Enhancing Mobile Apps to Use Sensor Hubs without Programmer Effort

Haichen Shen, Aruna Balasubramanian, Anthony LaMarca, David Wetherall
Continuous sensing apps

- Step Counting
- Fall Detection
- Driver Monitor
- Theft Detection

Healthcare apps: Ambulation
Lifestyle monitoring: BeWell, Acoustic
Participatory sensing: MobiPerf
But it drains the battery

A Google User - August 22, 2012 - Samsung Galaxy Nexus with version 3.0.120704r635
★★★★★ Destroys your battery
Appears to be little more than a tool to see how much time you spend talking to other people. Assumes you are asleep if you're not using your phone. Basic reports and absolutely abuses your battery. Uninstalled.

A Google User - August 27, 2012 - Samsung Galaxy Tab with version 3.0.120704r635
★★★★★ Battery Issue
Sucks up all your battery

A Google User - August 22, 2012 - Droid Bionic with version 3.0.120704r635
★★★★★ Major battery and memory hog
This app does what it says, but it alone was consuming about 10% of my battery and about half my memory. As if I didn't have enough performance issues already with my phone. Uninstalled.

Problem: CPU frequently wakes up to process sensor data
Sensor hub: low power processor

~1.5 mW at 2MHz
Existing approaches make it hard to leverage sensor hub

### APIs
- Provided by software company, e.g. Apple, Google
  - Easy to program
  - Only support a set of pre-defined events
  - Require programmer effort

### Hardware SDK
- Provided by hardware manufacturer, e.g. TI TivaWare
  - Full control of sensor hub
  - Compatibility
  - Require programmer effort
MobileHub: leverage sensor hub without programmer effort

MobileHub System

Analyze app and rewrite binary

Optimized app: Same semantics, but more energy efficient
MobileHub example

Challenge: How does MobileHub know when the application needs to be triggered?
MobileHub system overview

**Step 0: Sensor traces**
- Original app
- Optimized app

**Step 1 Dynamic taint tracking:**
Track app notifications for a series of sensor inputs
- Taint log

**Step 2 Learning:**
Learn how changes in sensor values result in app notifications
- Classifier model

**Step 3a Sensor hub program:**
Implement the classifier in the sensor hub, corresponding to the app
- Sensor parameter

**Step 3b Application rewriting:**
Rewrite app to offload sensing to the sensor hub
MobileHub system overview

**Step 0: Sensor traces**
- Original app
- Optimized app

**Step 1 Dynamic taint tracking:**
Track app notifications for a series of sensor inputs

**Step 2 Learning:**
Learn how changes in sensor values result in app notifications

**Step 3b Application rewriting:**
Rewrite app to offload sensing to the sensor hub

**Step 3a Sensor hub program:**
Implement the classifier in the sensor hub, corresponding to the app
Why do we need taint tracking?

- Goal: to track when a sensor value leads to an app notification.
- Observing the app notifications alone is insufficient.
- Use taint tracking to track the sensor data from when it was recorded to when it was used by the application.
void onSensorChange(sensorEvent){
    \textcolor{blue}{\textbf{Taint tag}}\textbf{X} = sensorEvent.val;
    \textcolor{blue}{\textbf{Taint tag}}\textbf{Y} = (\textbf{X} + 1) / 2;
    \textcolor{blue}{\textbf{Taint tag}}\textbf{if} (\textbf{Y} > \textbf{THRESHOLD}) {
        \textcolor{blue}{\textbf{Taint tag}}\textbf{stepCounter}++;
        \textcolor{blue}{\textbf{Taint tag}}\textbf{display}(\textbf{stepCounter});
    };
}
Challenge: implicit flow tracking

• Most taint tracking platforms only track explicit flow
• Without implicit flow tracking, we could only track 20% of triggers for sensing apps
• Use instrumentation to force implicit flow tracking
  • Built on top of TaintDroid [Enck_OSDI2010]
Instrumentation for implicit flow tracking

```java
void onSensorChange(sensorEvent){
    X = sensorEvent.val;
    Y = (X + 1) / 2;
    tag = getTaintTag(Y);
    if (Y > THRESHOLD) {
        stepCounter++;
        Taint(stepCounter, tag);
        display(stepCounter);
    }
}
```

Use static analysis to identify all taint blocks and instrument the app binary automatically.
Step 0: Sensor traces

Step 1 Dynamic taint tracking:
Track app notifications for a series of sensor inputs

Taint log

Step 2 Learning:
Learn how changes in sensor values result in app notifications

Step 3a Sensor hub program:
Implement the classifier in the sensor hub, corresponding to the app

Step 3b Application rewriting:
Rewrite app to offload sensing to the sensor hub

Original app

Optimized app

MobileHub System

Classifier model

Sensor parameter
Learning a buffer policy

• Hard to use a classifier to model the app logic
• Simply learn the statistical properties and distinguish between idle and active periods
Goal: find a proper buffer size

- Predict active and idle periods
- Reduce the number of notification delays

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>Buffer</td>
</tr>
</tbody>
</table>

Less energy saving

More energy saving but notification delay
Step 0: Sensor traces

Step 1 Dynamic taint tracking: Track app notifications for a series of sensor inputs

Taint log

Step 2 Learning: Learn how changes in sensor values result in app notifications

Original app

Step 3a Sensor hub program: Implement the classifier in the sensor hub, corresponding to the app

Step 3b Application rewriting: Rewrite app to offload sensing to the sensor hub

Optimized app

MobileHub System

Classifier model

Sensor parameter

MobileHub system overview
Implementation

• Implemented in Android
  – Taint tracking system
  – Interface with sensor hub
  – App binary rewriter

• Prototype
  – Implemented classifier on sensor hub
Evaluation

• Does the prototype work?

• Does MobileHub improve power consumption on real traces?

• Does MobileHub work for a large number of apps?
Prototype measurement

![Energy Consumption Graph](image)

- **MobileHub-instrumented App**
- **Unmodified App**

**Pedometer**

Energy Consumption (J)
Evaluation using real sensor traces

- Trace collection from 21 participants
  - 10 traces for sleeping, driving, and daily life
  - 5 traces for other activities

- Downloaded 20 apps from Google Play
<table>
<thead>
<tr>
<th>Name</th>
<th>Google Play Store ID</th>
<th>Task</th>
<th>Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>nWalk</td>
<td>pl.rork.nWalk</td>
<td>Step counting</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>pedometer</td>
<td>bagi.levente.pedometer</td>
<td>Step counting</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>stepcounter</td>
<td>Stepcounter.Step</td>
<td>Step counting</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>appsone</td>
<td>net.appsone.android.pedometer</td>
<td>Step counting</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>virtic</td>
<td>jp.virtic.apps.WidgetManpok</td>
<td>Step counting</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>walking</td>
<td>cha.health.walking</td>
<td>Step counting</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>lodecode</td>
<td>com.lodecode.metaldetector</td>
<td>Metal detector</td>
<td>Magnetometer</td>
</tr>
<tr>
<td>imkurt</td>
<td>com.imkurt.metaldetector</td>
<td>Metal detector</td>
<td>Magnetometer</td>
</tr>
<tr>
<td>tdt</td>
<td>com.tdt.magneticfielddetector</td>
<td>Metal detector</td>
<td>Magnetometer</td>
</tr>
<tr>
<td>multunus</td>
<td>com.multunus.falldetector</td>
<td>Fall detector</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>iter</td>
<td>com.iter.falldetector</td>
<td>Fall detector</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>t3lab</td>
<td>it.t3lab.fallDetector</td>
<td>Fall detector</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>fall</td>
<td>com.fall</td>
<td>Fall detector</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>jietusoft</td>
<td>com.jietusoft.earthquake</td>
<td>Earthquake detector</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>vibration</td>
<td>ycl.vibrationsensor</td>
<td>Earthquake detector</td>
<td>Orientation</td>
</tr>
<tr>
<td>posvic</td>
<td>cz.posvic.fitnessbar.sleeptrack</td>
<td>Sleep monitoring</td>
<td>Gyroscope</td>
</tr>
<tr>
<td>myway</td>
<td>myway.project.sleepmanagement</td>
<td>Sleep monitoring</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>driving</td>
<td>jp.co.noito.Accelerometer</td>
<td>Driver monitoring</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>motion</td>
<td>com.app.accelerometer</td>
<td>Motion detector</td>
<td>Accelerometer</td>
</tr>
<tr>
<td>thefthead</td>
<td>com.thefthead.appfinalsettings</td>
<td>Theft detector</td>
<td>Accelerometer</td>
</tr>
</tbody>
</table>
Trace evaluation methodology

• Run each app on the phone receiving sensor values from a trace file
• Trace file embeds the buffering policy

Power Accounting:
• Measure the power consumption of phone
• Deduct the standby power consumption
Notification delay

- Notification is delayed by at least 0.5s

<table>
<thead>
<tr>
<th>App</th>
<th>Task</th>
<th>#Delay/#Notifications</th>
<th>Max delay (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nWalk</td>
<td>Step Counting</td>
<td>1/3914</td>
<td>1.86</td>
</tr>
<tr>
<td>imkurt</td>
<td>Fall Detection</td>
<td>2/142</td>
<td>0.98</td>
</tr>
<tr>
<td>posvic</td>
<td>Sleep Monitor</td>
<td>1/36</td>
<td>0.64</td>
</tr>
<tr>
<td>thefthead</td>
<td>Anti-theft</td>
<td>6/65</td>
<td>2.80</td>
</tr>
</tbody>
</table>
Conclusion

• Design and implement MobileHub that rewrites application to leverage sensor hub without programmer effort
• Experiment with 20 sensing apps, and reduce power consumption by 74% in median
• MobileHub delays 1.5% app notifications across all apps on average
Thank you!

haichen@cs.washington.edu
Sensor Hub Service
Dynamic vs static buffer

![Graph showing comparison between Static Buffer and MobileHub. The x-axis represents Static/Average Buffer Size, and the y-axis represents the Percentage of delayed notifications. The graph includes data points for different buffer sizes and the percentage of delayed notifications for both Static Buffer and MobileHub.]