Cellular Networks and Mobile Computing
COMS 6998-8, Spring 2012

Instructor: Li Erran Li
(lierranli@cs.columbia.edu)

http://www.cs.columbia.edu/~coms6998-8/

2/20/2012: Ebugs, Power Models, Profiling and Debuging
Announcements

• To obtain physical access to Gateway Lab, contact mjg2203@columbia.edu

• Contact TA Hemin Merchant to provision your iOS devices

• Contact TA Jiawen Sun to get Amazon EC2 credits (one representative from each project team)

• Programming assignment 1 will be due on Monday, Feb 27th
Outline

• The Rise of Ebugs
• Methods of Measuring Power Usage
• Power Models
  – Usage based
  – System call trace based
• Profiling
• Conclusion
The Rise of Energy Bugs

Single Symptom: **Severe, Unexpected** Battery Drain

Apps Need Not Crash
No Blue Screen Of Death

Common Perception:
Kill some apps to fix

Courtesy: Pathak et al
User Frustration
(Dialer App EBug)

Comment 24 by mgil...@gmail.com, Aug 14, 2011
This defect is a real P.I.T.A. - I don't want to use my phone as a phone because I have to restart it every time. If I forget then it's usually 30-40% battery gone by the end of the day.

Comment 30 by hansheng...@gmail.com, Aug 15, 2011
Bring your charger with you and keep it charged!!! That's the only way the phone can last a day. It's a irritating bug!!!

Comment 239 by egork...@gmail.com, Nov 6 (6 days ago)
GOOGLE!!!!!!! DO SOMETHING WITH THIS ISSUE!!! FASTER PLEASE!!!!
Crawling Internet Forums

1. 4 Online Mobile Forums
   (≈400 mobile devices; Several mobile OSes)
   1.2M Posts
   grep “power drain”, “battery drain”, etc.
   39K Posts
   K-means
   1000 Clusters
   EBug Taxonomy
   Human

2. 2 Mobile Bugs/Issues Repository
   Indepth Analysis

2/20/12

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Pathak et al
Ebug Taxonomy

- **Hardware**: 23%
  - OS
    - No Sleep Bug
    - Apps
    - Loop Bug
    - Immortality Bug
- **Software**: 35%
  - External Service
  - Network Signal Strength
  - Wireless Handovers
- **External**: 12%
- **Unknown**: 30%

Courtesy: Pathak et al
Hardware EBug

- Hardware: 23%
- Software: 35%
- External: 12%
- Unknown: 30%

Battery

External Hardware

Sim Card

SDCard

Exterior Hardware Damage

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Pathak et al
Ebug Taxonomy

- Hardware - 23%
  - Apps
    - No Sleep Bug
      - Loop Bug
      - Immortality Bug
  - OS
    - External Service
  - Network Signal Strength
  - Wireless Handovers

- Software - 35%
  - Ebug
    - External - 12%
    - Unknown - 30%
OS Ebugs

Why does OS Leak Energy?

- Hard to infer
- OS Processes
- System Configuration

Courtesy: Pathak et al

2/20/12

Cellular Networks and Mobile Computing (COMS 6998-8)
Apps EBug: No Sleep Bug

• **Aggressive Sleeping Policies:** Smartphone OSes freeze system after brief inactivity

• **Power encumbered Programming:** Programmer has to manage sleep/wake cycle of components

• **No Sleep Bug:** At least one component is kept awake due to mismanagement
External Conditions

- External Services (<1%)
- Network Signal Strength (11%)
- Wireless Handovers (<1%)

Courtesy: Pathak et al
EDB: Energy Debugging Framework

Narrowing Down to One Entity

- Hardware Tests
- History Based Diagnosis
- Network Test

Source Test
Component Test
External Hardware

OS
App

Network Related?

Pinpointing the Root Cause

- eoprof
- eoprof

Instrumented OS
Configuration
Power Measurement

Instrumented OS
Instrumented App

Bug Detection And Patching

- Source code
- AutoTools/Manual
- Profiler Logs

Bug Fix Patch
Mobile Programming EcoSystem: The EBug Blame Game

Network Operators

App Developers

Framework Developers

Kernel Developers

Hardware Manufacturers

Firmware/OEM Developers

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Pathak et al
Outline

• The Rise of Ebugs

• Methods of Measuring Power Usage

• Power Models
  – Usage based
  – System call trace based

• Profiling

• Conclusion
Measuring Power Usage

• Approach 1: Use power meter (offline)
  – Buy an expensive equipment ($770)
  – Problems:
    • Only reports entire device energy consumption

• Approach 2: Use built-in battery sensor (online)
iOS Battery API

- Use **UIDevice** class to obtain information and notifications about
  - charging state (property `batteryState`)
  - charging level (property `batteryLevel`)

```objective-c
1. [[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];
2. NSArray *batteryStatus = [[NSArray arrayWithObjects:
    @"Battery status is unknown.",
    @"Battery is in use (discharging).",
    @"Battery is charging.",
    @"Battery is fully charged.", nil];
3. if ([[UIDevice currentDevice] batteryState] == UIDeviceBatteryStateUnknown)
   NSLog(@"%@", [batteryStatus objectAtIndex:0]);
4. else{
    NSString *msg = [[NSString stringWithFormat:
        @"Battery charge level: %0.2f\n",
        [[UIDevice currentDevice] batteryLevel] * 100,
        [batteryStatus objectAtIndex:[[UIDevice currentDevice] batteryState]] ];
5. NSLog(@"%@", msg);
6. }
```
Android Battery API

- Sample updates stored in files:
  - Current: /sys/class/power_supply/battery/batt_chg_current
  - Voltage: /sys/class/power_supply/battery/batt_vol
  - Capacity: /sys/class/power_supply/battery/capacity

```java
1. File fcur = new File("/sys/class/power_supply/battery/batt_chg_current");
2. if (fcur.exists())
3.    ...
```

- File names are vendor dependent
- Access using Android Debug Bridge (adb)
  - <sdk>platform-tools
  - Command: adb shell
Outline

• The Rise of Ebugs
• Methods of Measuring Power Usage
  • Power Models
    – Usage based
    – System call trace based
• Profiling
• Conclusion
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(?)
Generic Power Modeling

Actual Power Consumption

Power meter

Training Phase

Model

Prediction Phase

Predicted Power Consumption

Triggers

Triggers

Courtesy: Pathak et al
Smartphone Power Modeling: Utilization Based (1/3)

Actual Power Consumption

Power meter

Training Phase

Model

Prediction Phase

Triggers Utilization

Triggers Utilization

Linear Regression (LR) and Superimposition

Model = $\left(\text{Util}_{\text{Net}}\right) \times E_{\text{Net}} + \left(\text{Util}_{\text{CPU}}\right) \times E_{\text{CPU}} + \left(\text{Util}_{\text{Disk}}\right) \times E_{\text{Disk}}$
Smartphone Power Modeling: Utilization Based (2/3)

• PowerTutor model

\[
(\beta_{uh} \times freq_h + \beta_{ul} \times freq_l) \times util + \beta_{CPU} \times CPU_{on} + \beta_{br} \times brightness \\
+ \beta_{Gon} \times GPS_{on} + \beta_{Gsl} \times GPS_{sl} + \beta_{Wi-Fi}_{l} \times Wi-Fi_l \\
+ \beta_{Wi-Fi}_{h} \times Wi-Fi_h + \beta_{3G\_idle} \times 3G_{idle} + \beta_{3G\_FACH} \times 3G_{FACH} \\
+ \beta_{3G\_DCH} \times 3G_{DCH}
\]

\(\beta\) : power coefficient.
util, brightness and etc.: system variables.

• Sesame paper has two optimizations: model molding, principle component analysis (PCA)
Smartphone Power Modeling: Utilization Based (3/3)

Model = (Util_{Net}) \times E_{Net} + (Util_{CPU}) \times E_{CPU} + (Util_{Disk}) \times E_{Disk}

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)

Cellular Networks and Mobile Computing (COMS 6998-8)
Energy scales linearly with amount of work

(1) Send packets @ < 50pkts/s
+100-125mA

(2) Send packets @ > 50pkts/s
+325mA

WM6.5 on Tytn II

Courtesy: Pathak et al

2/20/12

Cellular Networks and Mobile Computing (COMS 6998-8)
Components power consumption add linearly

WM6.5 on HTC Touch

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

Spin_CPU(2M) (i = 1)

Send(2mb) (i = 5)

Spin_cpu(2M) (i = 5)

(3) for (i in 1 to 5){
    Send(2mb);
    Spin_CPU(2M);
}

Sleep();
Socket.close();

Network tail

Socket close

(2) Spin_CPU(10M);
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)
System Calls As Power Triggers

Key observation: System call is the interface through which an application communicates with the underlying system (hardware) and outside world (Internet, GPS, etc.)

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

Key Idea: Use System Calls as triggers in power modeling
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

Courtesy: Pathak et al
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

WM6.5 on HTC Touch

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

Measured power consumption + system calls (trigger)

Modeled power consumption

Current (mA)

Time (sec)

File Read

Base +0 mA

High disk +190 mA

Disk Tail +110 mA

Free

Inactivity for 5 sec

FSM

Courtesy: Pathak et al
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**

**SEND**

- **Base State** +0mA
  - Send done
  - Send > 50 Pkts/sec

- **Low Net** +125 mA
  - Send done
  - Send < 50 Pkts/sec

- **High Net** +325 mA
  - Send done
  - Send > 50 Pkts/sec

**CLOSE**

- **Base State** +0mA
  - Socket close

- **Net Tail** +280 mA
  - Socket close

- **Base State** +0mA
  - Send done
  - Send > 50 Pkts/sec

- **Low Net** +125 mA
  - Send done
  - Send < 50 Pkts/sec

- **High Net** +325 mA
  - Send done
  - Send > 50 Pkts/sec

- **Net Tail** +280 mA
  - Socket close

---

WM6.5 on HTC Tytn II

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Pathak et al

2/20/12
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)

Courtesy: Pathak et al
Snapshot of FSM for Entire Phone

**Base State**
- High CPU +130 mA
- Low Net +125 mA
- Disk: Read/write/open/close/create/delete

**Low Net**
- Send < 50 Pkts/sec
- Send done

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

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

**Disk Tail**
- Disk Tail +75 mA
- Timeout 3s

**Net Tail + CPU**
- CPU +300 mA
- Send
- Send done

**CPU (ctx_in)**
- ctx_out

**Timeout**
- 1.7s
- 3s

**WM6.5 on HTC Tytn II**

Courtesy: Pathak et al
End-To-End Energy Estimation Error

FSM: under 4%
LR: 1% – 20%

Courtesy: Pathak et al
Fine-Grained Energy Estimation

CDF of energy estimation error per 50ms time interval

FSM: 80th percentile error less than 10% for all apps
LR: 10th percentile error less than 10% for all apps

FSM based on System calls

Linear Regression (State-of-art)

Courtesy: Pathak et al
Outline

• The Rise of Ebugs
• Methods of Measuring Power Usage
• Power Models
  – Usage based
  – System call trace based
• Profiling
• Conclusion
Energy Profiling

• eprof published in Eurosys 2012
• **QCOM Trepn Profiler**
  - Trepn leverages hardware sensors built into the Snapdragon MDP
  - Analyze power consumption of hardware blocks in the Snapdragon MDP, including:
    • CPU (system and auxiliary)
    • GPS
    • Bluetooth
    • Camera
    • Audio
    • Memory
    • Network data (optimizes data transfer frequency)
• Accounting policies for asynchronous power
  – Tail power state energy consumption: attributed to last trigger
  – Concurrent accesses: divided among multiple system calls
  – Wakelocks and exotic components: attributed to the entities that acquired the wakelock
eprof Architecture

Code Instrumentation & Logging
- application source code
- compiler
- application binary
- install on mobile
- application run on mobile

Energy logging instrumentation embedded
OS/Framework enabled with system call/routine tracing

Data Presentation
- Energy split per entity during current run
- Bundle information
  - IO energy optimization

Energy Accounting
- eprof energy accounting
  - routine trace
  - system call trace
  - trace

2/20/12 Cellular Networks and Mobile Computing (COMS 6998-8)
eprof Implementation

• SDK routine tracing: extend Android routing profiling framework
• NDK routine tracing: use gprof port of NDK
  – \texttt{http://code.google.com/p/android-ndk-profiler/}
• System call tracing: insert ADB logging APIs in framework code and log CPU (sched.switch) scheduling event in kernel using systemtap
  – \texttt{http://www.cyanogenmod.com/}
### eprof Evaluation

- **Most energy spent on I/O**

<table>
<thead>
<tr>
<th>App</th>
<th>Runtime</th>
<th>#Routine calls (#Threads)</th>
<th>% Battery</th>
<th>3rd-Party Modules Used</th>
<th>Where is the energy spent inside an app?</th>
</tr>
</thead>
<tbody>
<tr>
<td>browser</td>
<td>30s</td>
<td>1M (34)</td>
<td>0.35%</td>
<td>-</td>
<td>38% HTTP; 5% GUI; 16% user tracking; 25% TCP cond.</td>
</tr>
<tr>
<td>angrybirds</td>
<td>28s</td>
<td>200K (47)</td>
<td>0.37%</td>
<td>Flurry[7], Khronos[41]</td>
<td>20% game rendering; 45% user tracking; 28% TCP cond.</td>
</tr>
<tr>
<td>fchess</td>
<td>33s</td>
<td>742K (37)</td>
<td>0.60%</td>
<td>AdWhirl[42]</td>
<td>50% advertisement; 20% GUI; 20% AI; 2% screen touch</td>
</tr>
<tr>
<td>nytimes</td>
<td>41s</td>
<td>7.4M (29)</td>
<td>0.75%</td>
<td>Flurry[7], JSON[43]</td>
<td>65% database building; 15% user tracking; 18% TCP cond.</td>
</tr>
<tr>
<td>mapquest</td>
<td>29s</td>
<td>6M (43)</td>
<td>0.60%</td>
<td>SHW[44], AOL, JSON[43]</td>
<td>28% map tracking; 20% map download; 27% rendering</td>
</tr>
</tbody>
</table>
Performance Optimization

- Energy bundle: continuous period of an I/O component actively consuming power

<table>
<thead>
<tr>
<th>App</th>
<th>Total I/O Energy</th>
<th>Bundles</th>
<th>#I/O Routines /total routines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Handset:tytn2 running WM6.5</td>
<td></td>
</tr>
<tr>
<td>pslide</td>
<td>92%</td>
<td>3 (3 Disk)</td>
<td>2/21</td>
</tr>
<tr>
<td>pup</td>
<td>57%</td>
<td>3 (3 NET)</td>
<td>3/32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Handset:magic running Android</td>
<td></td>
</tr>
<tr>
<td>syncdroid</td>
<td>50%</td>
<td>4 (1 NET, 3 DISK)</td>
<td>8/0.9K</td>
</tr>
<tr>
<td>streamer</td>
<td>31%</td>
<td>3 (3 NET)</td>
<td>4/1.1K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Handset:passion running Android</td>
<td></td>
</tr>
<tr>
<td>browser</td>
<td>69%</td>
<td>3 (2 Net, 1 GPS)</td>
<td>5/3.4K</td>
</tr>
<tr>
<td>angrybirds</td>
<td>80%</td>
<td>4 (3 NET, 1 GPS)</td>
<td>5/2.2K</td>
</tr>
<tr>
<td>fchess</td>
<td>75%</td>
<td>2 (2 NET)</td>
<td>7/3.7K</td>
</tr>
<tr>
<td>nytimes</td>
<td>67%</td>
<td>2 (1 NET, 1 GPS)</td>
<td>16/6.8K</td>
</tr>
<tr>
<td>mapquest</td>
<td>72%</td>
<td>3 (2 NET, 1 GPS)</td>
<td>14/7.1K</td>
</tr>
<tr>
<td>pup</td>
<td>70%</td>
<td>1 (1 NET)</td>
<td>3/1.1K</td>
</tr>
</tbody>
</table>
Performance Optimization (Cont’d)

• Energy bundle: continuous period of an I/O component actively consuming power

```
photo.send()
socket.connect()
```

```
photo.send()
socket.write()
```

```
photo.readPhoto();
photo.ComputeHash()
```

```
UploadButton.Click()
```

```
Net Start
```

```
3G rampup
+120mA
```

```
3G Tail
+150mA
```

```
Net High
+265mA
```

```
Base
+0mA
```

```
Net Send
```

```
Done
```

```
3
```

```
Timeout 2.5s
```

```
Timeout 6s
```

```
2/20/12
```

Cellular Networks and Mobile Computing
(COMS 6998-8)
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
Conclusion and Future work

• Ebugs need to be dealt with
• Fine-grained energy modeling and profiling very important to pinpoint energy bottleneck and ebugs
  — Accounting is tricky
  — I/O energy consumption is a major part
• Display energy modeling and profiling is still lacking
Online Resources

• ded: decompiling android application tool
  – [http://siis.cse.psu.edu/ded/](http://siis.cse.psu.edu/ded/)
Questions?