Bootstrapping Energy Debugging on Smartphones:
A First Look at Energy Bugs in Mobile Devices

Anjali Singhal
MTP
Introduction

• Smartphones face a new class of abnormal system behaviors, namely, energy bugs.
• We define an energy bug, or ebug, as an error in the system, either application, OS, hardware, firmware or external that causes an unexpected amount of high energy consumption by the system as a whole.
• Root causes:
  – Programming mistake
  – Inappropriate API usage
  – Flaws in design of apps or OS
  – Wear and tear
  – External conditions
Continued..

• Ebugs do not lead to an application crash, or OS screen of death but the applications and the whole system continue to run, providing their intended services normally, with the exception of consuming unexpectedly large amount of energy.

• This makes bug detection and root cause tracing much more difficult.
Reporting Ebugs

• Bug reports of Android were downloaded which contained info about various ebugs, their causes and fixes, reported online by users.
• Internet forums dedicated to mobile related discussions were crawled which provide much wider and updated coverage of the problems that users have experienced.
• Regex were applied on the test of posts containing keywords ‘battery’, ‘drain’, ‘power’, etc. And they were clustered based on the words they contained using k-means. Then manually those clusters having ebugs were selected.
Ebugs

- **Hardware Bugs**
  - **Battery**: A faulty battery does not hold the complete charge upon charging due to charger damage, old battery, and water damage. Sometimes device also showed incorrect battery statistics.
  - **Exterior Hardware Damage**: Example, external damage to buttons caused home button to be oversensitive, therefore unlocking the phone and turning CPU & backlight on.
  - **SIM**: Due to scratches etc, its internal resistance increases drains more power.
  - **SD card**: Corrupted SD card could trigger buggy apps into looping state accessing hardware repeatedly.
  - **External Hardware**: Erroneous phone chargers, Music speakers, keyboards are also the sources of drain.
Ebugs

• Software Bugs
  – OS bugs: OS updates, by user or forced, was a large fraction of user complaints regarding ebugs. Root causes were:
    • OS Processes: Buggy OS processes eg., suspend process in android was observed to run in background keeping CPU awake.
    • Configuration Changes: Simple Configuration change i.e. incorrect profile to SetCPU for overclocking and underclocking the kernel.
Ebbugs

• Software Bugs
  – Application bugs:
    • No Sleep bug: It do not allow atleast one component of the phone to sleep. It’s a situation where an app acquires a wakelock for a component which wakes the component up, but does not release it even when job is completed. Causes:
      – Programming mistake
      – Sleep conflict

• Sleep Conflict: Aggressive sleeping policies of modern smartphones force CPU to aggressively sleep after an inactivity period. If component is switched to high power state, and CPU sleeps during this process then the code login to bring back the component back to sleep wont run until CPU is woken.
Ebugs

• Software Bugs
  – Application bugs:
    • Loop bug: A loop bug happens where a part of an application enters a looping state performing periodic but unnecessary tasks, draining significant battery. Example conditions that trigger the clients by repeatedly trying to connect to the remote server after password change.
    • Immortality bug: An immortality bug is a situation where a buggy application that drains battery, upon being explicitly killed by the user, respawns, enters the same buggy state.
  – Triggered by External Condition:
    • Weak wireless signal strength causes NIC drivers to compensate by increasing its Tx/Rx power which can significantly increase the energy drain.
  – Unknown Bugs: Root causes not known
Energy Debugging

• An ebug is first detected by a user through the obvious symptom: the battery drain rate suddenly becomes very high for no apparent reason.

• Users use task killers to kill suspicious app (by seeing the fraction of system energy consumed per process or keeping track of CPU wakelocks)

• Goal: To find exact root cause for unexpected high rate of energy consumption.
Energy Debugging Framework

Narrowing Down to One Entity
- Hardware Tests
- History Based Diagnosis
- Network Test
- Source Test
- Component Test
- External Hardware
- Network Related?

Pinpointing the Root Cause
- Instrumented OS
- Configuration
- Power Measurement
- eproff

Bug Detection And Patching
- Source code
- AutoTools/
- Manual
- Profiler Logs

Run on Mobile
- Can run on out-of-box mobile

On Cloud
- Requires instrumented OS/App + app src code

Bug Fix Patch
Energy Debugging

• Narrowing Down to One Entity
  – Hardware tests:
    • Source test: Test if battery is faulty.
    • Component Test: For each component testing tools are designed.
  – History based diagnosis for software: In either OS or application. Take snapshot of phone on daily basis, when ebug is detected, EDB takes snapshot and calculates diff.

• Narrowing Down to Software Component
  – Eprof
  – Eoprof
Conclusion

• Need for fully automatic tools to detect the root causes of ebugs has been barely explored.
• By static and dynamic analysis software bugs discussed above can be discovered.
• Further a roadmap towards developing a systematic diagnosing framework for treating these ebugs was proposed in this paper.
References