Improving the Display Energy of Smartphone Apps

Ding Li, Mian Wan, Yuchen Jin, Angelica Huyen Tran, William G.J. Halfond

University of Southern California

Work supported by NSF grant 1321141
Energy Consumption of System APIs vs. Bytecode vs. Outliers

Breakdown of app execution energy

- 85% API
- 13% Bytecode
- 2% Outliers

Bytecode does not consume a significant amount of energy.
Energy Consumed by the Idle State of An Application

Only optimizing code is insufficient, energy consumed during idle-state also needs to be optimized.

- Code running, 38%
- Waiting for input, 37%
- Sleep, 25%

62% of energy consumed when not “executing”
Importance of Display Energy

AVERAGE SCREEN SIZE OF NEW SMARTPHONE MODELS
4906 smartphone models, launched from Jan 2007 to May 2014
@somospostpc
Smartphone Display: OLED

- Popular technology for smartphone displays
- More energy efficient than prior technologies
- Different energy consumption patterns
Display Oriented Techniques

• Dim the display
  – Good start, but more can be done

• Invert colors:
Goal

Automatically transform the implementation of a web application so that the web pages it generates consume less energy, but maintain aesthetics, when displayed on an OLED smartphone.
Challenges

1. Identify the colors used in a web application – Necessary to compute the web pages that can be generated by the application

2. Determine the relationship of colors in the web application – Important ones: adjacency and enclosure

3. Transform colors into a more efficient scheme – Maintain usability and attractiveness (aesthetics)

```java
public void print_html()
{
    print("<body bgcolor="white" style="color:black;">\n    println("<table><tr>\n    int a=1;
    if(a==0){
        println("<td>hi</td>\n    }
    else{
        println("<td style="background-color:red; color:yellow;">ha</td>\n    }
    for(int i=0;i<2;i++){
        println("<td style="background-color:green; color:blue;">usc</td>\n    }
    println("</tr></table>\n    println("</body>\n```
## Approach Overview

1. Compute the set of generated HTML pages
2. Determine visual relationships in pages
   - Example: adjacent and contained
3. Identify colors that have visual relationships
4. Solve for a new color scheme
   - Is more energy efficient
   - Maintains similar color differences
5. Rewrite application to use new color scheme

**Phase 1**

**Phase 2**

**Phase 3**
Phase 1: HTML Output Analysis

A. Compute the set of HTML pages that could be generated by the application at runtime

B. Determine visual relationships among HTML elements in the pages
   - Example: adjacent and contained
Phase 1A: HTML Output Graph

- A projection of the CFG, based on work by Moller and Schwarz
- The nodes are the instructions that print content to the web page
  - For example: `JspWriter.println()`
  - The value is represented as FSA
  - FSA generated based on work by Yu and colleagues
- The edges are the paths in CFG that connect each printing instruction

```html
<body bgcolor="white" style="color:black">
  <table>
    <tr>
      <td>hi</td>
      <td style="background-color:red; color:yellow;">ha</td>
      <td style="background-color:green; color:blue;">usc</td>
    </tr>
  </table>
</body>
```
2: Color Transformation

Color Conflict Graph (CCG)

- Shows visual relationships of colors in a page
- BCCG: weights are in \{a, b, c\}
  - \(a > b > c > 0\)
  - \(a\): parent-child
  - \(b\): siblings
  - \(c\): everything else
2: Color Transformation

Building the Color Conflict Graph

1. Basic unit is color definition (CD)
   - CSS based
   - HTML based

2. Perform reachability analysis over visual relationship graph

3. “Reaching CDs” define edges in CCG
2: Color Transformation

**BCCG**: weights are in \{a, b, c\}, \(a > b > c > 0\)
- a: parent-child
- b: siblings
- c: everything else
2: Color Transformation

Generate the color transformation scheme (CTS)

1. Let $S = \langle C_0, C_1, C_2, \ldots, C_k \rangle$ nodes of the CCG
2. Let $S'$ be the new coloring, where $C_0 =$black
3. Compute $S'$ that results in similar color differences as in $S$, i.e. minimize:

$$\sum_{i=0}^{k} \sum_{j=0}^{k} w_{ij} |\text{Dist}(C_i, C_j) - \text{Dist}(C_i', C_j')|$$

4. Optimization problem is NP-Hard, use simulated annealing to approximate optimal solution
Phase 3: Output Modification

1. Dynamically generated HTML pages
   – Insert instrumentation to replace HTML printing instructions
   – Replace original colors with new colors

2. Template based frameworks
   – Use CSS parser to identify entries to be replaced
   – Replace entries by rewriting CSS and HTML
Evaluation

• **RQ 2:** How much energy is saved by the transformed web pages?
• **RQ 3:** To what degree do users accept the appearance of the transformed web pages?
RQ2: Energy Savings

Loading Energy decrease: 25%
Display Power decrease: 40%
RQ3: User Acceptance

Users asked to rate before/after color transformation produced by our approach
1. How do you rate the readability?
2. How do you rate the appearance?
3. If the version on the right could save you X% of the energy, at what battery level would you choose to use it?
   a) Always – regardless of battery level
   b) Most of the time
   c) Only when the battery level is low
   d) Only when the battery level is critical
   e) Never
Attractiveness

Average decrease of 17%

Readability

Average decrease of 15%
• 60% choose transformed app for general usage
• 97% choose transformed app for battery critical
Where to Apply Display Optimization Techniques?

• Apply to the whole app
  – Some UIs may already be energy-efficient
  – Don’t want to use automatically transformed colors

• Apply according to developers’ intuition
  – The judgement is subjective and error-prone
Identify Display Energy Inefficiencies

• **Goal** – to identify the UIs that are not energy efficient
  – Display Energy Hotspot (DEH): a UI of a mobile app whose energy consumption is higher than an energy-optimized but functionally equivalent one

• **Intuition**: use color transformation to generate an energy efficient baseline and estimate how much energy can be saved through power modeling.
Overview of dLens

1. Target App
2. Replay and Capture
3. Establish Optimization Baseline
4. Predict Display Energy
5. Rank UIs
6. UI Ranking

- Workload
- DEP
1. Workload Replay and Screenshot Capture

Workload

<event, timestamp>

APK

Replay and Capture Mechanism

Screen shots

<screenshot, timestamp>
2. Establish Optimization Baseline

- To quantify the optimization potential for a UI, we need an optimization baseline
- How to generate it?
  - Generate reasonably optimized version of the UI
  - Use this version of UI as a baseline
2. Establish Optimization Baseline
3. Predict Display Energy

Step 1: Screenshots

<screenshot, timestamp>

Step 2: Transformed Screenshots

Prediction Module

Power & Energy of screenshots

DEP
3. Predict Display Energy

• For screenshot $s_i$, we get its energy estimate
  \[ E(s_i, t_i, t_{i+1}) = P(s_i) \times (t_i - t_{i+1}) \]

• As for power, its power is the sum of each pixel’s power:
  \[ P(s_i) = \sum_{k \in |s_i|} C(R_k, G_k, B_k) \]

• At the granularity of a pixel, its power model $C(R_k, G_k, B_k)$ is defined in a Display Energy Profile (DEP)
How to Construct a DEP

\[ C(R, G, B) = rR + gG + bB + c \]
4. Prioritize the User Interfaces

**inputs**: power and energy of original screenshot $s$ and its transformed one $s'$

\[
\Delta P = P_S - P_{S'}, \\
\Delta E = E_S - E_{S'}
\]

\[
IsDEH(s,p) = \begin{cases} 
true, & p > 0 \\
false, & p \leq 0
\end{cases}, \quad p \in \{\Delta P, \Delta E\}
\]

Sort the screenshots in descending order based on the magnitude of $\Delta P$ and $\Delta E$
# Example of the Output of dLens

<table>
<thead>
<tr>
<th>Rank</th>
<th>Screenshot</th>
<th>$\Delta P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1.png" alt="Screenshot" /></td>
<td>155.10</td>
</tr>
<tr>
<td>2</td>
<td><img src="image2.png" alt="Screenshot" /></td>
<td>154.46</td>
</tr>
<tr>
<td>3</td>
<td><img src="image3.png" alt="Screenshot" /></td>
<td>153.37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>Screenshot</th>
<th>$\Delta E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image4.png" alt="Screenshot" /></td>
<td>2339.09</td>
</tr>
<tr>
<td>2</td>
<td><img src="image5.png" alt="Screenshot" /></td>
<td>2147.31</td>
</tr>
<tr>
<td>3</td>
<td><img src="image6.png" alt="Screenshot" /></td>
<td>1575.40</td>
</tr>
</tbody>
</table>
Evaluation

• **RQ 1**: How accurate is the dLens analysis?

• **RQ 2**: How generalizable are the dLens results across devices?

• **RQ 3**: How long does it take to perform the dLens analysis?

• **RQ 4**: What is the potential impact of the dLens analysis?
## Subject Applications and Devices

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (MB)</th>
<th>Screenshots</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>23.7</td>
<td>116</td>
<td>554</td>
</tr>
<tr>
<td>Facebook Messenger</td>
<td>12.9</td>
<td>55</td>
<td>268</td>
</tr>
<tr>
<td>FaceQ</td>
<td>17.9</td>
<td>96</td>
<td>470</td>
</tr>
<tr>
<td>Instagram</td>
<td>9.7</td>
<td>93</td>
<td>429</td>
</tr>
<tr>
<td>Pandora internet radio</td>
<td>8.0</td>
<td>75</td>
<td>278</td>
</tr>
<tr>
<td>Skype</td>
<td>19.9</td>
<td>65</td>
<td>254</td>
</tr>
<tr>
<td>Snapchat</td>
<td>8.8</td>
<td>142</td>
<td>465</td>
</tr>
<tr>
<td>Super-Bright LED Flashlight</td>
<td>5.1</td>
<td>20</td>
<td>51</td>
</tr>
<tr>
<td>Twitter</td>
<td>13.7</td>
<td>101</td>
<td>388</td>
</tr>
<tr>
<td>WhatsApp Messenger</td>
<td>15.3</td>
<td>65</td>
<td>242</td>
</tr>
</tbody>
</table>

μOLED

Galaxy S2

Galaxy Nexus
RQ1: Accuracy of Power Model

The average estimation error rate varied from 5% to 8% across these 3 devices.
RQ2: Generalizability

DEH results for one device can typically represent the results for many other similar devices.

The rankings are almost identical ($\bar{R} = 0.9929$)
## RQ3: Analysis Time

<table>
<thead>
<tr>
<th>Name</th>
<th>Time for Color Transformation (s)</th>
<th>Time for Estimation (s)</th>
<th>Overall (s)</th>
<th>Per UI(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>1,470</td>
<td>7</td>
<td>1,477</td>
<td>12</td>
</tr>
<tr>
<td>Facebook Messenger</td>
<td>997</td>
<td>3</td>
<td>1,001</td>
<td>18</td>
</tr>
<tr>
<td>FaceQ</td>
<td>1,145</td>
<td>5</td>
<td>1,151</td>
<td>12</td>
</tr>
<tr>
<td>Instagram</td>
<td>2,799</td>
<td>6</td>
<td>2,806</td>
<td>30</td>
</tr>
<tr>
<td>Pandora internet radio</td>
<td>1,418</td>
<td>4</td>
<td>1,423</td>
<td>19</td>
</tr>
<tr>
<td>Skype</td>
<td>871</td>
<td>3</td>
<td>875</td>
<td>13</td>
</tr>
<tr>
<td>Snapchat</td>
<td>1,444</td>
<td>8</td>
<td>1,453</td>
<td>10</td>
</tr>
<tr>
<td>Super-Bright LED Flashlight</td>
<td>863</td>
<td>1</td>
<td>865</td>
<td>43</td>
</tr>
<tr>
<td>Twitter</td>
<td>1,316</td>
<td>6</td>
<td>1,323</td>
<td>13</td>
</tr>
<tr>
<td>WhatsApp</td>
<td>897</td>
<td>3</td>
<td>901</td>
<td>13</td>
</tr>
</tbody>
</table>
RQ4: Potential Impact

• We searched for DEHs in a large set of Android apps from Google Play
• After automatically taking screenshots, we manually checked all screenshots and removed invalid screenshots
  – In total, we collected screenshots of 962 apps
• We used dLens to analyze these apps’ initial pages
398 apps contain DEHs

One app consumes 101% more energy
Summary

• **Understand energy consumption**
  – Idle state energy consumption is significant
  – Display is a major part of this

• **Change energy consumption**
  – Automatically rewrite web pages so they use more energy efficient color scheme

• **Detect DEHs in mobile apps**
  – Present in 398 (41%) apps of 962 Android apps
Improving the Display Energy of Smartphone Apps

Ding Li, Mian Wan, Yuchen Jin, Angelica Huyan Tran, William G.J. Halfond

University of Southern California

Work supported by NSF grant 1321141