Cellular Networks and Mobile Computing COMS 6998-11, Spring 2012

Instructor: Li Erran Li
(lierranli@cs.columbia.edu)

http://www.cs.columbia.edu/~lierranli/coms6998-11Fall2012/

9/25/2012: Ebug debugging, Power Models, and Profiling

Announcements

- Contact TA Jaimin Shah (jps2178@columbia.edu) to provision your iOS devices
- Programming assignment 2 will be due on Tuesday, Oct 2nd

Outline

- The Rise of Ebugs
- Debugging no-sleep Ebugs
- Methods of Measuring Power Usage
- Power Models
  - Usage based
  - System call trace based
- Profiling
- Conclusion

The Rise of Energy Bugs

Single Symptom:

Severe, Unexpected Battery Drain

Apps Need Not Crash
No Blue Screen Of Death

Common Perception:
Kill some apps to fix

Courtesy: Pathak et al

User Frustration (Dialer App EBug)

Comment 24 by mgil...@gmail.com, Aug 14, 2011

This defect is a real P.I.T.A. - I don't want to use my phone as a phone because I have to restart it every time. If I forget then it's usually 30-40% battery gone by the end of the day.

Comment 30 by hansheng...@gmail.com, Aug 15, 2011

Bring your charger with you and keep it charged!!! That's the only way the phone can last a day. It's a irritating bug!!!

Comment 239 by egork...@gmail.com, Nov 6 (6 days ago)

GOOGLE!!!!!!!! DO SOMETHING WITH THIS ISSUE!!! FASTER PLEASE!!!!

Courtesy: Pathak et al

Crawling Internet Forums

1. **4 Online Mobile Forums**

1.2M Posts
(~400 mobile devices; Several mobile OSes)

→ **39K Posts** → **1000 Clusters**

grep “power drain”, “battery drain”, etc.

K-means

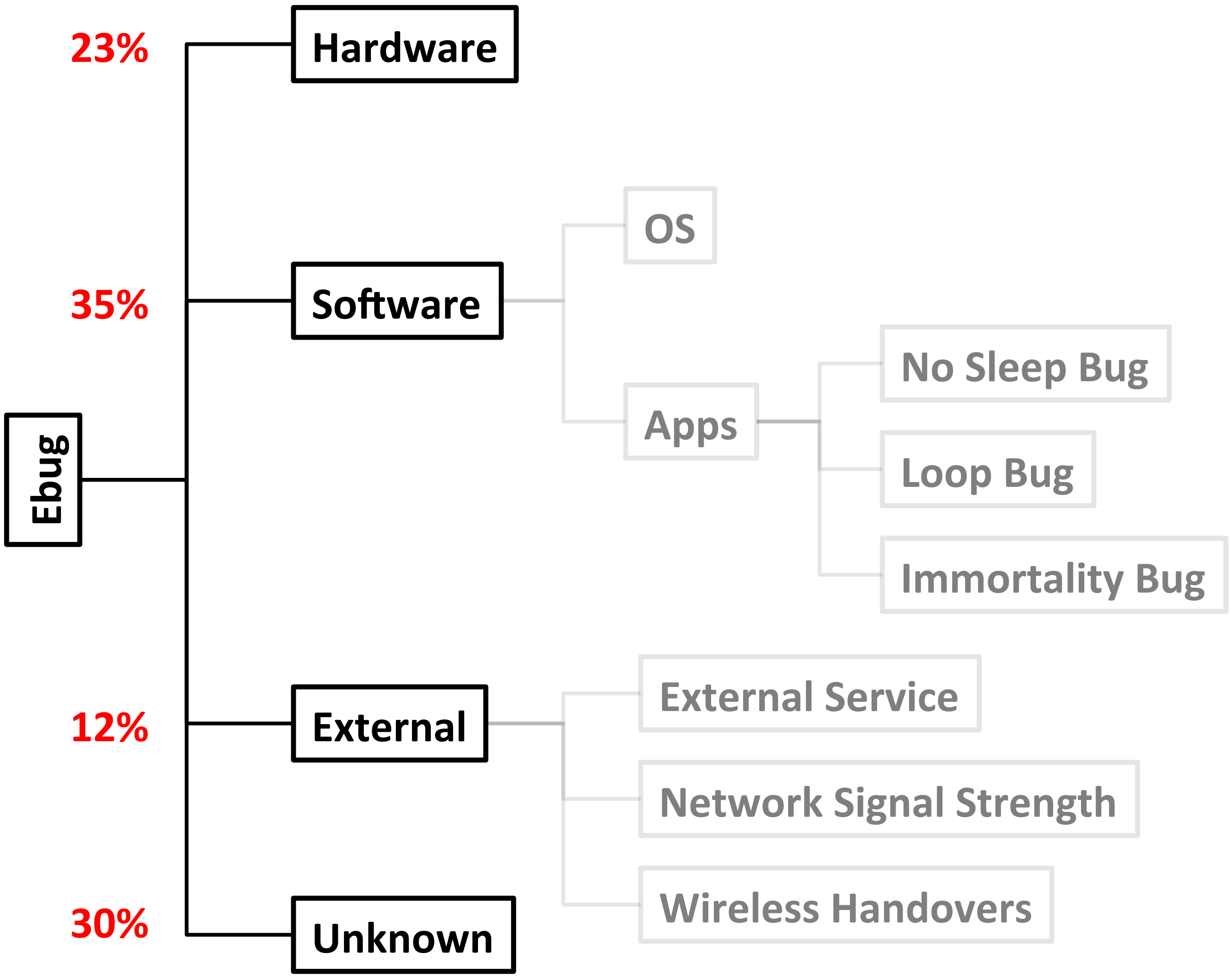
Human → **EBug Taxonomy**

....

2. **2 Mobile Bugs/Issues Repository** → **Indepth Analysis**

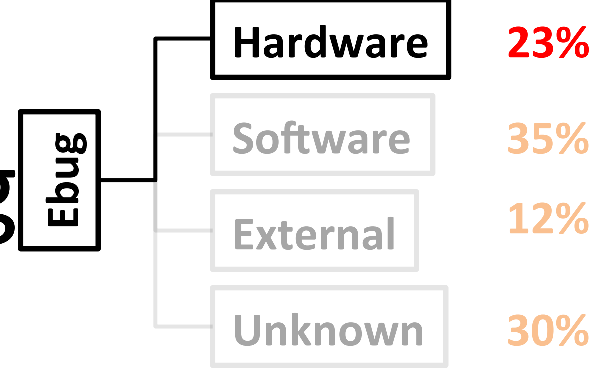
Courtesy: Pathak et al

Ebug Taxonomy



Courtesy: Pathak et al

Hardware EBug



Battery

Sim Card

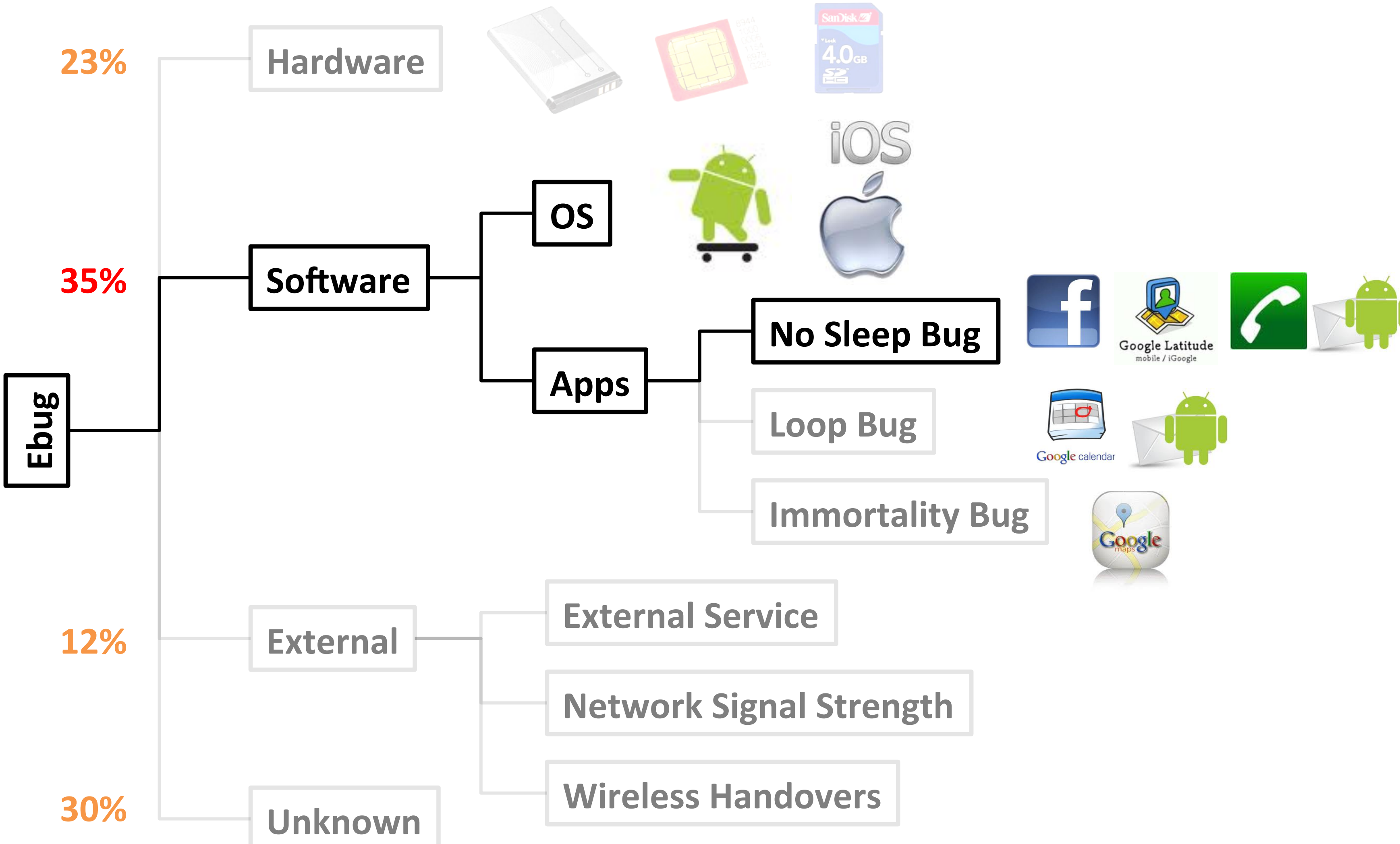
External Hardware

Exterior Hardware Damage

SDCard

Courtesy: Pathak et al

Ebug Taxonomy



Courtesy: Pathak et al

OS Ebugs

Why does OS Leak Energy?

– Hard to infer

– OS Processes

– System Configuration

HOT TOPICS APPLE ANDROID GOOGLE REPUBLIC WIRELESS FACEBOOK ADOBE

iPhone 4S Battery Life Bugs Got You Down?

Battery life on the iPhone 4S: the new 'death grip'?

By **Doug Gross**, CNN
updated 4:17 PM EST, Tue November 1, 2011 | Filed under: Mobile

iPhone battery fix coming 'in a few weeks'

By **Doug Gross**, CNN
updated 11:19 AM EST, Thu November 3, 2011 | Filed under: Mobile

IOS Version: 4.0 – 4.3.3 (5% posts)

2.5% posts

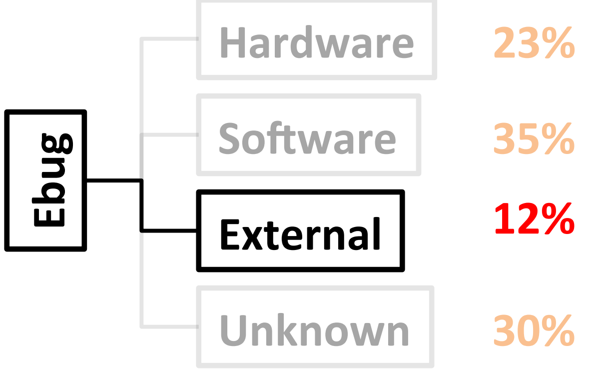
Courtesy: Pathak et al

Apps EBug: No Sleep Bug

- Aggressive Sleeping Policies: Smartphone OSes freeze system after brief inactivity
- Power encumbered Programming: Programmer has to manage sleep/wake cycle of components
- No Sleep Bug: At least one component is kept awake due to mismanagement

Courtesy: Pathak et al

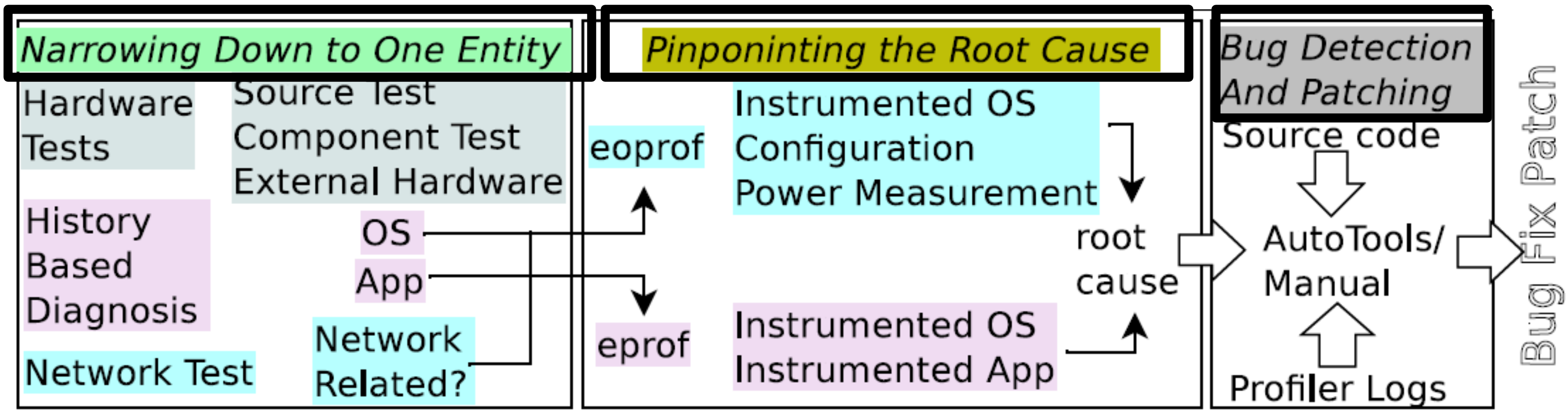
External Conditions



- External Services (<1%)
- Network Signal Strength (11%)
- Wireless Handovers (<1%)

Courtesy: Pathak et al

EDB: Energy Debugging Framework



Courtesy: Pathak et al

Mobile Programming EcoSystem: The EBug Blame Game

Network Operators

App Developers

Hardware Manufacturers

Framework Developers

Firmware/OEM Developers

Kernel Developers

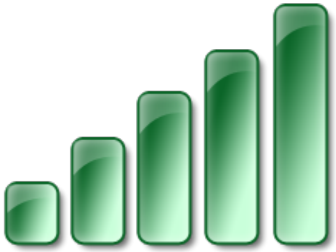
Courtesy: Pathak et al

Outline

- The Rise of Ebugs
- Debugging no-sleep Bugs
- Methods of Measuring Power Usage
- Power Models
  - Usage based
  - System call trace based
- Profiling
- Conclusion

Smartphones are Energy Constrained

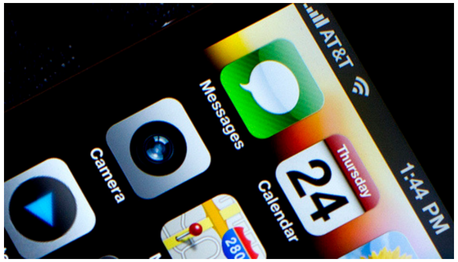
Battery energy density only doubled in last 15 years



3G/4G

GPS

CPU



Screen

Camera

WiFi

Courtesy: Jindal et al

Paradigm Shift in Power Management

- Desktop/Server/Laptop: Default ON
  - CPU turned off when idle for long time
- Smartphones: Default OFF
  - Smartphone OSes aggressively turn off Screen/CPU after brief user inactivity
  - Helps increasing standby time period

Courtesy: Jindal et al

Consequences of Aggressive Sleeping Policy

```
public void DoNetwork ( )
{

      Sync_Over_Network ( );   //Sync state with remote server
                               //Can take up to a minute

}
```

- Background jobs
- User is not always touching phone

Need a mechanism to explicitly keep components awake.

Keep the screen on !

Courtesy: Jindal et al

Android API to Explicitly Keep Components Awake

```
PowerManager pm = (PowerManager) getSystemService
(Context.POWER_SERVICE);
PowerManager.WakeLock wakelock= pm.newWakeLock
(PowerManager.PARTIAL_WAKE_LOCK, null);
```

API	Component (s)
PARTIAL_WAKE_LOCK	CPU
SCREEN_DIM_WAKE_LOCK	CPU and Screen (DIM)
SCREEN_BRIGHT_WAKE_LOCK	CPU and Screen (Bright)
Camera.startPreview() Camera.release()	Camera

Courtesy: Jindal et al

Android API to Explicitly Keep Components Awake (Cont'd)

```
public void DoNetwork ( ) {

        wakelock.acquire ( );         //Don't let CPU sleep till I say so

        Sync_Over_Network ( );    //Sync state with remote server
                                  //Can take up to a minute

}       wakelock.release ( );         //CPU is now free to sleep
```


Courtesy: Jindal et al

iOS API to Explicitly Keep Components Awake

- Screen: **idleTimerDisabled** property in **UIApplication**
  - If NO, the idle timer turns off the device's screen after a specified period of inactivity
- GPS: enable/disable location updates, and set the distance filter and accuracy levels
  - Core Location uses the available GPS, cell, and Wi-Fi networks to determine the user's location
  - Core Location minimizes the use of these radios
  - Setting the accuracy and filter values gives Core Location the option to turn off hardware altogether in situations where it is not needed.
- WiFi: **UIRequiresPersistentWiFi** key in the app's Info.plist file
  - System will not shut down the Wi-Fi hardware while your app is running.

Power Encumbered Programming

App programmer has to manage sleep/wake cycle of components

Courtesy: Jindal et al

NoSleep Energy Bugs

An error in smartphone app, where a component is woken up, **but not put to sleep**

Component	**Impact (Google Nexus)**
CPU	**50-60%** battery in 12 hrs
Screen	**100%** battery in 4-6 hrs
GPS	**100%** battery in 3-5 hrs

Courtesy: Jindal et al

Collecting and Characterizing Bugs

- Online mobile forums
- Bug lists
- Open source code repositories
- Running the detection tool

Courtesy: Jindal et al

Types of NoSleep Bugs

NoSleep Code Paths

NoSleep Race Conditions

NoSleep Dilations

Courtesy: Jindal et al

(a) NoSleep CodePaths

At least one path in program wakes up a component ut it to sleep

public void DoNetw
{
try
{
wakelock. U sleep till I say so
Sync_Over with remote server **;throws**
to a minute **;exception**

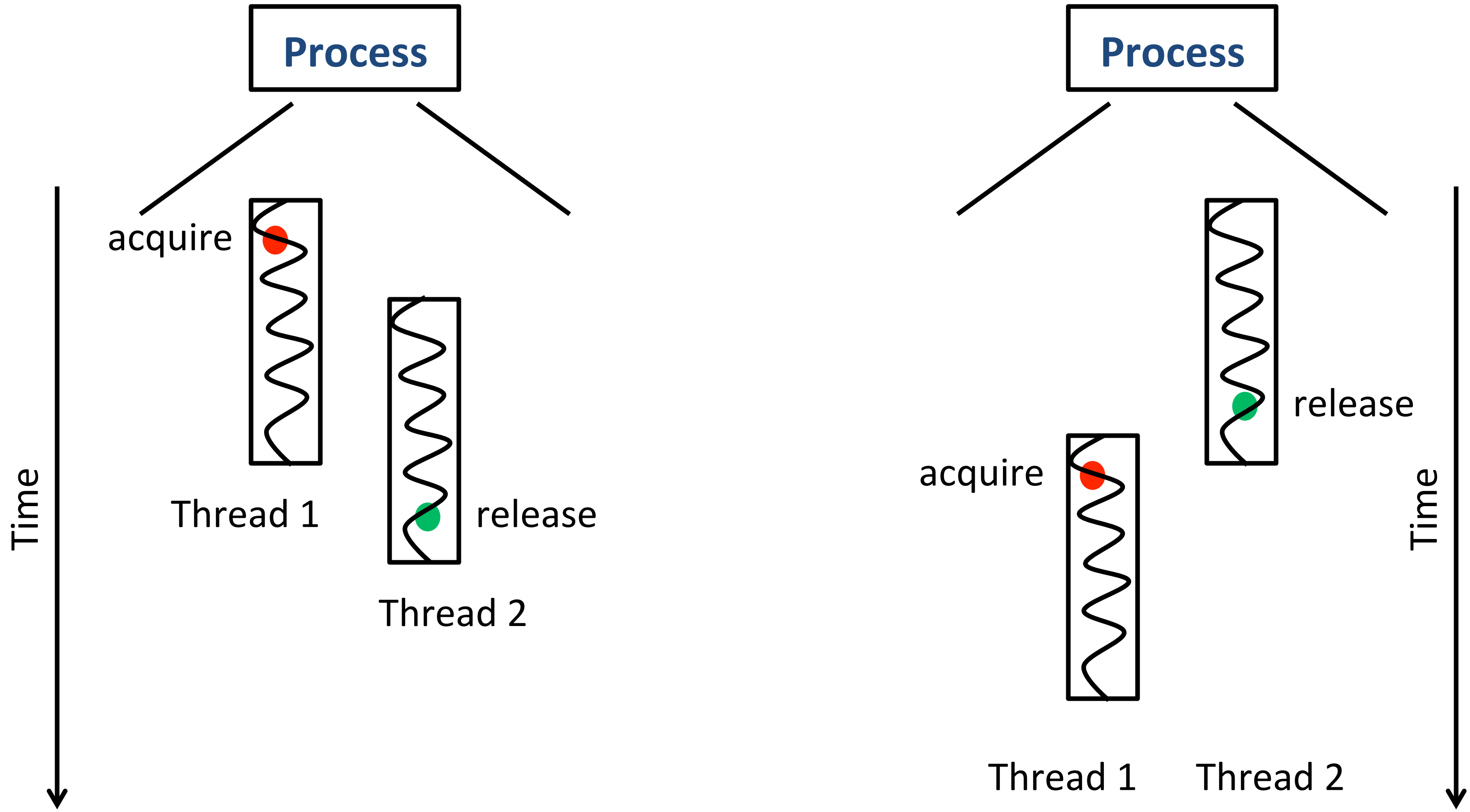
wakelock.release ( ); //CPU is now free to sleep

} catch (SomeException e){
Print.Error (e); //Print Error for Debugging purposes
}

}

Courtesy: Jindal et al

(b) NoSleep Race Condition (1)



Courtesy: Jindal et al

(b) NoSleep Race Conditions (2)

```
public class MainThread
{
    boolean KillFlag = false; //Kill Flag

    wakelock.acquire ( );  //CPU C

    WorkerThread.start ();//start w
     .......
     .......
     ......
    KillFlag = true; //Kill the worke
    WorkerThread.interrupt();//stop worker

    wakelock.release ( ); //CPU can sleep now
    .......
    .......
}
```

```
public class WorkerThread
{
          e(true){ //Infinite Loop
          f(KillFlag) break; //Exit
          NetSync(); //Critical Section

          vakelock.release ( ); //CPU can sleep

          Thread.Sleep (3 Mins);//Sleep for 3 mins

          wakelock.acquire ( );  //CPU Cant Sleep

      }
}
```

Buggy Execution

(c) NoSleep Dilations

Components are eventually turned off but are kept on for unusually long duration of time

public void onCreate( )
{
        ~~wakelock.acquire ( ) ;~~
}

public long startNewTrack( )
{
\+   wakelock.acquire ( ) ;
        ..
}

Courtesy: Jindal et al

Detecting NoSleep Bugs

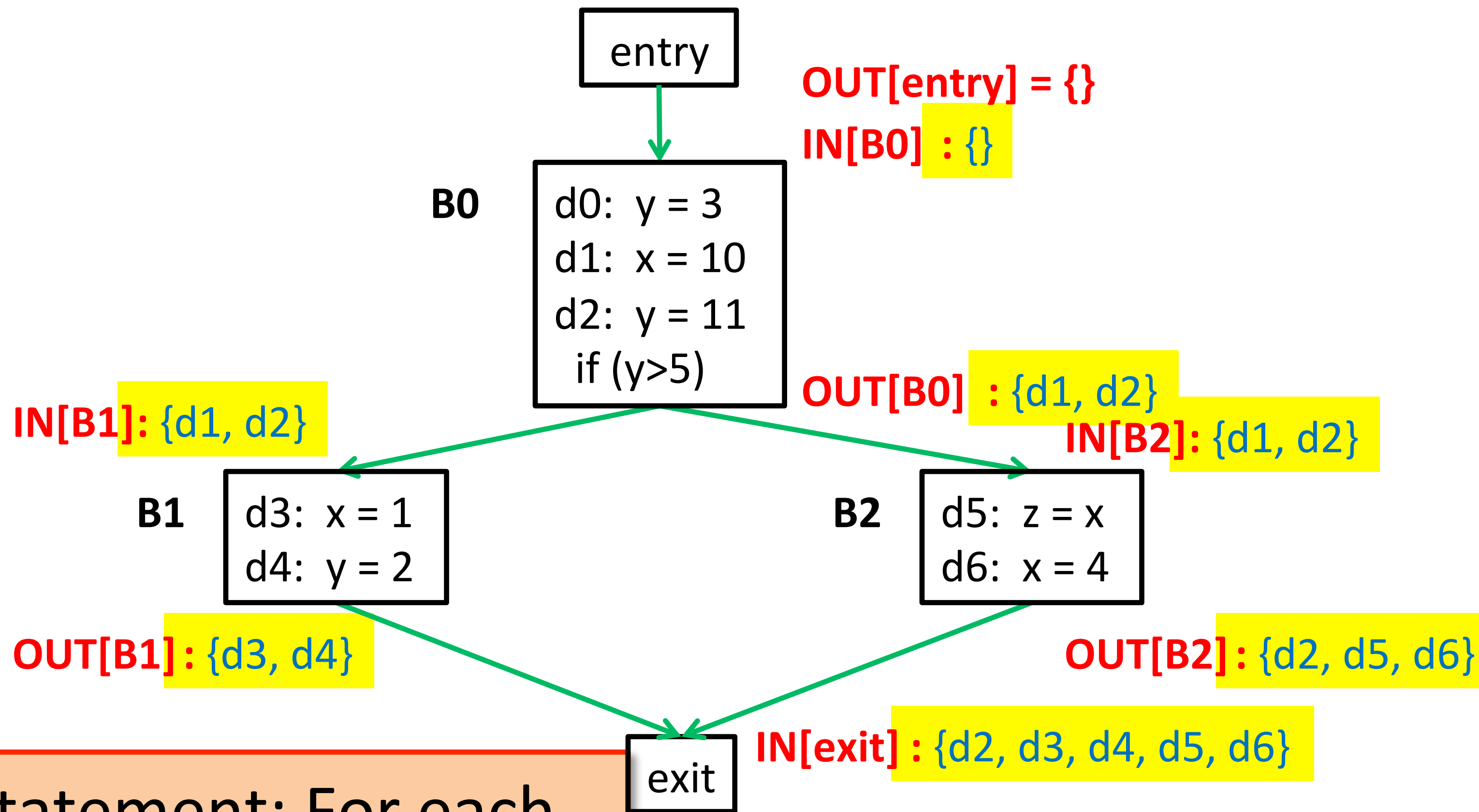
- NoSleep Code Paths
- NoSleep Race Conditions

- Static Data Flow Analysis
  - Reaching Definitions Data Flow Problem

Courtesy: Jindal et al

Reaching Definitions Data Flow Problem

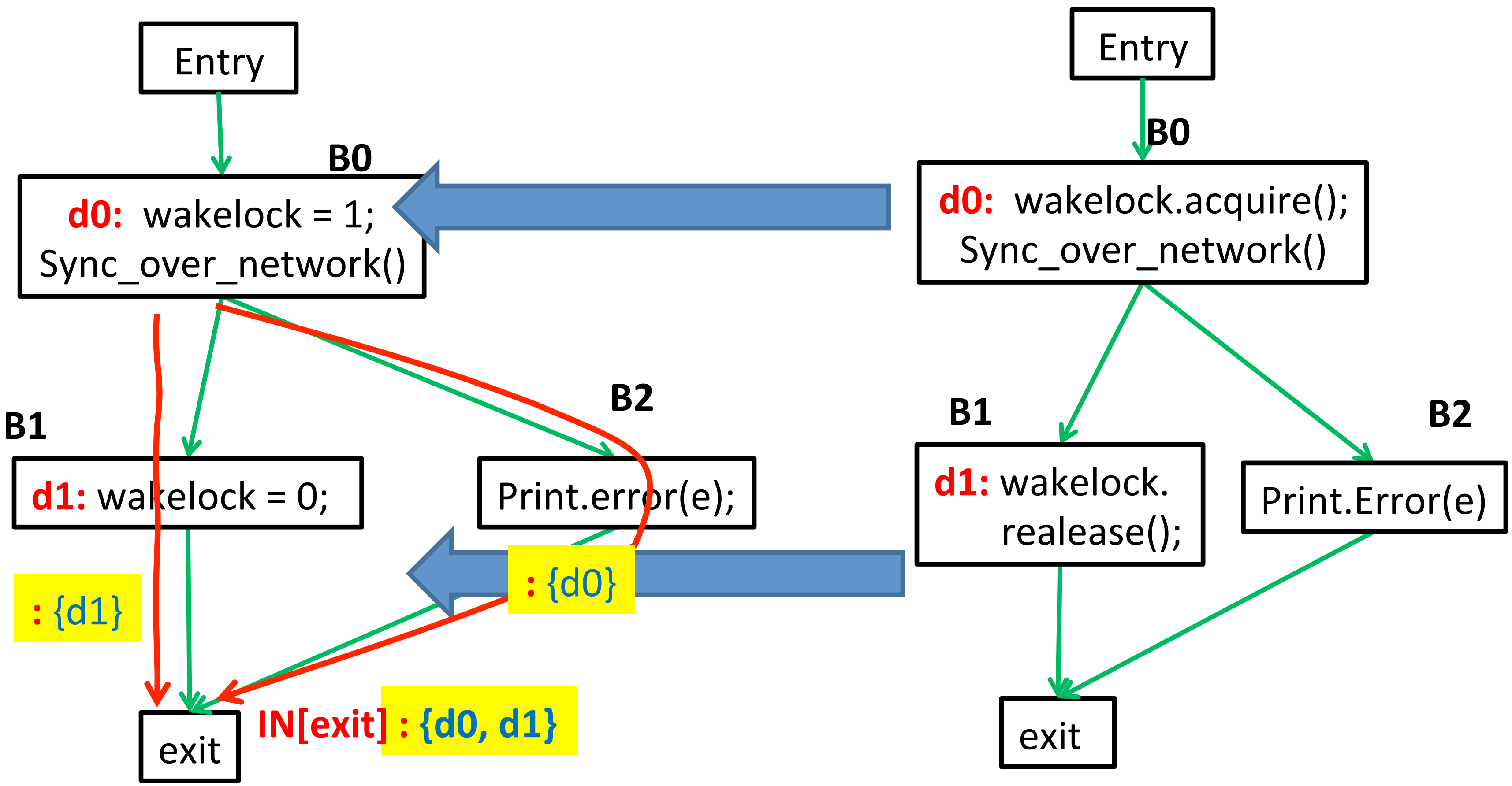
```
y = 3;
x = 10;
y = 11;

if( y > 5 ) {
    x = 1;
    y = 2;
} else {
    z = x;
    x = 4;
}
```



Problem Statement: For each point in the program, determine which definition of variable are reachable

Courtesy: Jindal et al

NoSleep Code Path DataFlow Analysis



Apply reaching definitions to NoSleep CodePaths

Courtesy: Jindal et al

Additional Details of Analysis in Paper

- Event based programming
- Java runtime issues
- Special code paths: reducing false positives
- Multi-threaded programs

Courtesy: Jindal et al

Implementation

- Soot Implementation
  - Open source framework for analyzing java bytecode from Sable research group at McGill University
  - Added ~2 KLOC
- Runs on Java byte code
  - No need for source code

Courtesy: Jindal et al

Evaluation: NoSleep Code Paths

500 Apps → **187** → **86** → **55**

Apps Manipulating components | **Apps which could be decompiled** | **Find NoSleep Bugs**

Total Apps 86	**BUG in reality**	**No BUG in reality**
Tool reports BUG (55)	True Positive **42**	False Positive **13**
Tool reports no BUG (31)	False Negative **0**	True Negative **31**

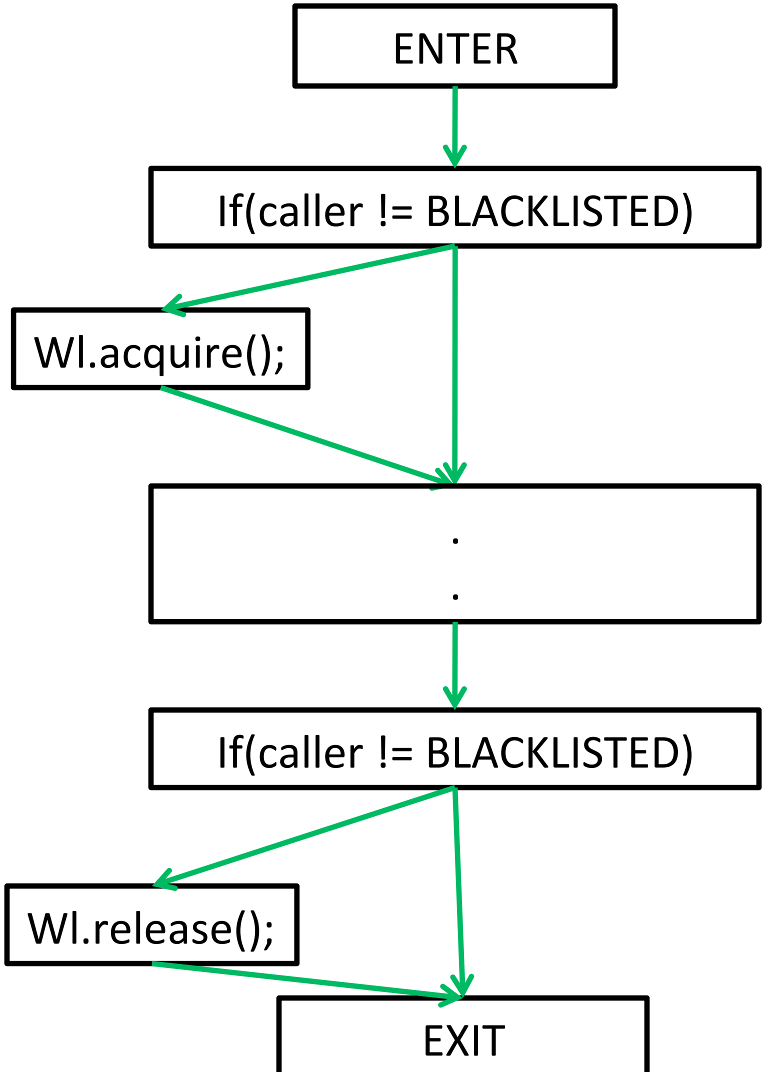
Courtesy: Jindal et al

Causes of Detected NoSleep Bugs

Category	Number of Apps	Examples
Incorrect Wakelock Handling in Events	26	MyTracks
If, else + exception Paths	12	Facebook
Forgot Release	4	K9Mail

Courtesy: Jindal et al

Common Cause of False Positives



Courtesy: Jindal et al

Summary

- Paradigm shift in power management
  - Power encumbered programming
- NoSleep energy bugs
  - Code paths, race, dilation
- Reaching definition data flow analysis
  - Found 30 new previously unreported bugs

Courtesy: Jindal et al

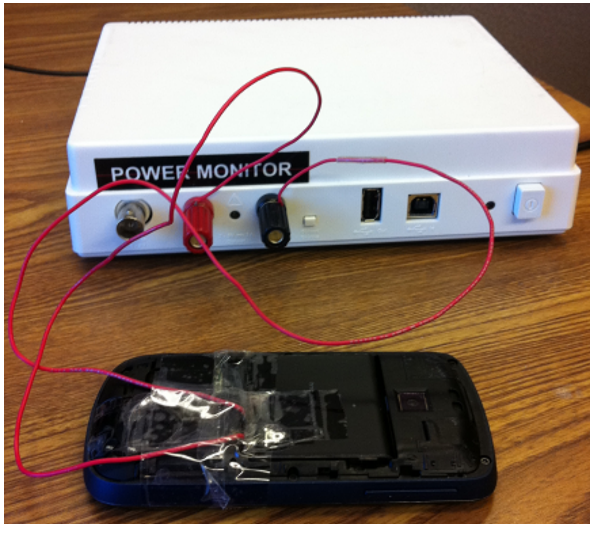
Future Work

- Comprehensive approaches to No-Sleep bug
  - Detection: e.g. Symbolic execution
  - Prevention: e.g. Better abstraction for power management API
  - Mitigation: e.g. Runtime detection, Limiting damage of the bug

Courtesy: Jindal et al

Measuring Power Usage

- Approach 1: Use power meter (offline)
  - Buy an expensive equipment ($770)
  - Problems:
    - Only reports entire device energy consumption



- Approach 2 : Use built-in battery sensor (online)

iOS Battery API

- Use `UIDevice` class to obtain information and notifications about
  - charging state (property `batteryState`)
  - charging level (property `batteryLevel`)

```
[[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];
  NSArray *batteryStatus = [NSArray arrayWithObjects:
                                     @"Battery status is unknown.",
                                     @"Battery is in use (discharging).",
                                     @"Battery is charging.",
                                     @"Battery is fully charged.", nil];
  if ([[UIDevice currentDevice] batteryState] == UIDeviceBatteryStateUnknown)
      NSLog(@"%@", [batteryStatus objectAtIndex:0]);
  else
  {
      NSString *msg = [NSString stringWithFormat:
                               @"Battery charge level: %0.2f%%\n%@",
                              [[UIDevice currentDevice] batteryLevel] * 100,
                              [batteryStatus objectAtIndex:[[UIDevice currentDevice]
                                         batteryState]] ];
      NSLog(@"%@", msg);
  }
```

Android Battery API

- Sample updates stored in files:
  - Current: /sys/class/power_supply/battery/batt_chg_current
  - Voltage: /sys/class/power_supply/battery/batt_vol
  - Capacity: /sys/class/power_supply/battery/capacity

```
1.  File fcur = new File("/sys/class/power_supply/
    battery/batt_chg_current");
2. if (fcur.exists())
3.      …
```

- File names are vendor dependent
- Access using Android Debug Bridge (adb)
  - <sdk>platform-tools
  - Command: adb shell

Outline

- The Rise of Ebugs
- Debugging no-sleep Bugs
- Methods of Measuring Power Usage
- Power Models
  - Usage based
  - System call trace based
- Profiling
- Conclusion

Smartphone is Energy Constrained

- Energy: One of the most critical issues in smartphones
  - Limited battery lifetime

- Battery energy density only doubled in last 15 yrs

- Smartphone capability has increased drastically
  - Multiple Components: GPS, 3G, retina display, ….

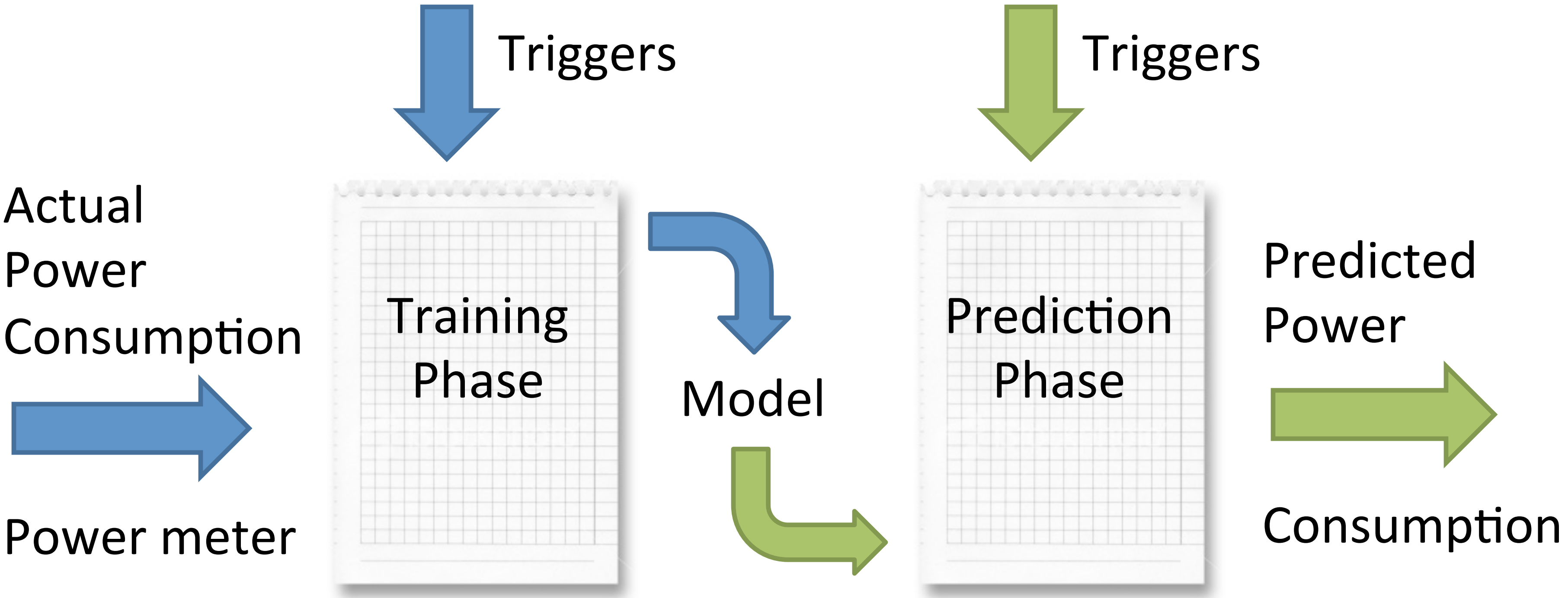
Courtesy: Pathak et al

Towards Understanding Energy Drain

- Key Question: Where is energy being spent?
  - Which component/process/thread/function(?)

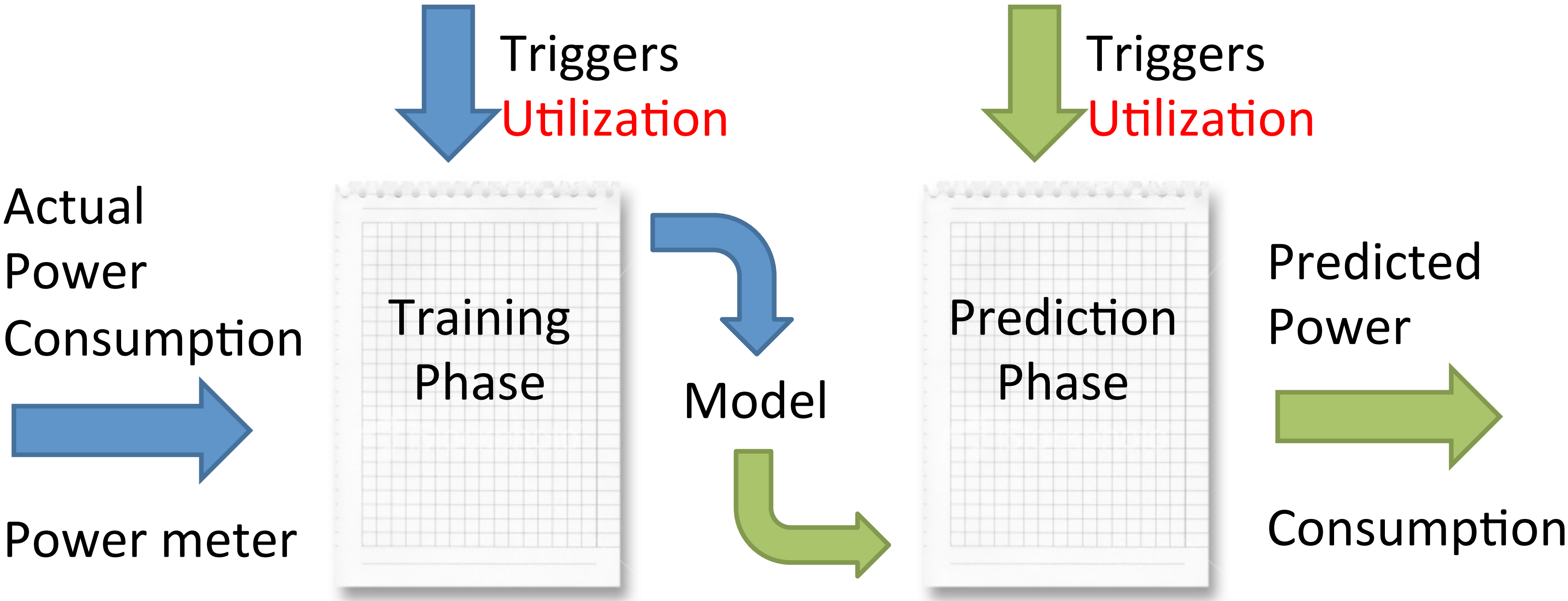
Courtesy: Pathak et al

Generic Power Modeling



Courtesy: Pathak et al

Smartphone Power Modeling: Utilization Based (1/3)



Linear Regression (LR) and Superimposition

$$\text{Model} = (\text{Util}_{Net}) * E_{Net} + (\text{Util}_{CPU}) * E_{CPU} + (\text{Util}_{Disk}) * E_{Disk}$$

Courtesy: Pathak et al

Smartphone Power Modeling: Utilization Based (2/3)

- PowerTutor model

$$(\beta_{uh} \times freq_h + \beta_{ul} \times freq_l) \times util + \beta_{CPU} \times CPU\_on + \beta_{br} \times brightness + \beta_{Gon} \times GPS\_on + \beta_{Gsl} \times GPS\_sl + \beta_{Wi\text{-}Fi\_l} \times Wi\text{-}Fi_l + \beta_{Wi\text{-}Fi\_h} \times Wi\text{-}Fi_h + \beta_{3G\_idle} \times 3G_{idle} + \beta_{3G\_FACH} \times 3G_{FACH} + \beta_{3G\_DCH} \times 3G_{DCH}$$

β : power coefficient.
util, brightness and etc.: system variables.

- Sesame paper has two optimizations: model molding, principle component analysis (PCA)

Smartphone Power Modeling: Utilization Based (3/3)

Model = $(Util_{Net}) * E_{Net} + (Util_{CPU}) * E_{CPU} + (Util_{Disk}) * E_{Disk}$

Fundamental (yet intuitive) assumption

(Only active) Utilization => power consumption

Second assumption

Energy scales linearly with amount of work

Third assumption

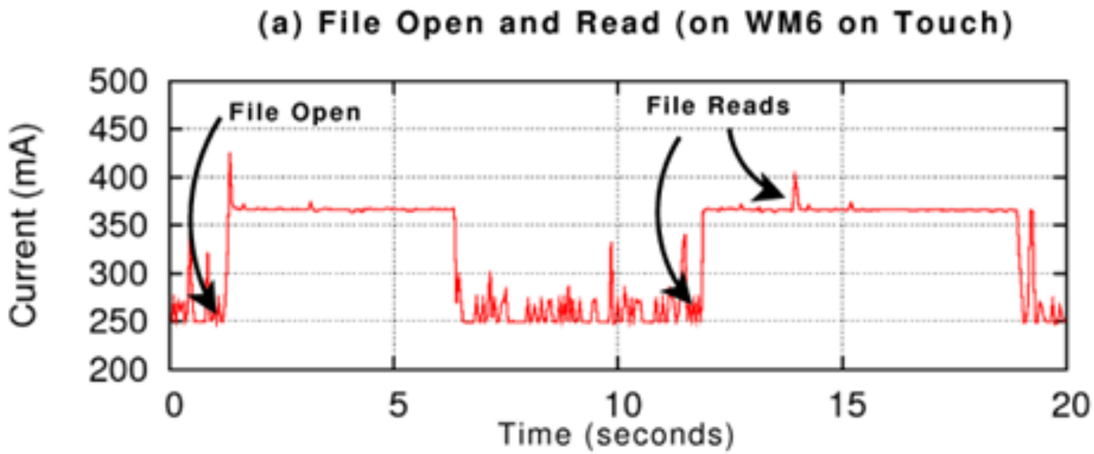
Components power consumption add linearly

Desired Feature

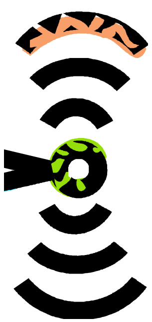
Which process/thread/function? Hard to correlate

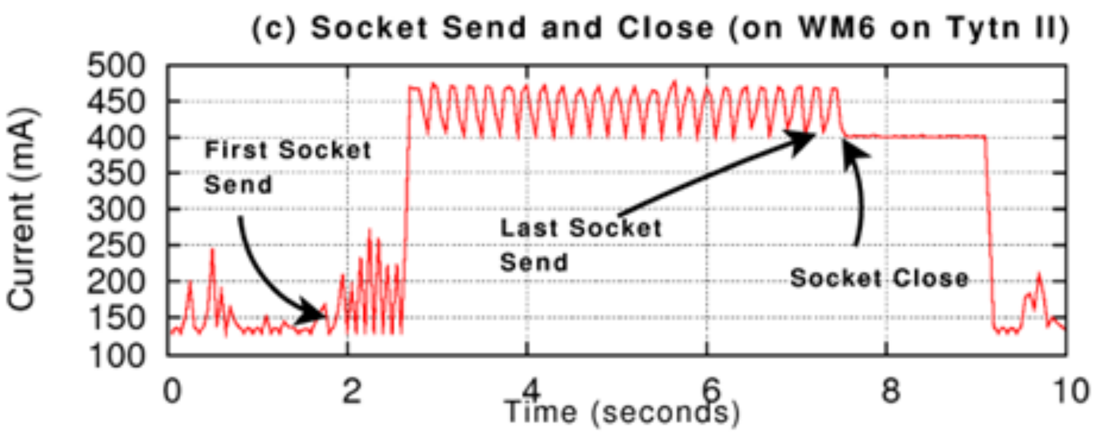
Courtesy: Pathak et al

(Only active) Utilization => Power Consumption



File open/delete/close/create change power state



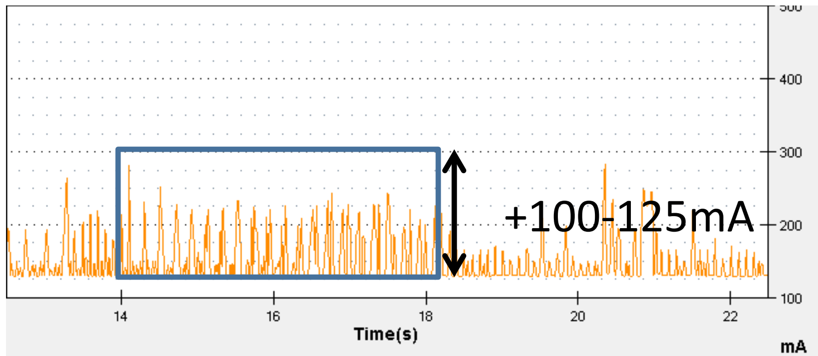


Several components have tail states (3G, disk, wifi, gps)

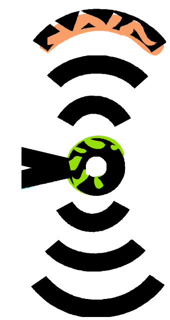
Courtesy: Pathak et al

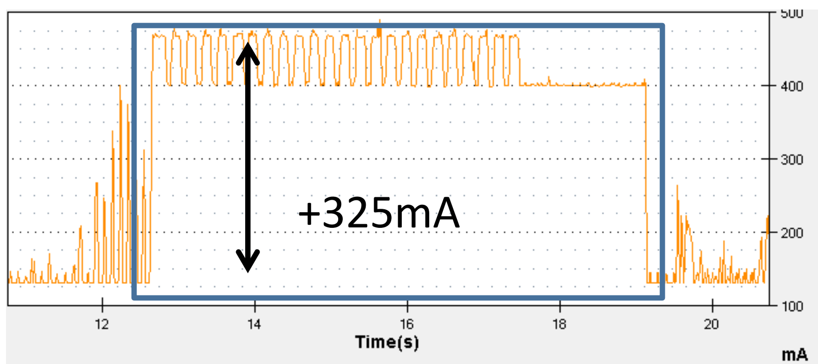
Energy scales linearly with amount of work

WM6.5 on Tytn II



(1) Send packets @ < 50pkts/s



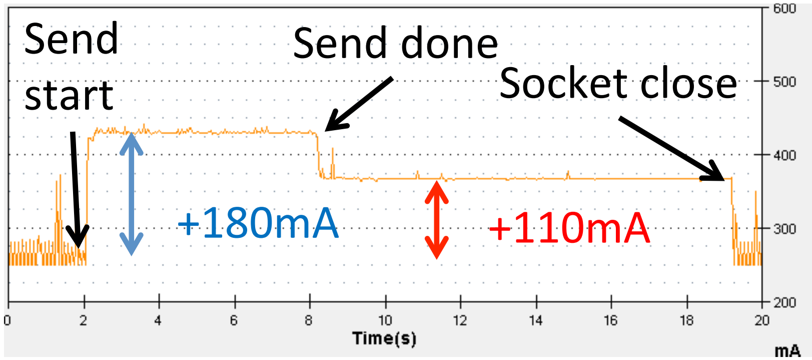


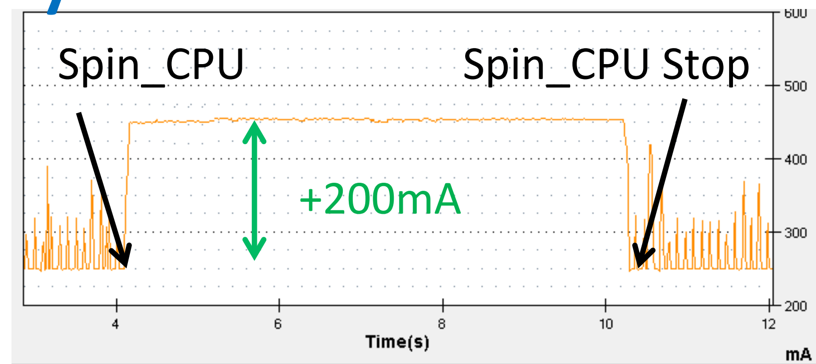
(2) Send packets @ > 50pkts/s

Courtesy: Pathak et al

Components power consumption add linearly

WM6.5 on HTC Touch

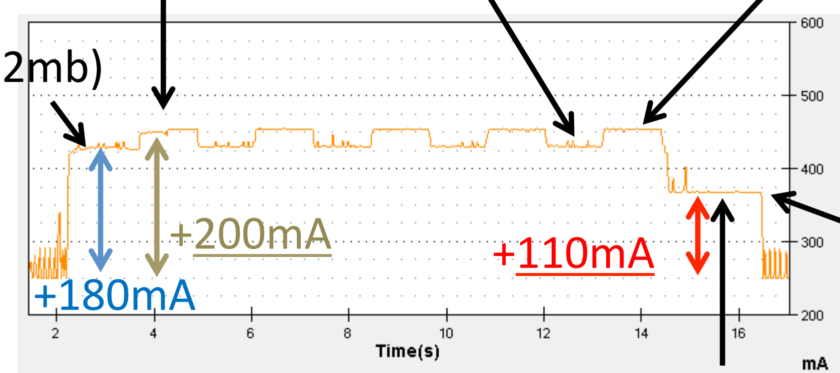




(1) Send(10mb);
sleep();
Socket.close();

(2) Spin_CPU(10M);

(3)
for (i in 1 to 5){
Send(2mb);
Spin_CPU(2M);
}
Sleep();
Socket.close();



Courtesy: Pathak et al

What have we learnt so far?

Simple (state-of-art) energy modeling assumptions are wrong

There exits a notion of power states

What have we hinted so far?

Device drivers have intelligent power control rules

System calls play a role in power consumption

Challenges in fine-grained power modeling?

Device drivers are closed source (no code/no information)

Courtesy: Pathak et al

System Calls As Power Triggers

Key observation: System call is the interface through which an application communicates with the underlying system (hardware) and outside world (Internet, GPS, etc.)

Key Idea: Use System Calls as triggers in power modeling

Advantages:

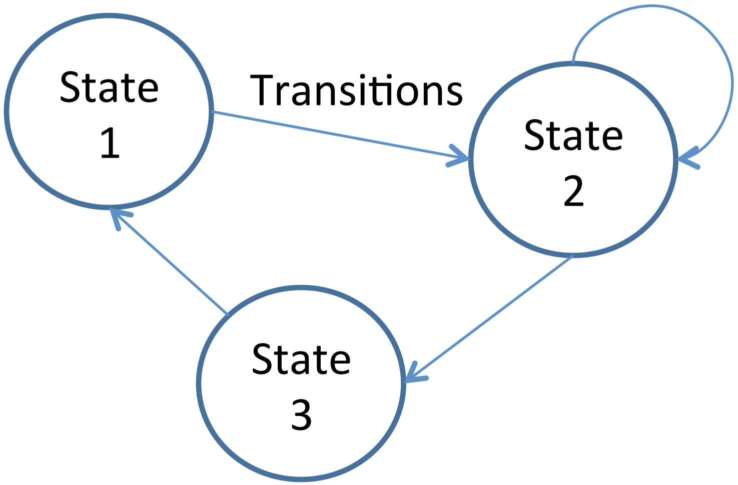
- Encapsulates utilization based triggers
  - Parameters of system calls
- Captures power behavior of ones that do not necessarily imply utilization
- Can be traced back to process, thread, function
  - Eases energy accounting

Courtesy: Pathak et al

Finite-State-Machine (FSM) as Power Model Representation

We Use Finite-State-Machine (FSM)

- Nodes: Power states
  - Base State: No activity on phone
  - Productive state: Actual utilization
  - Tail state: No-useful work
- Edges: Transition rules
  - System calls (start/completion)
  - Workload (Ex: 50 pkts/sec)
  - Timeout



Courtesy: Pathak et al

FSM Power Model Construction

- Systematic ‘Brute Force’ Approach
  - Step 1 : Model Single System Call
  - Step 2 : Model Multiple System Calls for Same Component
  - Step 3 : Model Multiple Components (Entire Phone)
- Requires domain knowledge
  - Semantics of system calls

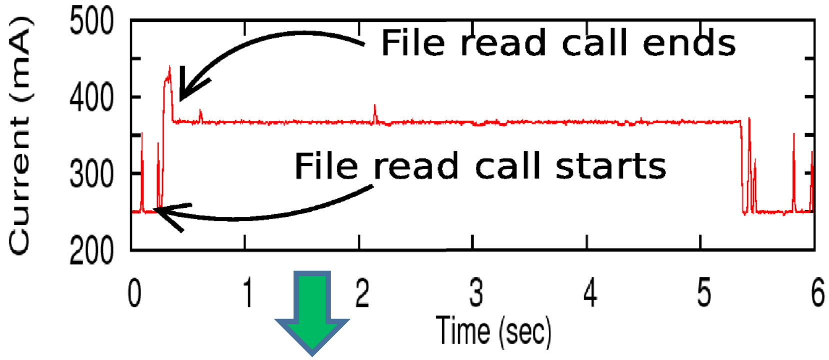
Courtesy: Pathak et al

Step 1: Single System Call FSM

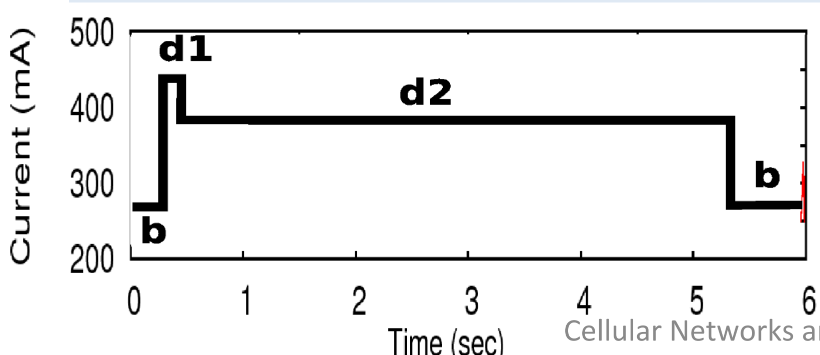
WM6.5 on HTC Touch

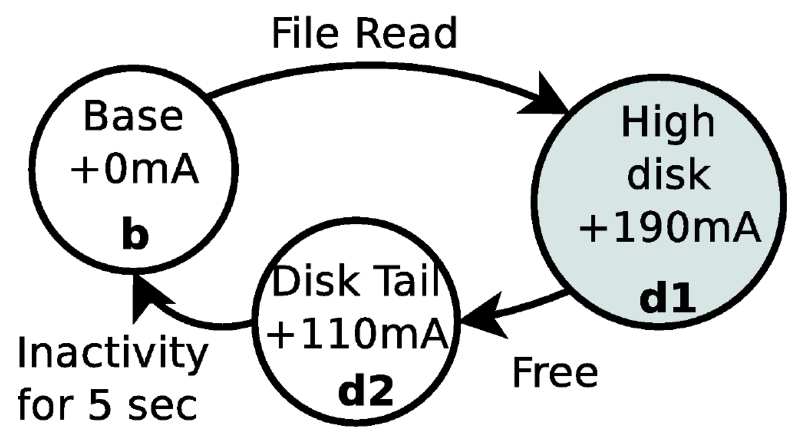
System call: **read** (fd, buf, size);

Measured power consumption + system calls (trigger)



Modeled power consumption





FSM

Courtesy: Pathak et al

Step 2: Modeling Multiple System Calls of Same Component

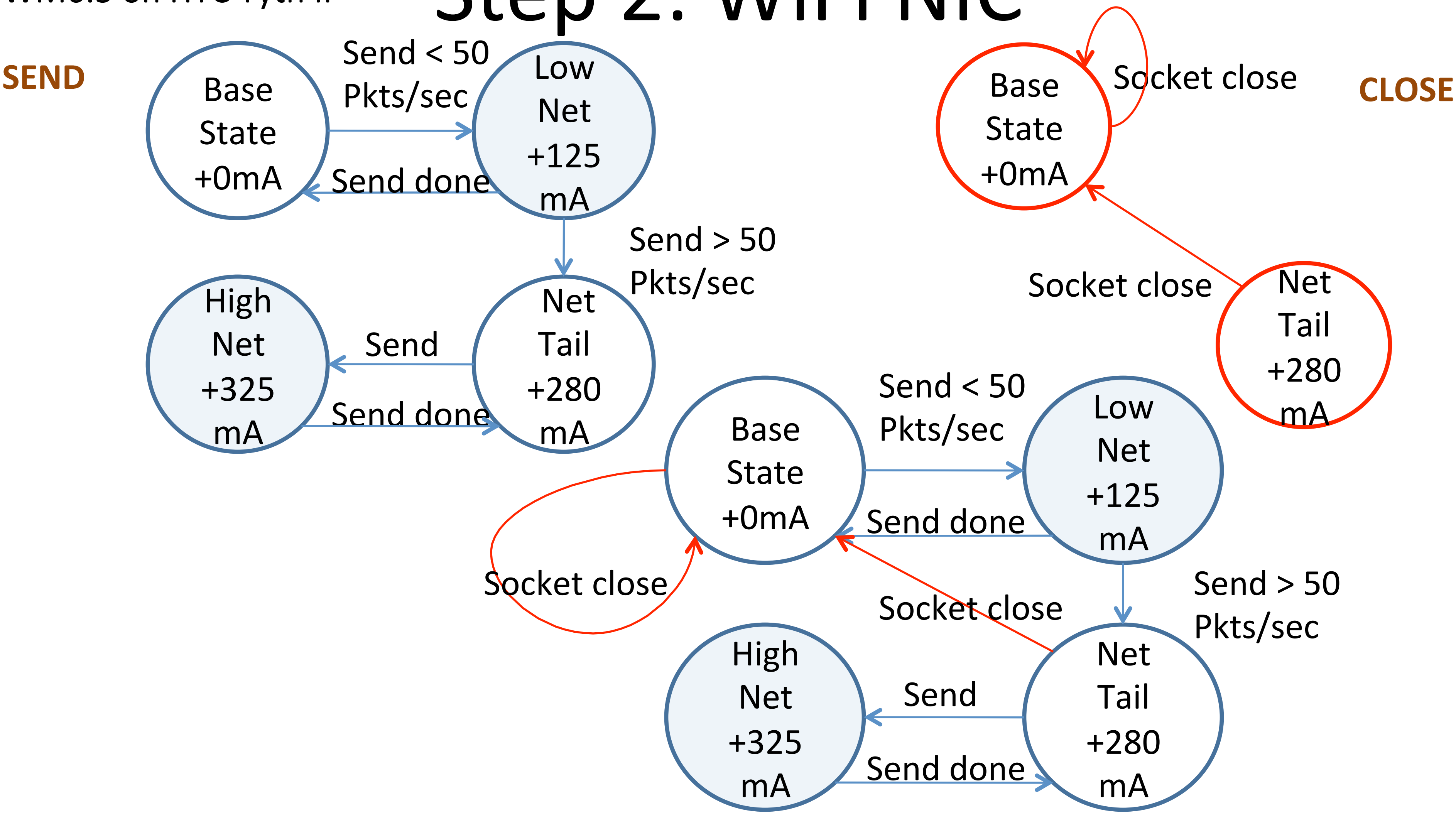
- Observation: A component can only have a small finite number of power states

- Methodology
  - Identify and merge similar power states
  - Obey programming order
  - Model concurrent system calls

Courtesy: Pathak et al

WM6.5 on HTC Tytn II

Step 2: WiFi NIC



Courtesy: Pathak et al

Step 3: Modeling Multiple Components

- Observation: Different components may interact with each other’s power consumption

- Methodology
  - Try to reach different combination of states
  - Construct new states and transitions in FSM

Courtesy: Pathak et al

Implementation

- Windows Mobile 6.5
  - Extended CeLog
- Android
  - System Tap: Logs kernel events
  - Android debugging framework: Custom logging in Dalvik VM

Courtesy: Pathak et al

Evaluation: Handsets Used

HTC Tytn II

Windows Mobile Win 6.5 (CE 5.2)

HTC Touch

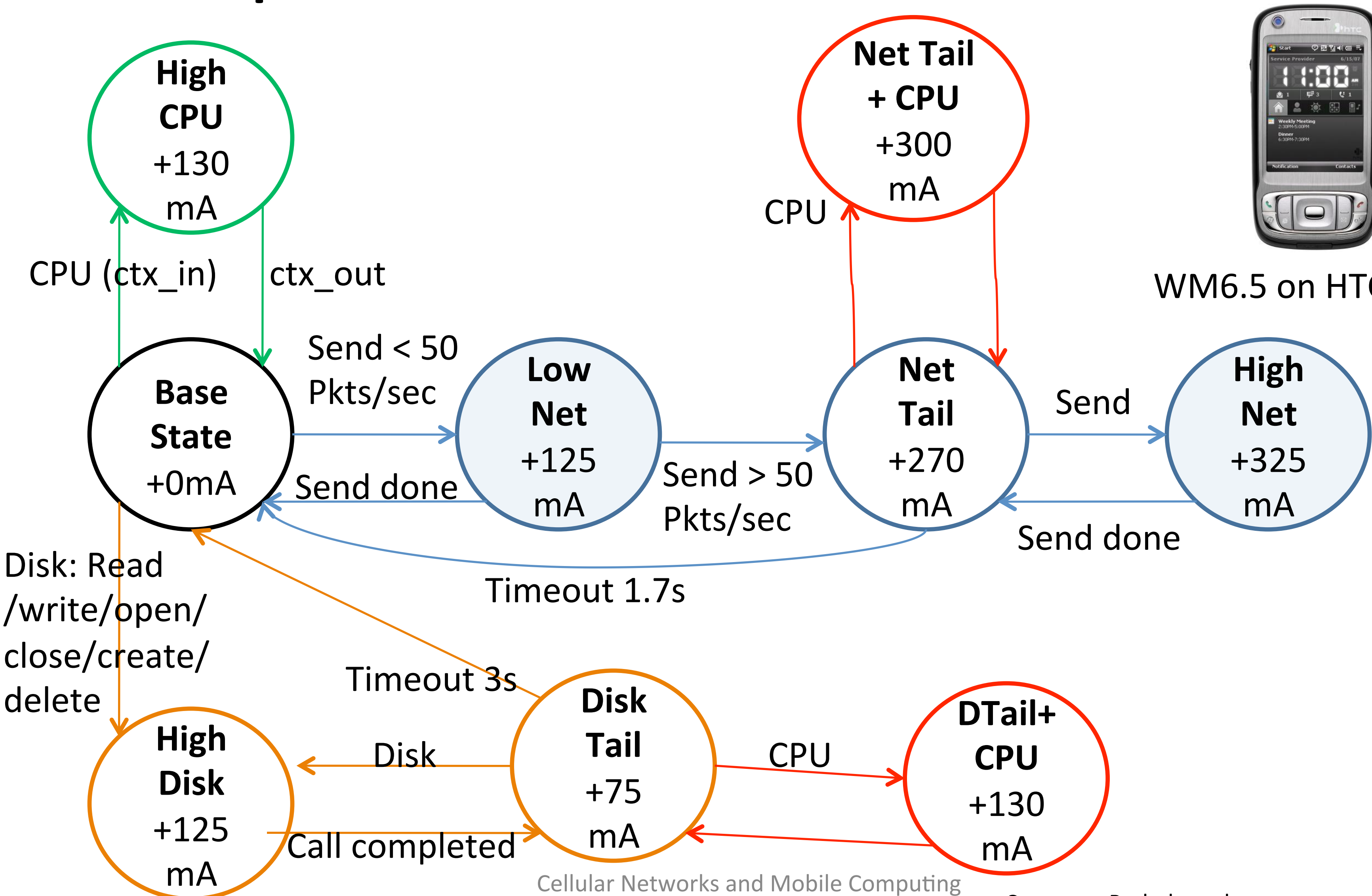
Windows Mobile Win 6.5 (CE 5.2)

HTC Magic

Android (Linux 2.6.34)

Courtesy: Pathak et al

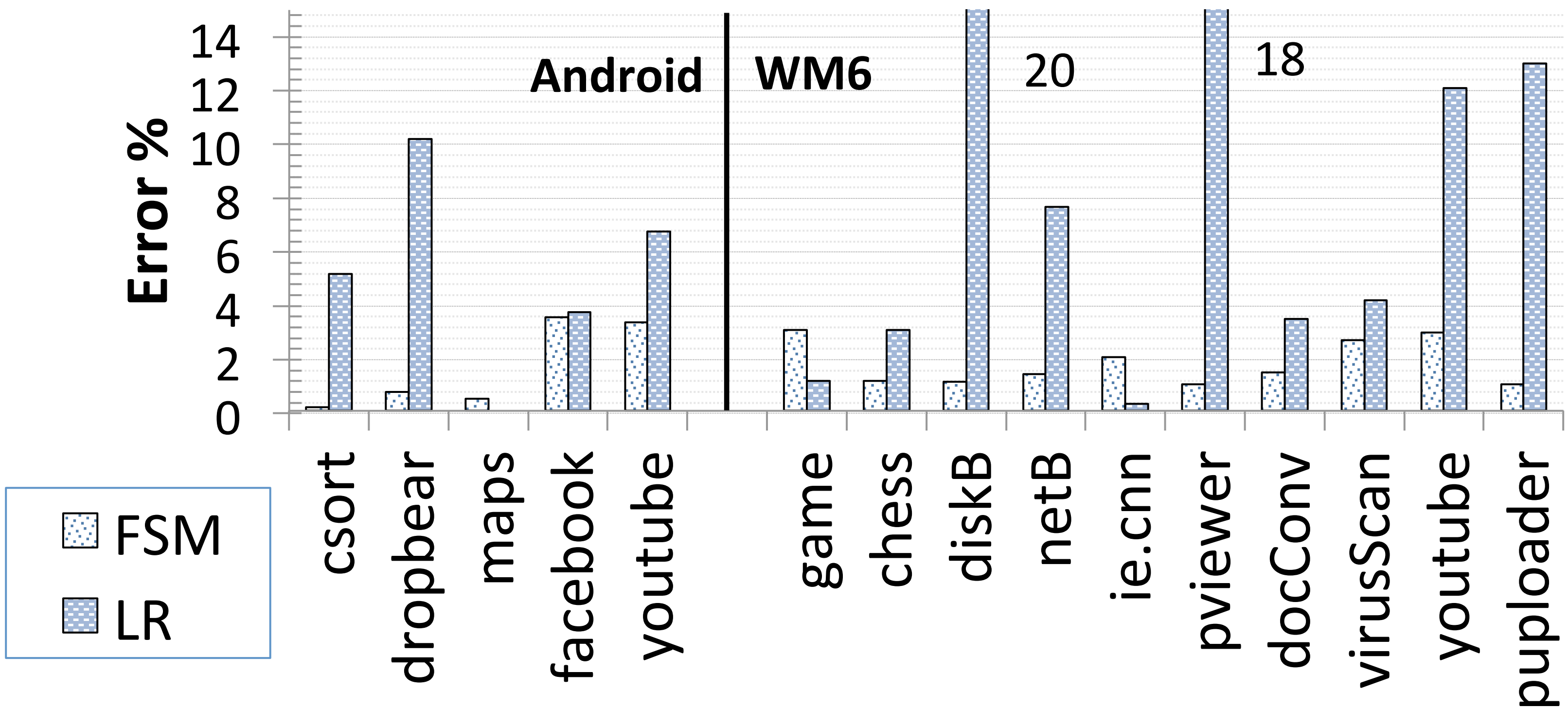
Snapshot of FSM for Entire Phone



WM6.5 on HTC Tytn II

Courtesy: Pathak et al

End-To-End Energy Estimation Error

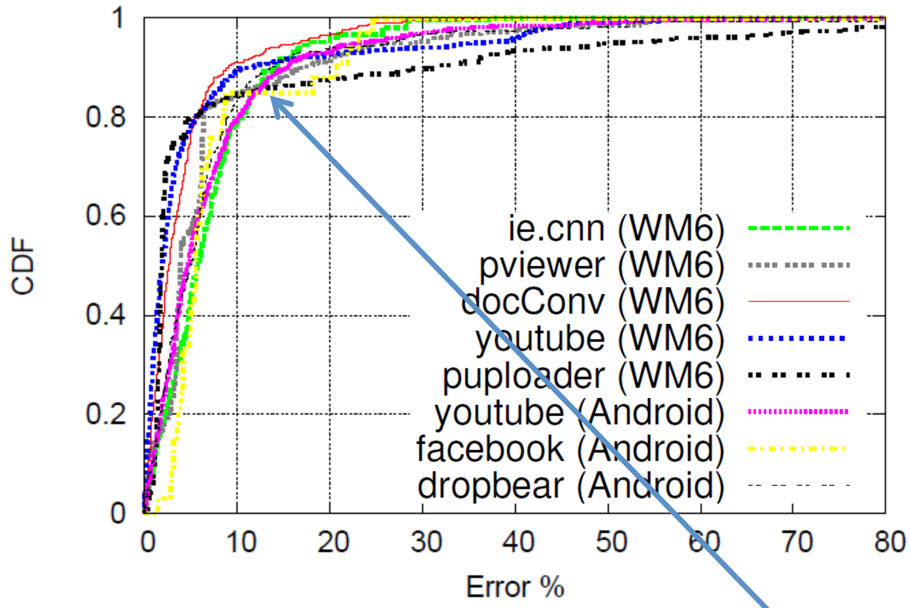


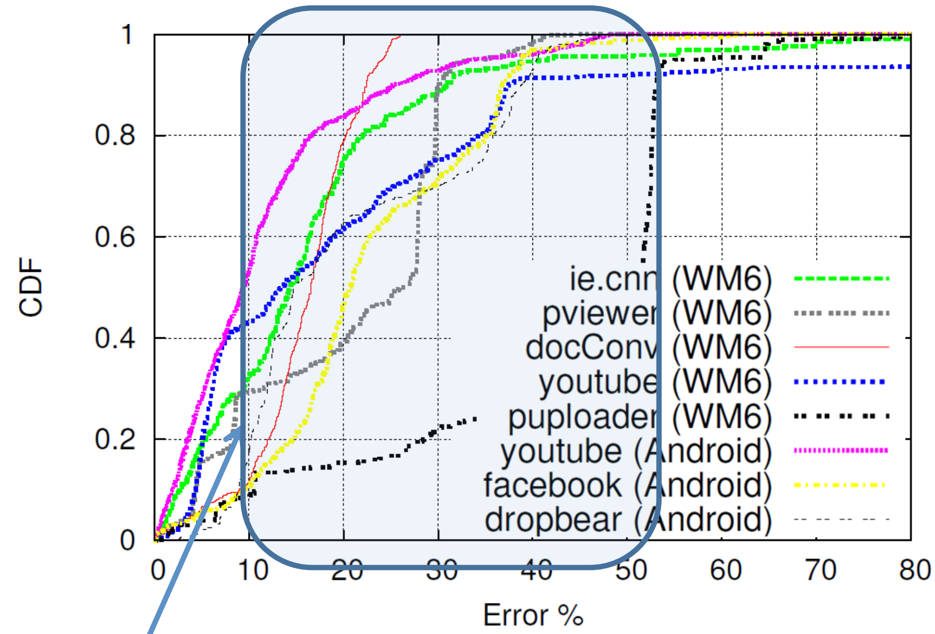
FSM: under 4%
LR: 1% – 20%

Courtesy: Pathak et al

Fine-Grained Energy Estimation

CDF of energy estimation error per 50ms time interval





FSM based on System calls

Linear Regression (State-of-art)

FSM: 80th percentile error less than 10% for all apps
LR: 10th percentile error less than 10% for all apps

Courtesy: Pathak et al

Outline

- The Rise of Ebugs
- Methods of Measuring Power Usage
- Power Models
  - Usage based
  - System call trace based
- Profiling
- Conclusion

Energy Profiling

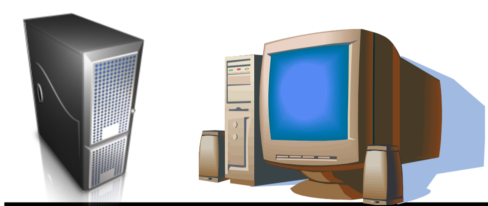
- eprof published in Eurosys 2012
- QCOM Trepn Profiler
  - Trepn leverages hardware sensors built into the Snapdragon MDP
  - Analyze power consumption of hardware blocks in the Snapdragon MDP, including:
    - CPU (system and auxiliary)
    - GPS
    - Bluetooth
    - Camera
    - Audio
    - Memory
    - Network data (optimizes data transfer frequency)

Courtesy: Pathak et al

Towards Energy Profiling

- Performance

- Energy: One of the most critical issues in smartphones
- Battery energy density only doubled in last 15 yrs



1970's-80's

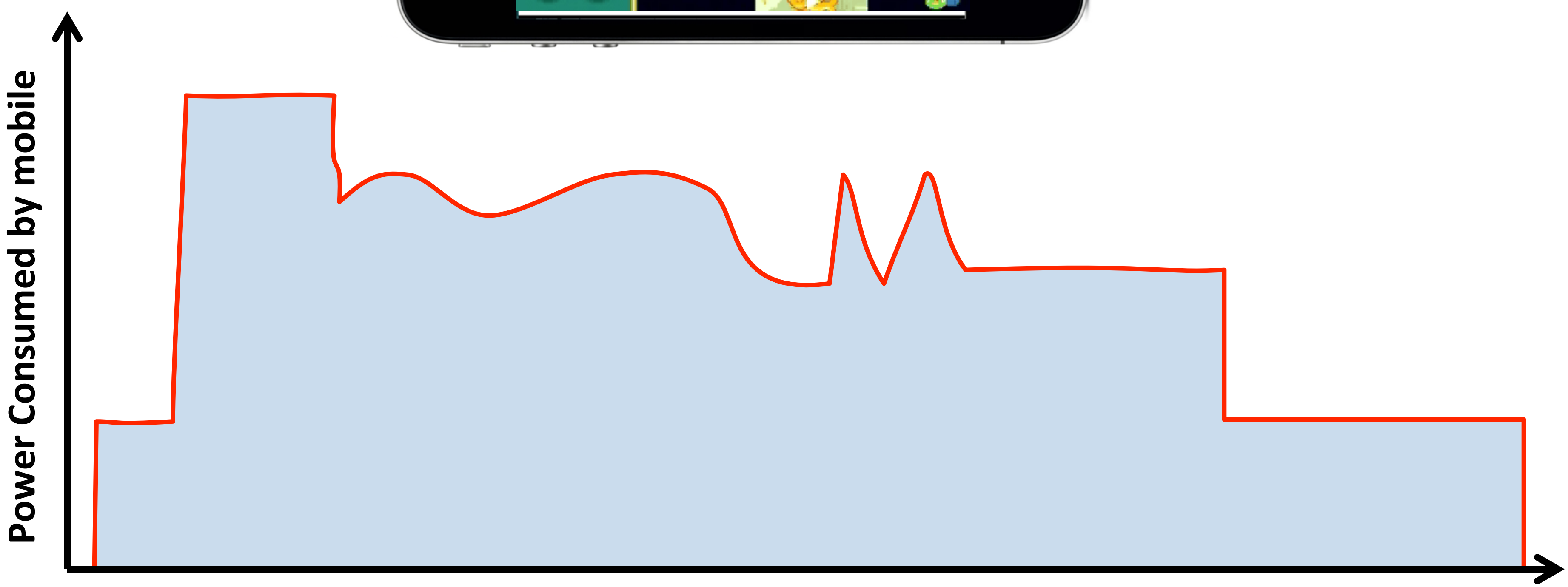
2010

Gnu Profiler (gprof PLDI'82)

Energy Profiler…?

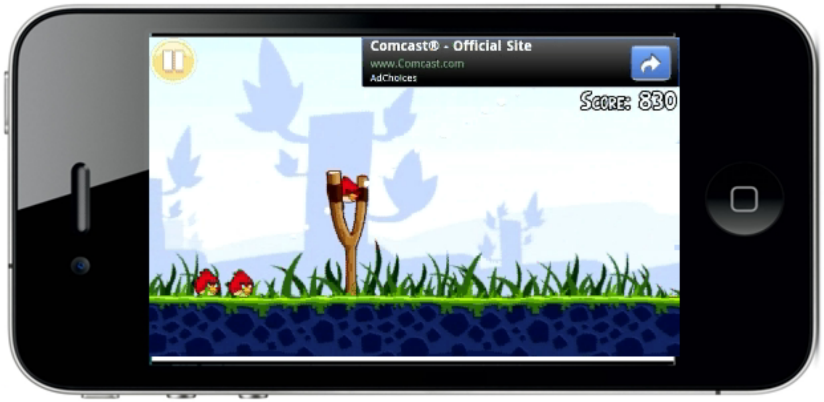
Courtesy: Pathak et al

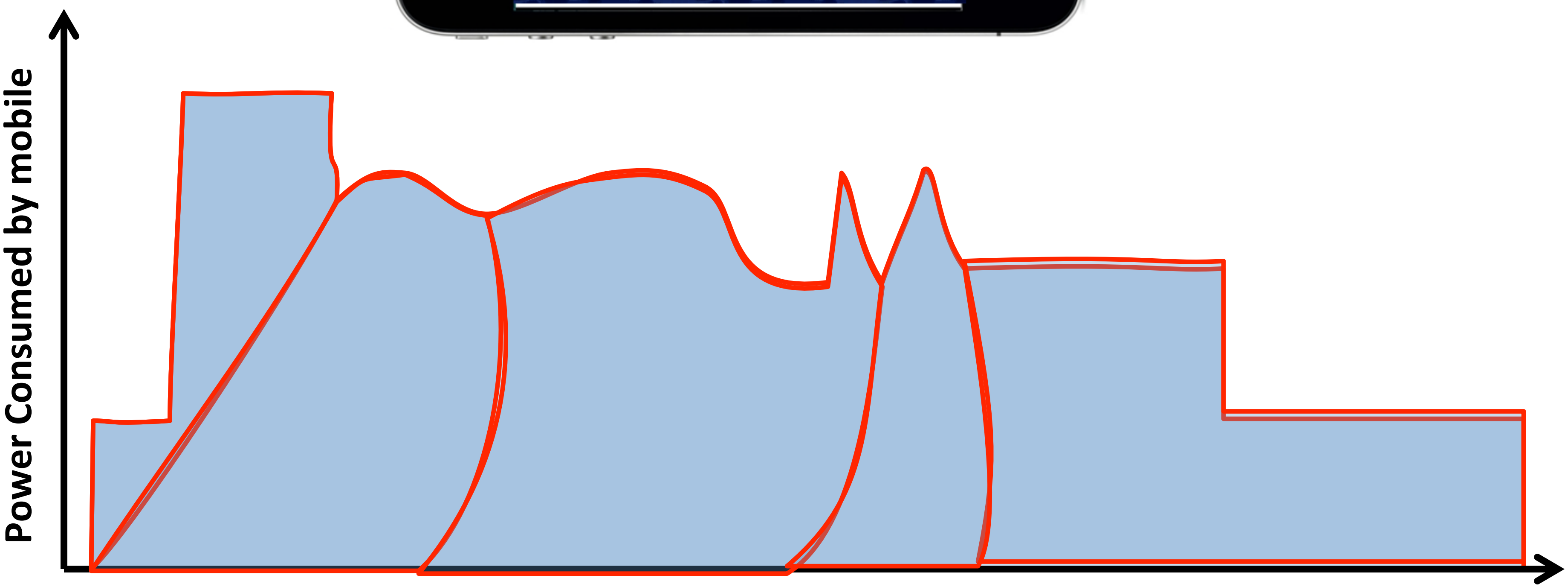
Approach (a) : Power Meters



Courtesy: Pathak et al

Approach (a) : Power Meters





Courtesy: Pathak et al

Approach (b) : Software Energy Profiling

- Tracking power activities

- Tracking app activities

- Mapping power activities to app activities

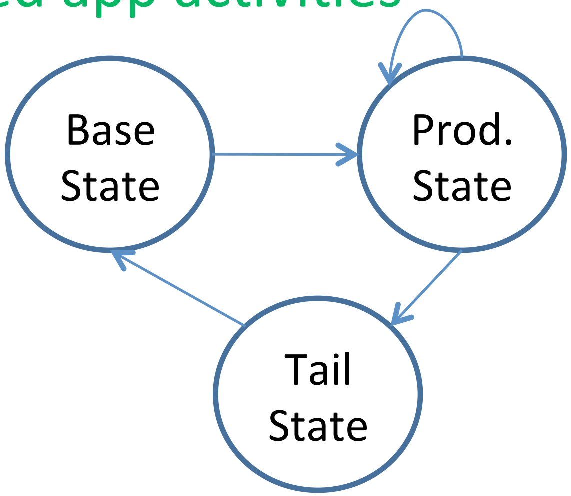
Courtesy: Pathak et al

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



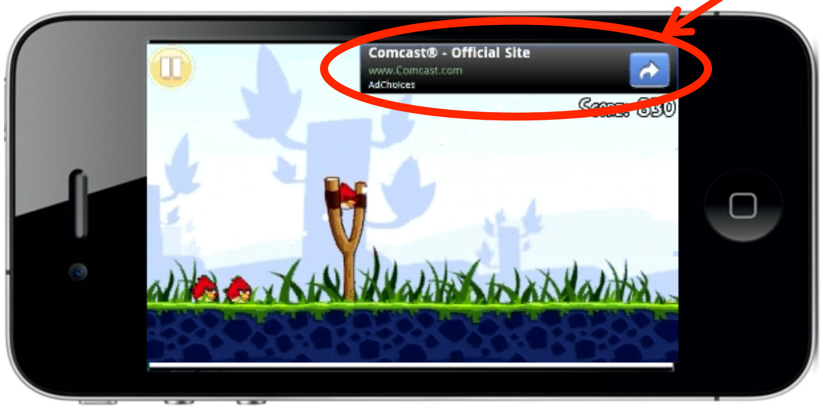
Courtesy: Pathak et al

Tracking App Activities

- Granularity of Energy Accounting

Multi Threading:

Third Party Ad Module



Multiple Routines:

Collect information
Upload information
Download ads

Multi Processing:

Courtesy: Pathak et al

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

Courtesy: Pathak et al

Where is Energy Spent Inside My App?

- ✔ Tracking power activities
  - System call based online power model
- ✔ Tracking app activities
  - IO: Backtrace system calls
  - CPU: Just like gprof [PLDI '82]
- Mapping power activities to app activities
  - Accounting Policy: Lingering Energy Consumption

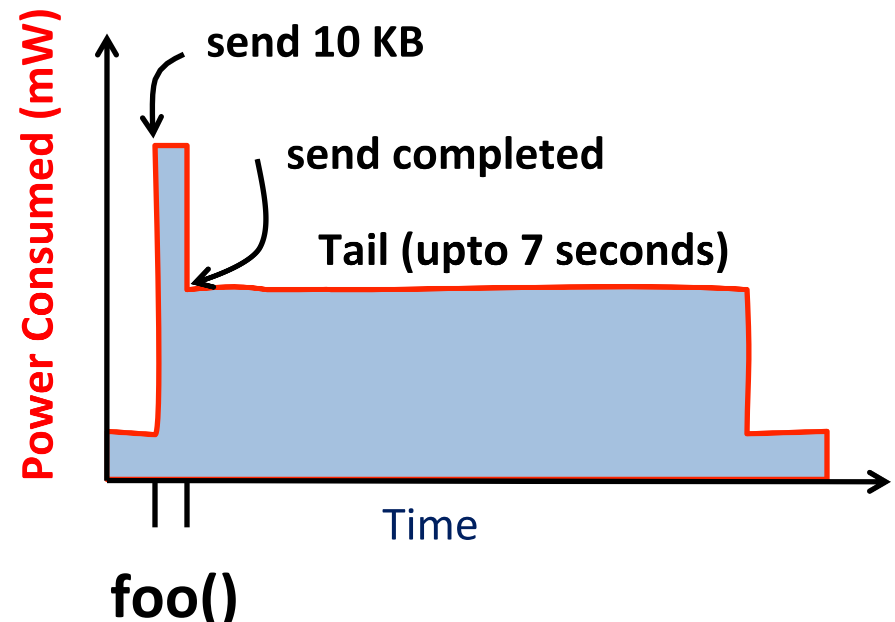
Courtesy: Pathak et al

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**



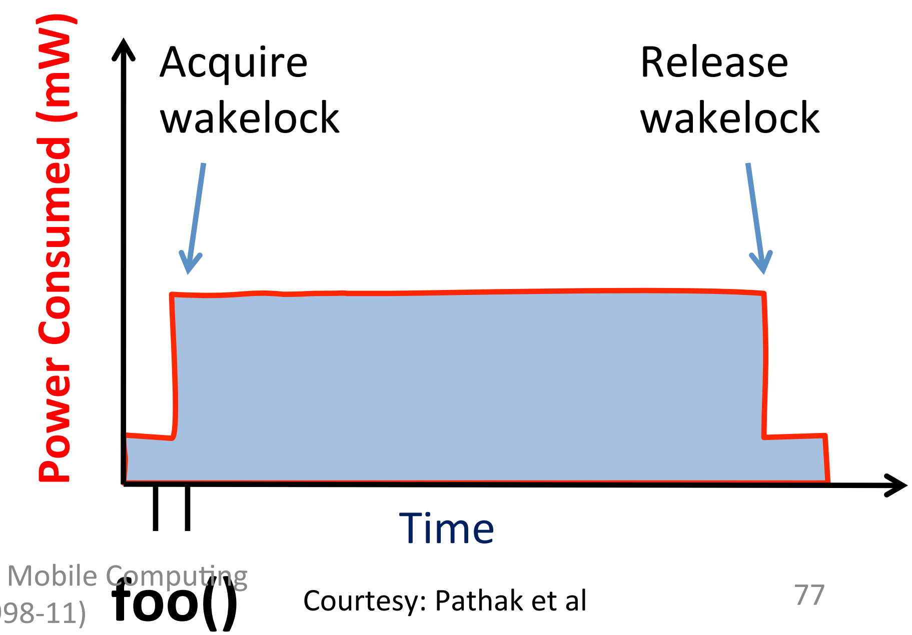
Courtesy: Pathak et al

Lingering Energy Consumption (b) Persistent State Wakelock

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

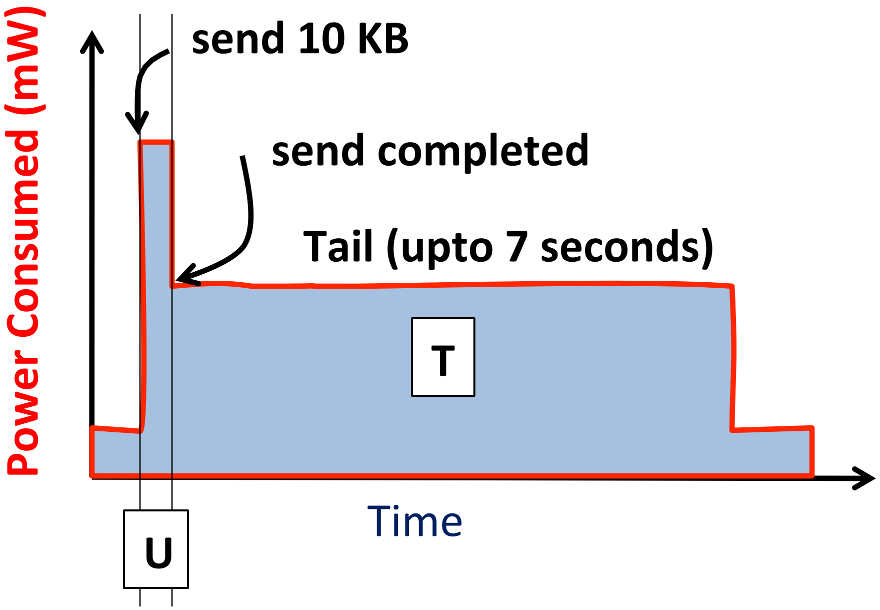
- Power encumbered Programming: Programmer has to manage sleep/wake cycle of components

Keep the screen on !



Courtesy: Pathak et al

Lingering Energy Consumption
Case 1: Single Call Single Tail

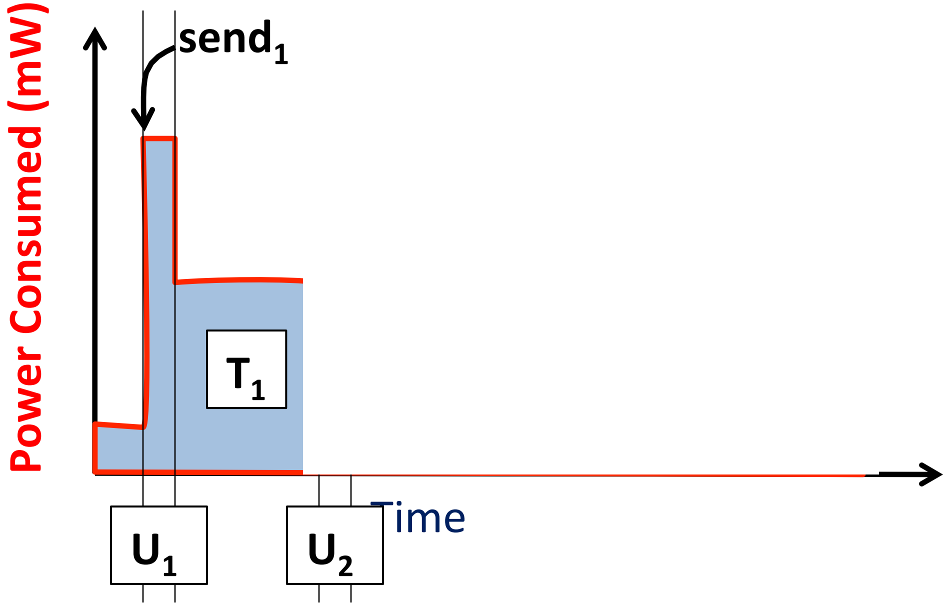


1. Energy represented in terms of an energy tuple **(U, T)**

2. (U, T) is attributed to entity (s) containing send system call

Courtesy: Pathak et al

Lingering Energy Consumption Case 2: Multiple Calls Multiple Tails

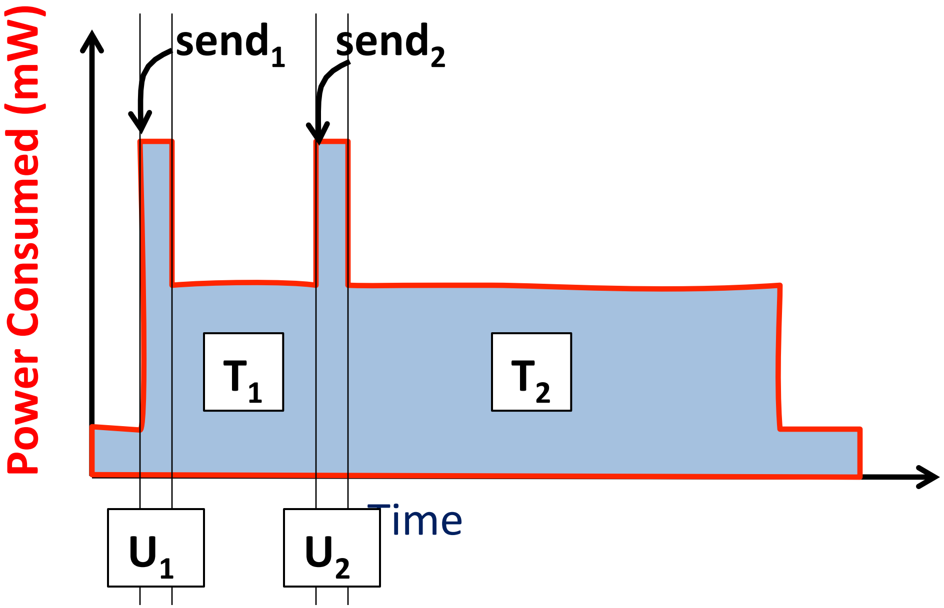


How to split tail T_2 among?

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~~

Courtesy: Pathak et al

eprof System
(Android and Windows Mobile)

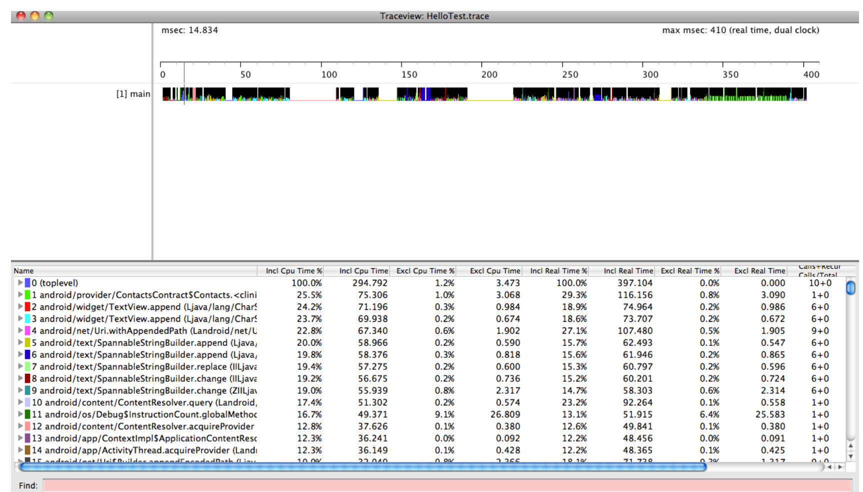
Logging Overhead:
2-15% Run Time,
1-13% Run Energy

Courtesy: Pathak et al

eprof Implementation

- SDK routine tracing: extend Android routine profiling framework
  - http://developer.android.com/reference/android/os/Debug.html
- NDK routine tracing: use gprof port of NDK
  - http://code.google.com/p/android-ndk-profiler/
- System call tracing: insert ADB logging APIs in framework code and log CPU (sched.switch) scheduling event in kernel using systemtap
  - http://www.cyanogenmod.com/

```
AndroidManifest.xml
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE
" />

MainActivity.java
import android.os.Debug;
---
public void onCreate(Bundle savedInstanceState) {
…
Debug.startMethodTracing("HelloTest");
…
Debug.stopMethodTracing();
}

adb pull /mnt/sdcard/HelloTest.trace .
traceview HelloTest.trace
```



eprof Implementation (Cont'd)

- Augmented TraceView in Dalvik
  - gprof-like tracing + syscall tracing

- eprof API: *StartEnergyTracing()*
  StopEnergyTracing()
  - Needs app recompile

Modified Android framework to trace each app by default
(no need for app source)

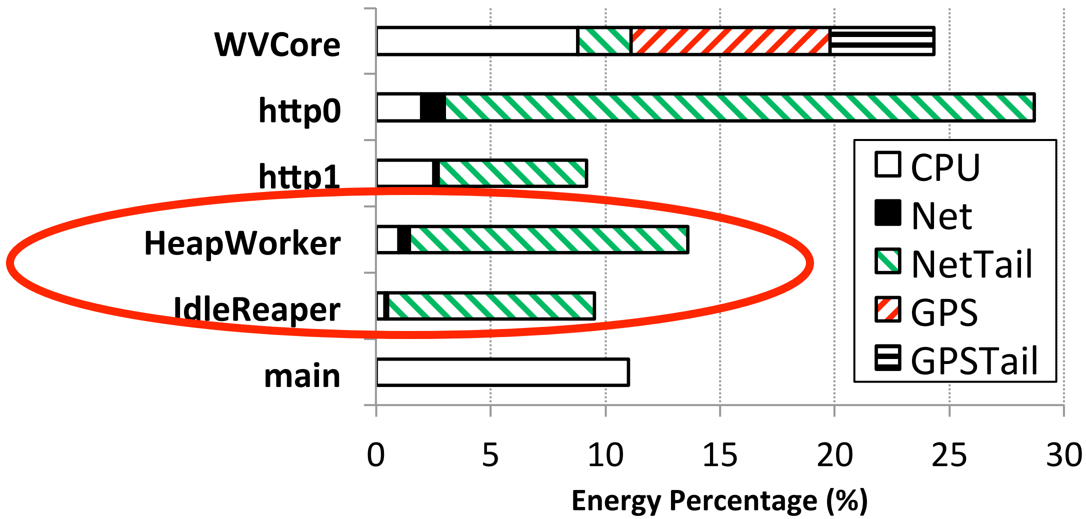
Courtesy: Pathak et al

Using eprof

App	Num Routine calls	Num Threads	3^{rd} Party Modules
Browser	1M	34	--
AngryBirds	200K	47	FLURRY KHRONOS GROUP
FChess	742K	37	adwhirl
NYTimes	7.4M	29	FLURRY
MapQuest	6M	43	SKYHOOK WIRELESS AOL

Courtesy: Pathak et al

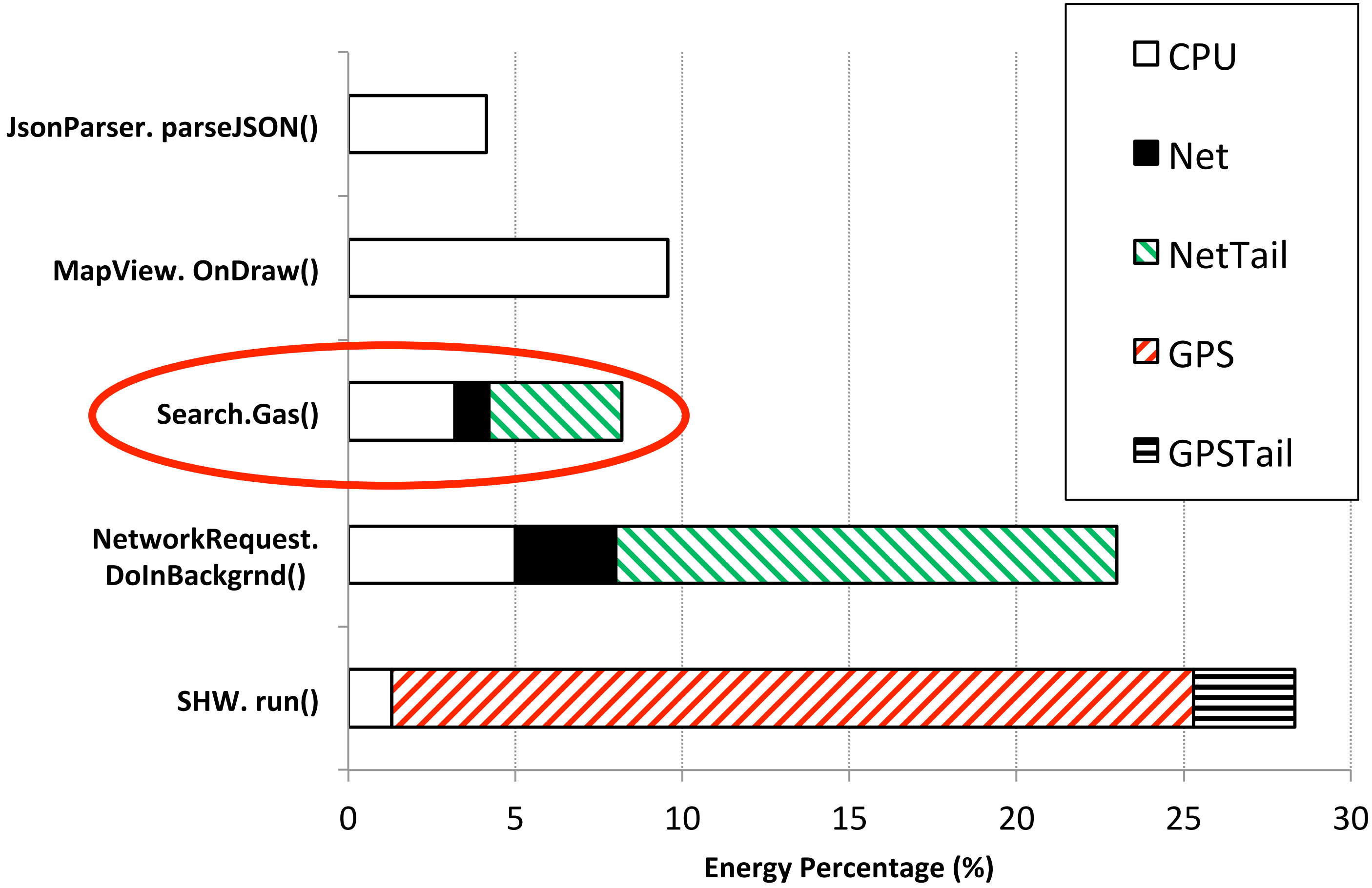
Case Studies: (a) Android Browser Google Search



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

Courtesy: Pathak et al

Case Studies: (b) Map Quest

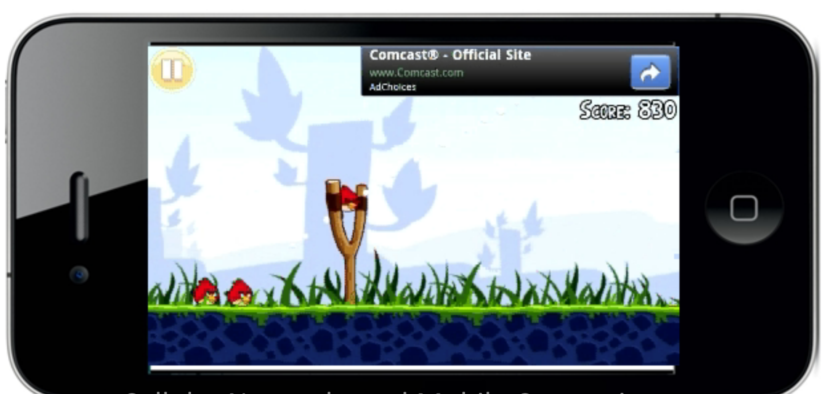


Courtesy: Pathak et al

Case Studies: (c) – Angry Birds

Google Nexus (3G)

Activity	Energy %
User Tracking & Uploading Information	45%
Fetching Ads	28%
Game Play	20%

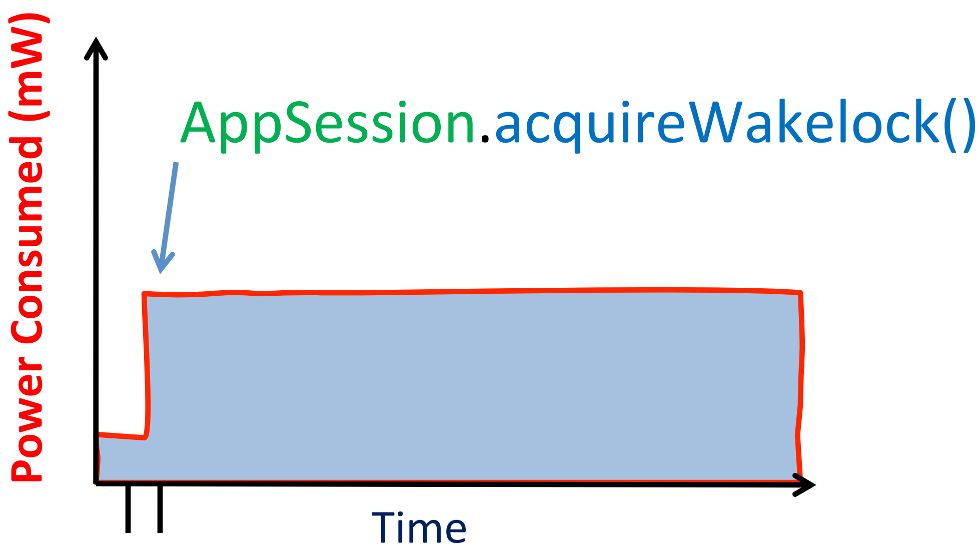


Courtesy: Pathak et al

Case Studies (d): Facebook Wakelock Bug

Google Nexus (WiFi)

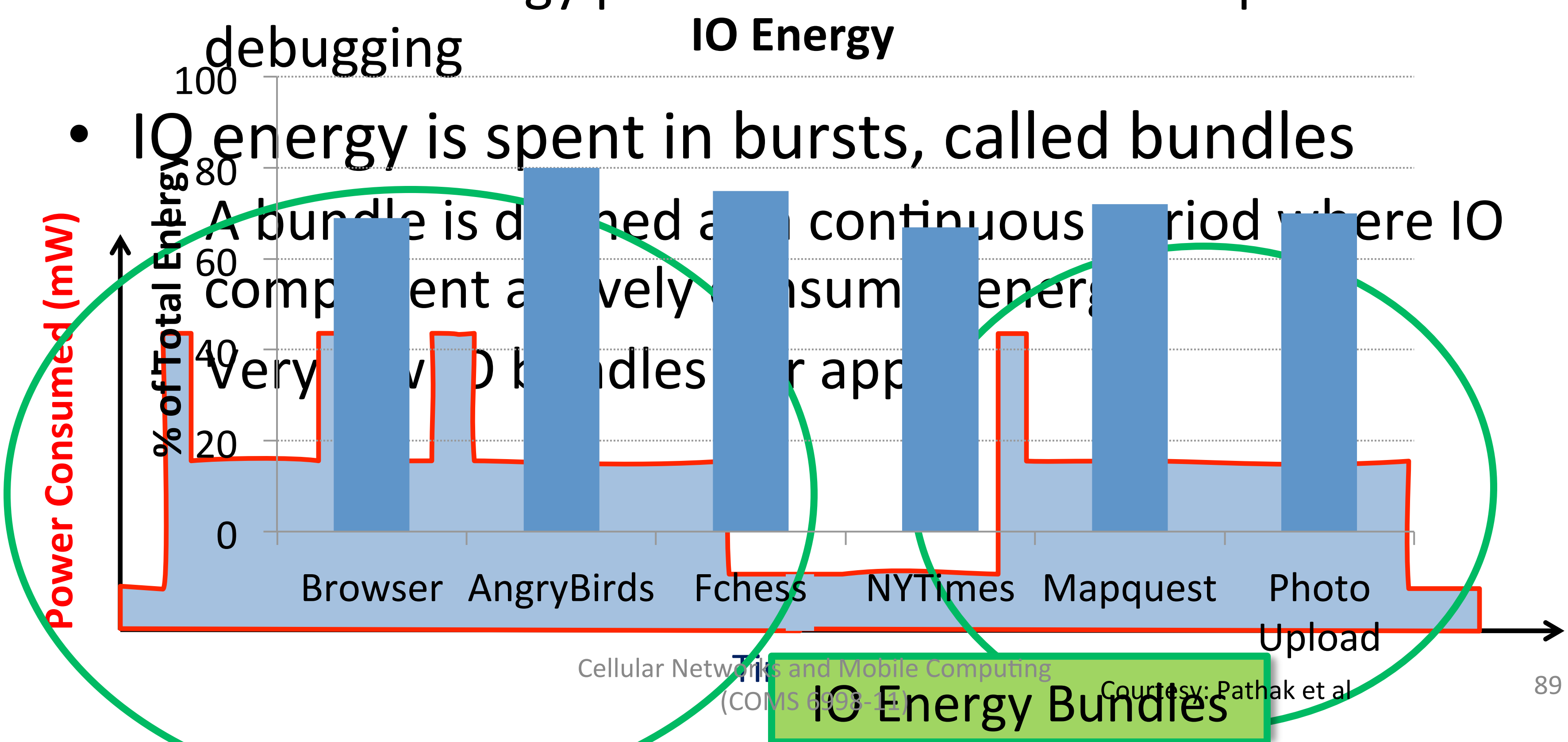
FaceBookService: 25%



Courtesy: Pathak et al

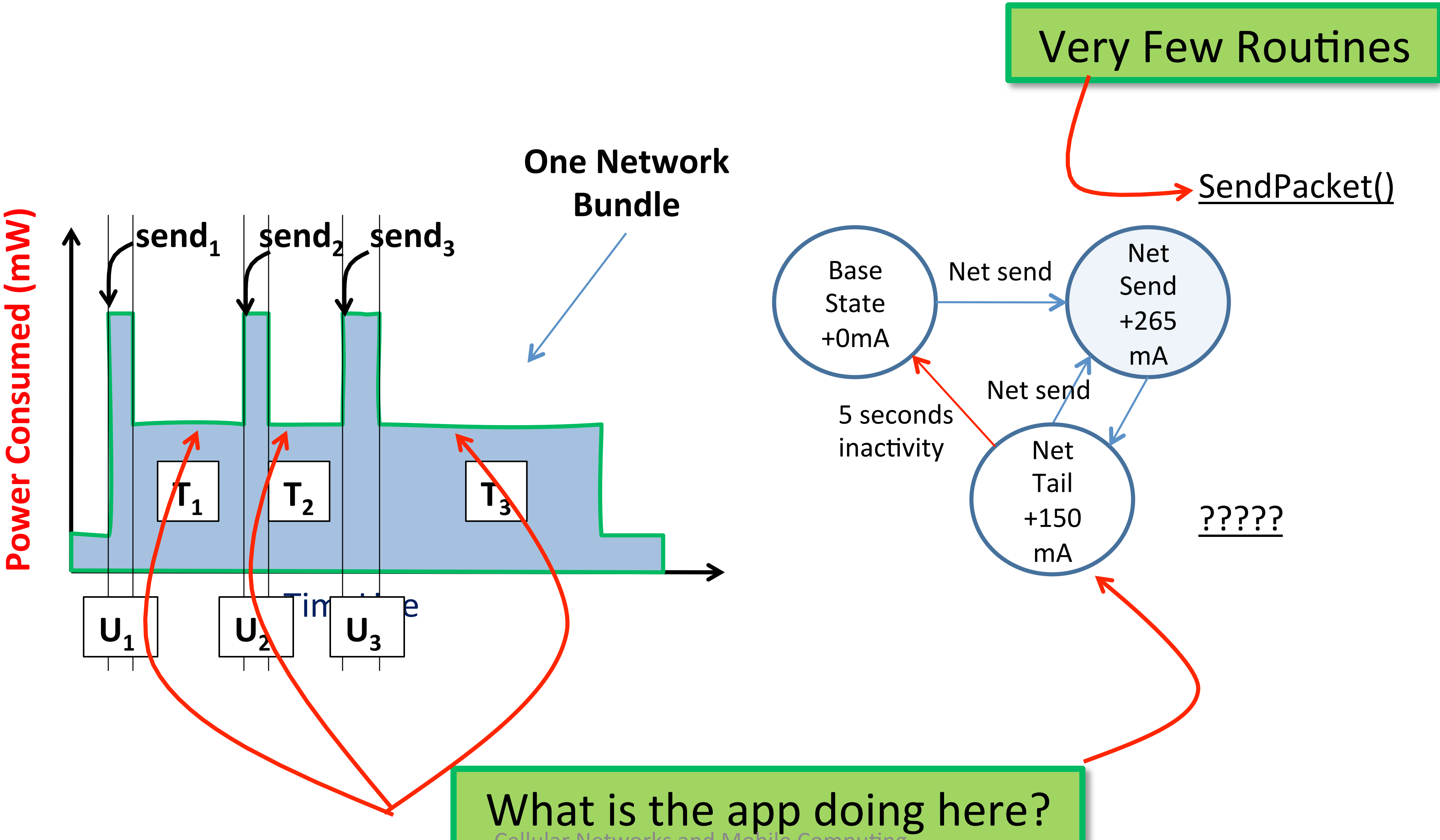
App Energy Drain Characteristics

- 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 ener
  - Very w D b dles r app



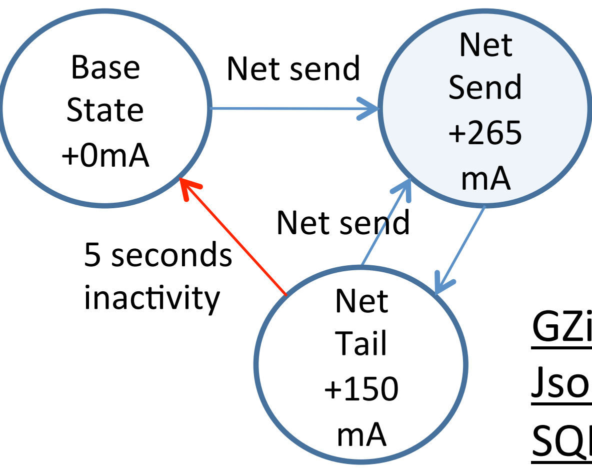
Courtesy: Pathak et al

IO Bundle Representation



Optimizing IO Energy using Bundles

Why is a bundle so long?



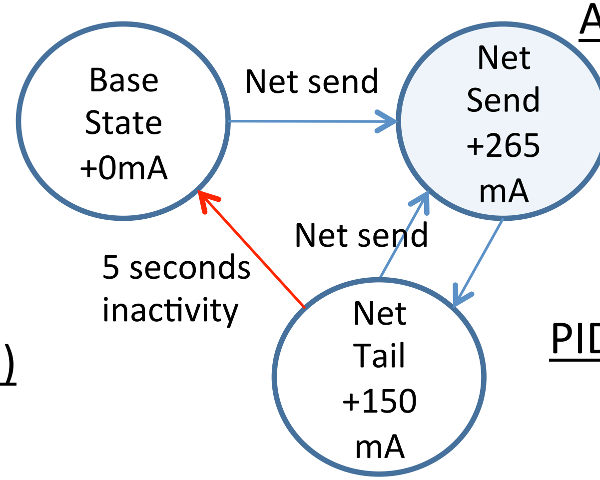
DownloadMgr.Read()

18

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

The New York Times ON THE WEB

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

Courtesy: Pathak et al

Summary

- 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

Conclusion and Future work

- Ebugs need to be dealt with
  - No-sleep ebug debugging studied
- Fine-grained energy modeling and profiling very important to pinpoint energy bottleneck and ebugs
  - Accounting is tricky
  - I/O energy consumption is a major part
- Display energy modeling and profiling is still lacking

Online Resources

- Soot: open source framework for analyzing java bytecode
  - http://www.sable.mcgill.ca/soot/
- ded: decompiling android application tool
  - http://siis.cse.psu.edu/ded/
- SDK routine tracing http://developer.android.com/reference/android/os/Debug.html
- gprof: NDK routine tracing
  - http://code.google.com/p/android-ndk-profiler/
- systemtap: system call tracing

Questions?