GreenDroid: Diagnosis of Energy Inefficiency in Android Applications

S.C. Cheung

in collaboration with

Yepang Liu

The Hong Kong University of Science and Technology

Chang Xu

State Key Lab for Novel Software, Nanjing University

Yepang Liu, Chang Xu, and S.C. Cheung. Where Has My Battery Gone? Finding Sensor Related Energy Black Holes in Smartphone Applications. In *Proceedings of the 11th IEEE International Conference on Pervasive Computing and Communications (PERCOM 2013)*, pp. 2-10, San Diego, California, USA, Mar 2013.

Smartphone apps

Google play

1 million applications (November 2013)

50 billion downloads (November 2013)

Energy Problem

Full Network access

Frequent sensor usage

3D rendering

Energy Problem

Full Network access

Frequent sensor usage

3D rendering

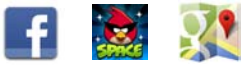
Annual density improvement very slow (6%)

Energy Inefficiency

2/29

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

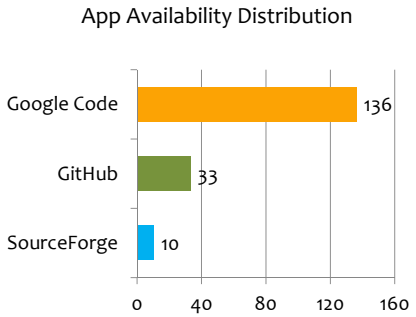
Investigation
- Diagnosis difficulty
- Common causes

Automated diagnosis
- State exploration
- Energy efficiency analysis

Evaluation
- Effectiveness
- Efficiency

Investigated Subjects

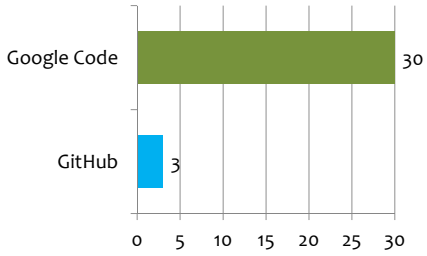
174 popular open-source Android apps (randomly selected)



Investigated Subjects

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

App Availability Distribution



Source Repository (24 affected apps)
✓ Bug reports
✓ Comments on bug reports
✓ Bug fixing patches
✓ Patch reviews
✓ Revision commit logs

Observations

Diagnosis difficulty

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

Observations

Diagnosis difficulty

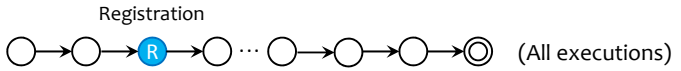
- Reproduce problem (extensive testing, energy profiling)
- Figure out root cause (instrumentation, runtime logging)

Problem causes

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

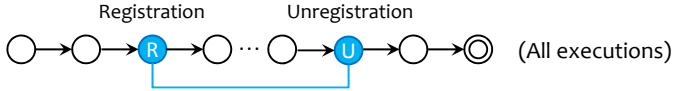
Patterns

- Missing sensor deactivation



Patterns

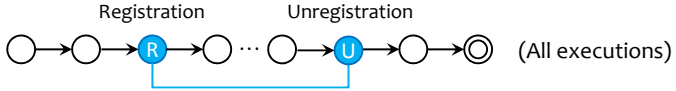
- Missing sensor deactivation



"Always un-register sensor listeners timely!" – Android documentation

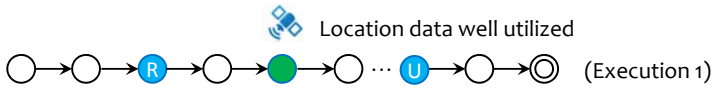
Patterns

- Missing sensor deactivation



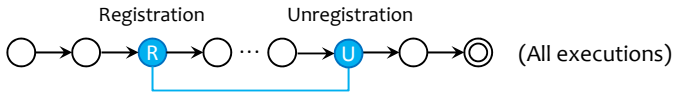
"Always un-register sensor listeners timely!" – Android documentation

- Sensory data underutilization



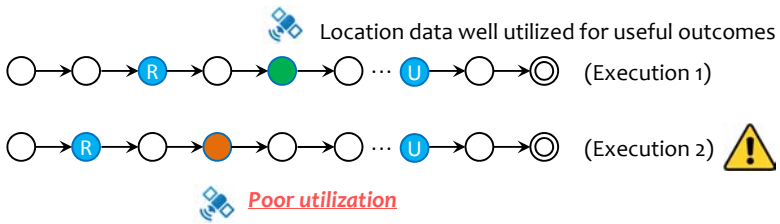
Patterns

- Missing/untimely sensor deactivation



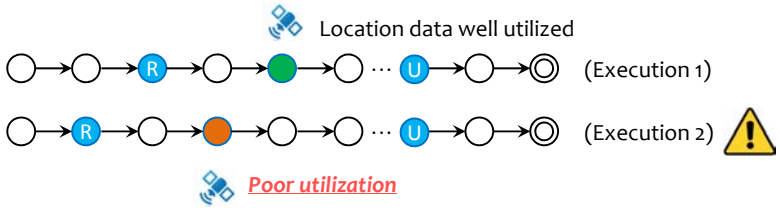
"Always un-register sensor listeners timely!" – Android documentation

- Sensory data underutilization



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

Sensory Data Underutilization (Examples)



> *"GeoHashDroid should slow down sensor update significantly if nothing besides the notification bar is listening."*
> *(GeoHashDroid Issue 24)*

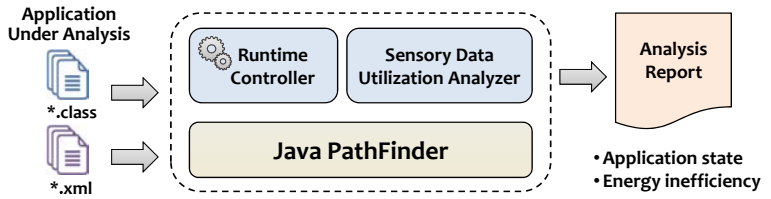
> *"GPS sensor should be timely disabled if location data are used to update an invisible map."*
> *(Osmdroid Issue 53)*

Approach Overview (GreenDroid)

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

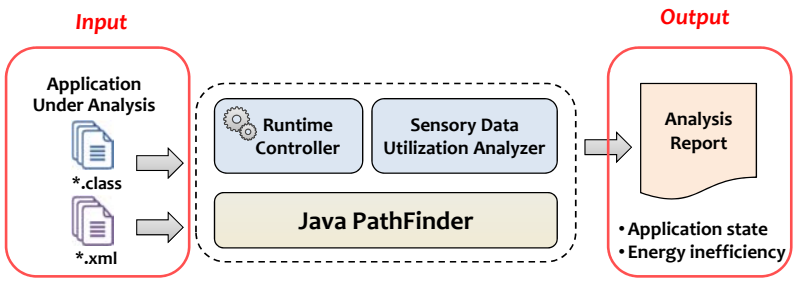
Approach Overview (GreenDroid)

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



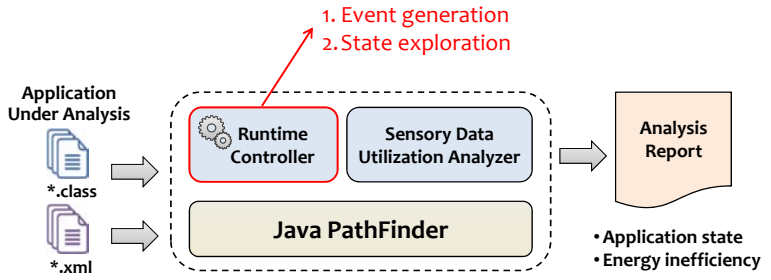
Approach Overview (GreenDroid)

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



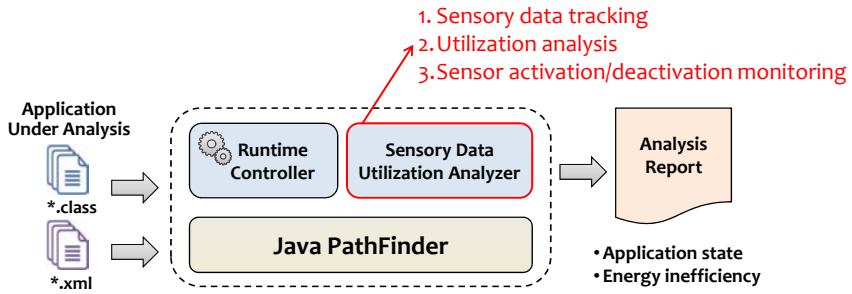
Approach Overview (GreenDroid)

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



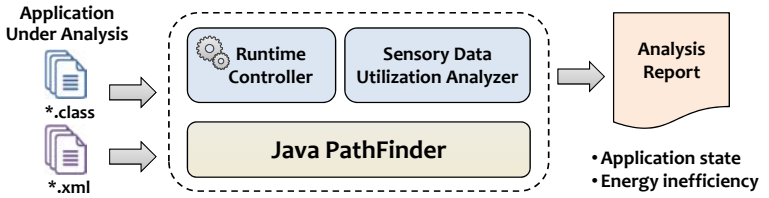
Approach Overview (GreenDroid)

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



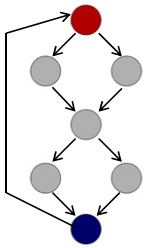
Approach Overview (GreenDroid)

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



App Execution in JPF (Problems)

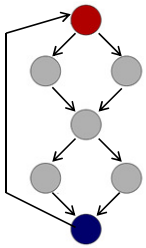
- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)



General Java programs
(explicit control flow)

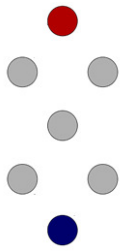
App Execution in JPF (Problems)

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

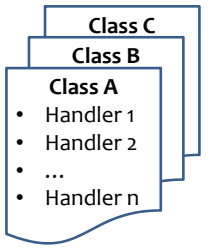


General Java programs
(explicit control flow)

VS



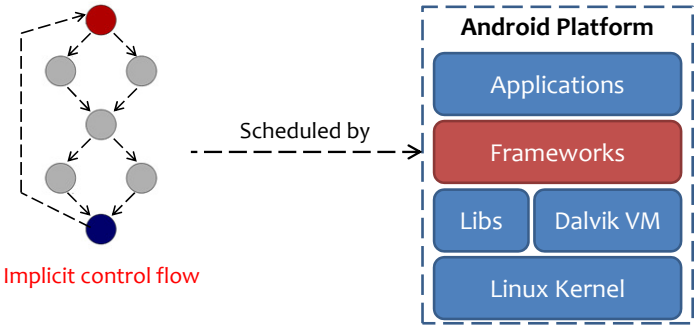
Android programs
(event-driven)



Loosely coupled handlers

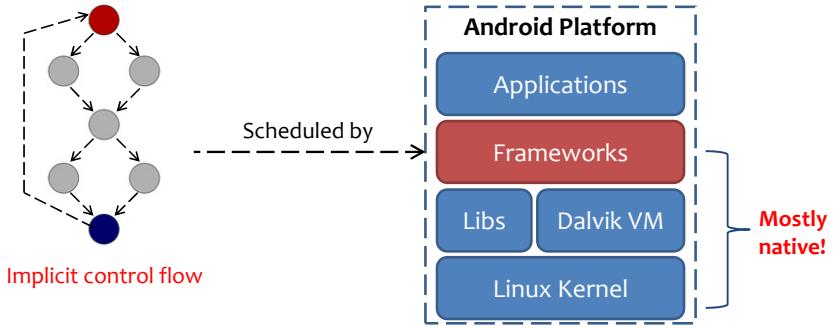
App Execution in JPF (Problems)

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



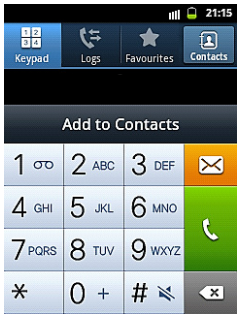
App Execution in JPF (Problems)

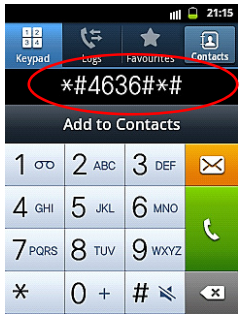
- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)



App Execution in JPF (Problems)

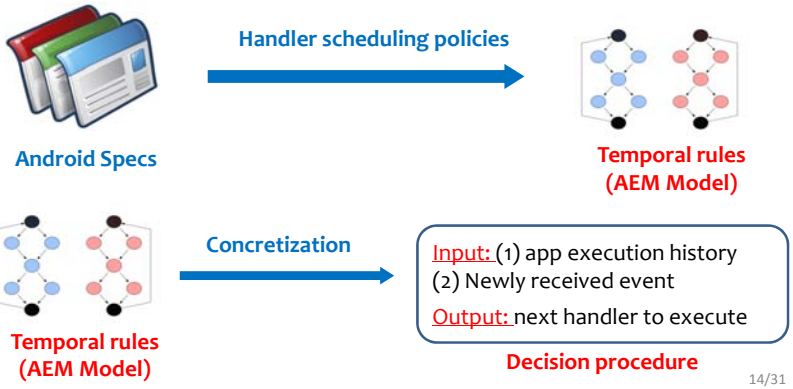
- Absence of explicit control flow (event-driven)
- Heavy reliance on native system libs (platform specific)
- Essentially interactive (valid user input generation)





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)



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"

GenPeer

```
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)

Native method

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

Identify native methods

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

GenPeer

```
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)

Native method

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

Identify native methods

Create stubs (native peers)

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

GenPeer

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)

Native method

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

Identify native methods

Create stubs (native peers)

Implement logics

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

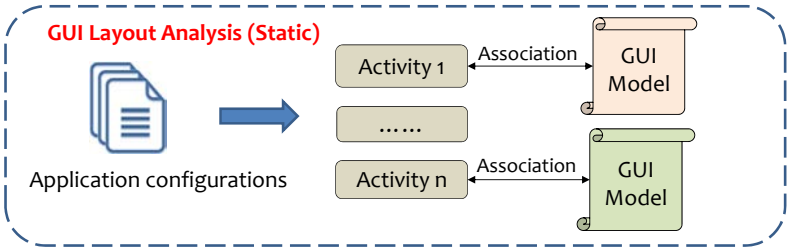
GenPeer

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;
  }
}
```

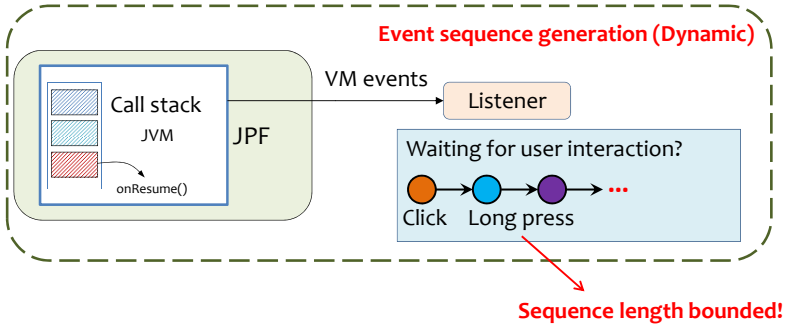
App Execution in JPF (Solutions)

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



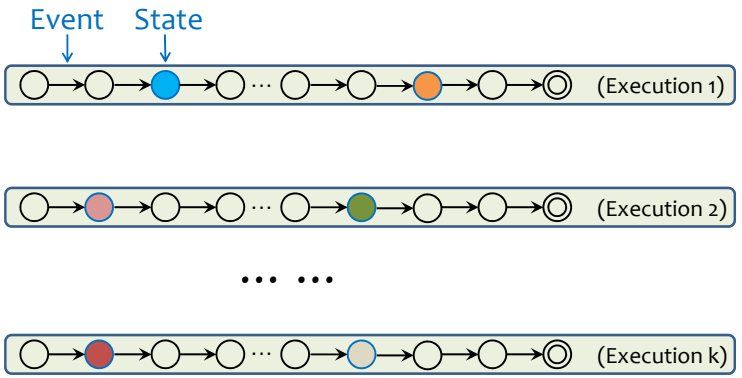
App Execution in JPF (Solutions)

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



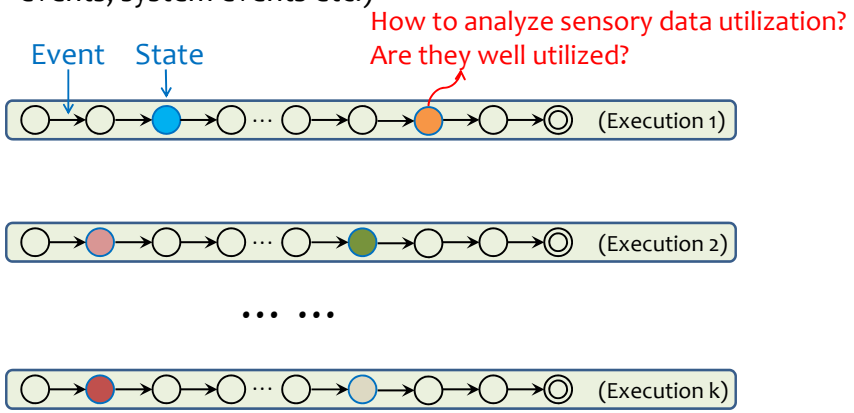
State Exploration

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



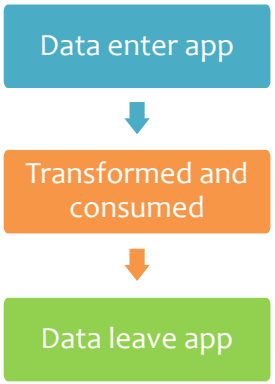
State Exploration

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



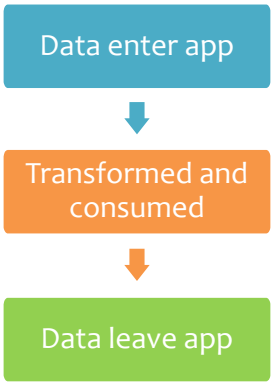
Sensory Data Tracking & Identification

Data Lifecycle:



Sensory Data Tracking & Identification

Data Lifecycle:

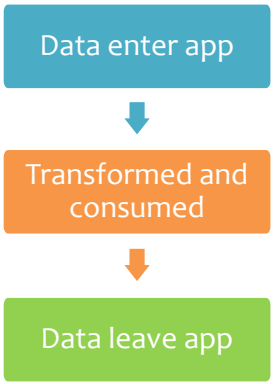


Sensory Data tracking process:

Taint source
- Taint the mock sensory data

Sensory Data Tracking & Identification

Data Lifecycle:



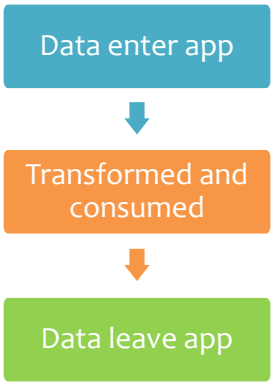
Sensory Data tracking process:

Taint source
- Taint the mock sensory data

Taint propagation
- Tracking and pinpointing sensory data
- Record sensory data usages

Sensory Data Tracking & Identification

Data Lifecycle:



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

Index	Bytecode Instruction	Taint Propagation Rule
1	***Const-op*** *C*	*T(stack[0]) = Ø*
2	***Load-op*** *index*	*T(stack[0]) = T(localVar$_{index}$)*
3	***LoadArray-op*** *arrayRef, index*	*T(stack[0]) = T(arrayRef) ∪ T(arrayRef [index])*
4	***Store-op*** *index*	*T(localVar$_{index}$) = T(stack'[0])*
5	***StoreArray-op*** *arrayRef, index*	*T(arrayRef [index]) = T(stack'[0])*
6	***Binary-op***	*T(stack[0]) = T(stack'[0]) ∪ T(stack'[1])*
7	***Unary-op***	*T(stack[0]) = T(stack'[0])*
8	***GetField-op*** *index*	*T(stack[0]) = T(stack'[0].instanceField) ∪ T(stack'[0])*
9	***GetStatic-op*** *index*	*T(stack[0]) = T(ClassName.staticField)*
10	***PutField-op*** *index*	*T(stack'[1].instanceField) = T(stack'[0])*
1	***PutStatic-op*** *index*	*T(ClassName.staticField) = T(stack'[0])*
12	***Return-op(non-void)***	*T(callerStack[0]) = T(calleeStack'[0])*

Example

Compute acceleration
Input: accEvent from accelerometer

```
float[] values = accEevent.values;
↓
float x = values[0];
float y = values[1];
float z = values[2];
↓
float g = GRAVERTIY_EARTH;
↓
float acc = (x*x + y*y + z*z) / (g*g);
```


Example

Tainted Data
accEvent

Compute acceleration
Input: accEvent from accelerometer

```
float[] values = accEevent.values;
↓
float x = values[0];
float y = values[1];
float z = values[2];
↓
float g = GRAVERTIY_EARTH;
↓
float acc = (x*x + y*y + z*z) / (g*g);
```


Example

Tainted Data
accEvent
values (Rule 8)

Compute acceleration
Input: accEvent from accelerometer

Field access

```
float[] values = accEevent.values;
↓
float x = values[0];
float y = values[1];
float z = values[2];
↓
float g = GRAVERTIY_EARTH;
↓
float acc = (x*x + y*y + z*z) / (g*g);
```

Example

Compute acceleration
Input: accEvent from accelerometer

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

```
float[] values = accEevent.values;
        ↓
float x = values[0];
float y = values[1];
float z = values[2];        Array element access
        ↓
float g = GRAVERTIY_EARTH;
        ↓
float acc = (x*x + y*y + z*z) / (g*g);
```


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 = accEevent.values;
        ↓
float x = values[0];
float y = values[1];
float z = values[2];
        ↓
float g = GRAVERTIY_EARTH;
        ↓
float acc = (x*x + y*y + z*z) / (g*g);
```

Arithmetic computation

Sensory data usage measurement

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

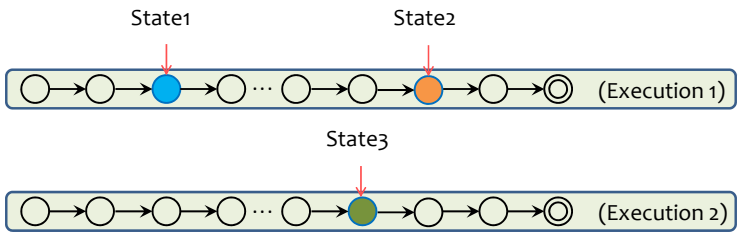
Usage Accumulation

Sensory data usage measurement

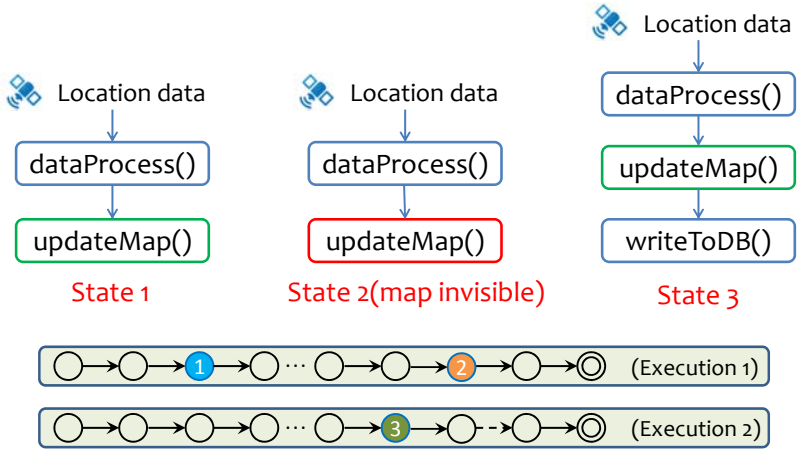
$$usage(s,d) = \sum_{i \in Instr(s,d)} weight(i,s) \times rel(i)$$

Usage Accumulation

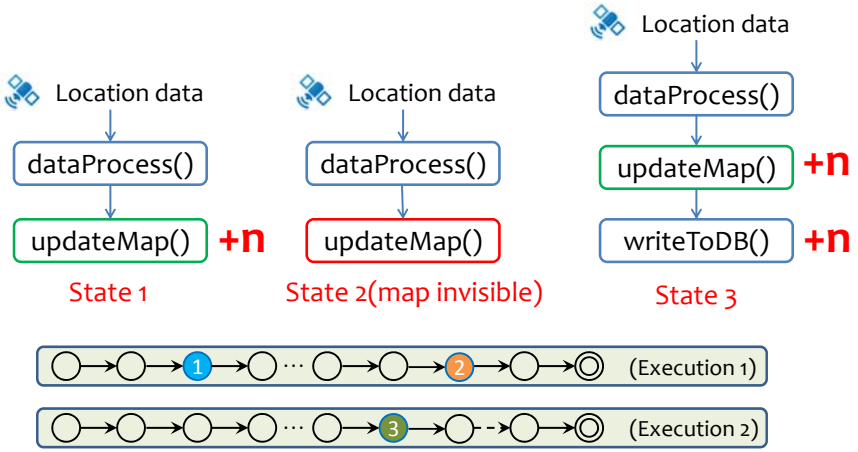
Osmdroid issue 53:



Sensory data usage measurement



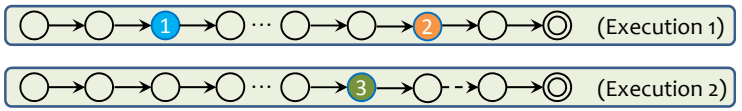
Sensory data usage measurement



Usage Comparison

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

Index	Usage	Utilization coefficient
State 1	n	0.5
State 2	**0**	**0.0**
State 3	2n	1

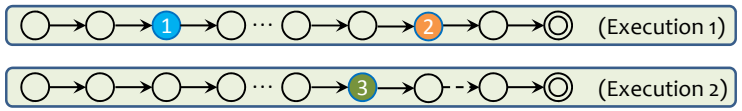


Usage Comparison

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

Index	Usage	Utilization coefficient
State 1	n	0.5
State 2	**0**	**0.0**
State 3	2n	1

Report
- Event sequence
- Sensory data usage details

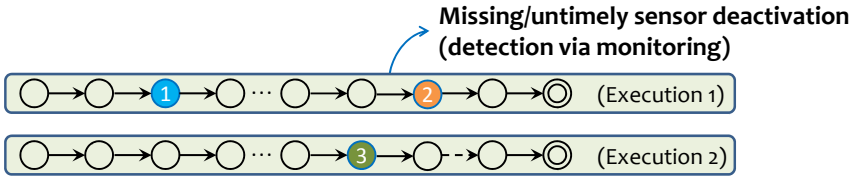


Usage Comparison

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

Index	Usage	Utilization coefficient
State 1	n	0.5
State 2	**0**	**0.0**
State 3	2n	1

Report
- Event sequence
- Sensory data usage details



Evaluation

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

Subjects

Application	Basic Information			
	Revision No.	Lines of code	Downloads	Availability
Osmdroid	750	18,091	10K—50K	Google Play
Zmanim	322	4,893	10K—50K	Google Play
Omnidroid	863	12,427	1K—5K	Google Play
DroidAR	204	18,106	1K—5K	Google Code
Recycle-locator	68	3,241	1K—5K	Google Play
GPSLogger	15	659	1K—5K	Google Code
Ushahidi	9d0aa75	10,186	5K—10K	Google Play
Sofia Public Transport Nav.	114	1,443	10K—50K	Google Play
Geohash Droid	V0.8.1-pre2	6,682	10K—50K	Google Play

Effectiveness

Energy Problem	Problem type	New problem
OsmDroid issue 53	*Sensory data underutilization*	*No*
Zmanim issue 50/56	*Sensory data underutilization*	*No*
Sofia Public Transport Nav. issue 38	*Sensory data underutilization*	*No*
Geohash Droid issue 24	*Sensory data underutilization*	*No*
DroidAR issue 27	*Missing sensor deactivation*	*No*
Recycle-Locator issue 33	*Missing sensor deactivation*	*No*
Ushahidi issue 11	*Missing sensor deactivation*	*No*
Omnidroid issue 179	*Sensory data underutilization*	*Yes*
GPSLogger issue 7	*Sensory data underutilization*	*Yes*

GreenDroid found nine real problems. Six are caused by poor sensory data utilization. Three are caused by missing sensor deactivation.

Effectiveness

Energy Problem	Problem type	New problem
OsmDroid issue 53	*Sensory data underutilization*	*No*
Zmanim issue 50/56	*Sensory data underutilization*	*No*
Sofia Public Transport Nav. issue 38	*Sensory data underutilization*	*No*
Geohash Droid issue 24	*Sensory data underutilization*	*No*
DroidAR issue 27	*Missing sensor deactivation*	*No*
Recycle-Locator issue 33	*Missing sensor deactivation*	*No*
Ushahidi issue 11	*Missing sensor deactivation*	*No*
Omnidroid issue 179	*Sensory data underutilization*	*Yes*
GPSLogger issue 7	*Sensory data underutilization*	*Yes*

First seven problems were confirmed before our experiments. The Last two were new problems found by GreenDroid (both confirmed).

Effectiveness

Energy Problem	Problem type
OsmDroid issue 53	*Sensory data underutilization*
Zmanim issue 50/56	*Sensory data underutilization*
Sofia Public Transport Nav. issue 38	*Sensory data underutilization*
Geohash Droid issue 24	*Sensory data underutilization*
DroidAR issue 27	*Missing sensor deactivation*
Recycle-Locator issue 33	*Missing sensor deactivation*
Ushahidi issue 11	*Missing sensor deactivation*
*Omnidroid issue 179**	*Sensory data underutilization*
*GPSLogger issue 7**	*Sensory data underutilization*

New bugs found

"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

Application	Analysis Overhead	
	Time (seconds)	Space (MB)
Osmdroid (18 KLOC)	151	591
Zmanim	110	205
Omnidroid (12 KLOC)	220	342
DroidAR (18 KLOC)	276	217
Recycle-locator	43	153
GPSLogger	35	149
Sofia Public Transport Nav.	17	204
Ushahidi	32	175
Geohash Droid	185	229

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

Efficiency

Application	Analysis Overhead	
	Time (seconds)	Space (MB)
Osmdroid (18 KLOC)	151	591
Zmanim	110	205
Omnidroid (12 KLOC)	220	342
DroidAR (18 KLOC)	276	217
Recycle-locator	43	153
GPSLogger	35	149
Sofia Public Transport Nav.	17	204
Ushahidi	32	175
Geohash Droid	185	229

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

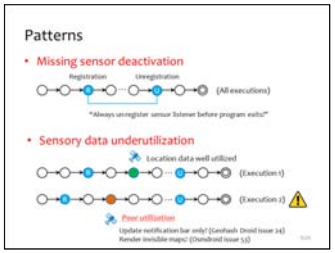
Discussion

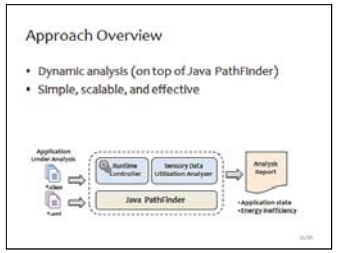
- Patching GPSLogger
  - Invited to provide a patch
  - Built the patch by following a real one (Geohash Droid issue 24)
  - Patch accepted and released online.
- GreenDroid limitations
  - Complex inputs generation (e.g., password)
  - Dynamic GUI updates (GUI models are extracted statically)

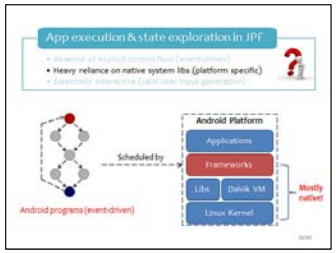
Future Work

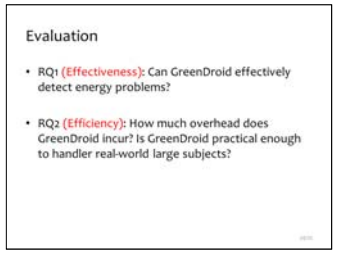
- More energy problem patterns
  - Initial evidence: 16% energy problems was caused by network issues (e.g., energy-inefficient data transmission)
- Integration to Android framework
  - Modify Dalvik VM for data utilization analysis
  - On-device detection of energy problems

Conclusion









Thank you!