – Base State: No activity on phone
  – Productive state: Actual utilization
  – Tail state: No-useful work

• **Edges:** Transition rules
  – System calls (start/completion)
  – Workload (Ex: 50 pkts/sec)
  – Timeout
FSM Power Model Construction

• Systematic ‘Brute Force’ Approach
  – Step 1: Model Single System Call
  – Step 2: Model Multiple System Calls for Same Component
  – Step 3: Model Multiple Components (Entire Phone)

• Requires domain knowledge
  – Semantics of system calls
Step 1: Single System Call FSM

System call: **read** (fd, buf, size);

**WM6.5 on HTC Touch**

Measured power consumption + system calls (trigger)

Modeled power consumption

File Read

- Base +0mA
- High disk +190mA

Disk Tail

- +110mA
- Inactivity for 5 sec
- Free

FSM
Step 2: Modeling Multiple System Calls of Same Component

• Observation: A component can only have a small finite number of power states

• Methodology
  – Identify and merge similar power states
  – Obey programming order
  – Model concurrent system calls
Step 2: WiFi NIC

WM6.5 on HTC Tytn II

SEND

Base State +0mA
Send < 50 Pkts/sec
Low Net +125 mA
Send done
High Net +325 mA
Send
Net Tail +280 mA
Send done

CLOSE

Base State +0mA
Socket close
Net Tail +280 mA
Socket close

Send > 50 Pkts/sec
Socket close
Send > 50 Pkts/sec
Socket close
Send > 50 Pkts/sec
Socket close
Step 3: Modeling Multiple Components

• Observation: Different components may interact with each other’s power consumption

• Methodology
  – Try to reach different combination of states
  – Construct new states and transitions in FSM
Implementation

• Windows Mobile 6.5
  – Extended CeLog

• Android
  – System Tap: Logs kernel events
  – Android debugging framework: Custom logging in Dalvik VM
Evaluation: Handsets Used

HTC Tytn II
Win 6.5 (CE 5.2)

HTC Touch
Win 6.5 (CE 5.2)

HTC Magic
Android (Linux 2.6.34)
Snapshot of FSM for Entire Phone

- **Base State**: +0mA
  - Send done
  - Send done
  - Send done
  - Disk: Read/write/open/close/create/delete
  - Timeout 1.7s

- **Low Net**: +125 mA
  - Send < 50 Pkts/sec
  - Send > 50 Pkts/sec
  - Call completed

- **Net Tail**: +270 mA
  - Send
  - Send done
  - DTail + CPU
  - +300 mA
  - CPU

- **High Net**: +325 mA
  - Send done

- **High CPU**: +130 mA
  - CPU (ctx_in)
  - ctx_out

- **High Disk**: +125 mA
  - Disk
  - Call completed

- **WM6.5 on HTC Tytn II**
End-To-End Energy Estimation Error

<table>
<thead>
<tr>
<th>Application</th>
<th>Error %</th>
<th>FSM: under 4%</th>
<th>LR: 1% – 20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>csort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropbear</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maps</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>facebook</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>youtube</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>game</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chess</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diskB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>netB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ie.cnn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pviewer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>docConv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>virusScan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>youtube</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uploade</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FSM: under 4%
LR: 1% – 20%
Fine-Grained Energy Estimation

CDF of energy estimation error per 50ms time interval

- FSM based on System calls
- Linear Regression (State-of-art)

FSM: 80\textsuperscript{th} percentile error less than 10\% for all apps
LR: 10\textsuperscript{th} percentile error less than 10\% for all apps
Paper Contains ...

• Detailed FSM construction
  – Handling special cases (CPU Frequency, WiFi Signal Strength)
  – FSM for 3 smartphones

• Detailed Accuracy Results
  – Why our model performs better than state-of-art

• Logging Overhead
  – Under 10% overhead on both the OSes

• Application: Energy Profiler
  – Call-Graph Energy profiler for smartphone apps
  – Generates source code heat map
Main Contributions

• **Developed** fine-grained energy modeling: Predicts fine grained energy consumption using FSM of mobile applications

• **Implemented** on Windows Mobile 6.x and Android

• **Demonstrated** improved accuracy in fine-grained energy estimation over state-of-art utilization based models