FlexDisplay: A Flexible Display Framework to Conserve Smartphone Battery Power

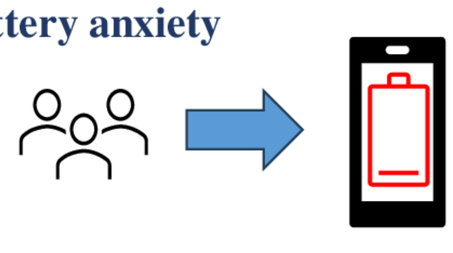
INDRAPRASTHA INSTITUTE of INFORMATION TECHNOLOGY DELHI

Anshak Goel, Manavjeet Singh, Deeptorshi Mondal, Sahil Goyal, Navneet Singh, Jian Xu, Mukulika Maity, Arani Bhattacharya

Stony Brook University

Motivation and Basic Objectives

Smartphone users suffer from **low-battery anxiety**



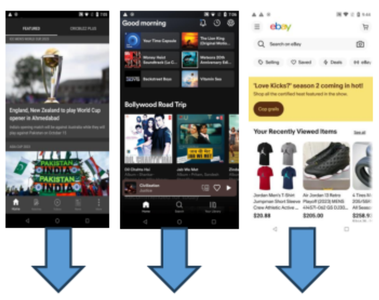
Prior Works: Dim screen brightness [1], reduce resolution [2], use less energy-intensive colors [3]

Rendering the content still consumes significant energy

Goal of FlexDisplay: Disable parts of user interfaces that do not provide important users, such as:

- Video while listening to podcast over Youtube
- Video of the participants in video-conferencing
- Images in podcast, news and online shopping

FlexDisplay uses a combination of app, middleware and static analysis of apps to allow users to disable user interfaces for diverse apps selectively



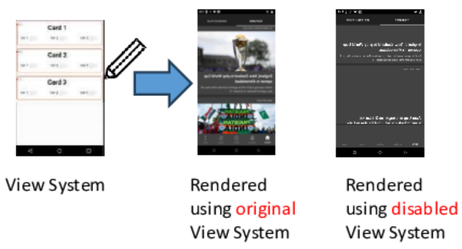
Not adding to information, but is rendered; FlexDisplay aims to disable it

Working of FlexDisplay: Four Steps

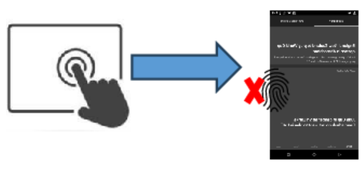
1. Identify relevant classes in the app's APK by static analysis

2. User utilizes our app to determine what to disable

3. Based on user's choice, use enhanced or original View System to either render or not render a UI component

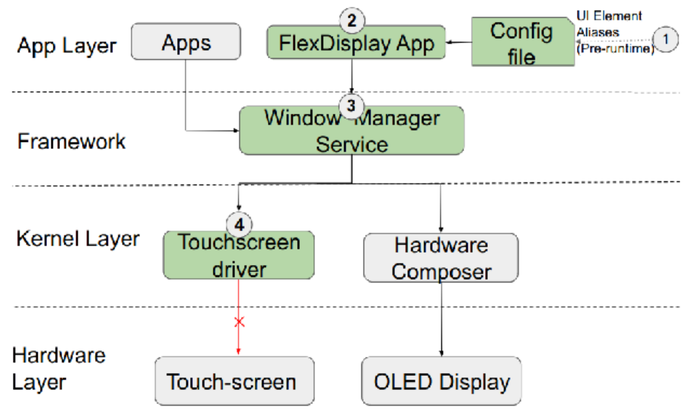


4. Disable mistapping on areas where UI is disabled by modifying adding conditions to the touch kernel module



FlexDisplay's above workflow is general enough to test on 15 apps across 8 categories on two smartphones

Internal Working of FlexDisplay

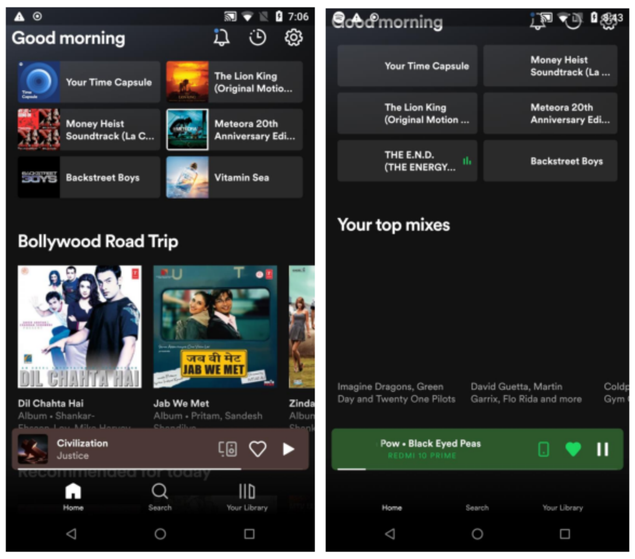


1. Static analysis using apktools. Uses the image class most frequently used, to minimize manual identification of UI elements.
2. An app is created to store the user's preferences. Users can utilize it any time to enable or disable a specific app's UI elements. Choice is currently stored on an external MicroSD card.
3. We create an Enhanced View System within Window Service Manager of Android Framework to add an additional check to see whether to render, based on the user's choice.
4. Touchscreen driver has an additional check to see whether the user has tapped an area within the disabled UI. If so, tap is ignored.

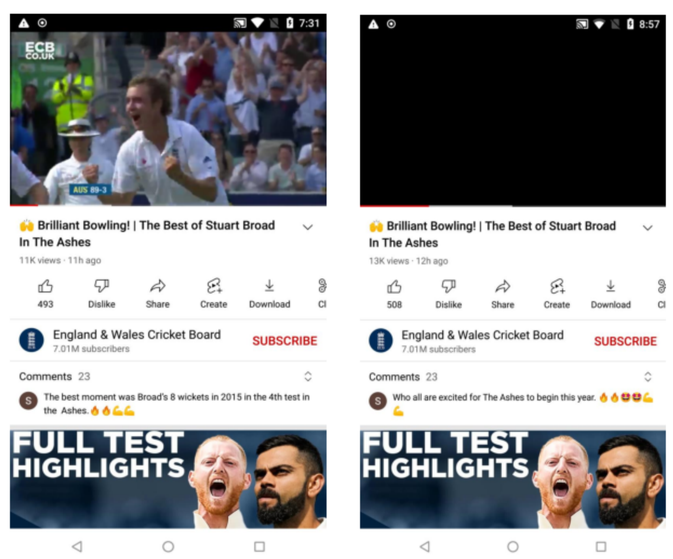
Overall Result

1. Median 14.18% of power savings across all 15 apps on Google Nexus 6 Smartphone, compared to only 5% for completely darkened display
2. Usability studies (with IRB approval) show that users on median give a rating of 3.9/5

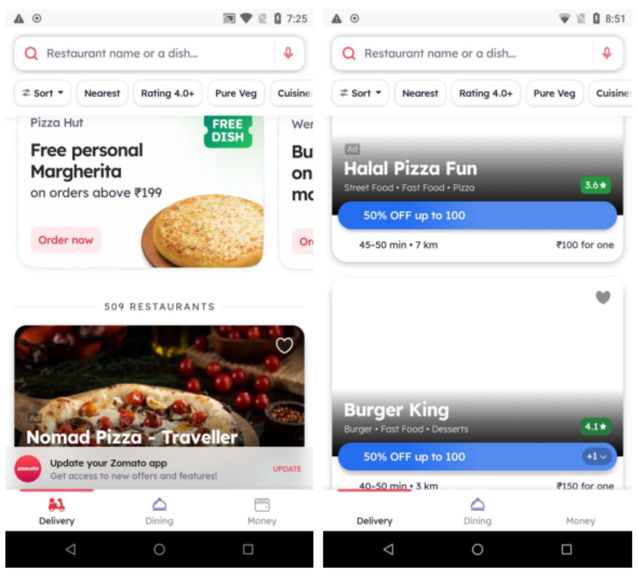
Representative Examples: Four (among 15) apps screenshots shown



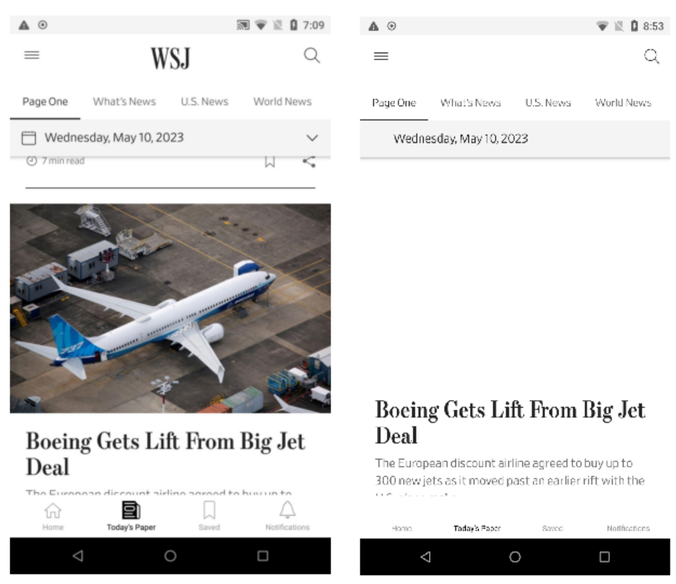
Disabling images on Spotify **saves 22.48%** of total power



Disabling images on Cricbuzz **saves 11.08%** of total power



Disabling images on Zomato (food ordering) **saves 13.22%** of total power



Disabling images on Wall Street Journal **saves 10.12%** of total power

Conclusion

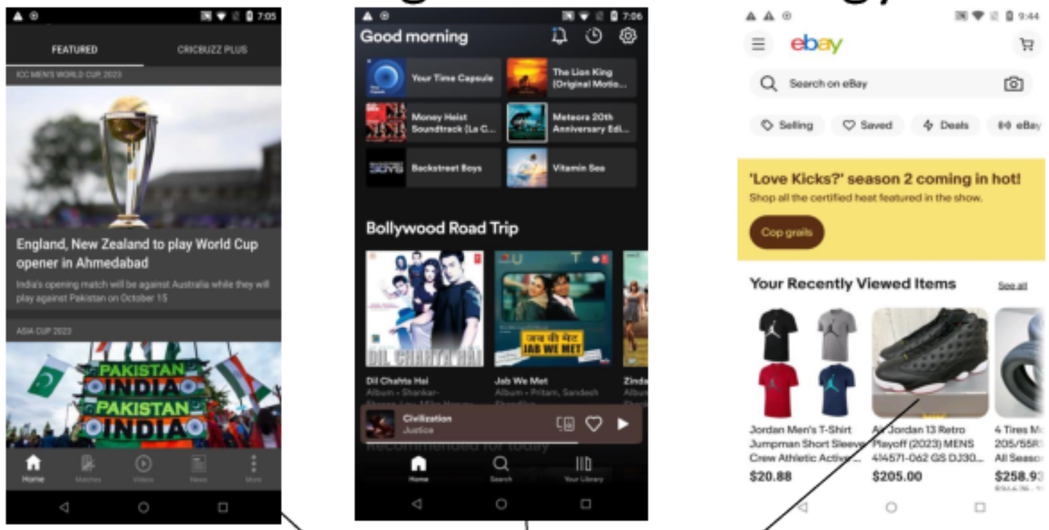
1. FlexDisplay presents a solution to the problem of low-battery anxiety of smartphone users
2. FlexDisplay first identifies UI components that can potentially be disabled, then uses user preferences to decide whether to actually disable
3. Our power measurement and user studies show that FlexDisplay is effective in practice

References

1. Z. Yan and C. W. Chen, "Rnb: Rate and brightness adaptation for rate-distortion-energy tradeoff in http adaptive streaming over mobile devices," in Proceedings of the 22nd Mobicom, pp. 308–319
2. Z. Yan and C. W. Chen, "Too many pixels to perceive: Subpixel shutoff for display energy reduction on OLED smartphones," in Proceedings of the 25th ACM Multimedia, pp. 717–725
3. H.-C. Chang, Y.-C. Yang, L.-Y. Yu, and C.-H. Lin, "Flash: Content-based power-saving design for scrolling operations in browser applications on mobile oled devices," in 2019 IEEE/ACM ISLPED, 2019. pp. 1–6

FlexDisplay: A Flexible Display Framework to Conserve Smartphone Battery Power

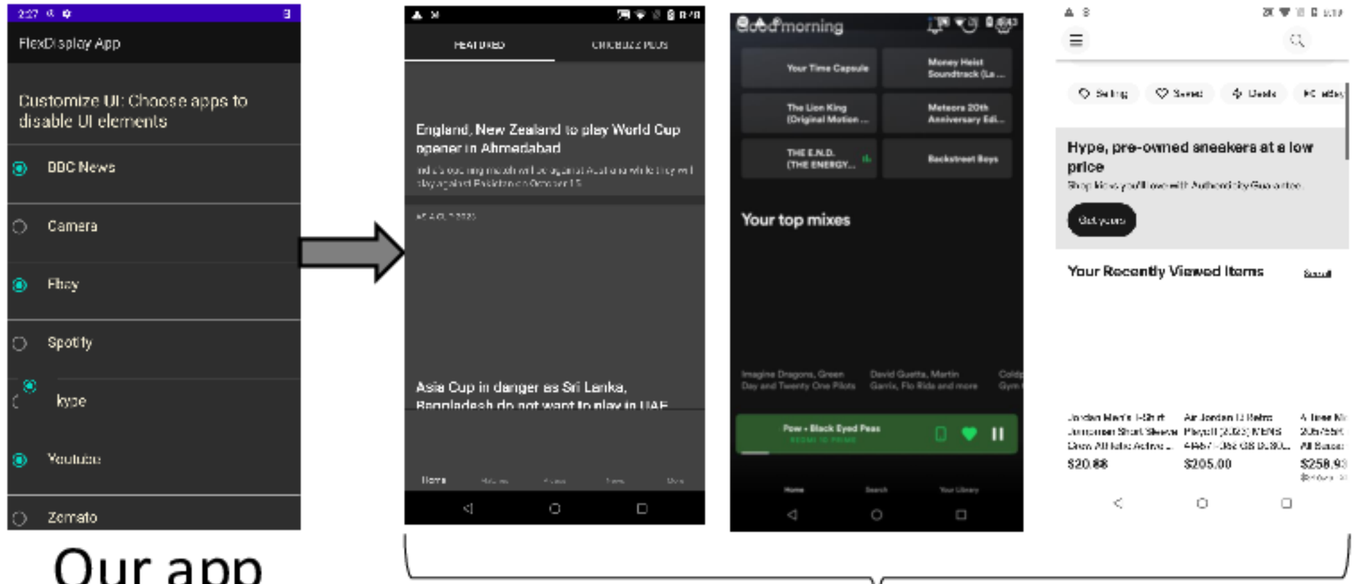
Motivation: Presence of UI components whose rendering consumes energy



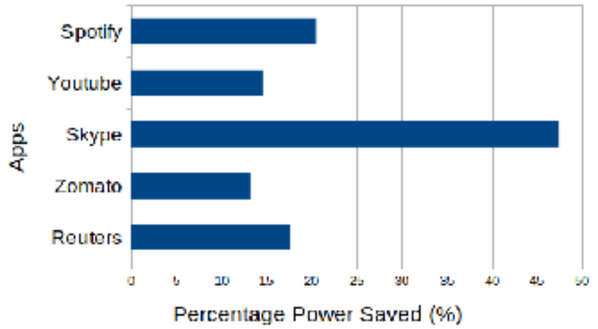
UI elements that present limited information, yet are consistently displayed and visually rendered -- wasting power.

How do we disable rendering of such UI components?

Overview: User uses an app to select which apps images/videos to disable



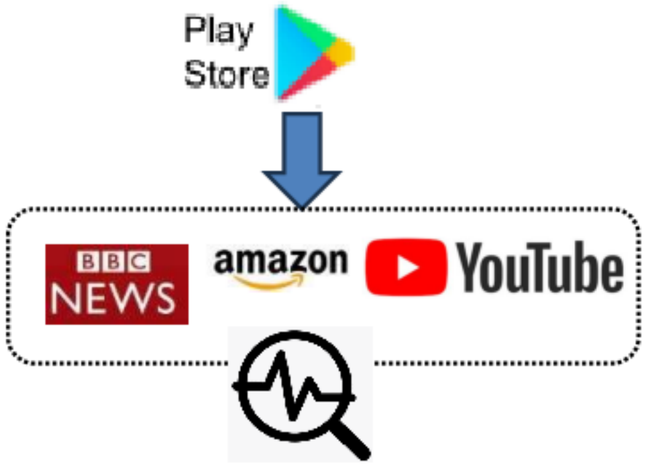
Power saved > 10% across all categories



FlexDisplay: A Flexible Display Framework to Conserve Smartphone Battery Power

Under the hood: Four steps

1. Identify relevant UI component by analyzing app



BBC: ImageView1
Amazon: ImageView1, ImageView2
...

2. Utilize our app to choose what to disable

Writes to

Configuration File

Zomato: ImageClass: Yes (X1,Y1)
BBC: ImageClass: No (X1,Y1)
...

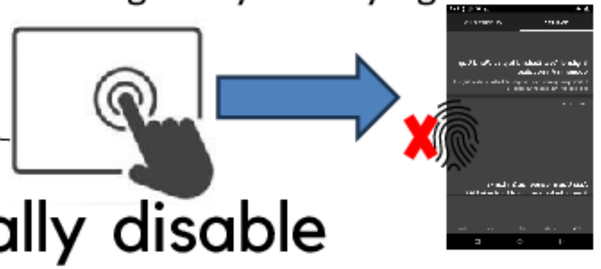
Reads from

3. Enhanced View System used to disable rendering

Rendered using original View System

Rendered using modified View System

4. Prevent mis-taps by disabling touch in the relevant region by modifying kernel module



FlexDisplay acts as a middleware to automatically disable UI components on a large number of apps to save smartphone power