
Abhilash Jindal
Y. Charlie Hu
Towards optimizing app battery drain

Source code energy profilers

DIFFPROF (this paper)

Is there room for improvement? How to do the optimization?

App-1

A: 10 μAh
B: 90 μAh
C: 20 μAh

App-1

A: 10 μAh

App-2

X: 10 μAh
B: 90 μAh
C: 20 μAh
Y: 20 μAh
Part I: Why diffing?

Key Observations
There is an app for that!
There are dozens of apps for that!

<table>
<thead>
<tr>
<th>Category</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Shopping</td>
<td>Walmart, Wish, eBay, Amazon, Flipkart, Snapdeal, Alibaba, Lazada</td>
</tr>
<tr>
<td>Music</td>
<td>Pandora, Apple Music, iHeartRadio, musical.ly, Music Player, SongFlip, Spinrilla</td>
</tr>
<tr>
<td>Chess</td>
<td>Chess Free, Real Chess, Chess - Play &amp; Learn, Chess, Chess With Friends, Chess Online</td>
</tr>
</tbody>
</table>
Similar apps implement similar functionalities

O1: Dozens of similar apps implement similar app functions

(a) Apple Music  (b) Google Play music  (c) iHeartRadio  (d) Pandora  (e) Spotify
Battery drain of similar apps differ a lot

O2: Battery drain among similar apps differ significantly (2.8x – 8.0x)
Comparing energy profiles can potentially be effective

O1: Dozens of similar apps

O2: Battery drain differ significantly

App-1
- A: 10 uAh
- B: 90 uAh
- C: 20 uAh

App-2
- X: 10 uAh
- Y: 20 uAh
Framework services dominate energy drain

O3: Framework services, common to all apps, drain up to 90% of total app energy drain
Comparing energy profiles will be effective

**O1: Dozens of similar apps**

**O2: Battery drain differ significantly**

**O3: Framework services dominate battery drain**
Part II: How to diff?

What should be the diffing granularity?
How to perform diffing?
What should be the diffing granularity?

• Similar apps perform similar core tasks
  • Music app performs music playback, UI updates such as progress bar, text boxes

• Diffing should be performed on app tasks
How to identify app tasks from energy profile?

• App tasks manifest as EFLASK (Erlenmeyer flask shaped)
  • Call path
  • Neck F-method
  • Subtree

• Identifying matching tasks boils down to matching EFLASK

Framework subtree

App task (EFLASK)
### What tree structures to diff?

<table>
<thead>
<tr>
<th>Tree type</th>
<th>Size</th>
<th>Path preserving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call tree</td>
<td>O(millions)</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic call graph</td>
<td>O(thousands)</td>
<td>No</td>
</tr>
<tr>
<td>Calling context tree</td>
<td>O(ten thousands)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
How to perform diffing?

• Previous tree matching algorithms are not applicable

• EFLASK matching algorithm
  • Simultaneously identifies EFLASKs and finds matching EFLASKs between two similar apps
Exact path matching

- Unique node with same path from root

Calling Context Tree

App -1

App -2
EFLASKs for matching tasks can vary

• Call paths can vary slightly
  • Use different mechanism to get same callback
  • Use different callbacks to receive similar events
EFLASKs for matching tasks can vary (2)

• Call paths can differ slightly

• **Neck F-methods may vary**
  • Use different classes that implement same APIs
    • HttpURLConnectionURLImpl, HttpsURLConnectionImpl
  • Use alternate APIs to perform same task
EFLASKs for matching tasks can vary (3)

- Call paths can differ slightly
- Neck F-methods may differ
- **F-method subtrees may vary**
  - Program state, call parameters determine F-method subtree
Prior approximate tree matching algorithms

- [Zhang et. al. Algorithmica 1995] produces maximal matching

- Drawback: matches EFLASKS with arbitrarily different paths
  - Maximize subtree overlap, disregard paths
## EFLASK matching algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Passes</th>
<th>Approach</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Path Matching</td>
<td>Top-down</td>
<td>Matches paths, disregards subtree</td>
<td>Can’t handle path variations</td>
</tr>
<tr>
<td>Approximate Tree Matching</td>
<td>Bottom-up</td>
<td>Maximizes subtree overlap, disregards path</td>
<td>Matches nodes with arbitrarily different paths</td>
</tr>
<tr>
<td>(Zhang et.al)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EFLASK matching algorithm</td>
<td>1 top-down, 1 bottom-up pass</td>
<td>Maximizes subtree overlap while respecting path similarity</td>
<td></td>
</tr>
</tbody>
</table>
EFLASK Matching Algorithm

\[ \forall (v, w) \in M : w \in C_\alpha(v) \]

1. Top down pass: Calculate \( C_\alpha(v) \)

\[
C_\alpha(v) = \{ w \in V(T_2) \mid \rho(s(v), s(w)) \leq \alpha \}
\]

2. Bottom up pass:
Calculate \( \mu_\alpha(v) \)

\[
\begin{align*}
\mu_\alpha(T_1(v), \theta) &= 0 \\
\mu_\alpha(\theta, T_2(w)) &= 0 \\
\mu_\alpha(T_1(v), T_2(w)) &= 0 \\
\mu_\alpha(T_1(v), T_2(w)) &= \max \\
&= \max_{w \in \text{child}(w) \in \text{child}(v)} \max_{v \in \text{child}(v)} \mu_\alpha(T_1(v), T_2(w)) \\
&= \max_{w \in \text{child}(w) \in \text{child}(v)} \mu_\alpha(T_1(v), T_2(w)) + \gamma(v, w)
\end{align*}
\]

3. Use backtracking to find matched nodes

4. Finds matching EFLASKS based on maximally matched nodes
Reducing unimportant call path variations – Collapsing app methods

• Internal app method names are arbitrary and often obfuscated

• App callback method names are well-defined by framework
  • foo.onClick overrides onClickListener.onClick
Part III: It works!
DIFFPROF implementation

• Built on top of Eprof [Pathak et. al. EuroSys’ 12]
• Diffing and GUI front-end
  • 5.7k lines Java code
Developer workflow with DIFFPROF

- Write automated tests for two similar apps
- Run the tests with Eprof and collect energy profiles
- Upload energy profiles to DIFFPROF

Tabular output

Graphical output
Evaluation

• Android’s UI Automator tests
• 8 app groups- 5 popular apps, 5 versions of one app, majority with 50M+ installs

<table>
<thead>
<tr>
<th>App Category</th>
<th>App group</th>
<th>Similar/Competing Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Instant Messaging</td>
<td>Whatsapp, Google Hangouts, Facebook Messenger, Line, TextNow</td>
</tr>
<tr>
<td>Email</td>
<td></td>
<td>Android mail, Aqua Mail, Email For Any, MailRU, myMail</td>
</tr>
<tr>
<td>Music &amp; Audio</td>
<td>Music streaming</td>
<td>Spotify, Pandora, Soundcloud, iHeartRadio, Free music</td>
</tr>
<tr>
<td>Personalization</td>
<td>Launcher</td>
<td>GO, CM Launcher 3D, APUS, Solo, Hola</td>
</tr>
<tr>
<td>Productivity</td>
<td>File explorer</td>
<td>ES, FX, Solid, File explorer, File manager</td>
</tr>
<tr>
<td>Shopping</td>
<td>Shopping</td>
<td>Wish, eBay, Amazon, Kohl, letgo</td>
</tr>
<tr>
<td>Tools</td>
<td>Antivirus</td>
<td>CM Security, AVG, DU, Mobile Security &amp; Antivirus, Kaspersky</td>
</tr>
<tr>
<td></td>
<td>Cleaning</td>
<td>Clean Master, DFNDR, Fast Cleaner, Turbo cleaner, DU, Ccleaner</td>
</tr>
</tbody>
</table>
## Evaluation – Matching task statistics

<table>
<thead>
<tr>
<th>App</th>
<th>Matched tasks’ energy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Antivirus</strong></td>
<td></td>
</tr>
<tr>
<td>AVG</td>
<td>92.73%</td>
</tr>
<tr>
<td>CMSecurity</td>
<td>79.64%</td>
</tr>
<tr>
<td>DU</td>
<td>85.90%</td>
</tr>
<tr>
<td>Kaspersky</td>
<td>73.99%</td>
</tr>
<tr>
<td>MobileSec</td>
<td>69.98%</td>
</tr>
<tr>
<td><strong>Cleaner</strong></td>
<td></td>
</tr>
<tr>
<td>CCleaner</td>
<td>76.57%</td>
</tr>
<tr>
<td>Clean Master</td>
<td>70.82%</td>
</tr>
<tr>
<td>DFNDR</td>
<td>73.52%</td>
</tr>
<tr>
<td>Fast Cleaner</td>
<td>94.89%</td>
</tr>
<tr>
<td>Turbo Cleaner</td>
<td>88.46%</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>78.86%</td>
</tr>
</tbody>
</table>
Evaluation – Case studies

• Found 12 energy optimizations in 9 popular apps
  • 3 of them already confirmed by developers
  • Saves 5.2% - 27.4%

<table>
<thead>
<tr>
<th>App</th>
<th>Task</th>
<th>Percentage of total energy drain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wish letgo</td>
<td>Bitmap.compress</td>
<td>15.9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.6%</td>
</tr>
<tr>
<td>Wish letgo</td>
<td>BitmapFactory.decodeStream</td>
<td>19.9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5%</td>
</tr>
<tr>
<td>Pandora 5.7</td>
<td>TextView.setText</td>
<td>28.1%</td>
</tr>
<tr>
<td>Pandora 8.3</td>
<td>ProgressBar.setProgress</td>
<td>20.2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.6%</td>
</tr>
</tbody>
</table>
EFLASK matching algorithm’s effectiveness:

Wish vs letgo
# DIFFPROF vs Eprof: Wish vs letgo

## DIFFPROF

<table>
<thead>
<tr>
<th>Task energy drain difference rank</th>
<th>Task Name</th>
<th>Wish energy drain (μAh)</th>
<th>Letgo energy drain (μAh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BitmapFactory.decodeStream</td>
<td>126.3</td>
<td>5.01</td>
</tr>
<tr>
<td>2</td>
<td>BitmapFactory.compress</td>
<td>100.9</td>
<td>7.14</td>
</tr>
<tr>
<td>3</td>
<td>LayoutInflater.inflate</td>
<td>62.46</td>
<td>17.03</td>
</tr>
</tbody>
</table>

## Eprof

<table>
<thead>
<tr>
<th>Method energy drain rank</th>
<th>Method name</th>
<th>Wish Energy drain (μAh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thread.run</td>
<td>395.0</td>
</tr>
<tr>
<td>2</td>
<td>ThreadPoolExecutor.runWorker</td>
<td>353.9</td>
</tr>
<tr>
<td>18</td>
<td>BitmapFactory.decodeStream</td>
<td>126.3</td>
</tr>
<tr>
<td>28</td>
<td>BitmapFactory.compress</td>
<td>100.9</td>
</tr>
</tbody>
</table>
Wish vs letgo: Energy optimization

• Setting breakpoint at Bitmap.compress method reveals
  • Wish uses png images with quality set to 100
  • letgo uses jpg images with quality set to 90
Conclusions

Why diffing?
- Dozens of similar apps
- Similar apps differ significantly in battery drain
- Framework services dominate battery drain

How to diff?
- App tasks manifest as EFLASK (call path, neck F-method, subtree)
- EFLASK matching algorithm

Diffing works!
- DIFFPROF matches tasks consuming 80% of total energy
- Found 12 energy optimization opportunities in 9 popular apps