Finding Sensor Related Energy Black Holes In Smartphone Applications

Yepang Liu\textsuperscript{1}, Chang Xu\textsuperscript{2}, S.C. Cheung\textsuperscript{1}

\textsuperscript{1}The Hong Kong University of Science and Technology
\textsuperscript{2}State Key Lab for Novel Software, Nanjing University

IEEE PerCom 2013
San Diego, California, USA
Smartphone apps

650,000+ applications
(September 2012)

25+ billion downloads
(September 2012)
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 Inefficiency
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

App Availability Distribution

- Google Code: 136
- GitHub: 33
- SourceForge: 10
Investigated Subjects

174 popular open-source Android apps

App Availability Distribution

Source Repository
(24 affected apps)

- ✓ Bug reports
- ✓ Comments on bug reports
- ✓ Bug fixing patches
- ✓ Revision commit logs
Observations

Diagnosis difficulty

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

• **Sensor listener misusage**

Registration

(All executions)
Patterns

- **Sensor listener misusage**

```
Registration       Unregistration
○ → ○ → ○ → ○ → ○ → ○ → ○ (All executions)
```

“Always un-register sensor listener before program exits!”
Patterns

• Sensor listener misusage

Registration \[ \rightarrow \] Unregistration

○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → ○ → --
Patterns

• **Sensor listener misusage**

  Registration → \( \mathbf{R} \) → Unregistration → U → \( \bigcirc \) (All executions)

  “Always un-register sensor listener before program exits!”

• **Sensory data underutilization**

  ![Diagram of sensory data utilization](image)

  Location data well utilized
  
  Registration → \( \mathbf{R} \) → \( \bigcirc \) → \( \bigcirc \) → \( \bigcirc \) → \( \bigcirc \) → Unregistration → U → \( \bigcirc \) (Execution 1)

  Poor utilization

  Registration → \( \mathbf{R} \) → \( \bigcirc \) → \( \bigcirc \) → \( \bigcirc \) → \( \bigcirc \) → \( \bigcirc \) → Unregistration → U → \( \bigcirc \) (Execution 2)
Sensory Data Underutilization

Location data well utilized

Execution 1

R U (Execution 1)

Location data poorly utilized

Execution 2

R U (Execution 2)

"GeoHashDroid should slow down sensor update significantly if nothing besides the notification bar is listening."

(GeoHashDroid 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
Approach Overview (GreenDroid)

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

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

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

1. Event generation
2. State exploration

- Application state
- Energy inefficiency
Approach Overview (GreenDroid)

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

1. Sensory data tracking
2. Utilization analysis
3. Sensor listener usage monitoring

Application Under Analysis

Runtime Controller

Sensory Data Utilization Analyzer

Java PathFinder

Analysis Report

- Application state
- Energy inefficiency
Approach Overview (GreenDroid)

- Dynamic analysis (on top of Java PathFinder)
- Goal: Simple, scalable, and effective
- **Major Challenges**
  - App execution and state exploration in Java PathFinder
  - Sensory data identification and utilization analysis (no metrics)

![Diagram of analysis process]

- Application state
- Energy inefficiency
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)

General Java programs
(explicit control flow)
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)

General Java programs (explicit control flow)

Android programs (event-driven)

Loosely coupled handlers
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)
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)

```
package x.y.z;
class MyClass {
   ...
   native String foo (int i, String s);
}

"java.gov.nasa.jpf.GenPeer x.y.z.MyClass > JPF_x_y_z_MyClass.java"
```

```
class JPF_x_y_z_MyClass {
   ...
   public static
      int foo__ILjava_lang_String__2 (MJEnv env, int objRef,
         int i, int sRef) {
         int ref = MJEnv.NULL;
         // <2do> fill in body
         return ref;
      }
}
```
Identify native methods

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

```java
cpyackage x.y.z;
class MyClass {
    ...
    native String foo (int i, String s);
}
```

"java gov.nasa.jpfl.GenPeer x.y.z.MyClass > JPF_x_y_z_MyClass.java"

class JPF_x_y_z_MyClass {
    ...
    public static
        int foo__ILjava_lang_String__2 (MJIE env, int objRef,
            int i, int sRef) {
            int ref = MJIE.NULL;
            // <2do> fill in body
            return ref;
        }
    }
```
Identify native methods

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

```
package x.y.z;

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

"java gov.nasa.jpl.GenPeer x.y.z.MyClass > JPF_x.y_z_MyClass.java"

Native peer

```
class JPF_x.y_z_MyClass {
    public static
    int foo__ILjava_lang_String__2 (MJIEnv env, int objRef,
        int i, int sRef) {
        int ref = MJIEnv.NULL;
        // <2do> fill in body
        return ref;
    }
}
```
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)

Identify native methods

Create stubs (native peers)

Implement logics

```
package x.y.z;
class MyClass {
    
    native String foo (int i, String s);
}

"java.gov.nasa.jpl.GenPeer x.y.z.MyClass > JPF_x.y.z.MyClass.java"

class JPF_x.y.z.MyClass {
    ...  
    public static 
        int foo_ILjava_lang_String__2 (MJIEnv env, int objRef, int i, int sRef) {
            int ref = MJIEnv.NULL;
            // <2do> fill in body
            return ref;
        }
    }
```
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)
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)

Event sequence generation (Dynamic)

Wait for user interaction?
- Click
- Long press

Sequence length bounded!
State Exploration

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

• State changes as the app continuous handles events (user events, system events etc.)

How to analyze sensory data utilization? Are they well utilized?
Sensory Data Tracking & Identification

Data Lifecycle:

- Data enter app
- Transformed and consumed
- Data leave app
**Sensory Data Tracking & Identification**

**Data Lifecycle:**
- Data enter app
- Transformed and consumed
- Data leave app

**Sensory Data tracking process:**
- Taint source
  - Taint the mock sensory data
Sensory Data Tracking & Identification

Data Lifecycle:
- Data enter app
- Transformed and consumed
- Data leave app

Sensory Data tracking process:
- Taint source
  - Taint the mock sensory data
- Taint propagation
  - Tracking and pinpointing sensory data
  - Record sensory data usages
Sensory Data Tracking & Identification

Data Lifecycle:

Data enter app

Transformed and consumed

Data leave app

Sensory Data tracking process:

Taint source

- Taint the mock sensory data

Taint propagation

- Tracking and pinpointing sensory data
- Record sensory data usages

Taint sink

- Terminate tainting
- Compute sensory data utilization
# 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><strong>Const-op</strong> C</td>
<td>$T(stack[0]) = \emptyset$</td>
</tr>
<tr>
<td>2</td>
<td><strong>Load-op</strong> index</td>
<td>$T(stack[0]) = T(localVar_{index})$</td>
</tr>
<tr>
<td>3</td>
<td><strong>LoadArray-op</strong> arrayRef, index</td>
<td>$T(stack[0]) = T(arrayRef) \cup T(arrayRef[\text{index}])$</td>
</tr>
<tr>
<td>4</td>
<td><strong>Store-op</strong> index</td>
<td>$T(localVar_{index}) = T(stack'[0])$</td>
</tr>
<tr>
<td>5</td>
<td><strong>StoreArray-op</strong> arrayRef, index</td>
<td>$T(arrayRef[\text{index}]) = T(stack'[0])$</td>
</tr>
<tr>
<td>6</td>
<td><strong>Binary-op</strong></td>
<td>$T(stack[0]) = T(stack'[0]) \cup T(stack'[1])$</td>
</tr>
<tr>
<td>7</td>
<td><strong>Unary-op</strong></td>
<td>$T(stack[0]) = T(stack'[0])$</td>
</tr>
<tr>
<td>8</td>
<td><strong>GetField-op</strong> index</td>
<td>$T(stack[0]) = T(stack'[0].instanceField) \cup T(stack'[0])$</td>
</tr>
<tr>
<td>9</td>
<td><strong>GetStatic-op</strong> index</td>
<td>$T(stack[0]) = T(ClassName.staticField)$</td>
</tr>
<tr>
<td>10</td>
<td><strong>PutField-op</strong> index</td>
<td>$T(stack'[1].instanceField) = T(stack'[0])$</td>
</tr>
<tr>
<td>11</td>
<td><strong>PutStatic-op</strong> index</td>
<td>$T(ClassName.staticField) = T(stack'[0])$</td>
</tr>
<tr>
<td>12</td>
<td><strong>Return-op</strong>(non-void)</td>
<td>$T(callerStack[0]) = T(calleeStack'[0])$</td>
</tr>
</tbody>
</table>
**Example**

**Compute acceleration**

Input: `accEvent` from accelerometer

```c
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

Tainted Data
accEvent

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

Field access

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
values (Rule 8)
Example

Compute acceleration
Input: accEvent from accelerometer

Tainted Data

<table>
<thead>
<tr>
<th>accEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>values (Rule 8)</td>
</tr>
<tr>
<td>x (Rule 3)</td>
</tr>
<tr>
<td>y (Rule 3)</td>
</tr>
<tr>
<td>z (Rule 3)</td>
</tr>
</tbody>
</table>

```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);
```

Array element access
Example

Compute acceleration
Input: accEvent from accelerometer

Tainted Data
accEvent
values (Rule 8)
x (Rule 3)
y (Rule 3)
z (Rule 3)
acc (Rule 6)

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);

Arithmetic computation
Sensory data usage measurement

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

Usage Accumulation
Sensory data usage measurement

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

**Usual Accumulation**

Osmdroid issue 53:

State 1

\[ \text{State 1} \]

State 2

\[ \text{State 2} \]

State 3

\[ \text{State 3} \]
Sensory data usage measurement

Location data
- dataProcess()
- broadcast()
- updateMap()

State 1

Location data
- dataProcess()
- broadcast()
- updateMap()

State 2 (map invisible)

Location data
- dataProcess()
- broadcast()
- updateMap()
- writeToDB()

State 3

(Execution 1)

(Execution 2)
Sensory data usage measurement

- Location data
  - dataProcess()
  - broadcast()
  - updateMap()
  - State 1
- Location data
  - dataProcess()
  - broadcast()
  - updateMap()
  - State 2 (map invisible)
- Location data
  - dataProcess()
  - broadcast()
  - updateMap()
  - writeToDB()
  - State 3

(Execution 1)

(Execution 2)
Usage Comparison

\[ \text{utilization coefficient}(s, d) = \frac{\text{usage}(s, d)}{\max_{s', d'} (\text{usage}(s', d'))} \]

<table>
<thead>
<tr>
<th>Index</th>
<th>Usage</th>
<th>Utilization coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>3n</td>
<td>0.75</td>
</tr>
<tr>
<td>State 2</td>
<td>2n</td>
<td>0.5</td>
</tr>
<tr>
<td>State 3</td>
<td>4n</td>
<td>1</td>
</tr>
</tbody>
</table>

(Execution 1)

(Execution 2)
Usage Comparison

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Usage</th>
<th>Utilization coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>3n</td>
<td>0.75</td>
</tr>
<tr>
<td>State 2</td>
<td>2n</td>
<td>0.5</td>
</tr>
<tr>
<td>State 3</td>
<td>4n</td>
<td>1</td>
</tr>
</tbody>
</table>

Report
- Event sequence
- Sensory data usage details
Usage Comparison

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Usage</th>
<th>Utilization coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>3n</td>
<td>0.75</td>
</tr>
<tr>
<td>State 2</td>
<td>2n</td>
<td>0.5</td>
</tr>
<tr>
<td>State 3</td>
<td>4n</td>
<td>1</td>
</tr>
</tbody>
</table>

Report
- Event sequence
- Sensory data usage details

Sensor listener misusage

(Execution 1)

(Execution 2)
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>Revision No.</th>
<th>Lines of code</th>
<th>Downloads</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Osmdroid</td>
<td>750</td>
<td>18,091</td>
<td>10K—50K</td>
<td>Google Play</td>
</tr>
<tr>
<td>Zmanim</td>
<td>322</td>
<td>4,893</td>
<td>10K—50K</td>
<td>Google Play</td>
</tr>
<tr>
<td>Omnidroid</td>
<td>863</td>
<td>12,427</td>
<td>1K—5K</td>
<td>Google Play</td>
</tr>
<tr>
<td>DroidAR</td>
<td>204</td>
<td>18,106</td>
<td>1K—5K</td>
<td>Google Code</td>
</tr>
<tr>
<td>Recycle-locator</td>
<td>68</td>
<td>3,241</td>
<td>1K—5K</td>
<td>Google Play</td>
</tr>
<tr>
<td>GPSLogger</td>
<td>15</td>
<td>659</td>
<td>1K—5K</td>
<td>Google Code</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</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>Zmanim Issue 56</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>DroidAR Issue 27</td>
<td>Sensor listener misusage</td>
<td>No</td>
</tr>
<tr>
<td>Recycle-Locator Issue 33</td>
<td>Sensor listener misusage</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 **seven real problems**. Five problems are caused by poor sensory data utilization. Two problems are caused by sensor listener misusage.
## 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</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>Zmanim Issue 56</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>DroidAR Issue 27</td>
<td>Sensor listener misusage</td>
<td>No</td>
</tr>
<tr>
<td>Recycle-Locator Issue 33</td>
<td>Sensor listener misusage</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>

*First five problems were confirmed before our experiments. The Last two were new problems found by GreenDroid*
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</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>Zmanim Issue 56</td>
<td>Sensory data underutilization</td>
<td>No</td>
</tr>
<tr>
<td>DroidAR Issue 27</td>
<td>Sensor listener misusage</td>
<td>No</td>
</tr>
<tr>
<td>Recycle-Locator Issue 33</td>
<td>Sensor listener misusage</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>

“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)
### Efficiency

<table>
<thead>
<tr>
<th>Application</th>
<th>Explored states</th>
<th>Time (seconds)</th>
<th>Space (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Osmdroid</td>
<td>120,189</td>
<td>151</td>
<td>591</td>
</tr>
<tr>
<td>Zmanim</td>
<td>54,270</td>
<td>110</td>
<td>205</td>
</tr>
<tr>
<td>Omnidroid (12 KLOC)</td>
<td>52,805</td>
<td>220</td>
<td>342</td>
</tr>
<tr>
<td>DroidAR (18 KLOC)</td>
<td>91,170</td>
<td>276</td>
<td>217</td>
</tr>
<tr>
<td>Recycle-locator</td>
<td>114,709</td>
<td>43</td>
<td>153</td>
</tr>
<tr>
<td>GPSLogger</td>
<td>58,824</td>
<td>35</td>
<td>149</td>
</tr>
</tbody>
</table>

*Thousands of states explored in a few minutes. Memory Consumption well supported by modern PCs even without optimization.*
### Efficiency

<table>
<thead>
<tr>
<th>Application</th>
<th>Analysis Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Explored states</td>
</tr>
<tr>
<td><strong>Osmdroid (18 KLOC)</strong></td>
<td>120,189</td>
</tr>
<tr>
<td><strong>Zmanim</strong></td>
<td>54,270</td>
</tr>
<tr>
<td><strong>Omnidroid (12 KLOC)</strong></td>
<td>52,805</td>
</tr>
<tr>
<td><strong>DroidAR (18 KLOC)</strong></td>
<td>91,170</td>
</tr>
<tr>
<td><strong>Recycle-locator</strong></td>
<td>114,709</td>
</tr>
<tr>
<td><strong>GPSLogger</strong></td>
<td>58,824</td>
</tr>
</tbody>
</table>

*Large subjects’ analysis overhead suggests that GreenDroid is practical enough to handle real world Android applications.*
Discussion

• Limitations
  – Imprecision in AEM model and native lib modeling
  – Complex inputs generation (e.g., password)
  – Limited subjects in evaluation

• Future work
  – Validate the effectiveness with more subjects
  – Investigate energy problems caused by other reasons (e.g., network)
Conclusion

Patterns

- Sensor listener misusage
  - Registration
  - Unregistration
  - (All executions)
  - "Always unregister sensor listener before program exists!"

- Sensory data underutilization
  - Location data well utilized
  - Poor utilization:
    - Update notification every 10 seconds
    - (Cost of Android issue #53)

App execution & state exploration in JPF

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

Approach Overview

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

Evaluation

- RQ1 (Effectiveness): Can we approach effectively detect energy inefficiency problems?
- RQ2 (Efficiency): How much overhead does our approach incur?
- RQ3 (Comparison): How does our approach compare with existing resource leak detection techniques?
Thank you!