Mobile Energy

Special Topics in Mobile Systems (FC5260)

Instructor: Venkat Padmanabhan

Note: includes slides generously made available by the authors of the papers being discussed

This Lecture: Energy

- Papers to be critiqued:
  - "*An Analysis of Power Consumption in a Smartphone*", Usenix ATC 2010
  - "*Fine Grained Energy Accounting on Smartphones with Eprof*", EuroSys 2012
- Other papers to read:
  - "*Carat: Collaborative Energy Debugging for Mobile Devices*", Usenix HotDep 2012
  - "*Who Killed My Battery: Analyzing Mobile Browser Energy Consumption*", WWW 2012

Top-Down View

Apps (Carat)

Modules (Browser)

Subroutines (E-Prof)

Hardware Components

NICTA

NICTA

An Analysis of Power Consumption in a Smartphone

Aaron Carroll and Gernot Heiser

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Problem

- Where and how is power consumed in a smartphone?

- Approach: fine-grained instrumentation of a real device

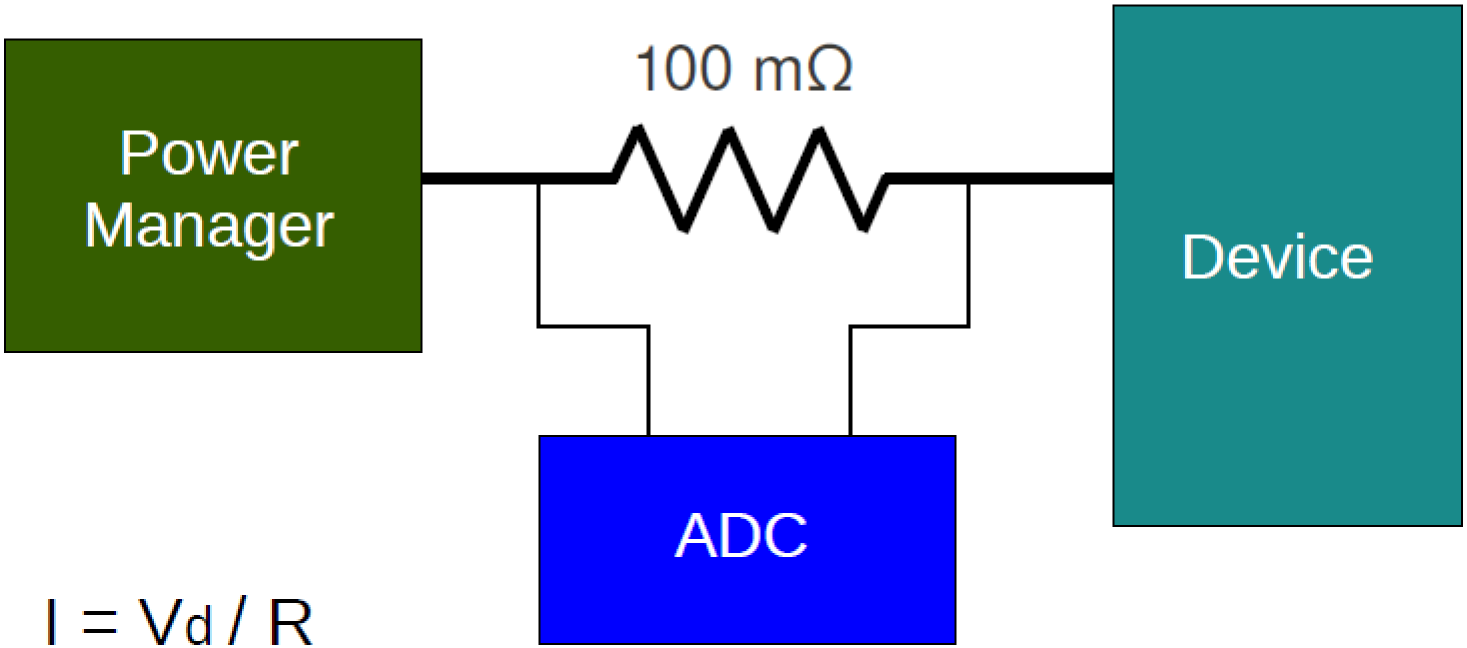
Methodology

NICTA

- OpenMoko Freerunner
  - 2.5G smartphone, c. 2008
  - 400 MHz ARM9
  - Lacking camera, 3G modem

  - Open design
  - Amenable to power instrumentation

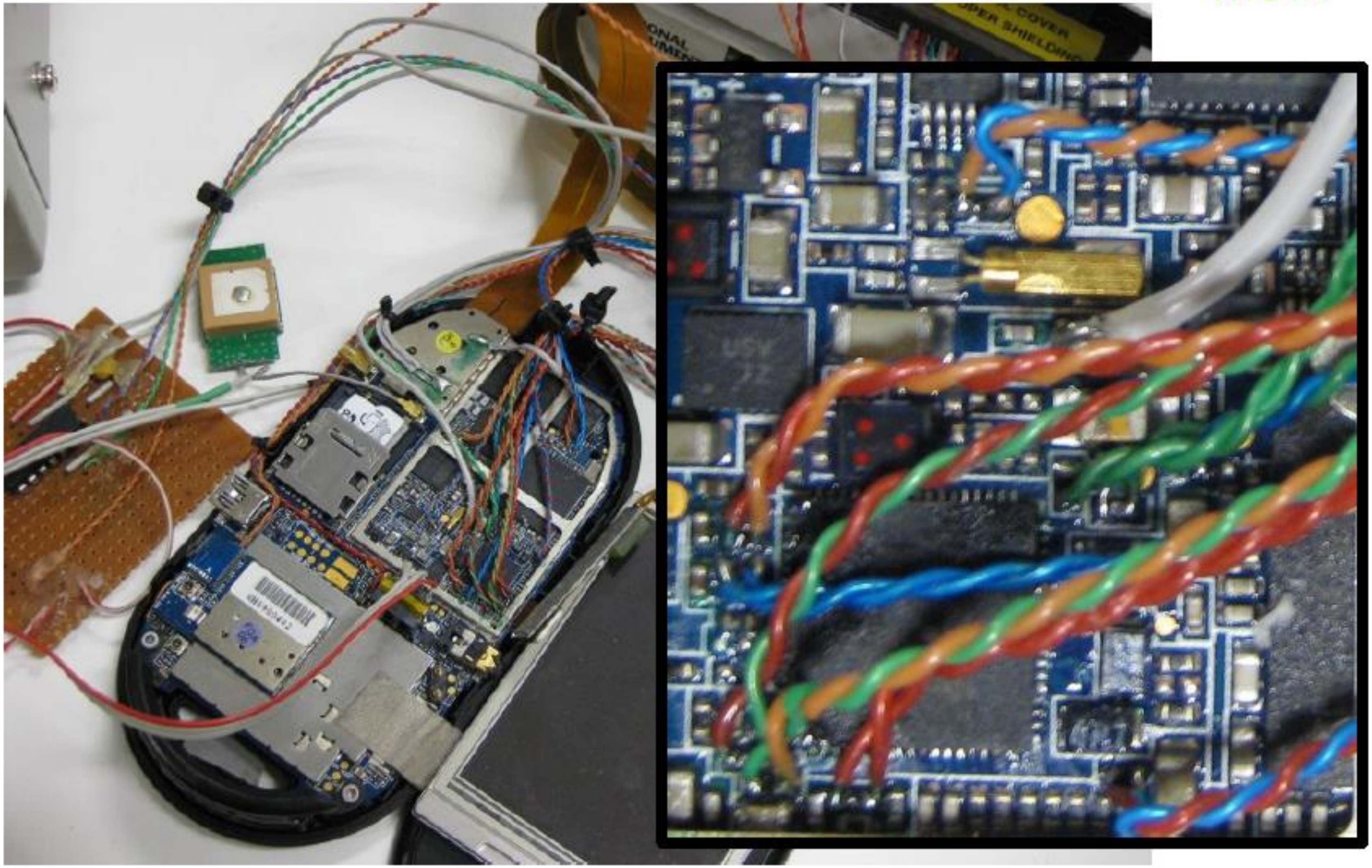
Methodology



$I = V_d / R$

$P = IV$

Methodology



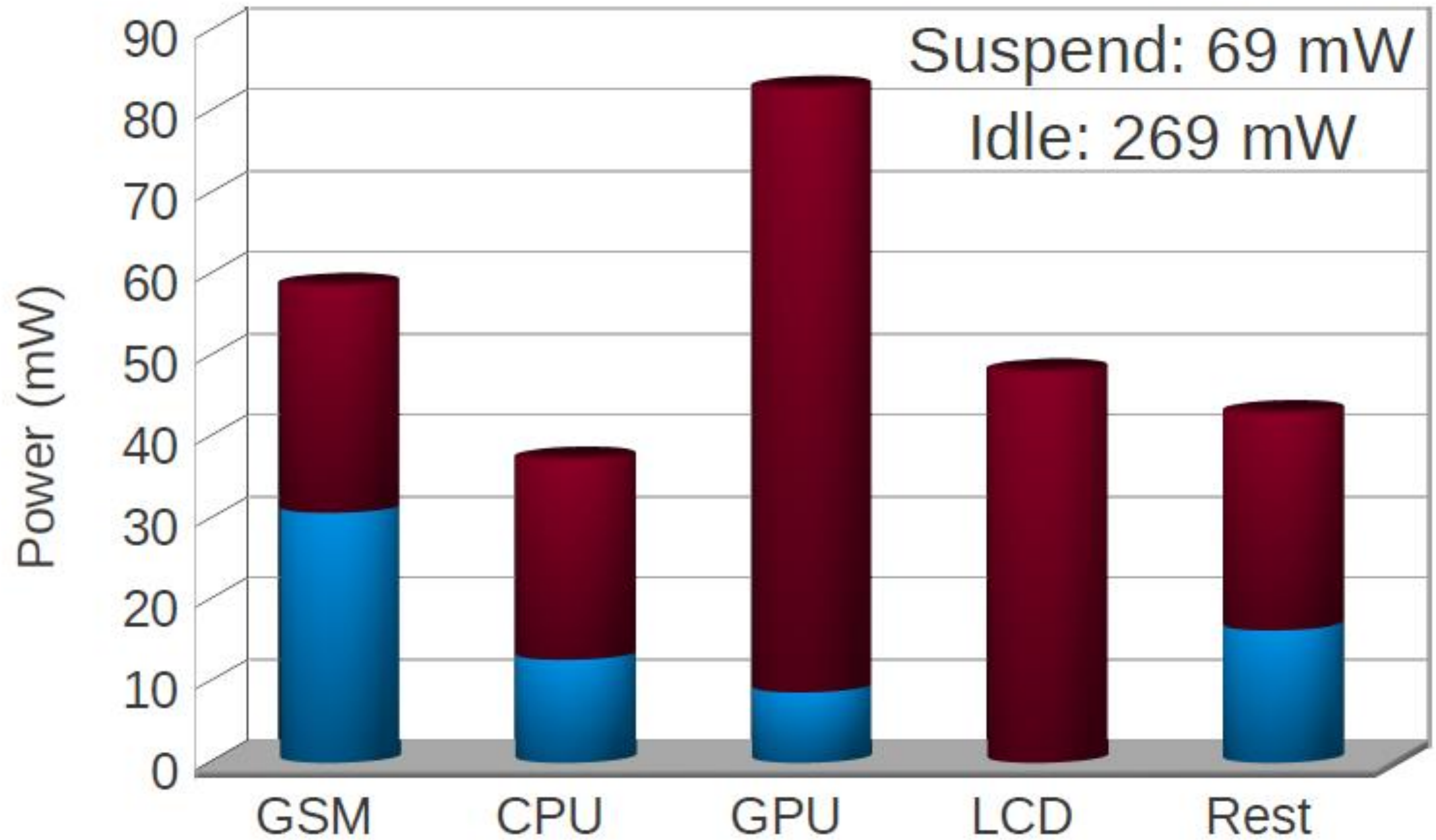
Methodology

- Instrumented components
  - CPU
  - RAM
  - GSM
  - GPS
  - Bluetooth
  - LCD panel
  - WiFi
  - Backlight
  - Audio codec
  - Amplifier
  - NAND flash
  - SD card

Benchmarks

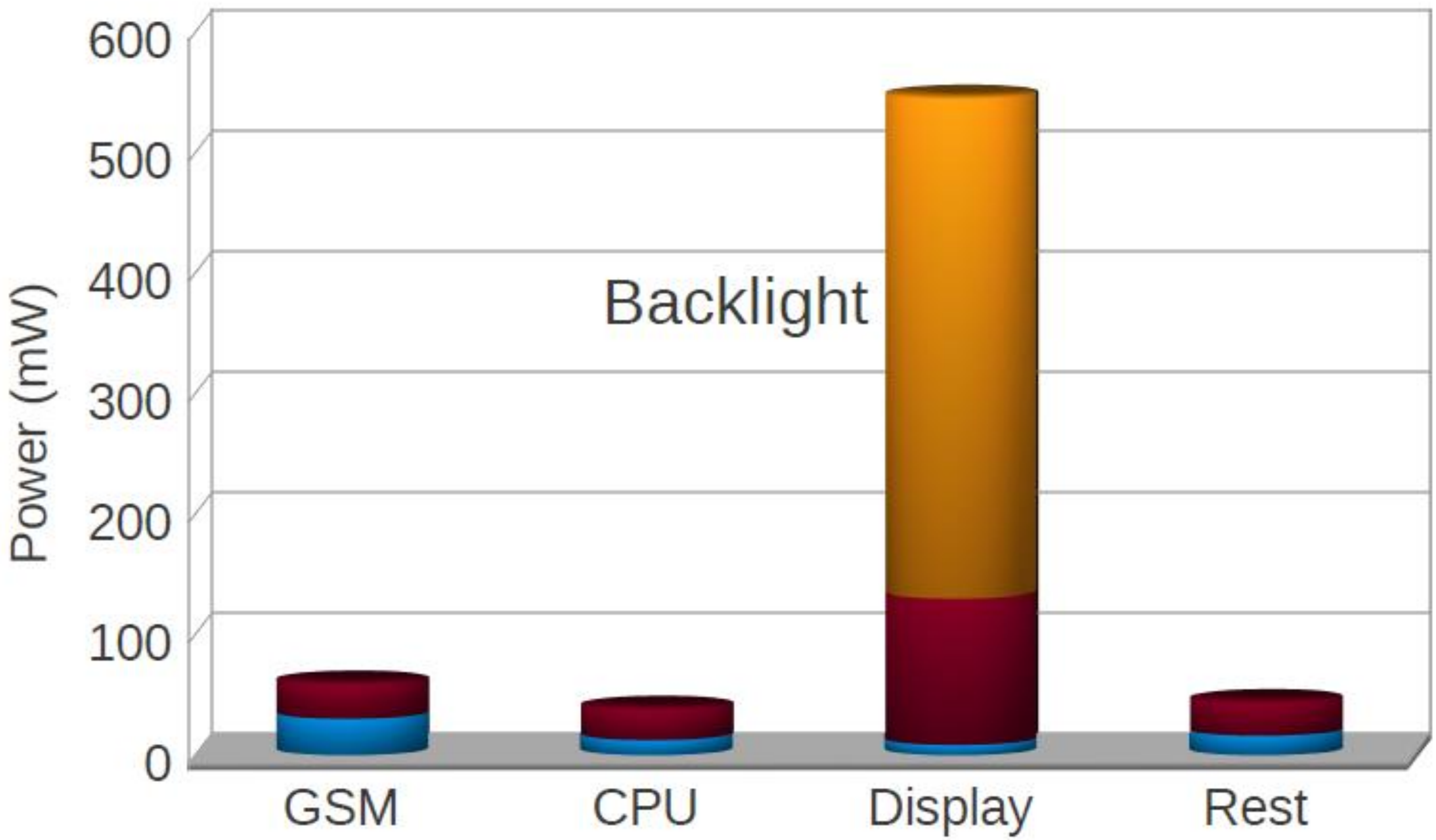
- Micro-benchmarks
  - Suspend
  - Idle
  - Backlight
  - CPU/RAM
  - Flash storage
  - Network
  - GPS
- Usage scenarios
  - Audio
  - Video
  - SMS
  - Email
  - Web
  - Call

Idle Power

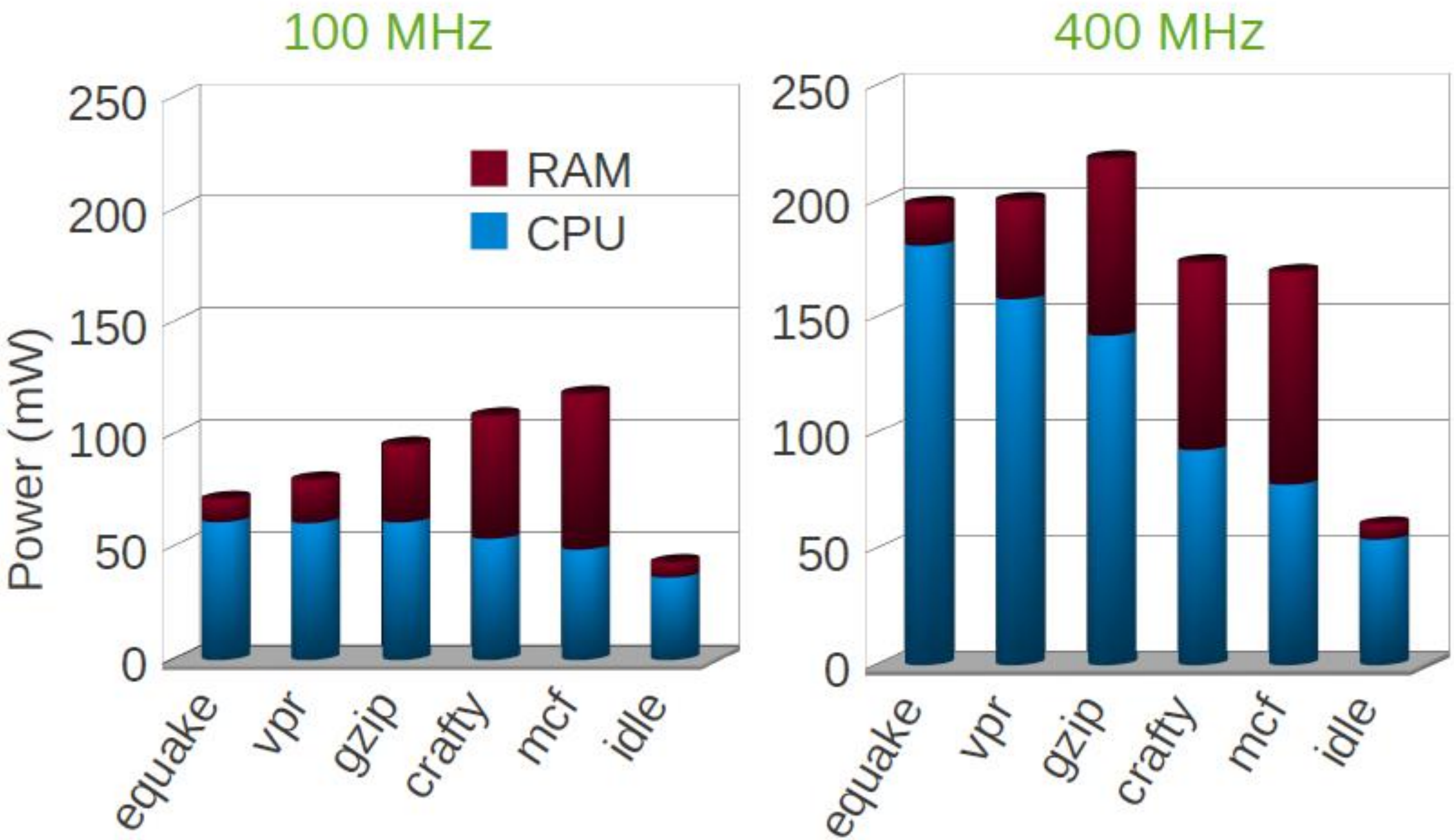


Suspend: 69 mW
Idle: 269 mW
Power (mW)
0
10
20
30
40
50
60
70
80
90
GSM
CPU
GPU
LCD
Rest

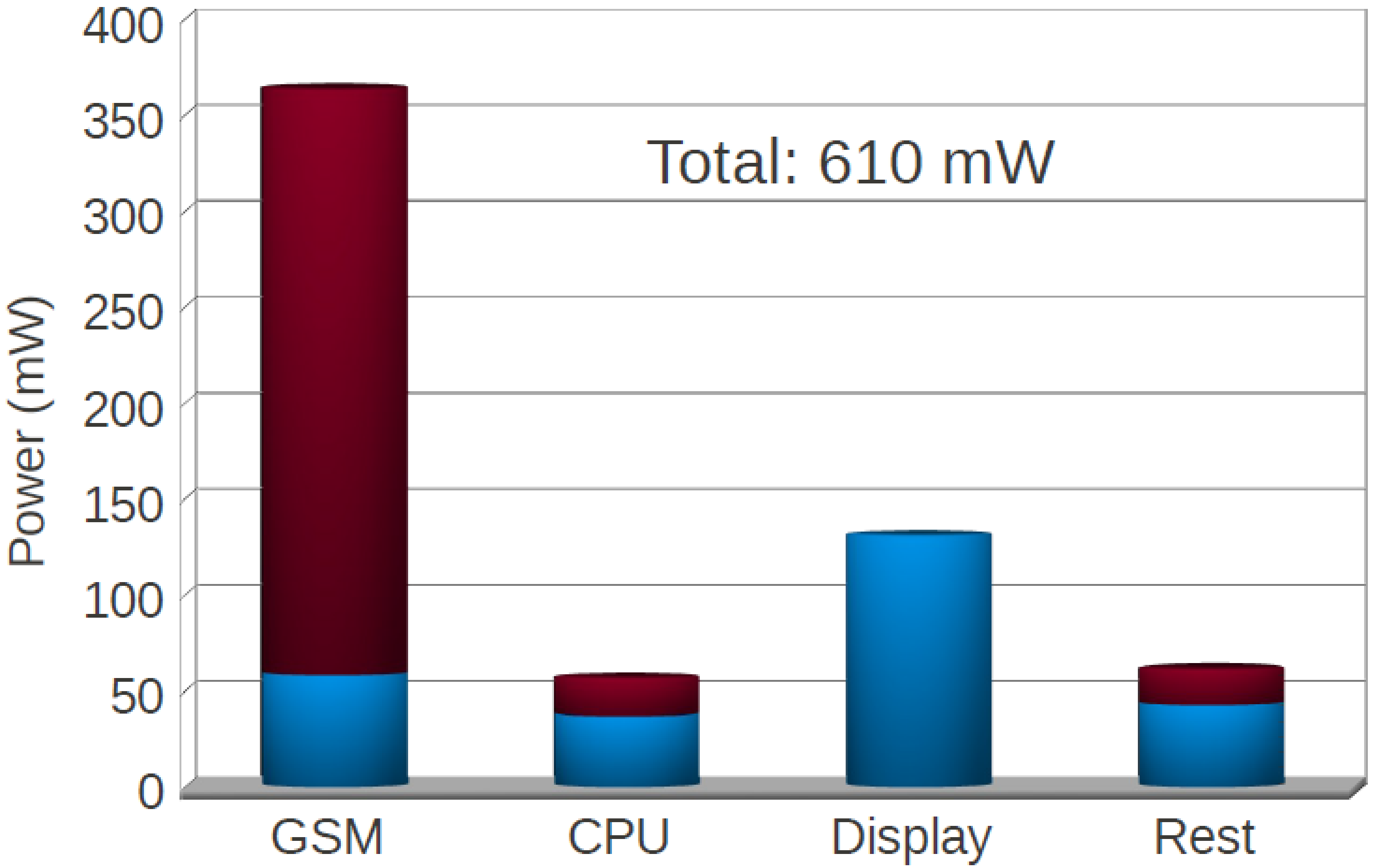
Display Power



CPU and RAM

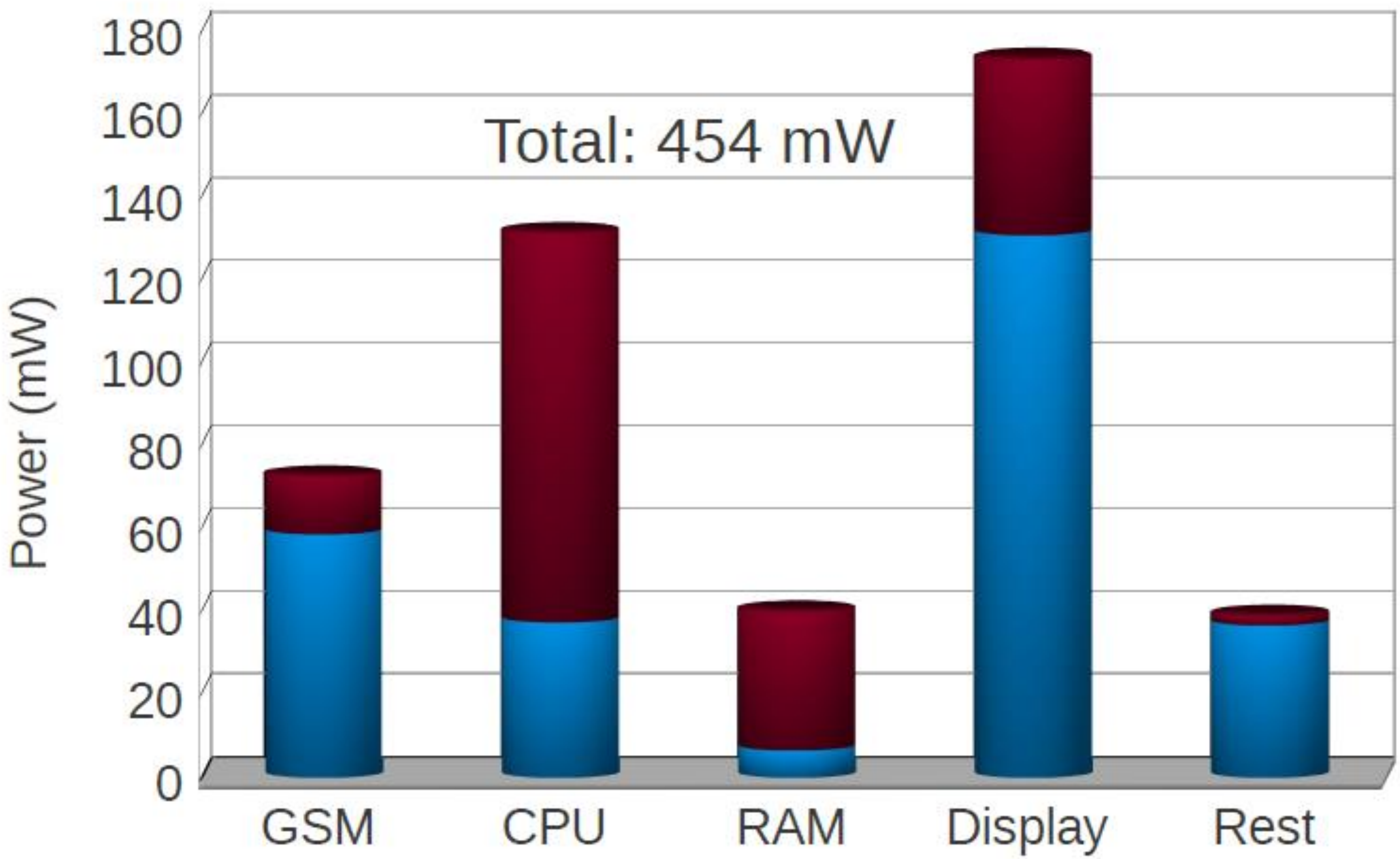

100 MHz
400 MHz
Power (mW)
250
200
150
100
50
0
RAM
CPU
equake
vpr
gzip
crafty
mcf
idle

Email



Power (mW)
400
350
300
250
200
150
100
50
0
Total: 610 mW
GSM
CPU
Display
Rest

Video

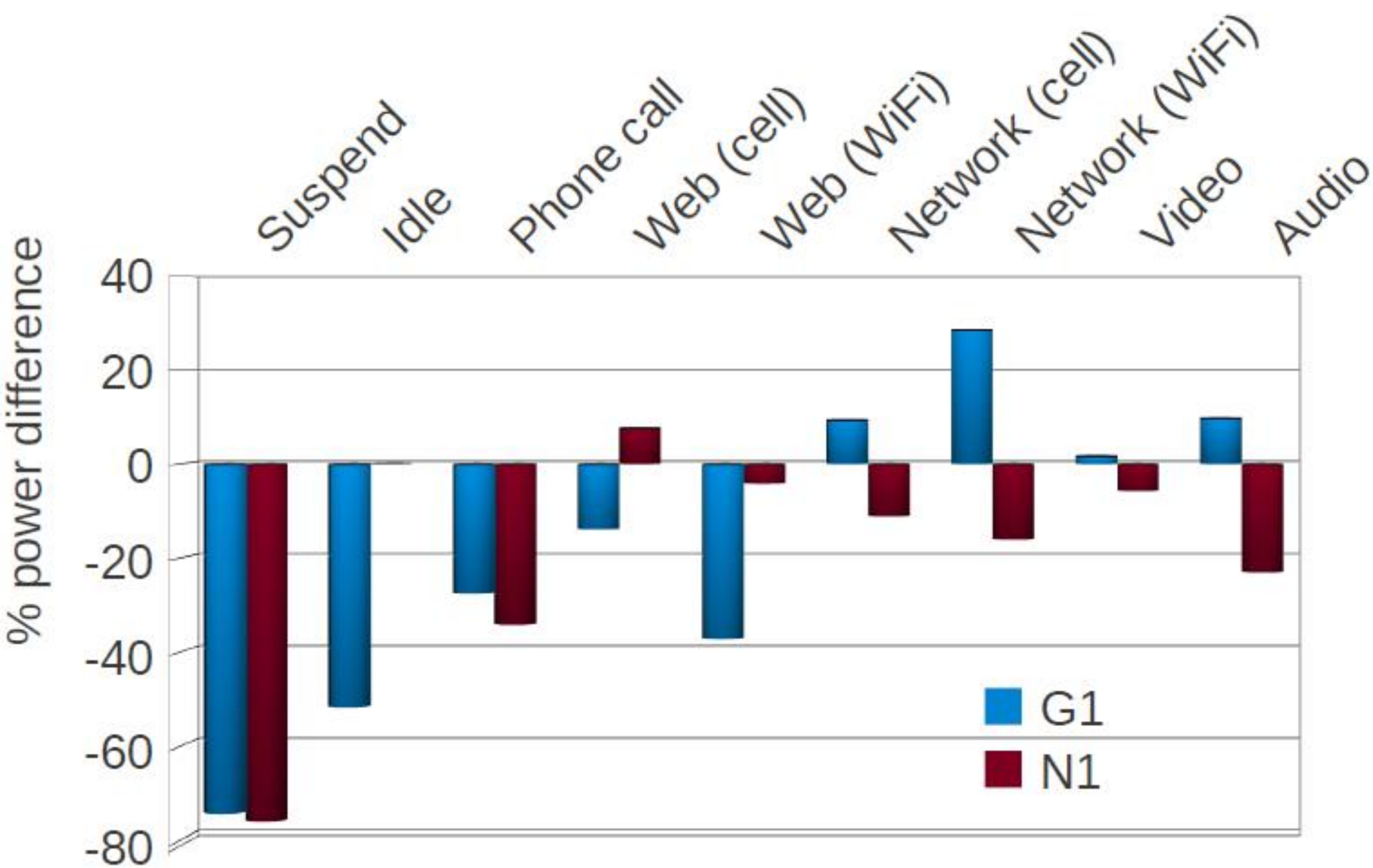


Total: 454 mW
Power (mW)
180
160
140
120
100
80
60
40
20
0
GSM
CPU
RAM
Display
Rest

Validation

- Benchmarks repeated on two devices:
  - HTC Dream (G1)
  - Google Nexus One (N1)
- Total system power only
- 3-4 years of mobile technology

Validation



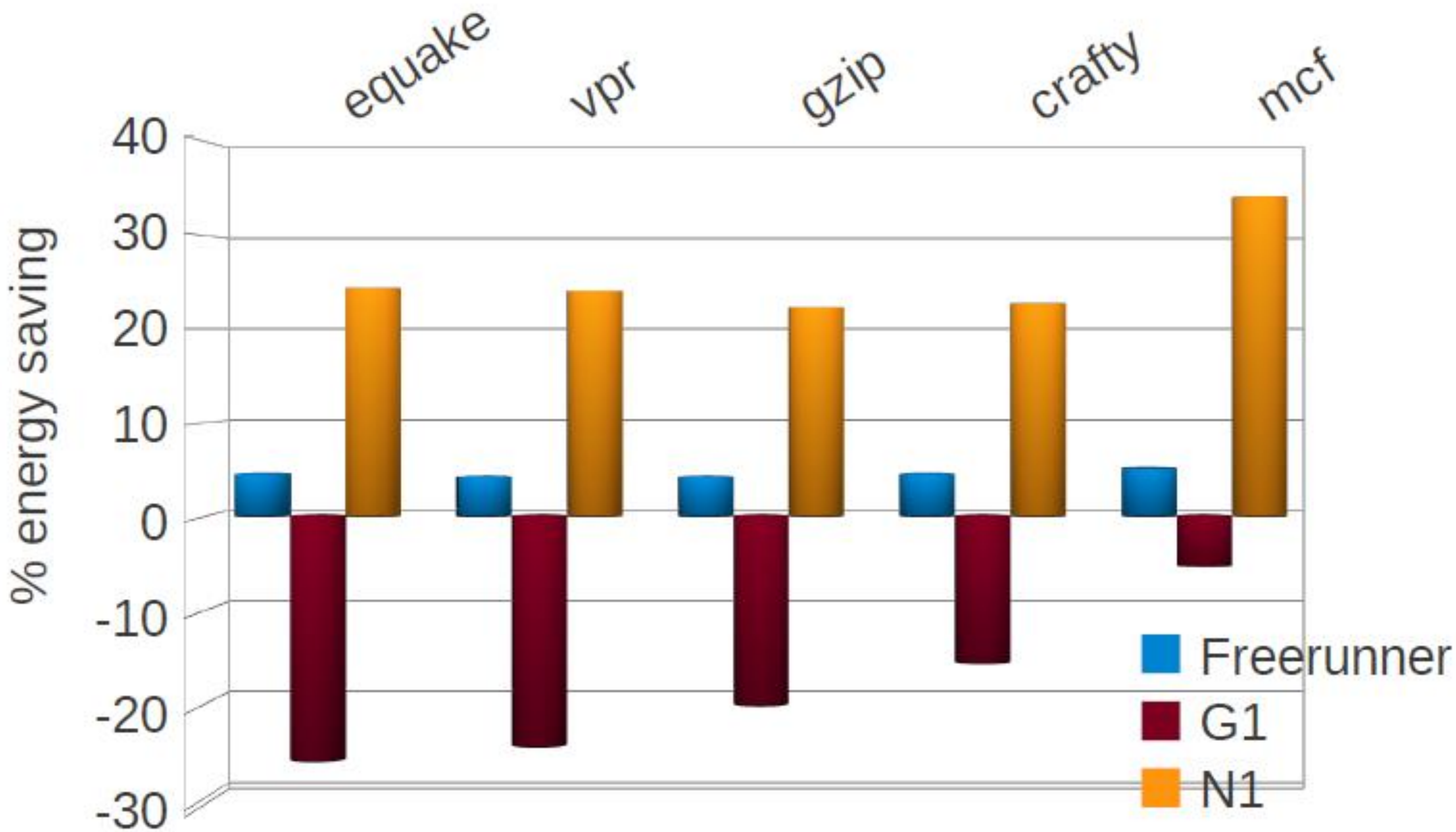
% power difference
40
20
0
-20
-40
-60
-80
Suspend
Idle
Phone call
Web (cell)
Web (WiFi)
Network (cell)
Network (WiFi)
Video
Audio
G1
N1

DVFS

- Dynamic Voltage and Frequency Scaling
- DVFS reduces power

  … but does it reduce energy?

DVFS



equake
vpr
gzip
crafty
mcf
40
30
20
10
0
-10
-20
-30
% energy saving
Freerunner
G1
N1

Conclusions

- Major consumers: display & cell radio
  - WiFi power low in most situations
- CPU can be significant
  - Future power driver
- Where power is **not** going:
  - RAM
  - Audio
  - Bluetooth
  - Storage

Conclusions

- Both dynamic and static power important
- DVFS hanging on (for now)
- Networking power not increasing

Where is the Energy Spent Inside My App?
Fine Grained Energy Accounting on Smartphones with eprof

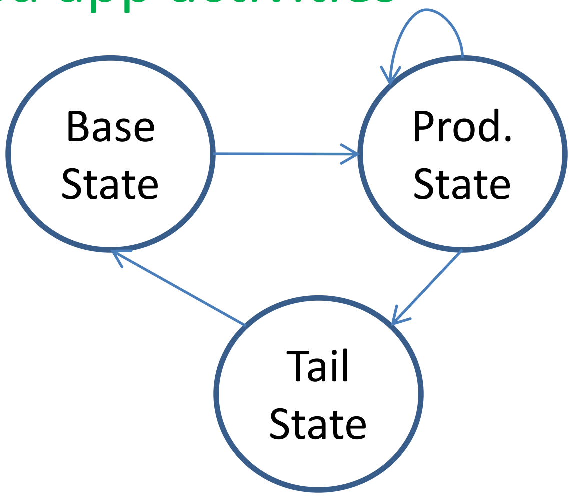
Abhinav Pathak
Y. Charlie Hu
Purdue University

Ming Zhang
Microsoft Research

Tracking Power Activities
Power Modeling

- State-of-art ‘utilization based’ power models are inaccurate on smartphones
  - Only active utilization => power consumption
  - Energy is consumed linearly w.r.t utilization

  - Hard to map power triggers to fine grained app activities



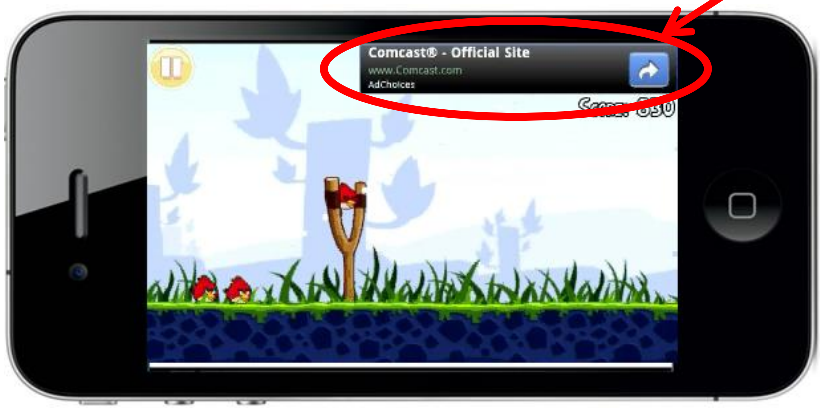
- System call triggered FSM based fine-grained power model [Eurosys ‘11]
  - Use system calls as power triggers
  - System calls drive Finite-State-Machine

Tracking App Activities

- Granularity of Energy Accounting

Multi Threading:

Third Party Ad Module



Multiple Routines:

Collect information
Upload information
Download ads

Multi Processing:

Tracking App Activities

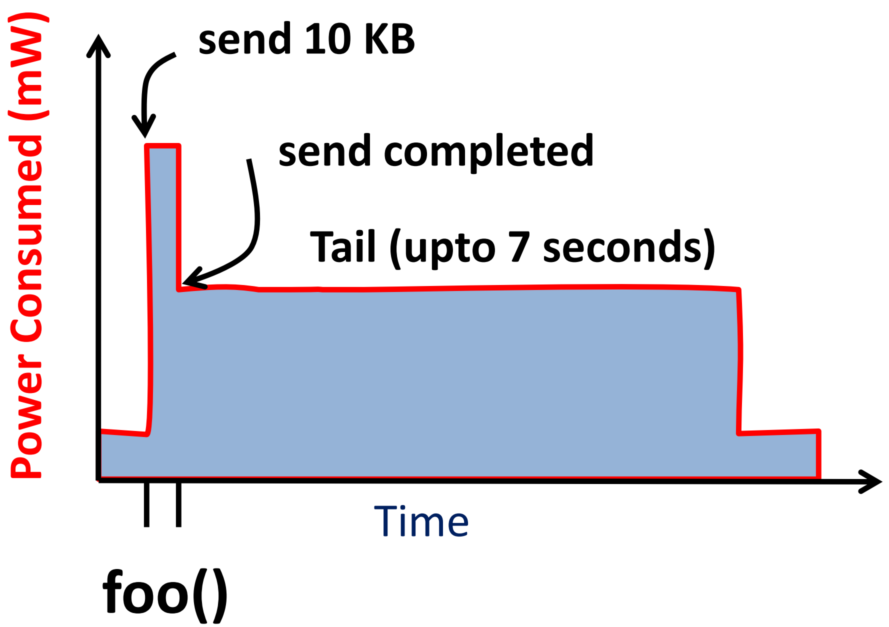
- Granularity of Energy Accounting
  - eprof supports per Process/Thread/Routine granularity
- I/O Devices
  - Track system call to program entity
    - Process – getpid()
    - Thread – gettid()
    - Routine – backtrace()
- CPU
  - Just like gprof [PLDI '82]
  - Periodic sampling of routine call stack

Lingering Energy Consumption
(a) Tail Energy

Effect on power/energy consumed by a component because of an activity lasts beyond the end of the activity

Components with tail:

- **Sdcard**
- **3G**
- **WiFi**
- **GPS**

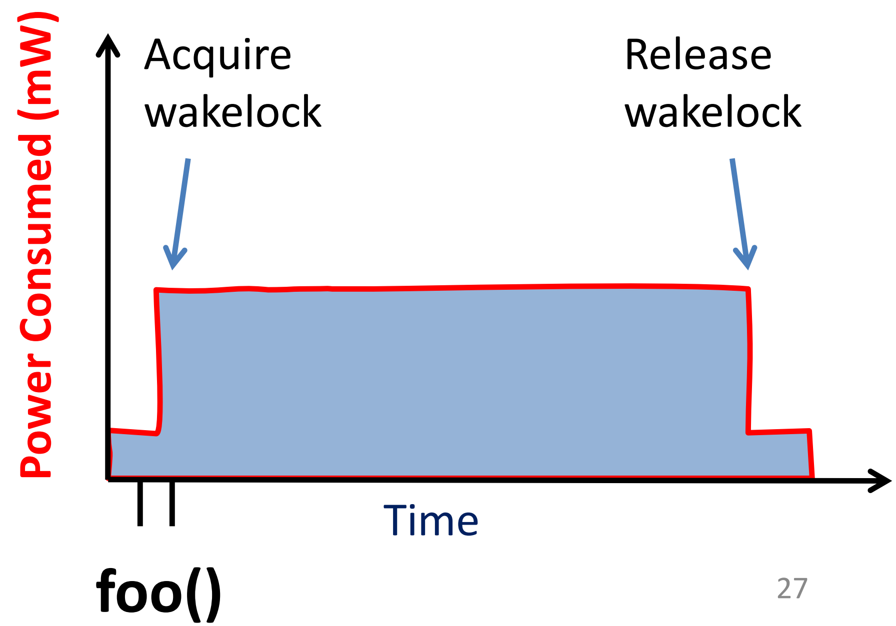


Lingering Energy Consumption
(b) Persistent State Wakelock

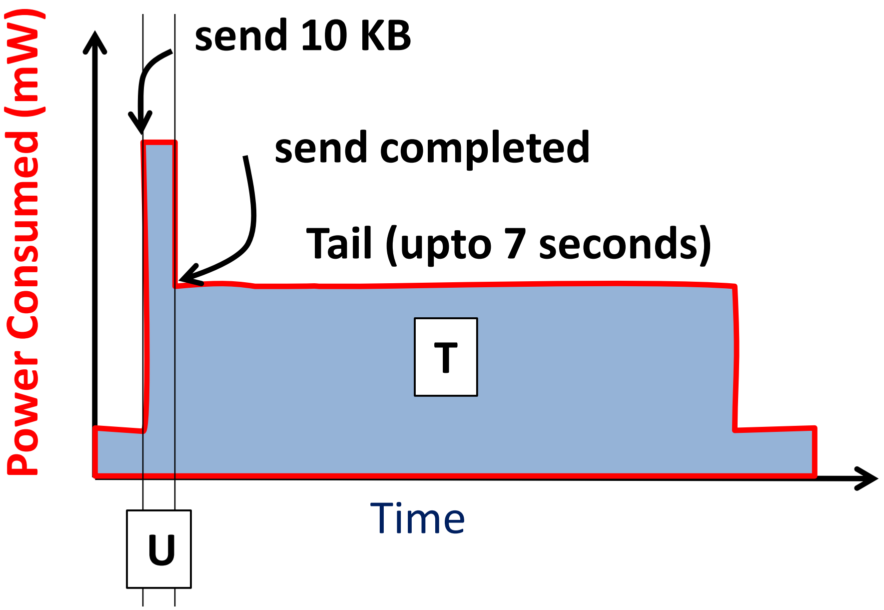
- Aggressive Sleeping Policies: Smartphone OSes freeze system after brief inactivity

- Power encumbered Programming: Programmer has to manage sleep/wake cycle of components

Keep the screen on !

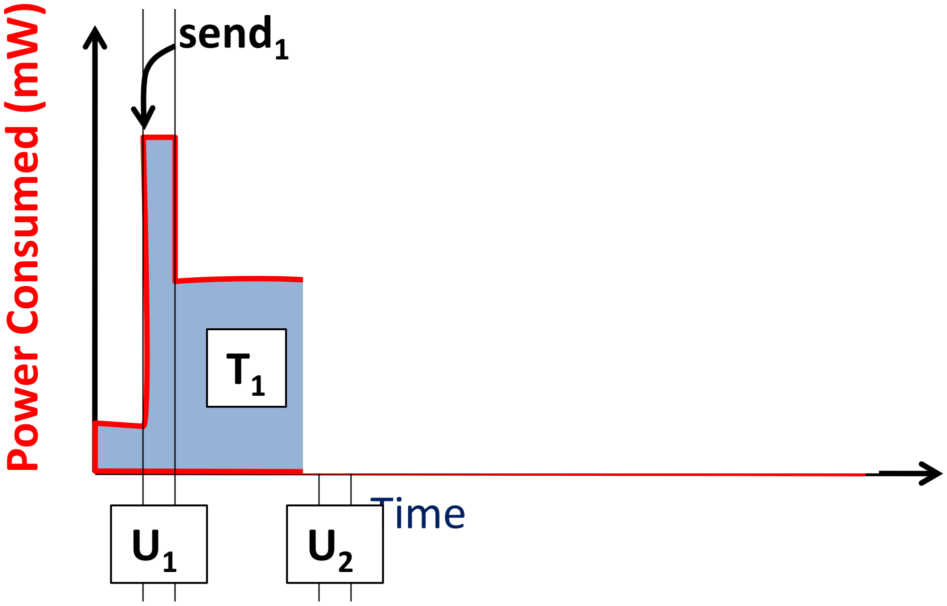


Lingering Energy Consumption
Case 1: Single Call Single Tail



1. Energy represented in terms of an energy tuple **(U, T)**
2. (U, T) is attributed to entity (s) containing send system call

Lingering Energy Consumption
Case 2: Multiple Calls Multiple Tails

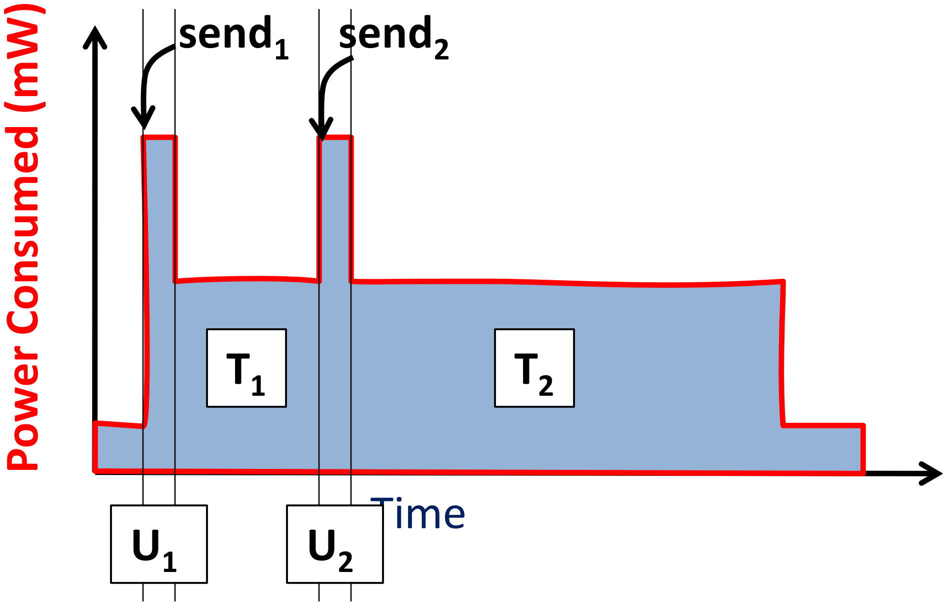


How to split tail T_2 among?

Average Policy: Split tail energy T_2 in weighted ratio

1. Not easy to define weights
2. Policy gets complicated in presence of multiple system calls

Lingering Energy Consumption
Case 2: Multiple Calls Multiple Tails



***Last-Trigger-Policy*:** Assign asynchronous (tail) energy to the last active system call

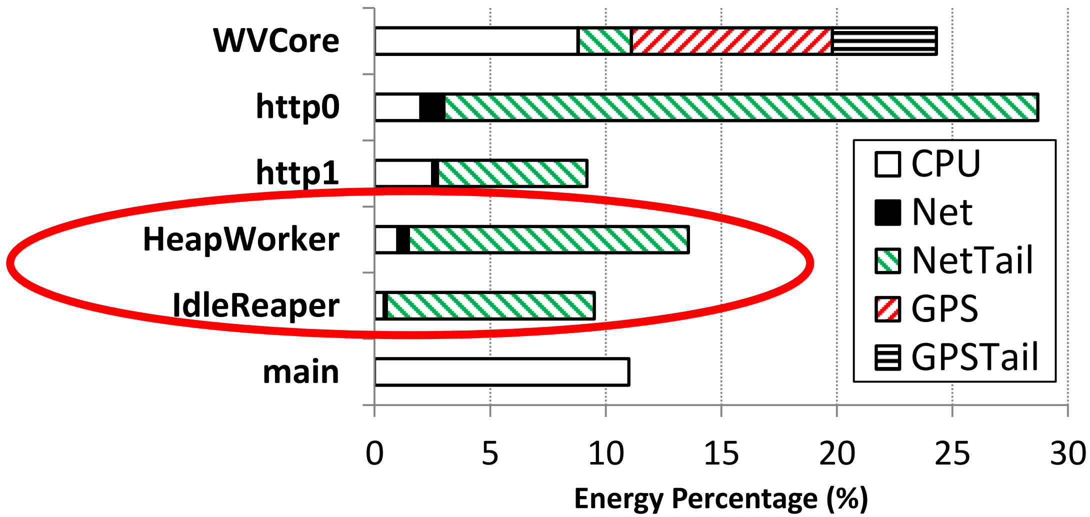
send$_1$: (U_1, T_1)
send$_2$: (U_2, T_2)

1. ~~Not easy to define weights~~
2. ~~Policy gets complicated in presence of multiple system calls~~

eprof System
(Android and Windows Mobile)

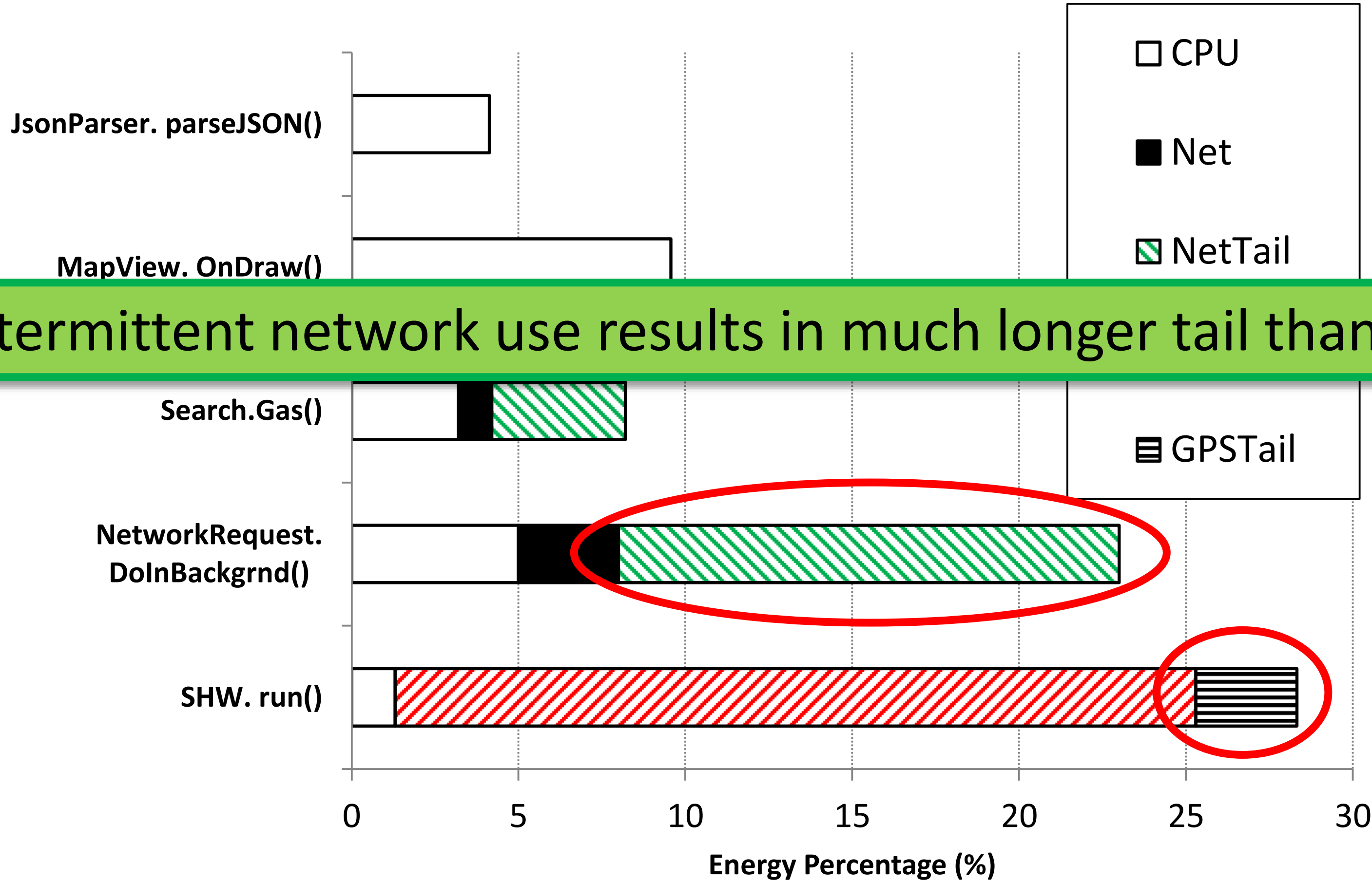
Logging Overhead:
2-15% Run Time,
1-13% Run Energy

Case Studies: (a) Android Browser Google Search



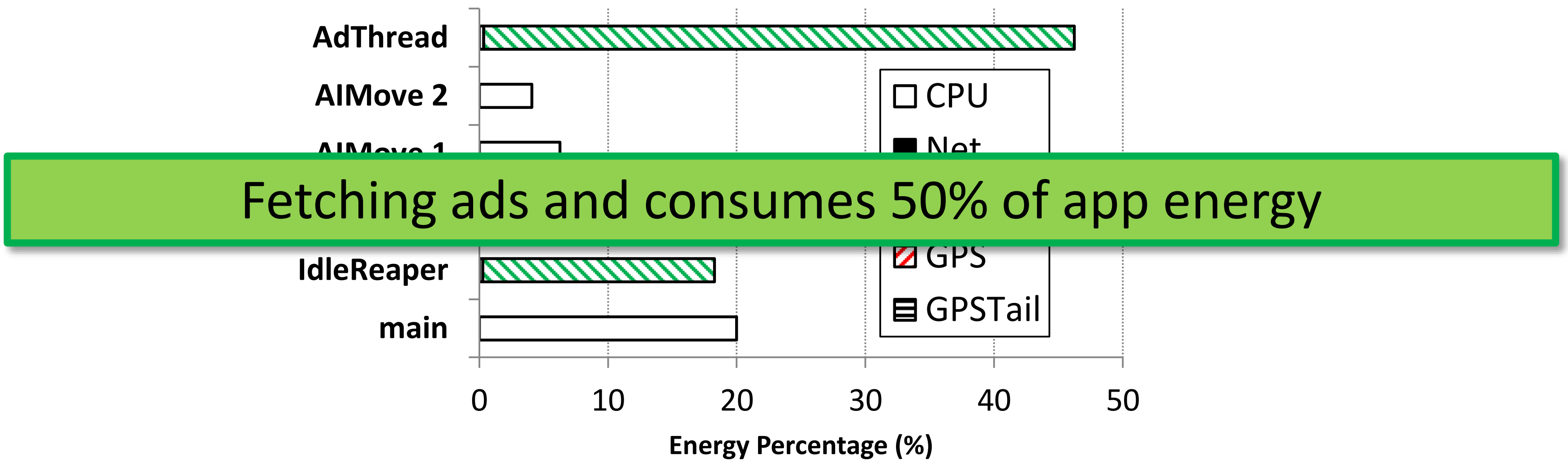
Activity	Energy %
HTTP	38%
TCP Conditioning	25%
User Tracking	16%
GUI Rendering	5%

Case Studies: (b) Map Quest



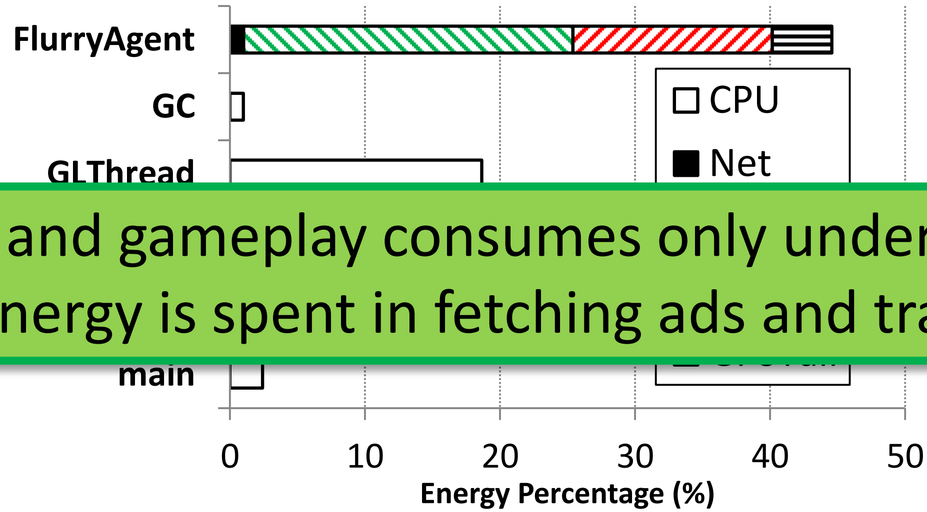
Intermittent network use results in much longer tail than GPS

Case Studies: (c) Free Chess



Activity	Energy %
Advertisement	50%
GUI Rendering	20%
AI	20%
Screen Touch Events	2%

Case Studies: (d) Angry Birds



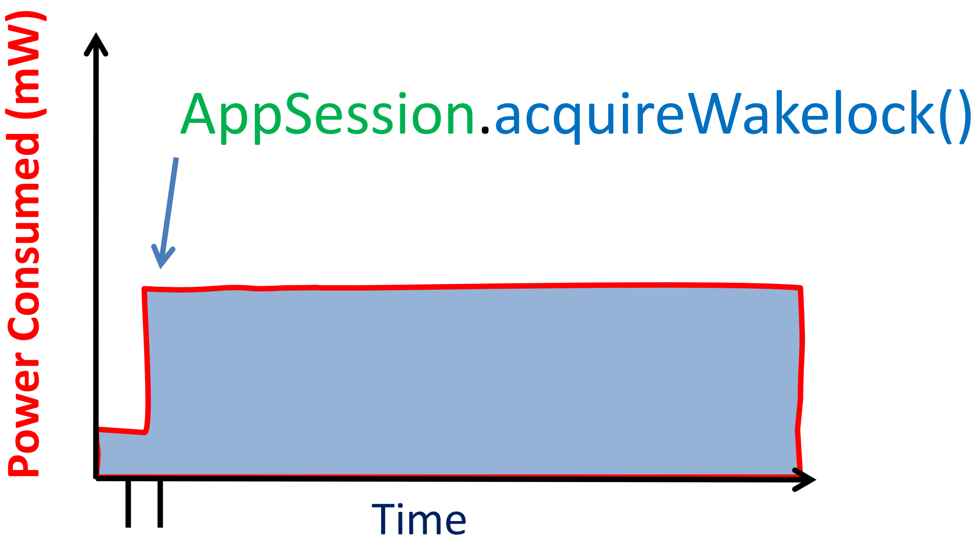
Rendering and gameplay consumes only under 30% of energy
Rest energy is spent in fetching ads and tracking user

Activity	Energy %
User Tracking	45%
TCP Conditioning	28%
Game Rendering	20%

Case Studies (e): Facebook Wakelock Bug

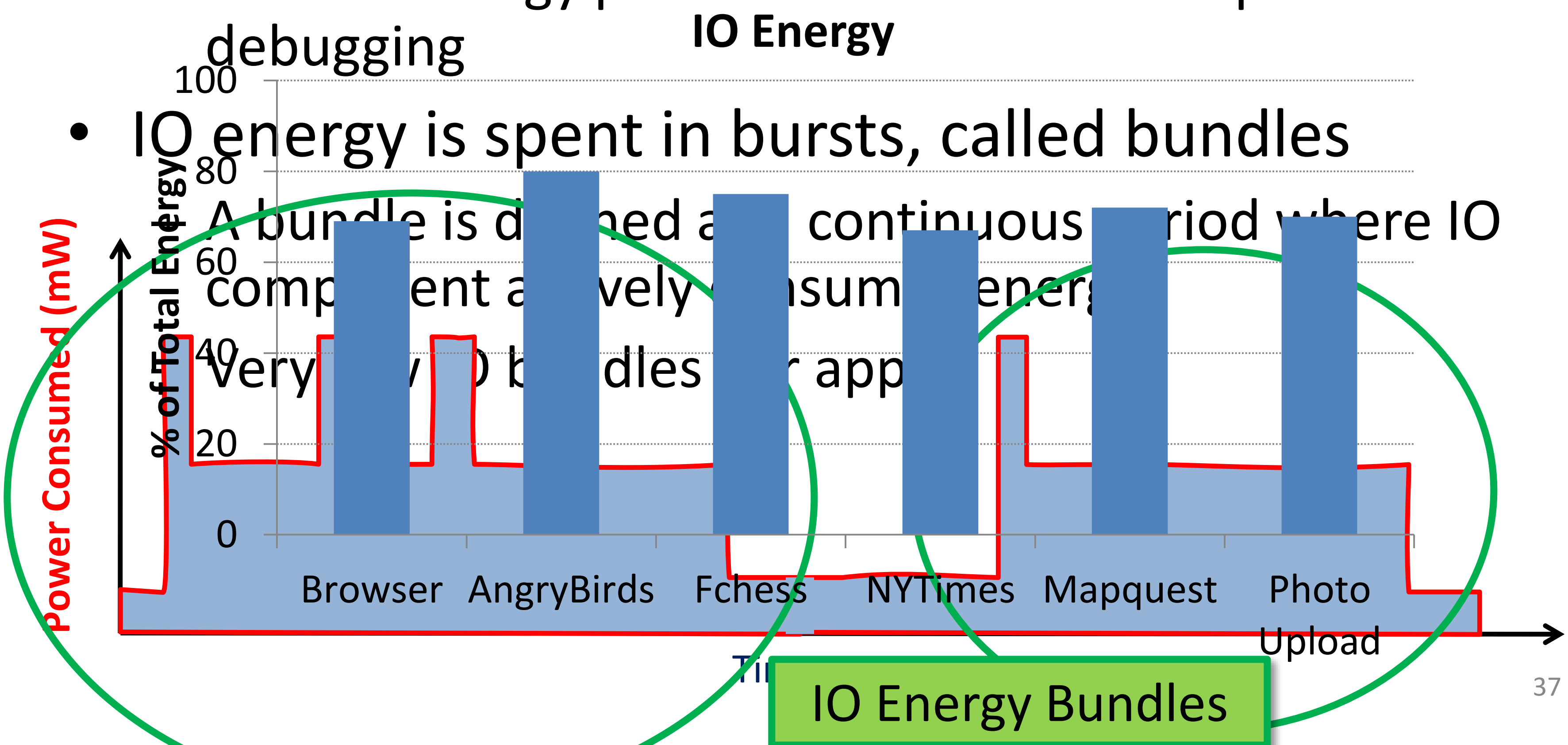
Google Nexus (WiFi)

FaceBookService: 25%

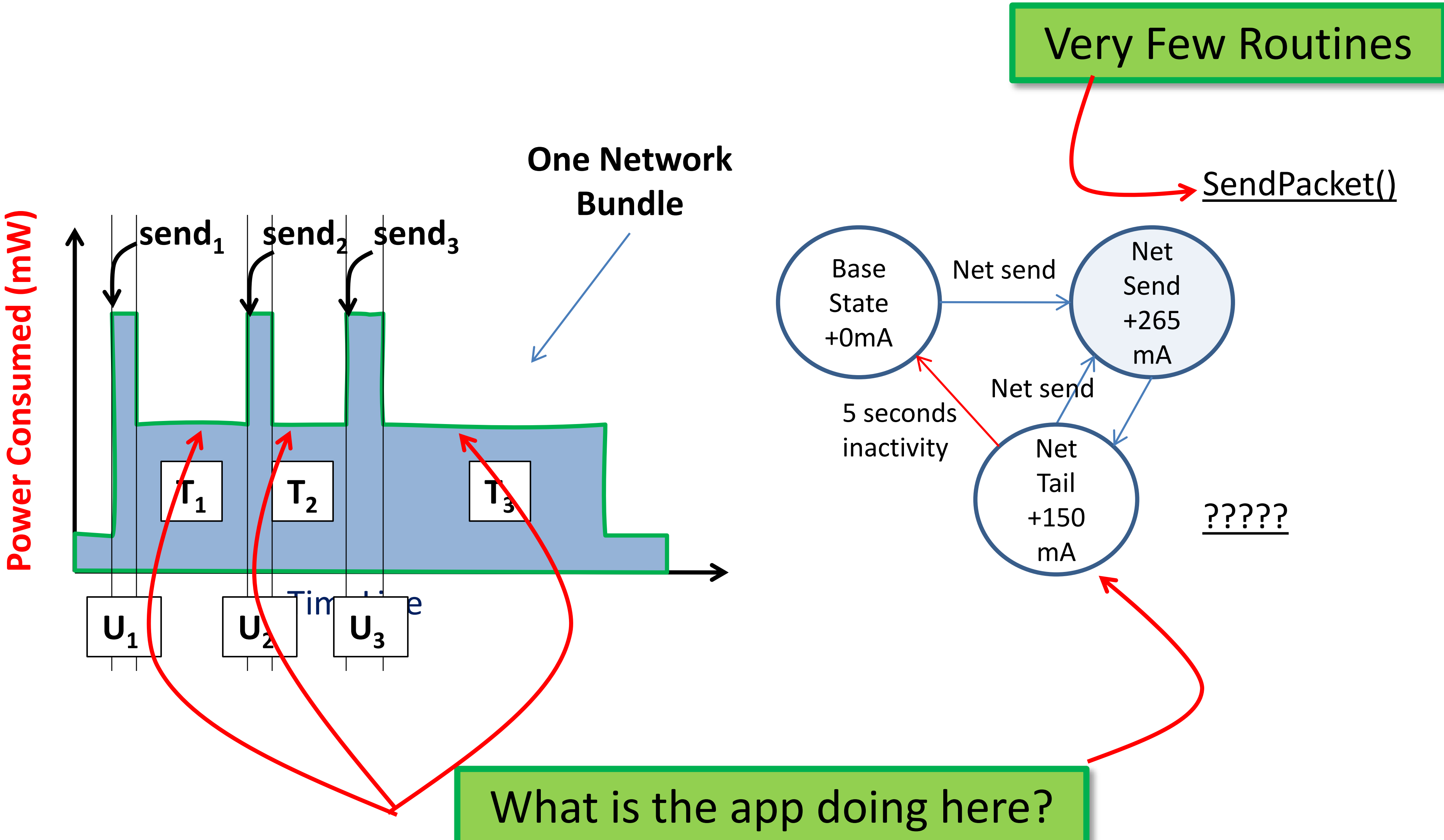


App Energy Drain Characteristics

- IO consumes the most energy
  - Most apps spent 50% - 90% of their energy in IO
  - A linear energy presentation does not help with debugging
- IO energy is spent in bursts, called bundles
  - A bundle is d… ned a… continuous … riod where IO comp… ent a… vely … nsum… energ…
  - Very … O b… dles … r app…

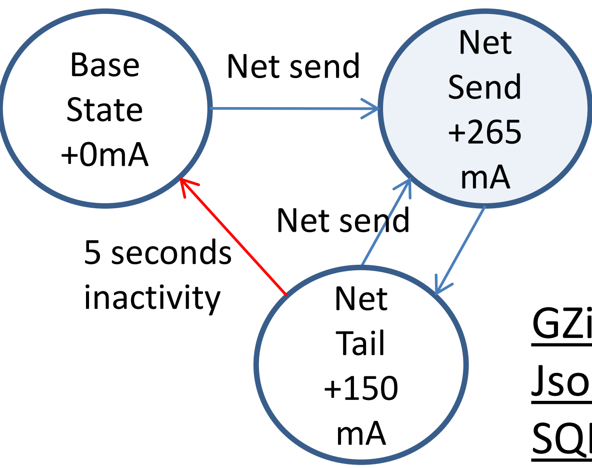


IO Bundle Representation



Optimizing IO Energy using Bundles

Why is a bundle so long?



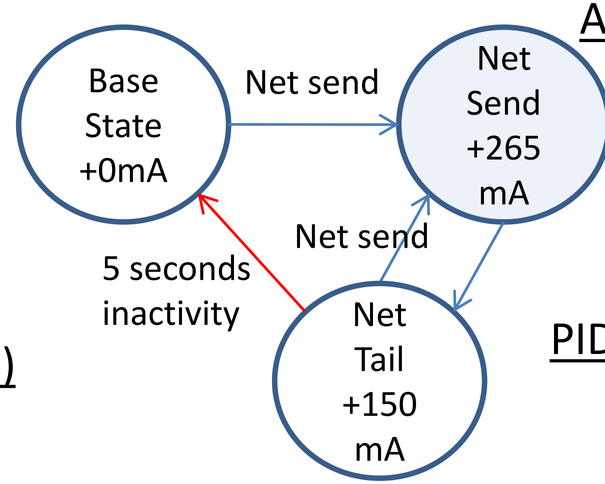
DownloadMgr.
Read()

18

GZipStream.read()
JsonNode.Deserialize()
SQLiteDB.insert()

The New York Times

Why are there so many bundles?



AdWhirl.FetchAd()

PID 0 (null loop)

Reduced energy consumption of 4 apps by 20-65% by minimizing number of bundles and reducing bundle lengths

Conclusion

- eprof: fine-grained energy profiler
  - Enables opportunities for in-depth study of app energy consumption
- Case studies of popular apps energy consumption
  - 65-75% of app energy spent in tracking user and fetching ads (for example, angrybirds)
- Bundles: IO energy representation
  - Helps debugging smartphone app energy

Top-Down View

- Apps (Carat)
- Modules (Browser)
- Subroutines (E-Prof)
- Hardware Components

Deutsche Telekom, Inc.
Silicon Valley Innovation Center

Who Killed My Battery: Analyzing Mobile Browser Energy Consumption

Narendran Thiagarajan[1], Gaurav Aggarwal[1], Angela Nicoara[2]
Dan Boneh[1], Jatinder Pal Singh[3]

[1]Department of Computer Science, Stanford University, CA
[2]Deutsche Telekom Innovation Labs, Silicon Valley Innovation Center, CA
[3]Department of Electrical Engineering, Stanford University, CA

April 18, 2012

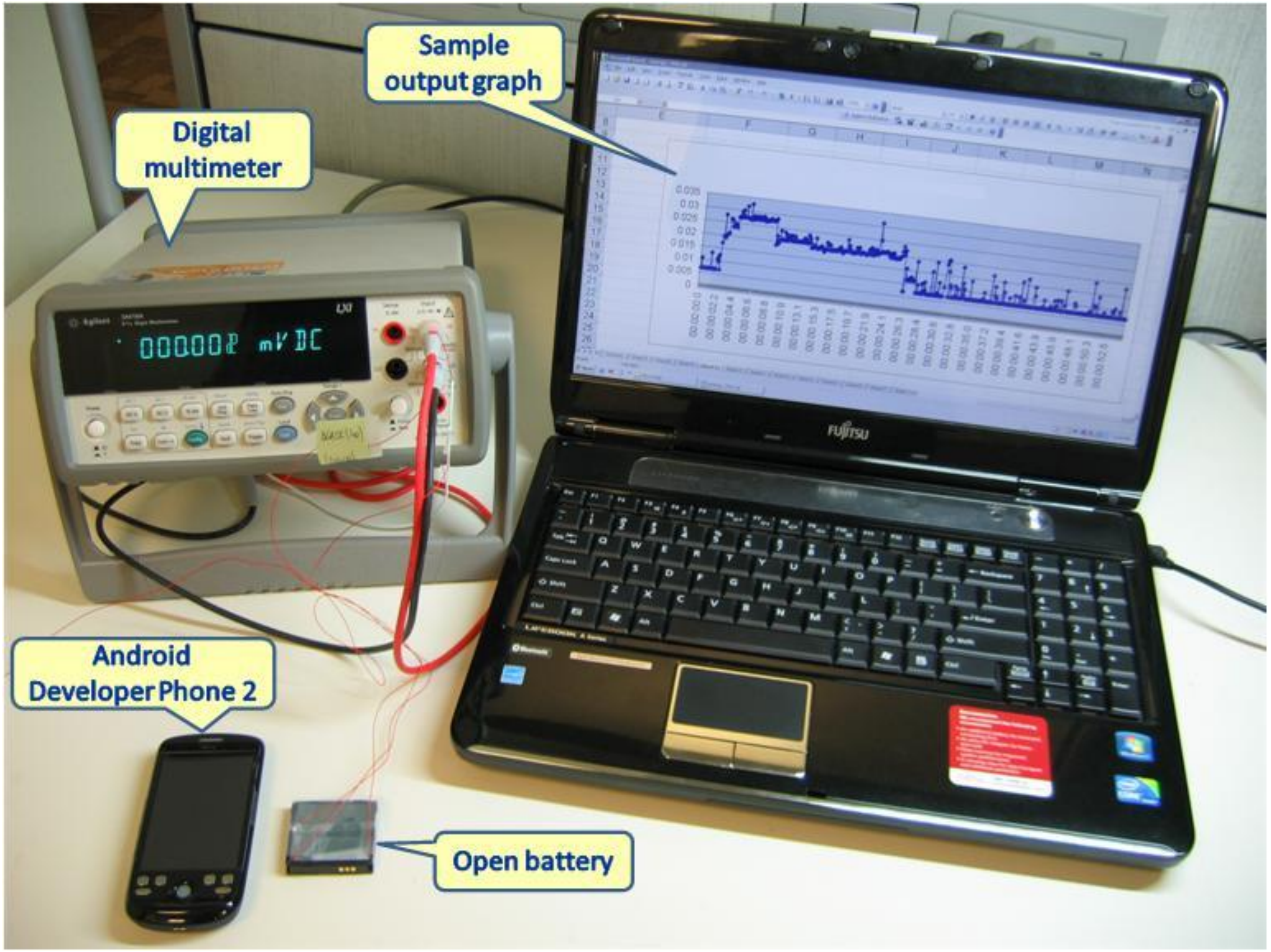
A Software Infrastructure for Measuring the Precise Energy Used by a Mobile Browser

Challenge: How much energy does the phone use to render a particular web page?

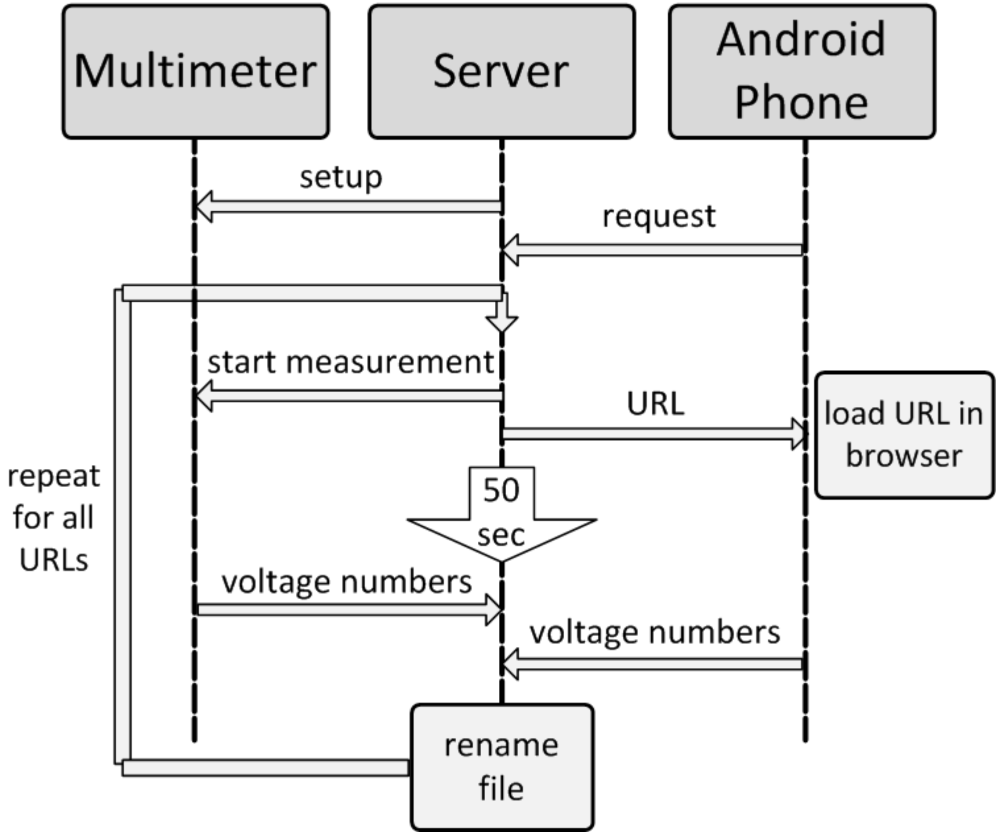
Impact of the structure of web pages on battery usage in phone browser?

How to design web pages to minimize the energy needed for rendering?

Automated Energy Measurement System



Automated Energy Measurement System



Server controls the phone and multimeter:

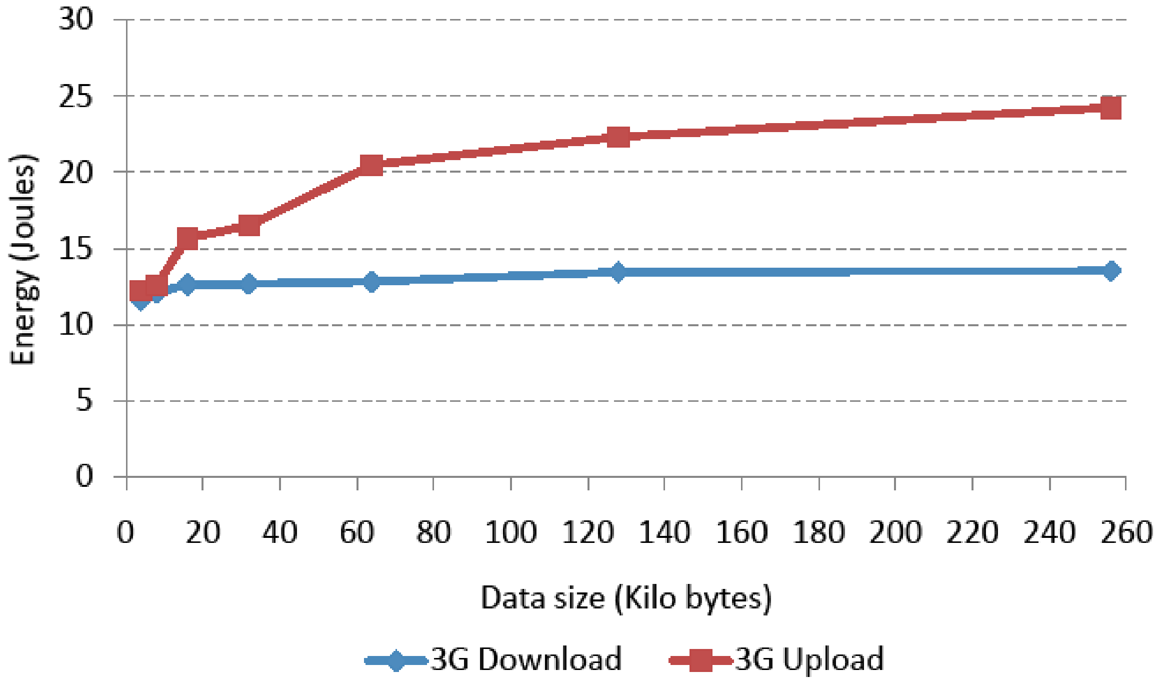
❶ Server communicates with the Browser Profiler app on the phone

❷ Server instructs the Browser Profiler app to request the running phone browser to repeatedly load a specific URL, either with or without caching

❸ Server starts the multimeter measurement

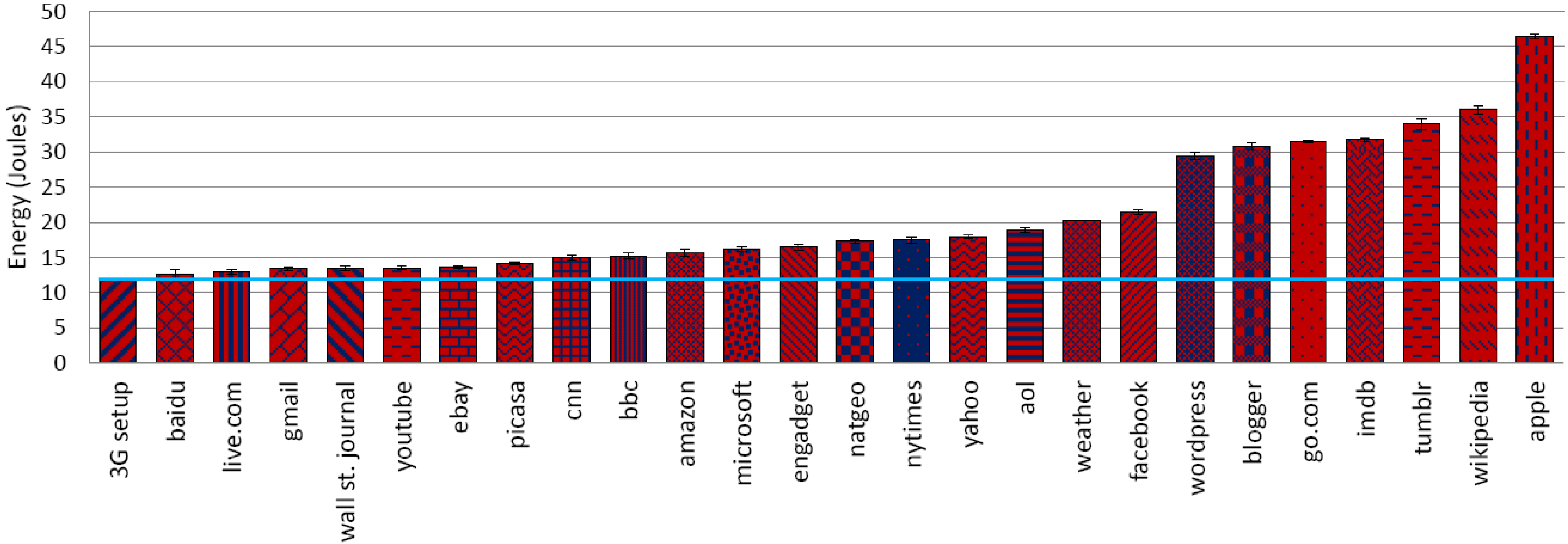
❹ All measurements recorded on the multimeter are transferred to the server for processing

Energy for Download & Upload Data over 3G



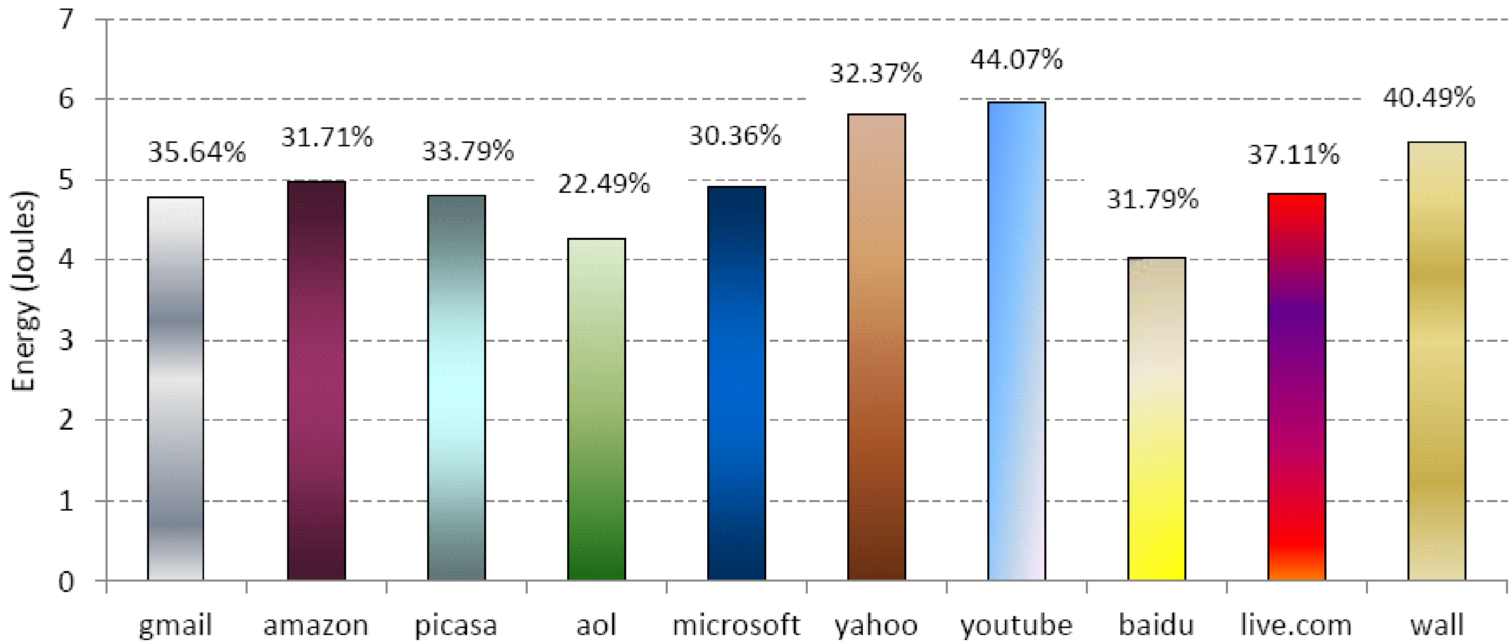
- Average energy needed for downloading & uploading 4kB to 256kB over 3G
- Setup cost of roughly 12 Joules before the first byte can be sent
- Download energy – mostly flat (up to 256kB)
- Upload energy – increases with the amount of data being uploaded

Energy Consumption of Top Web Sites



- Energy to download and render the web page
  (energy for 3G communication + parsing + rendering web page)
- Average power consumption when the browser is idle ⇨ 170 mW

Rendering Energy Consumption of Top Web Sites

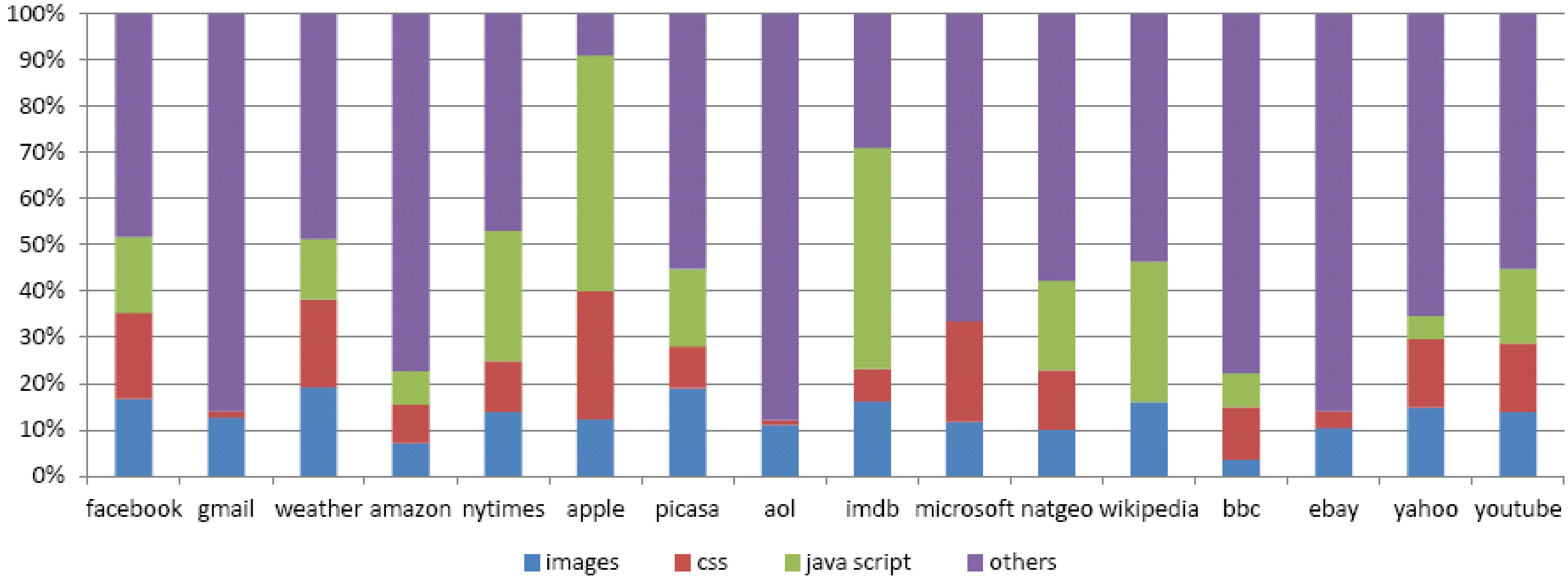


- Energy needed to parse and render the page (no energy for 3G communication)
- Energy used to render the content from local cache
- How the complexity of the web page affects the energy needed to render it
- Dynamic Javascript can greatly increase the power usage of a web page

Challenge: How much energy is used by different web elements?

STANFORD UNIVERSITY

Energy Consumption of Web Components (Transmission + Rendering)



Evaluation:

- Relative energy costs of individual web components

- **Results**:
  - ✓ CSS and Javascript – most energy consuming components in the transmission and rendering of a web site
  - "Others" – mainly includes the 3G connection setup and text rendering

Optimizing Mobile Web Pages

Reducing Javascript Power Consumption

- Javascript – one of the most energy consuming components in a web page
- Optimizations:
  - ✓ Shrinking Javascript on a mobile page to contain only functions used by the page greatly reduces energy cost

Reducing CSS Power Consumption

- Large CSS files with unused CSS rules consume more then minimum required energy
- Optimizations:
  - ✓ CSS should be web page specific and contain only the rules required by the elements in the web page

STANFORD UNIVERSITY

Guidelines for Designing Energy-Efficient Web Sites

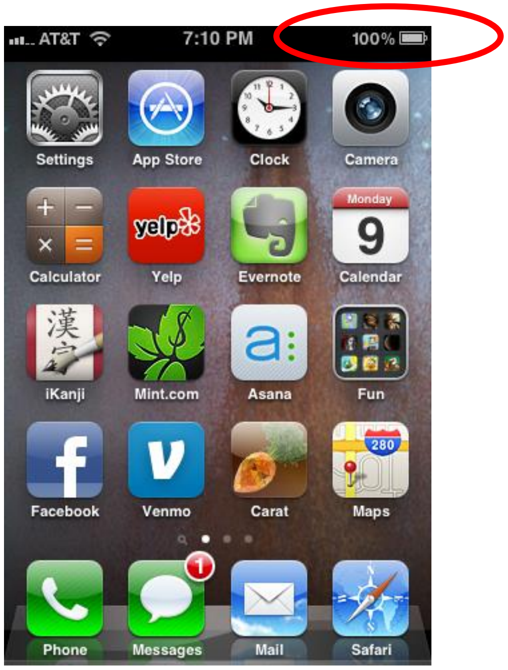
- JPEG is the best image format for the Android browser and holds for all image sizes
- Using HTML links instead of Javascript greatly reduces the rendering energy for the page
- Using links to third party tools can greatly increase the power usage of a phone
- Using simple HTML table element to position elements on the page instead of CSS saves energy
- Building a mobile site optimized for mobile devices conserves energy
- Guidelines also produce a faster UX and reduced data consumption

Carat: Collaborative Energy Diagnosis on Mobile Devices

Adam J. Oliner, Anand P. Iyer, and Ion Stoica
AMP Lab, UC Berkeley
Eemil Lagerspetz, Sasu Tarkoma
U of Helsinki

amplab UC BERKELEY
UNIVERSITY OF HELSINKI

Users' Dilemma



- Users love to use myriad apps
- But they hate it when their battery drains fast
- They then wonder:
  - ***Why*** is my battery draining? (hog)
  - Is that ***normal***? (bug)
  - What can I ***do*** about it? (action)

Prior Approaches

- Focus on specific issues
  - e.g., no-sleep bug
- Intrusive
  - e.g., instrumentation (software or harware)
- Sledgehammer
  - "Kill *all* background apps"
  - "Dim the screen"

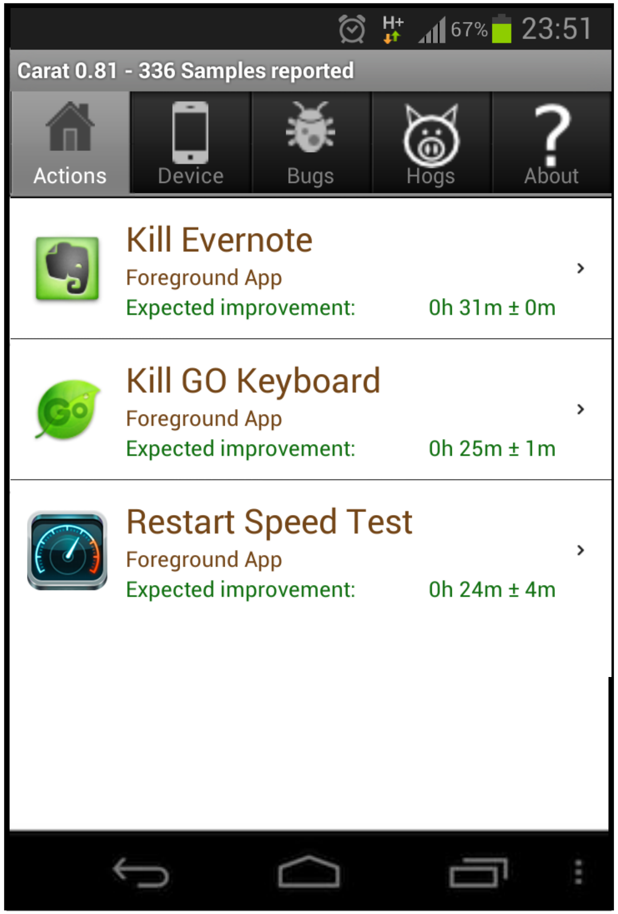
- Goal: a generic, software-only approach that doesn't require any hardware or OS mods

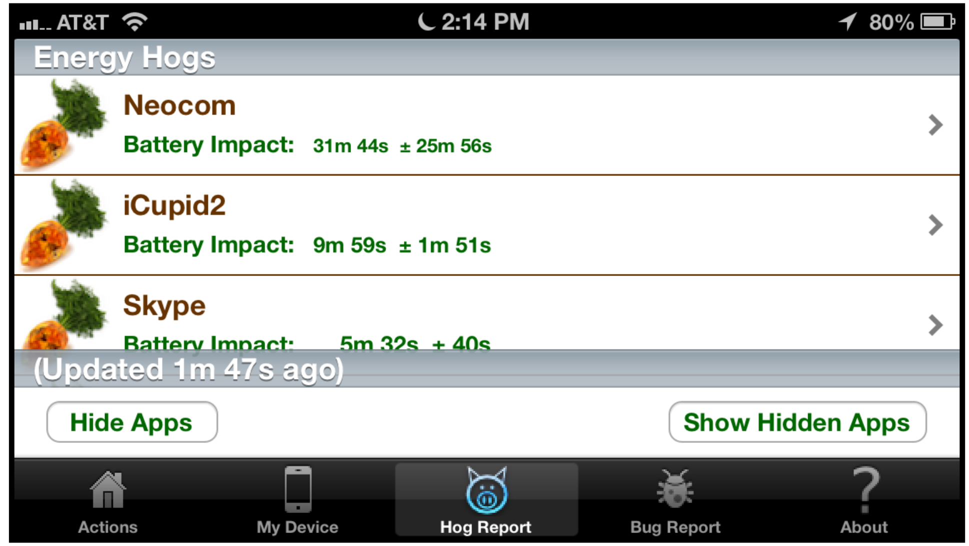
Collaborative Diagnostics

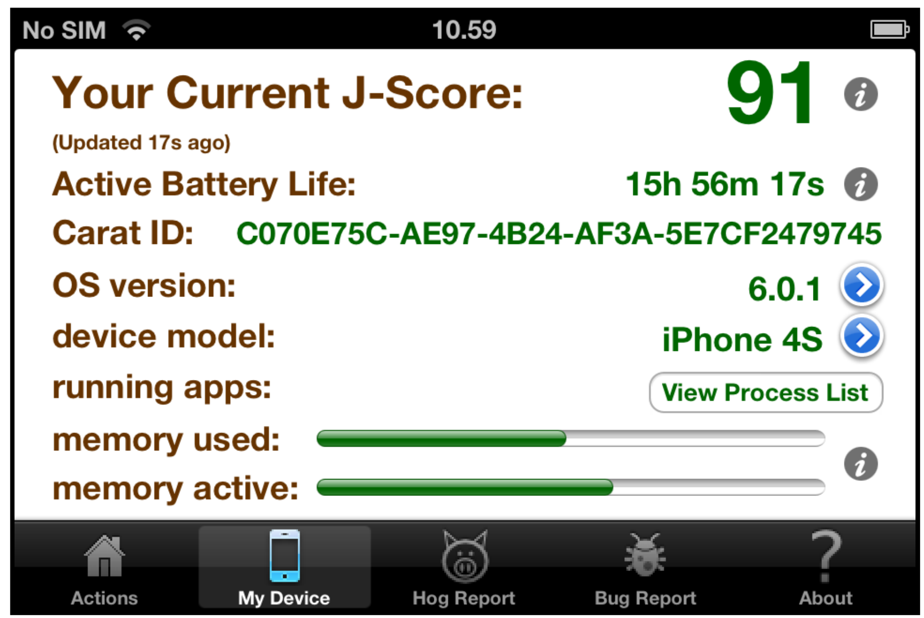
- Idea: use statistics to diagnose problems
- Assumption:
  - mass => norm
  - significant departure from mass => anomaly
- Previously used for a variety of problems:
  - Windows registry issues (STRIDER, 2003)
  - WiFi diagnostics (WiFiProfiler, 2006)
  - Home network diagnostics (NetPrints, 2009)
- Questions:
  - Which metrics to measure?
  - How to gather data from a population?
  - How to compare to identify anomalies?

Why Collaborative?

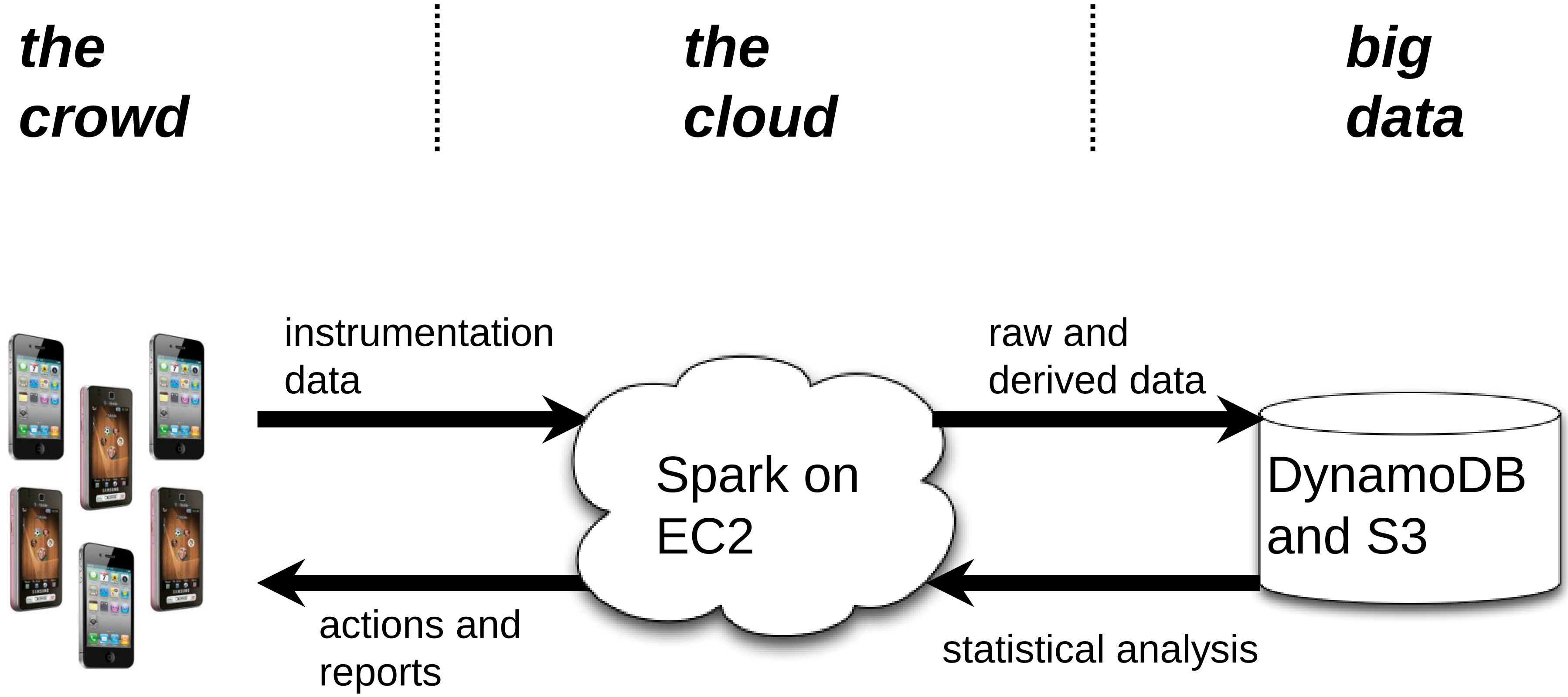
- Enables diagnosis
  - Nearly impossible on a single device
  - Normal? Trigger? Severity? Frequency?
- Distribute instrumentation overhead
- Compensate for biases and uncertainty


23:51
Carat 0.81 - 336 Samples reported
Actions
Device
Bugs
Hogs
About
Kill Evernote
Foreground App
Expected improvement: 0h 31m ± 0m
Kill GO Keyboard
Foreground App
Expected improvement: 0h 25m ± 1m
Restart Speed Test
Foreground App
Expected improvement: 0h 24m ± 4m


AT&T
2:14 PM
80%
Energy Hogs
Neocom
Battery Impact: 31m 44s ± 25m 56s
iCupid2
Battery Impact: 9m 59s ± 1m 51s
Skype
(Updated 1m 47s ago)
Hide Apps
Show Hidden Apps
Actions
My Device
Hog Report
Bug Report
About


No SIM
10.59
Your Current J-Score: 91
(Updated 17s ago)
Active Battery Life: 15h 56m 17s
Carat ID: C070E75C-AE97-4B24-AF3A-5E7CF2479745
OS version: 6.0.1
device model: iPhone 4S
running apps: View Process List
memory used:
memory active:
Actions
My Device
Hog Report
Bug Report
About

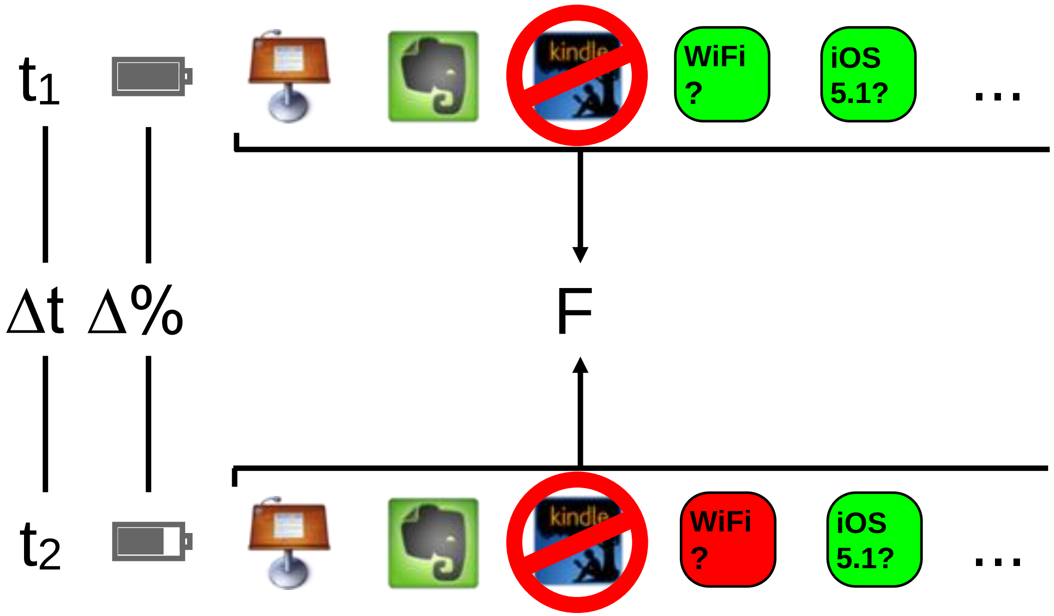
Carat



Carat Infrastructure

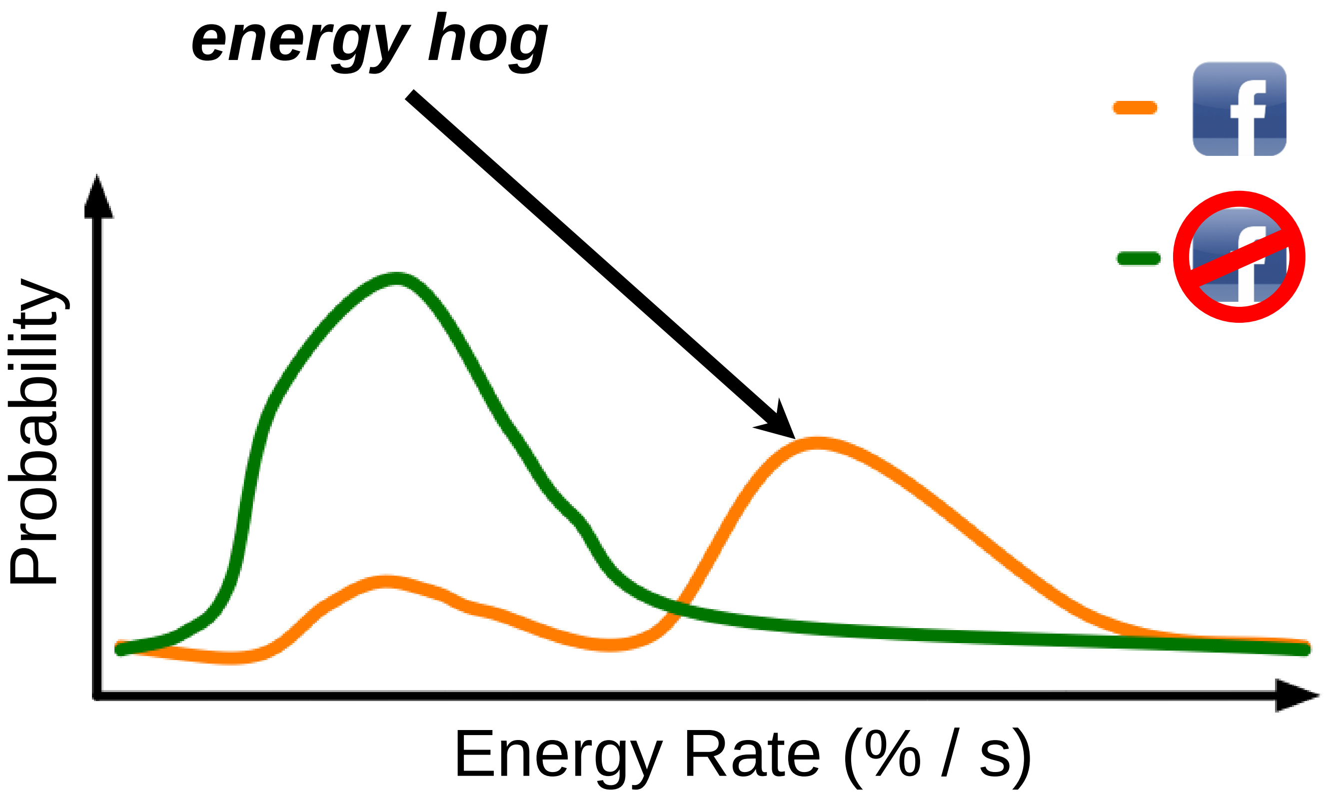
Carat Sampling

Computing Rates



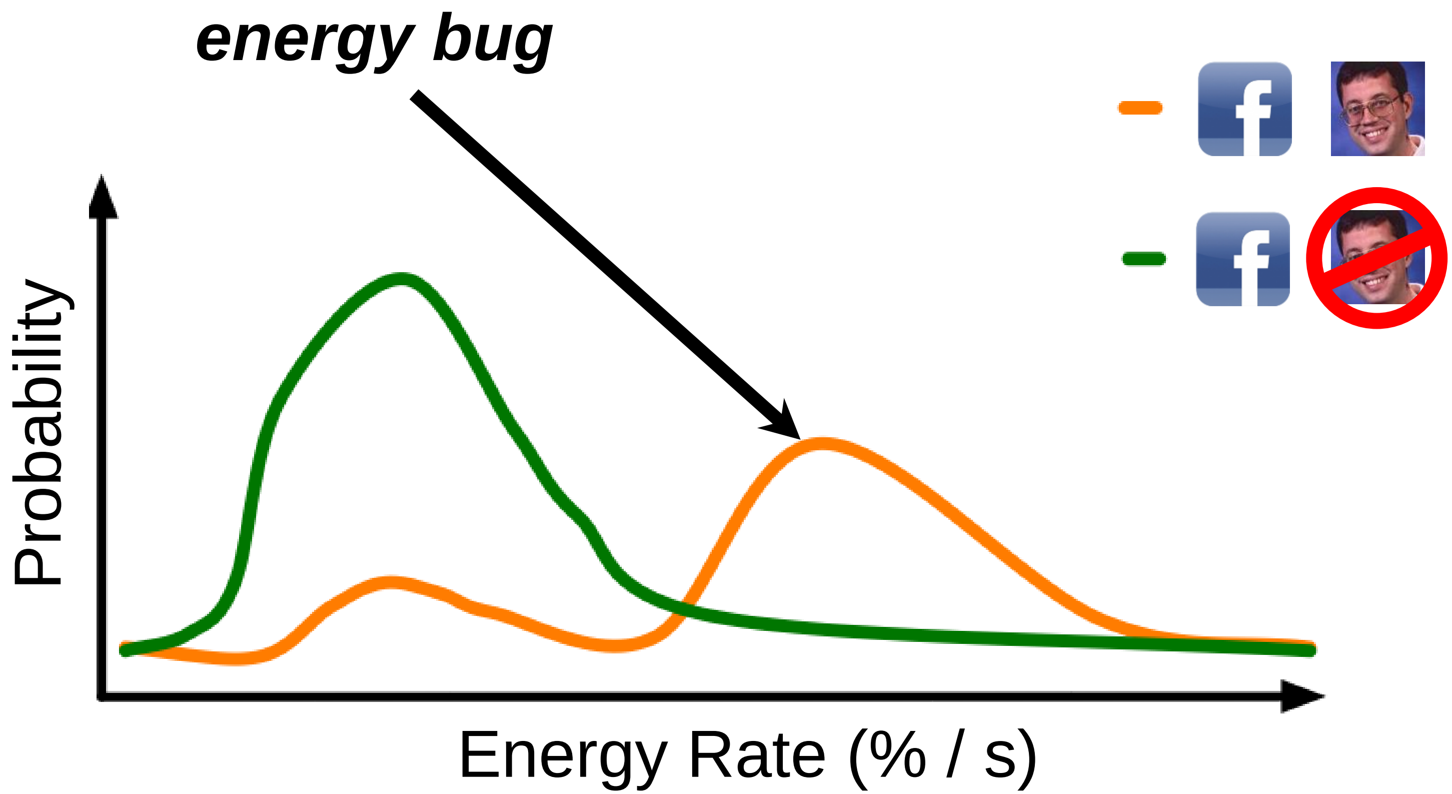
$\frac{\Delta\%}{\Delta t}$ = discharge rate (%/s)| F

Energy Anomalies



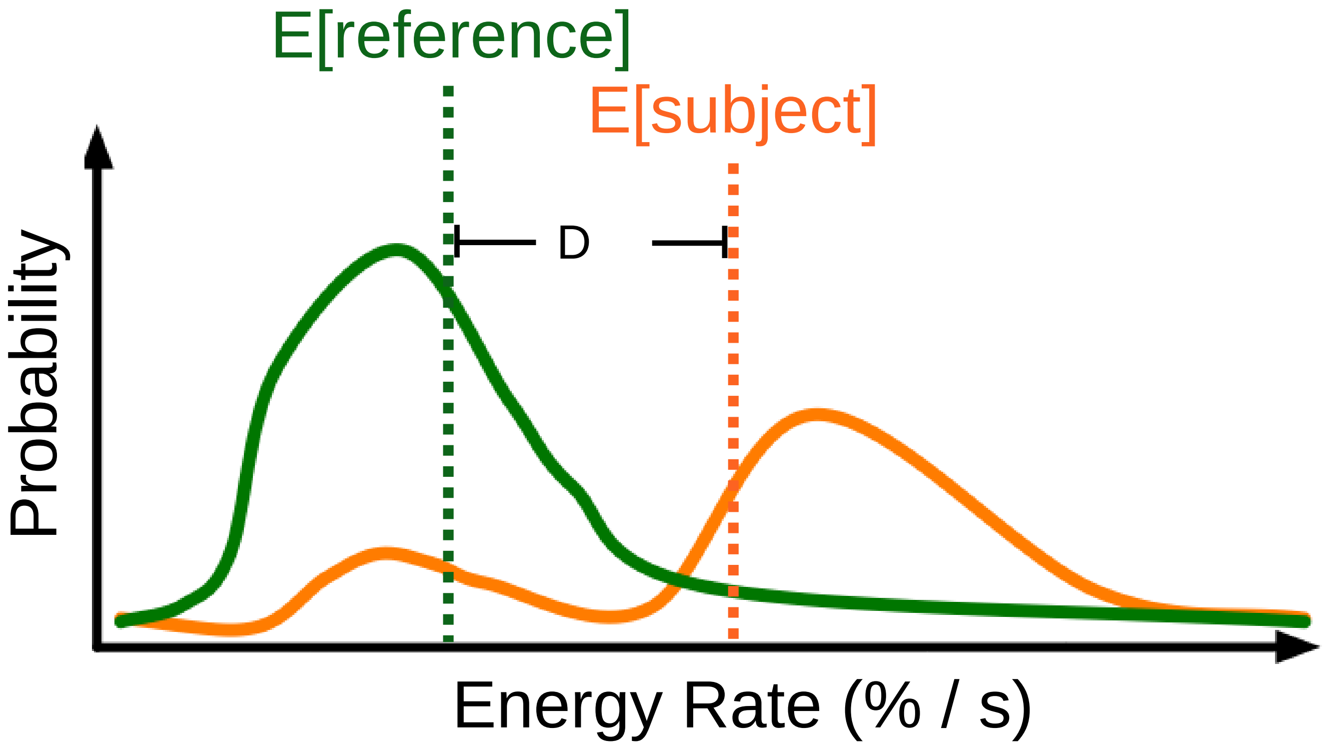
Energy Anomalies

(Given [Facebook icon] is not a Hog.)

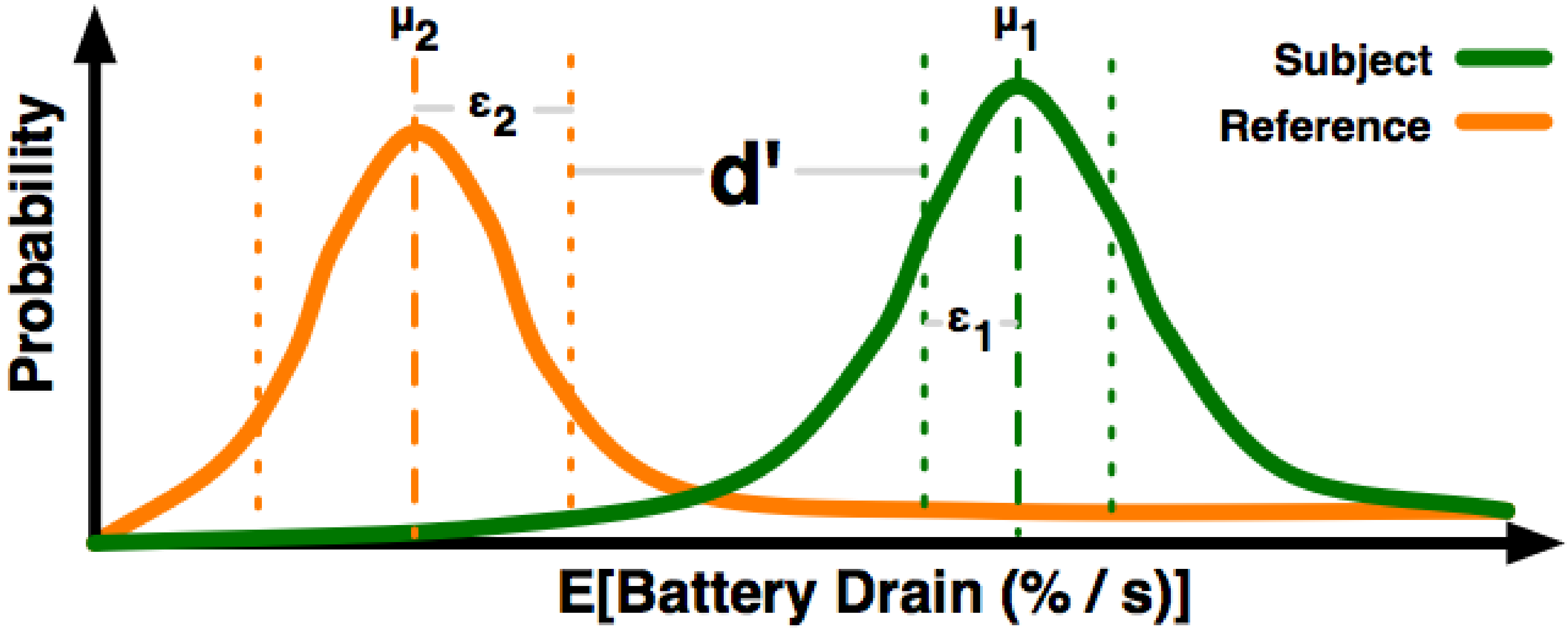


Without the crowd, there is no way to know whether this is normal.

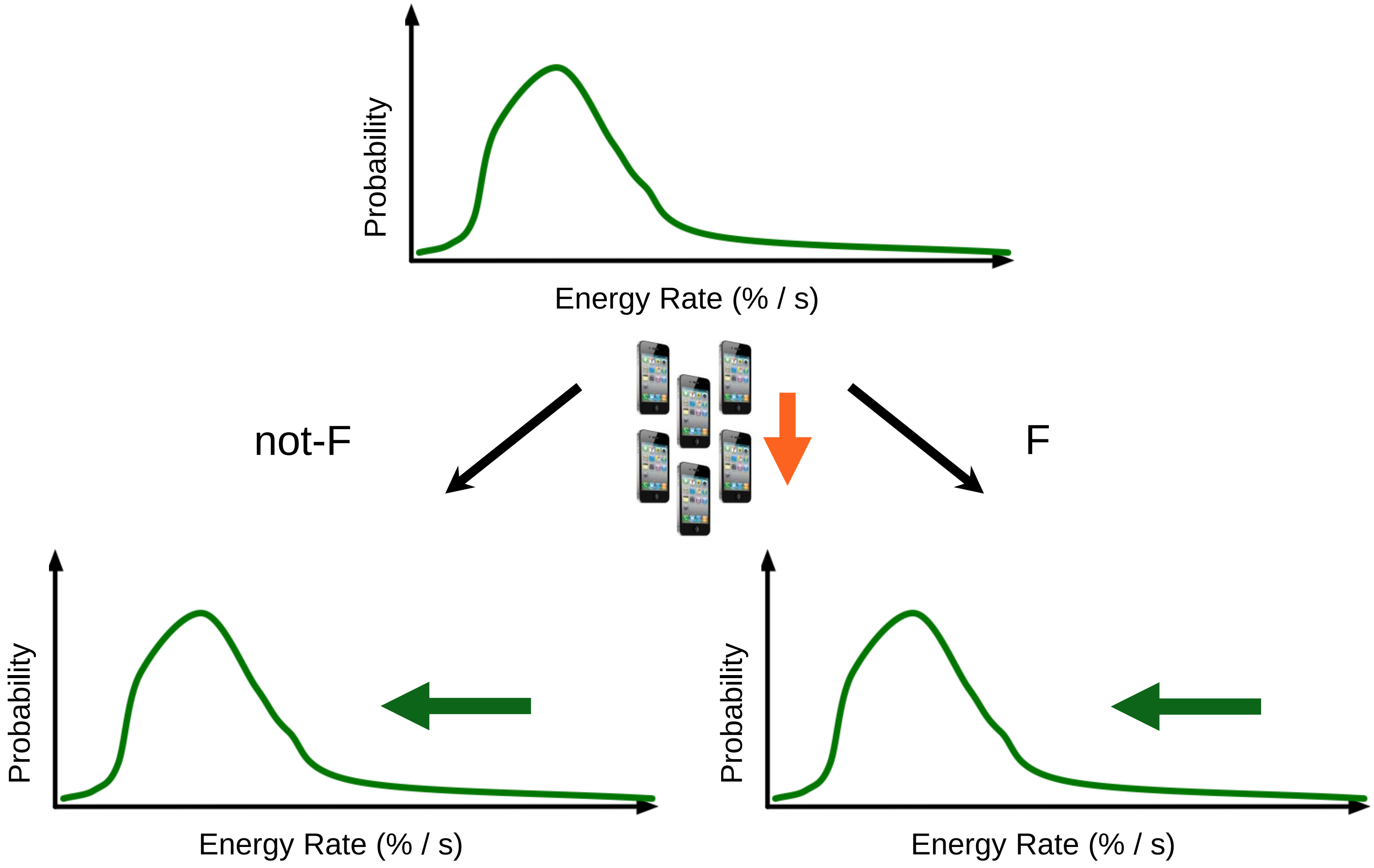
Original Distribution



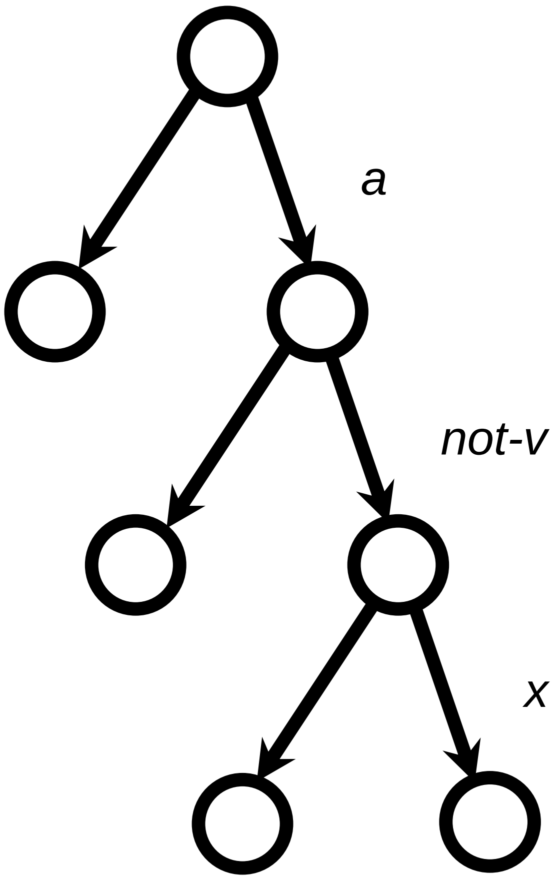
“Significant”



Classification



Diagnosis



"Killing app *a* will give *x*±*e* of battery life (95% confidence), as would upgrading the OS to version *v*."

Carat Today

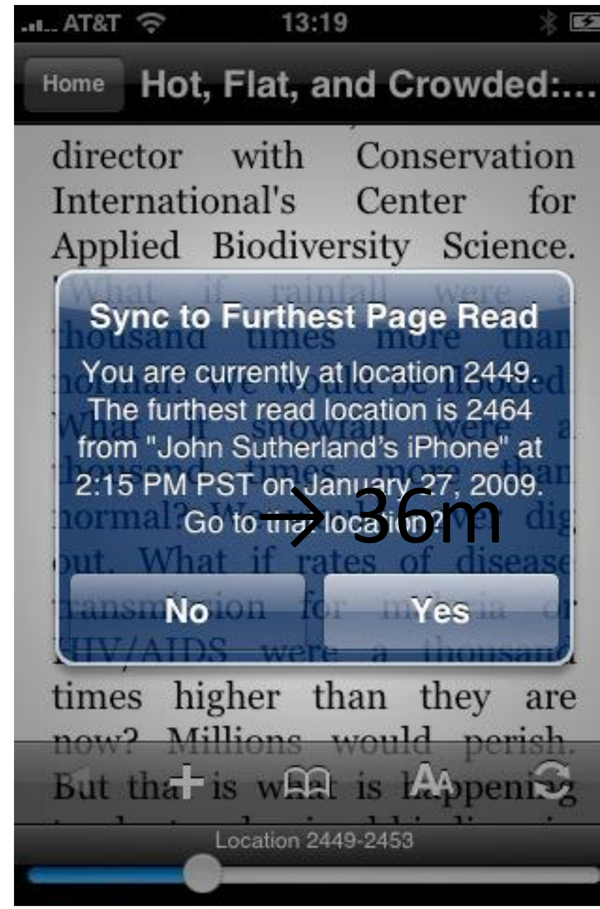
- 450,000+ devices
  - 60% iOS
  - 40% Android
- Tens of millions of samples

Energy Anomalies

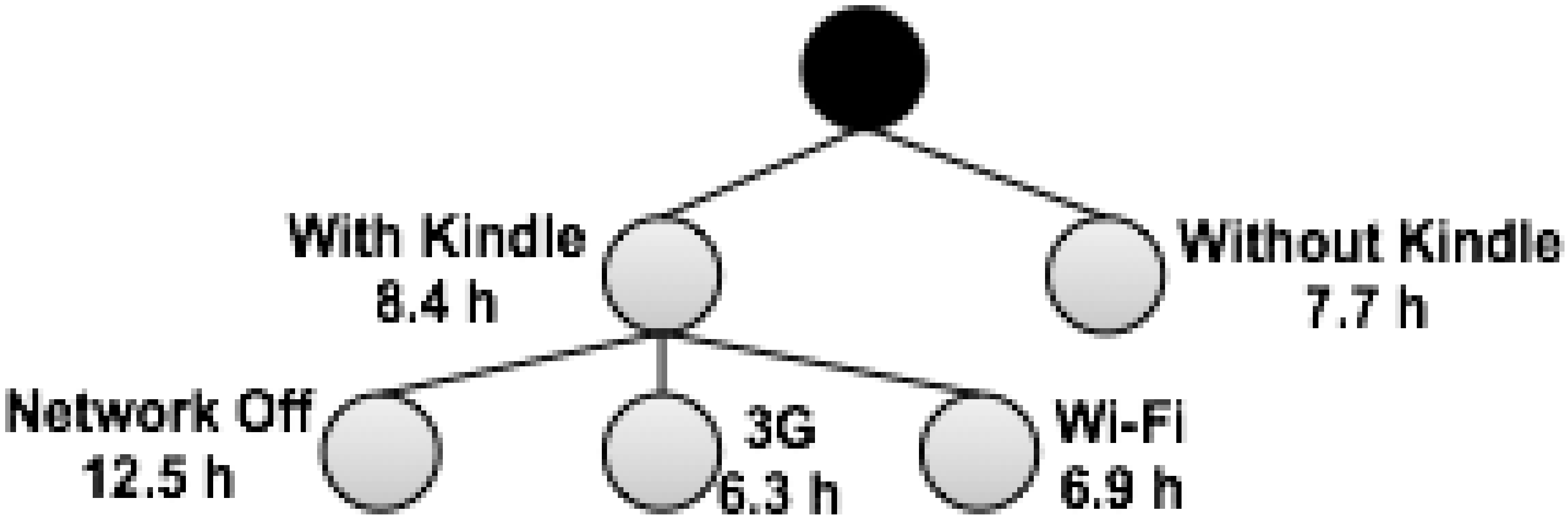
- Hogs
  - 11,256 hogs (9.4%)
  - e.g., Pandora and Skype
- Bugs
  - 483,354 buggy instances (5.3%)
  - e.g., Kindle, Facebook, and YouTube

Kindle Bug (iOS)

- E-book reader
- Bug on 3.9% of clients
- Forum: WhisperSync
- Confirmed by our data
- Turn on WiFi improvement



Kindle Diagnosis



Next Lecture: Mobile Communication