GreenDroid: Diagnosis of Energy Inefficiency in Android Applications

S.C. Cheung
in collaboration with
Yepang Liu
Chang Xu

The Hong Kong University of Science and Technology
State Key Lab for Novel Software, Nanjing University


Energy Problem

- Full Network access
- Frequent sensor usage
- 3D rendering

Energy Problem

- Full Network access
- Frequent sensor usage
- 3D rendering

Annual density improvement very slow (6%)
Energy Problem

- Problem magnitude
  - Thousands of apps are NOT energy efficient
  - Millions of users affected and complained
  - Phone batteries drained in a few hours
    (Pathak et al. Hotnets 2011)

- Major reasons
  - Hardware management burden (e.g., sensors)
  - Lack of dedicated QA, short time to market
  - Difficulty in problem diagnosis

Motivation

- What are the common causes of energy problems?
- Can we distill patterns to enable automated diagnosis?

Our Work

Investigation
- Diagnosis difficulty
- Common causes

Automated diagnosis
- State exploration
- Energy efficiency analysis

Evaluation
- Effectiveness
- Efficiency

Investigated Subjects

174 popular open-source Android apps (randomly selected)
Investigated Subjects

- 33 apps with bug reports on energy problems (problems in 24 apps have been fixed)

Observations

- **Diagnosis difficulty**
  - Reproduce problem (extensive testing, energy profiling)
  - Figure out root cause (instrumentation, runtime logging)

- **Problem causes**
  - Common causes (10/24): improper use of sensors

Patterns

- Missing sensor deactivation (All executions)
Patterns

• Missing sensor deactivation

Registration ➔ U ➔ R ➔ ⋯ ➔ U ➔ R ➔ (All executions)

“Always un-register sensor listeners timely!” – Android documentation

• Sensory data underutilization

Location data well utilized ➔ U ➔ ⋯ ➔ R ➔ (Execution 1)

Location data well utilized for useful outcomes ➔ U ➔ ⋯ ➔ R ➔ (Execution 1)

Poor utilization ➔ U ➔ ⋯ ➔ R ➔ (Execution 2)

Update notification bar only? (GeoHashDroid issue 24)
Render invisible maps? (Osmdroid issue 53)

“GeoHashDroid should slow down sensor update significantly if nothing besides the notification bar is listening.”
(NumHashDroid Issue 24)

“GPS sensor should be timely disabled if location data are used to update an invisible map.”
(Osmdroid Issue 53)
Approach Overview (GreenDroid)

- Dynamic analysis (on top of Java PathFinder)
- Goal: Simple, scalable, and effective

Input

Application Under Analysis

*.class

*.xml

Output

Runtime Controller

Sensory Data Utilization Analyzer

Java PathFinder

Analysis Report

• Application state
• Energy inefficiency

1. Event generation
2. State exploration
Approach Overview (GreenDroid)

• Dynamic analysis (on top of Java PathFinder)
• Goal: Simple, scalable, and effective

1. Sensory data tracking
2. Utilization analysis
3. Sensor activation/deactivation monitoring

App Execution in JPF (Problems)

• Absence of explicit control flow (event-driven)
• Heavy reliance on native system libs (platform specific)
• Essentially interactive (valid user input generation)
App Execution in JPF (Problems)

- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)

App Execution in JPF (Problems)

- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)

App Execution in JPF (Solutions)

- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)

Android Specs

Handler scheduling policies

Temporal rules (AEM Model)

Concretization

Input: (1) app execution history (2) Newly received event

Output: next handler to execute

Decision procedure
**App Execution in JPF (Solutions)**

- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)

```java
package x.y.z;

class MyClass {
    native String foo (int i, String s);
}
```

Identify native methods

Native method

```
public native String foo (int i, String s);
```

Create stubs (native peers)

Native peer

```
public String foo (int i, String s) {
    return "stub result";
}
```

Implement logics

```java
package x.y.z;

class MyClass {
    // stub for foo method
    public String foo (int i, String s) {
        return "stub result";
    }
}
```
App Execution in JPF (Solutions)

- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)

GUI Layout Analysis (Static)

Activity 1

Association

GUI Model

Activity n

Association

GUI Model

App Execution in JPF (Solutions)

- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)

GUI Layout Analysis (Static)

Application configurations

GUI

Model

GUI

Model

Event sequence generation (Dynamic)

JPF

Call stack

(onResume())

VM events

Listener

Waiting for user interaction?

Click

Long press

Sequence length bounded!

State Exploration

- State changes as the app continuously handles events (user events, system events etc.)

Event State

(Execution 1)

(Execution 2)

(Execution k)

How to analyze sensory data utilization?

Are they well utilized?

(Execution 1)

(Execution 2)

(Execution k)
### Taint Propagation Policy

<table>
<thead>
<tr>
<th>Index</th>
<th>Bytecode Instruction</th>
<th>Taint Propagation Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Const-op C</td>
<td>( T(\text{stack}[0]) = \emptyset )</td>
</tr>
<tr>
<td>2</td>
<td>Load-op index</td>
<td>( T(\text{stack}[0]) = T(\text{localVar}_{\text{index}}) )</td>
</tr>
<tr>
<td>3</td>
<td>LoadArray-op arrayRef, index</td>
<td>( T(\text{stack}[0]) = T(\text{arrayRef}) \cup T(\text{arrayRef}[\text{index}]) )</td>
</tr>
<tr>
<td>4</td>
<td>Store-op index</td>
<td>( T(\text{localVar}_{\text{index}}) = T(\text{stack}[0]) )</td>
</tr>
<tr>
<td>5</td>
<td>StoreArray-op arrayRef, index</td>
<td>( T(\text{arrayRef}[\text{index}]) = T(\text{stack}[0]) )</td>
</tr>
<tr>
<td>6</td>
<td>Binary-op</td>
<td>( T(\text{stack}[0]) = T(\text{stack}[0]) \cup T(\text{stack}[1]) )</td>
</tr>
<tr>
<td>7</td>
<td>Unary-op</td>
<td>( T(\text{stack}[0]) = T(\text{stack}[0]) )</td>
</tr>
<tr>
<td>8</td>
<td>GetField-op index</td>
<td>( T(\text{stack}[0]) = T(\text{stack}[0].\text{instanceField}) \cup T(\text{stack}[0]) )</td>
</tr>
<tr>
<td>9</td>
<td>GetStatic-op index</td>
<td>( T(\text{stack}[0]) = T(\text{Name}.\text{staticField}) )</td>
</tr>
<tr>
<td>10</td>
<td>PutField-op index</td>
<td>( T(\text{stack}[1].\text{instanceField}) = T(\text{stack}[0]) )</td>
</tr>
<tr>
<td>11</td>
<td>PutStatic-op index</td>
<td>( T(\text{Name}.\text{staticField}) = T(\text{stack}[0]) )</td>
</tr>
<tr>
<td>12</td>
<td>Return-op (non-void)</td>
<td>( T(\text{callerStack}[0]) = T(\text{calleeStack}[0]) )</td>
</tr>
</tbody>
</table>

### Example

**Compute acceleration**

Input: `accEvent` from accelerometer

```java
float[] values = accEvent.values;
float x = values[0];
float y = values[1];
float z = values[2];
float g = GRAVITY_EARTH;
float acc = (x*x + y*y + z*z) / (g*g);
```

**Tainted Data**

- `accEvent` (Rule 8)

---

**Field access**

Input: `accEvent` from accelerometer

```java
float[] values = accEvent.values;
float x = values[0];
float y = values[1];
float z = values[2];
float g = GRAVITY_EARTH;
float acc = (x*x + y*y + z*z) / (g*g);
```
**Example**

**Compute acceleration**
Input: `accEvent` from accelerometer

```plaintext
float[] values = accEvent.values;
float x = values[0];
float y = values[1];
float z = values[2];
float g = GRAVITY_EARTH;
float acc = (x*x + y*y + z*z) / (g*g);
```

**Sensory data usage measurement**

\[
\text{usage}(s,d) = \sum_{i \in \text{Instr}(s,d)} \text{weight}(i,s) \times \text{rel}(i) \\
\text{Usage Accumulation}
\]
Sensory data usage measurement

Location data

- dataProcess()
- updateMap()

State 1

- dataProcess()
- updateMap()

State 2 (map invisible)

- dataProcess()
- updateMap()

State 3

writeToDB()

Usage Comparison

usage\_coefficient(s, d) = \frac{usage(s, d)}{Max_{s',d'\in D}(usage(s', d'))}

<table>
<thead>
<tr>
<th>Index</th>
<th>Usage</th>
<th>Utilization coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>n</td>
<td>0.5</td>
</tr>
<tr>
<td>State 2</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>State 3</td>
<td>2n</td>
<td>1</td>
</tr>
</tbody>
</table>

Report

- Event sequence
- Sensory data usage details
Usage Comparison

\[ \text{utilization coefficient}(s, d) = \frac{\text{usage}(s, d)}{\text{Max}_{s',d' \in D}(\text{usage}(s', d'))} \]

<table>
<thead>
<tr>
<th>Index</th>
<th>Usage</th>
<th>Utilization coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>n</td>
<td>0.5</td>
</tr>
<tr>
<td>State 2</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>State 3</td>
<td>2n</td>
<td>1</td>
</tr>
</tbody>
</table>

Evaluation

- **RQ1 (Effectiveness):** Can GreenDroid effectively detect energy problems?
- **RQ2 (Efficiency):** How much overhead does GreenDroid incur? Is GreenDroid practical enough to handle real-world large subjects?

Subjects

<table>
<thead>
<tr>
<th>Application</th>
<th>Basic Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Osmdroid</td>
<td>Revision No. 750, Lines of code 18,091, Downloads 10K—50K, Availability Google Play</td>
</tr>
<tr>
<td>Zmanim</td>
<td>Revision No. 322, Lines of code 4,893, Downloads 10K—50K, Availability Google Play</td>
</tr>
<tr>
<td>Omnidroid</td>
<td>Revision No. 863, Lines of code 12,427, Downloads 1K—5K, Availability Google Play</td>
</tr>
<tr>
<td>DroidAR</td>
<td>Revision No. 204, Lines of code 18,106, Downloads 1K—5K, Availability Google Code</td>
</tr>
<tr>
<td>Recycle-locator</td>
<td>Revision No. 68, Lines of code 3,241, Downloads 1K—5K, Availability Google Play</td>
</tr>
<tr>
<td>GPSLogger</td>
<td>Revision No. 15, Lines of code 659, Downloads 1K—5K, Availability Google Code</td>
</tr>
<tr>
<td>Ushahidi</td>
<td>Revision No. 9d0a75, Lines of code 10,186, Downloads 5K—10K, Availability Google Code</td>
</tr>
<tr>
<td>Sofia Public Transport Nav.</td>
<td>Revision No. 114, Lines of code 1,443, Downloads 10K—50K, Availability Google Play</td>
</tr>
<tr>
<td>GeohashDroid</td>
<td>Revision No. V0.8.1-pre2, Lines of code 6,682, Downloads 10K—50K, Availability Google Play</td>
</tr>
</tbody>
</table>

Effectiveness

<table>
<thead>
<tr>
<th>Energy Problem</th>
<th>Problem type</th>
<th>New problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Osmdroid issue 53</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>Zmanim issue 50/56</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>Sofia Public Transport Nav. issue 38</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>Geohash Droid issue 24</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>DroidAR issue 27</td>
<td>Missing sensor deactivation</td>
<td>No</td>
</tr>
<tr>
<td>Recycle-locator issue 33</td>
<td>Missing sensor deactivation</td>
<td>No</td>
</tr>
<tr>
<td>Ushahidi issue 11</td>
<td>Missing sensor deactivation</td>
<td>No</td>
</tr>
<tr>
<td>Omnidroid issue 179</td>
<td>Sensory data underutilization</td>
<td>Yes</td>
</tr>
<tr>
<td>GPSLogger issue 7</td>
<td>Sensory data underutilization</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**GreenDroid found nine real problems. Six are caused by poor sensory data utilization. Three are caused by missing sensor deactivation.**
First seven problems were confirmed before our experiments. The last two were new problems found by GreenDroid (both confirmed).

“Completely true, Omnidroid does suck up way more energy than necessary. I'd be happy to accept a patch in this regard”. (Omnidroid issue 179)

Large applications of 18KLOC can be explored in a few minutes. Memory consumption is well supported by modern PCs even without optimization.
Discussion

- Patching GPSLogger
  - Invited to provide a patch
  - Built the patch by following a real one (Geohash Droid issue 24)
  - Patch accepted and released online.

- GreenDroid limitations
  - Complex inputs generation (e.g., password)
  - Dynamic GUI updates (GUI models are extracted statically)

Future Work

- More energy problem patterns
  - Initial evidence: 16% energy problems was caused by network issues
    (e.g., energy-inefficient data transmission)

- Integration to Android framework
  - Modify Dalvik VM for data utilization analysis
  - On-device detection of energy problems

Conclusion

Thank you!