AppScope: Application Energy Metering Framework for Android Smartphones using Kernel Activity Monitoring

Chanmin Yoon*, Dongwon Kim, Wonwoo Jung,
Chulkoo Kang, Hojung Cha

Dept. of Computer Science
Yonsei University, Korea

USENIX ATC' 12
2012.06.15

Motivation

Q) I find it is *100mW* when I just run my app, and it is *20mW* when I do nothing. I think *80mW* is consumed by my app. But it is *200mW* I run another app B and my app also run, and it is *160mW* when I just run app B, so my app also consume *40mW*? Which one is correct?

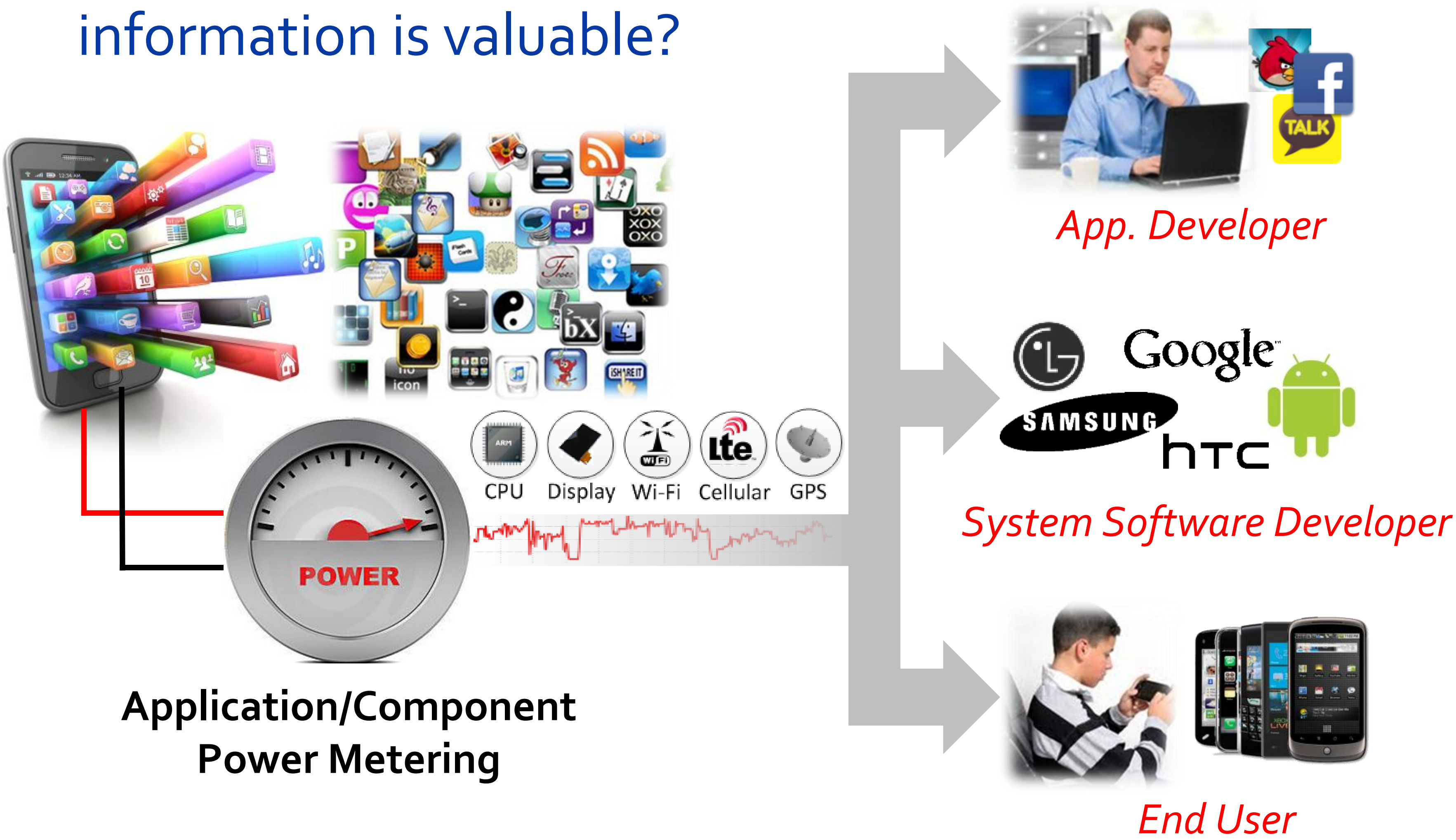
.., **So I want to know how to estimate power consumption correctly?**

A) …only use the radios when necessary. …

People want to know power consumption of their apps

Motivation

- Why application/component energy information is valuable?



Challenge

How can we estimate

Application Energy?

Power Models

- Linear regression models
  - a. MANTIS
  - b. Lasso regression
  - c. Others
- Non-linear regression models
  - a. Exponential
  - b. SVM
  - c. Others
- Finite-state machine models
  - System call-based

Utilization-based Model

$$E^{App} = \sum_{i=0}^{\#ofComp} \left(\beta_i \times x_i^{App}\right) \times d_i^{App}$$

β_i → *Power coefficient value*

x_i^{App} → *Utilization*

d_i^{App} → *Activated duration*

Hardware Component Usage

Challenge

How can we estimate
Application Energy?

Conventional methods to get Hardware component usage

- Reading hardware performance counter
  a. Very accurate
  b. Dependency on processor architecture

- Reading */proc* , */sys* file system
  a. Update rate problem
  e.g., CPU utilizations/frequencies
  b. GPS, display, cellular ?

- Using *BatteryStat class*
  a. Update rate problem
  b. Information granularity problem
  c. It's a Java class

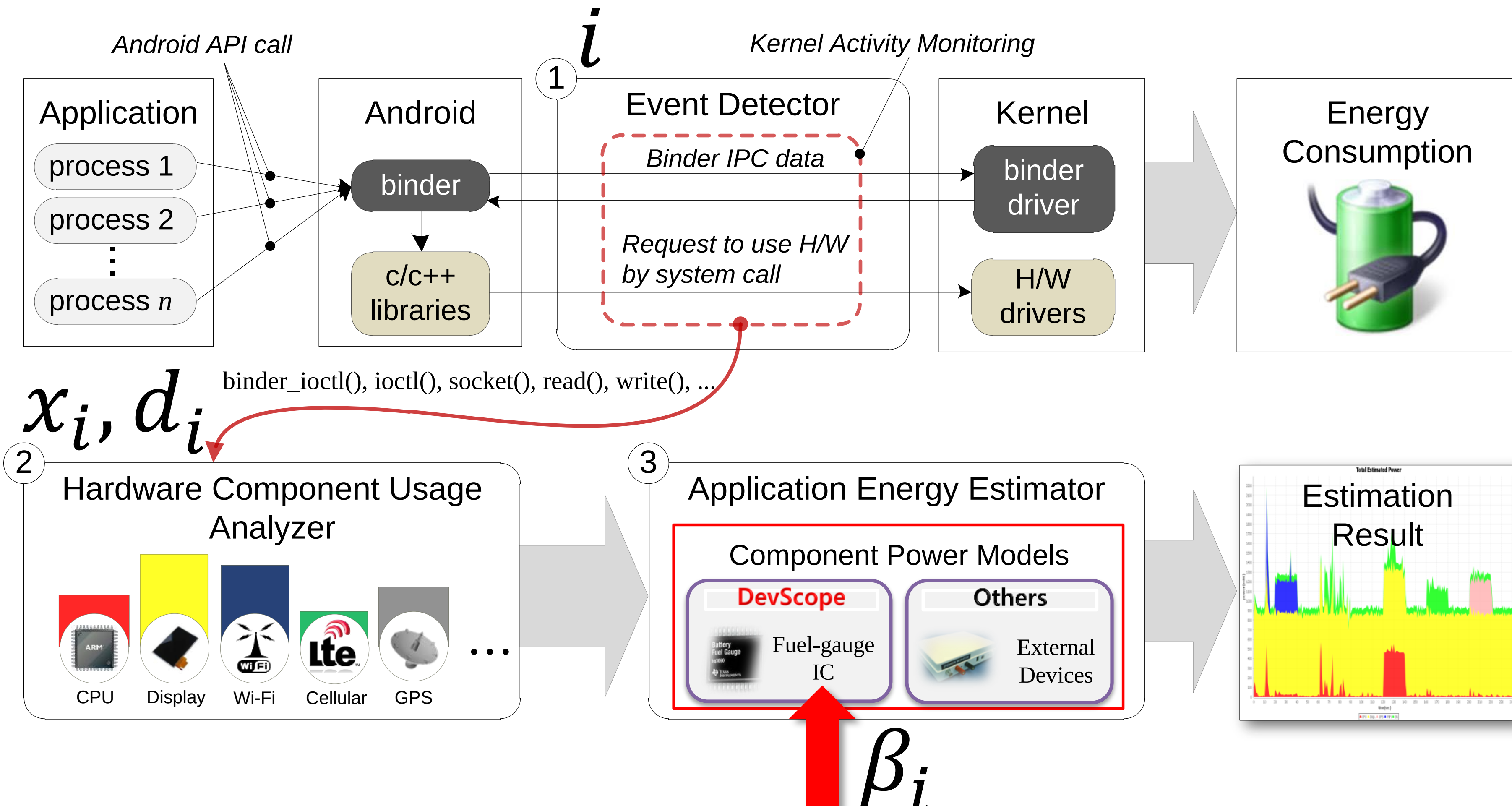
Limitations
Accuracy
Granularity
Real-time

Objectives

AppScope | **Application Energy Metering Framework for Android Smartphone**

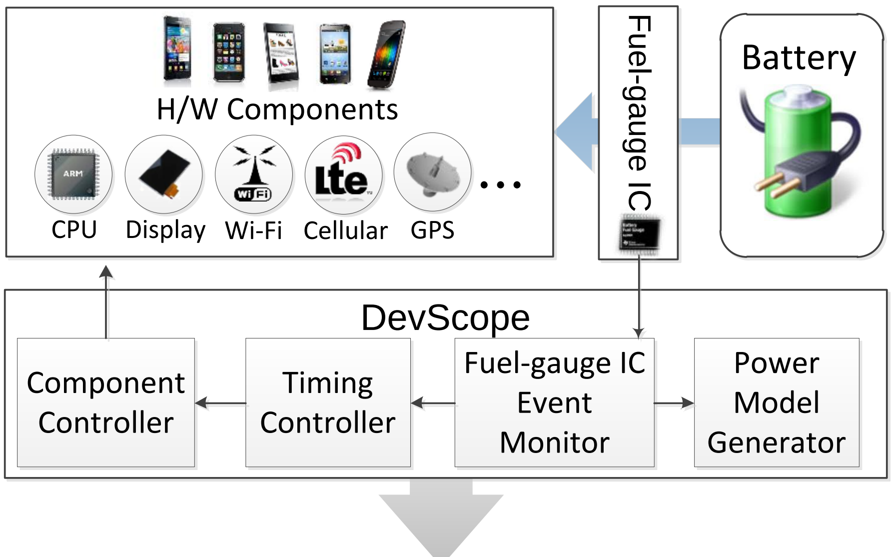
- Online and autonomous estimation in real-time
  - No external measurement device
- Fine-grained energy consumption information
  - Process & hardware component-level granularity
- System portability
  - No modification in system software

The AppScope Framework



AppScope employs the DevScope's power model

DevScope (CODES+ISSS 2012)



Component Power Model

Component	Model
CPU	$P^{CPU} = \beta^{CPU}_{freq} \times u + \beta^{idle}_{freq}$, u: utilization, $0 \le u \le 100$ freq: frequency index, $freq = 0,1,2 \cdots ,n$
LCD	$P^{LCD} = \beta^{LCD}_{b}$ b: brightness level, $\text{MIN}(level) \le b \le \text{MAX}(level)$
WiFi	$P^{WIFI} = \begin{cases} \beta^{WIFI}_{l} \times p + \beta^{base}_{l}, & if\ p \le t \\ \beta^{WIFI}_{h} \times p + \beta^{base}_{h}, & if\ p > t \end{cases}$, p: packet rate, t: threshold
Cellular(3G)	$P^{3G} = \begin{cases} \beta^{3G}_{IDLE}, & if\ RRC\ state\ is\ IDLE \\ \beta^{3G}_{FACH}, & if\ RRC\ state\ is\ FACH \\ \beta^{3G}_{DCH}, & if\ RRC\ state\ is\ DCH \end{cases}$
GPS	$P^{GPS} = \beta^{GPS}_{on},\ if\ GPS\ is\ on$

Online modeling

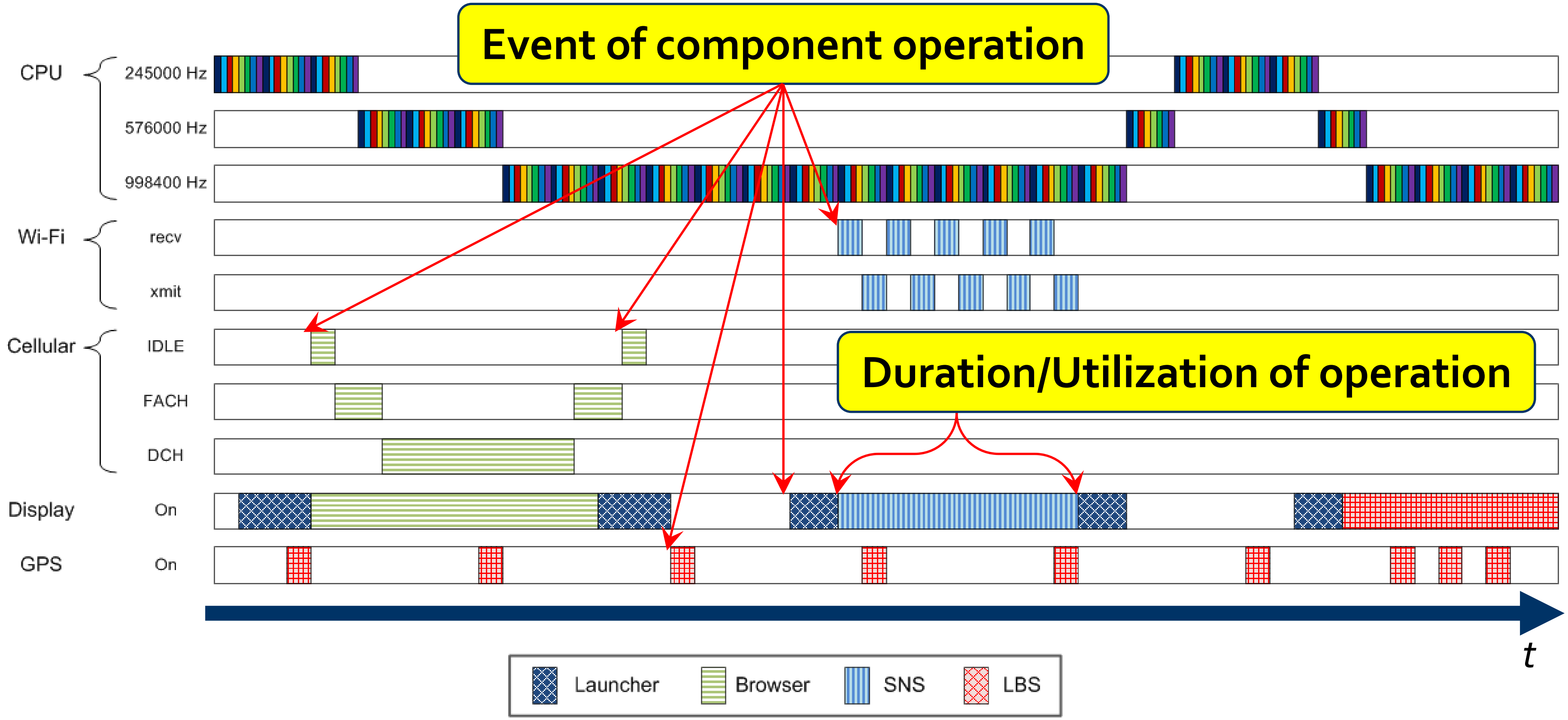
- Android application
- Assume built-in fuel-gauge IC

Non-intrusive power modeling

- Probe OS, H/W component
- Monitor fuel-gauge IC
- Component-specific
- Training set generation
  - Workload
  - Control scenario

Kernel Activity Monitoring

- How to detect hardware component operation?
  - Event-driven approach



AppScope Implementation with "*Kprobes*"

Kernel path

- System calls
- Kernel functions
- Binder calls

- Parameters
- IPC data

Instrumentation routine

Hooking Point → 0x81808c144 INT03 Break ⟷ Probe Handler → End

Return to original path

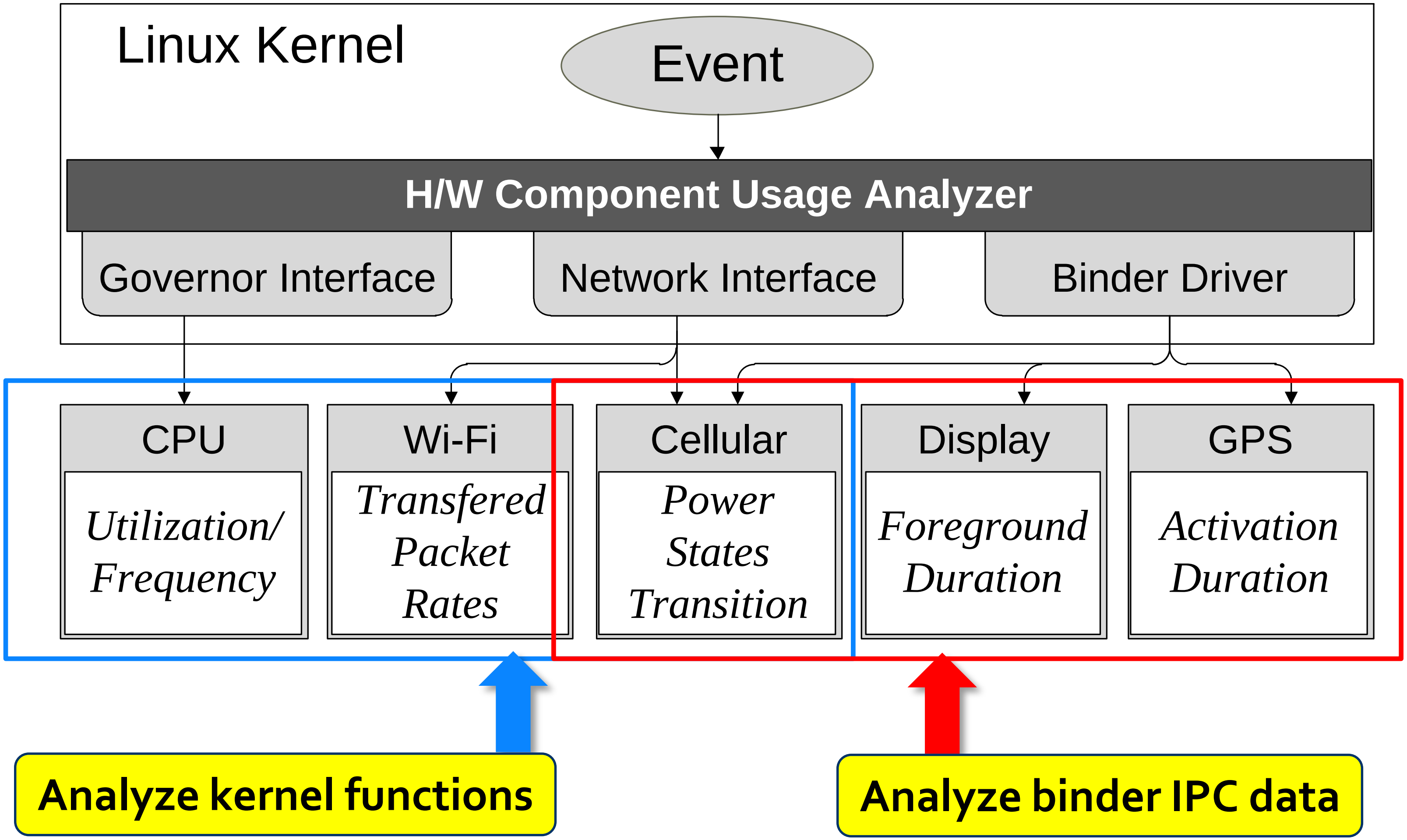
Event detector

- binder_ioctl()
- binder_transaction()
- cpufreq_cpu_put()
- sched_switch()
- dev_queue_xmit()
- netif_rx()
- …

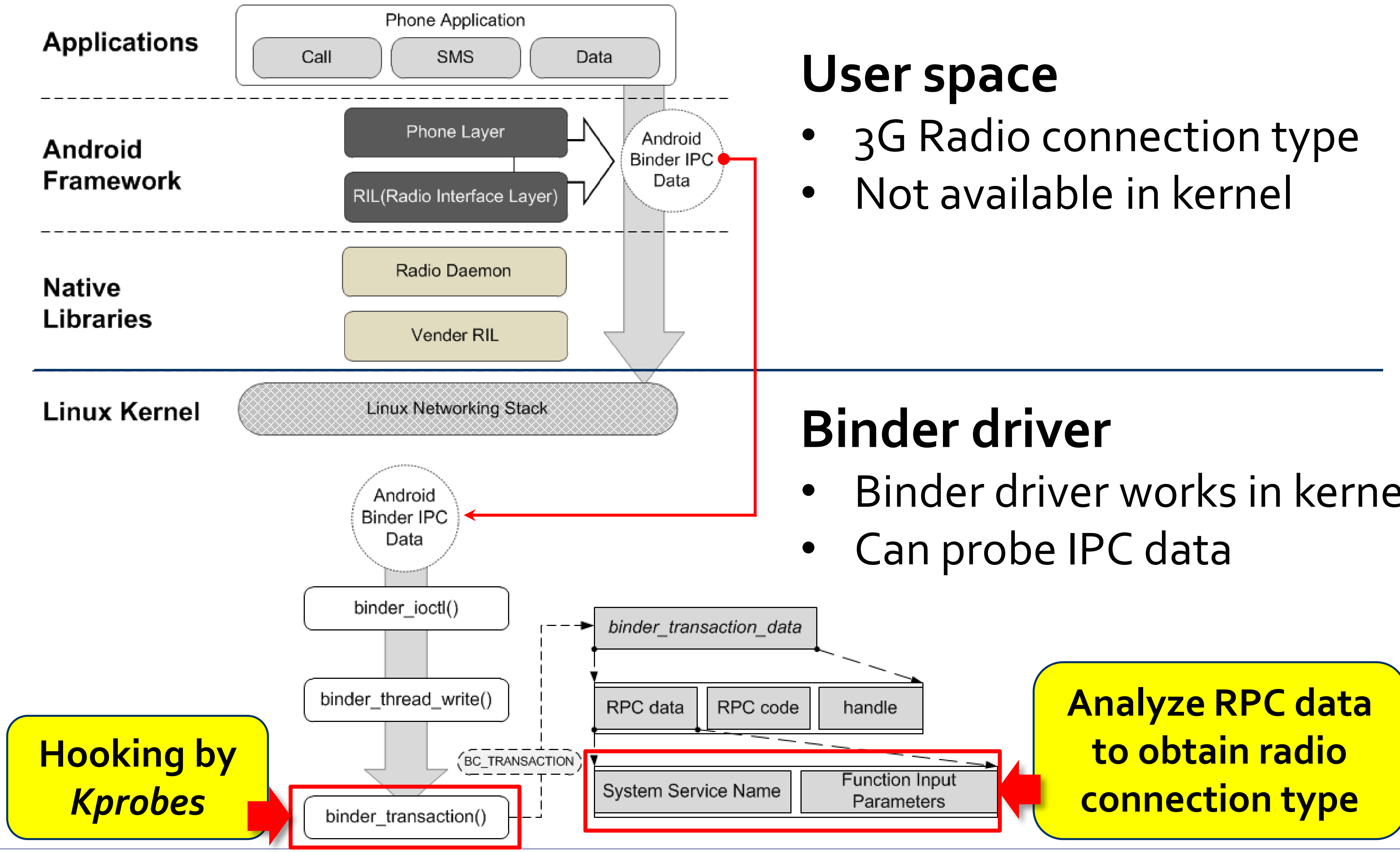
Hardware Component Usage Analyzer

- CPU frequency/utilization
- WiFi transmission packet rate
- LCD display duration
- GPS activated duration
- 3G network connection type
- …

Hardware Component Usage Analyzer



Why analyzing the Binder IPC?



User space

- 3G Radio connection type
- Not available in kernel

Binder driver

- Binder driver works in kernel
- Can probe IPC data

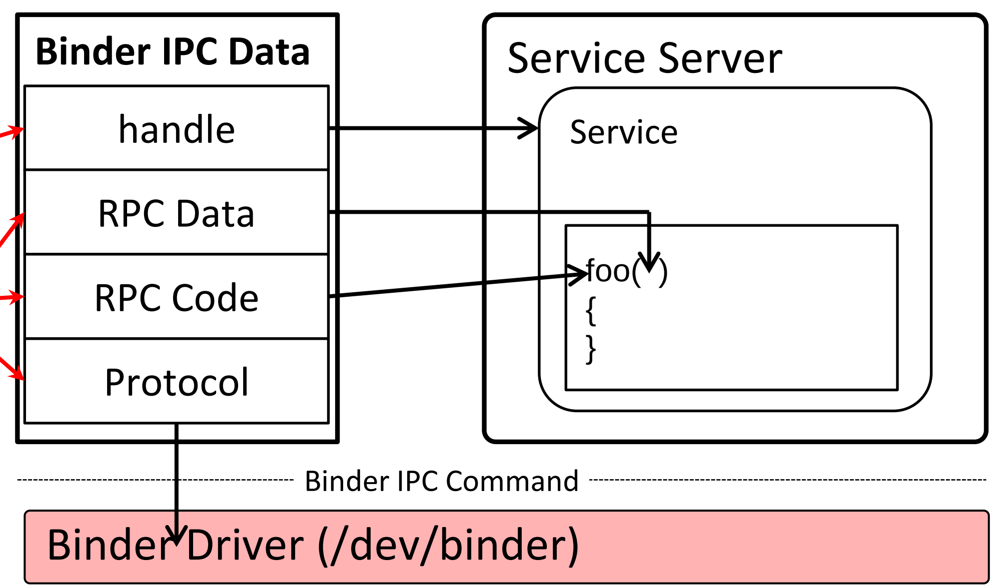
Inspecting Binder IPC data

1. **Hook binder_transaction()**
2. **Extract RPC code/data**
3. **Check RPC Code**
4. **Read RPC Data**

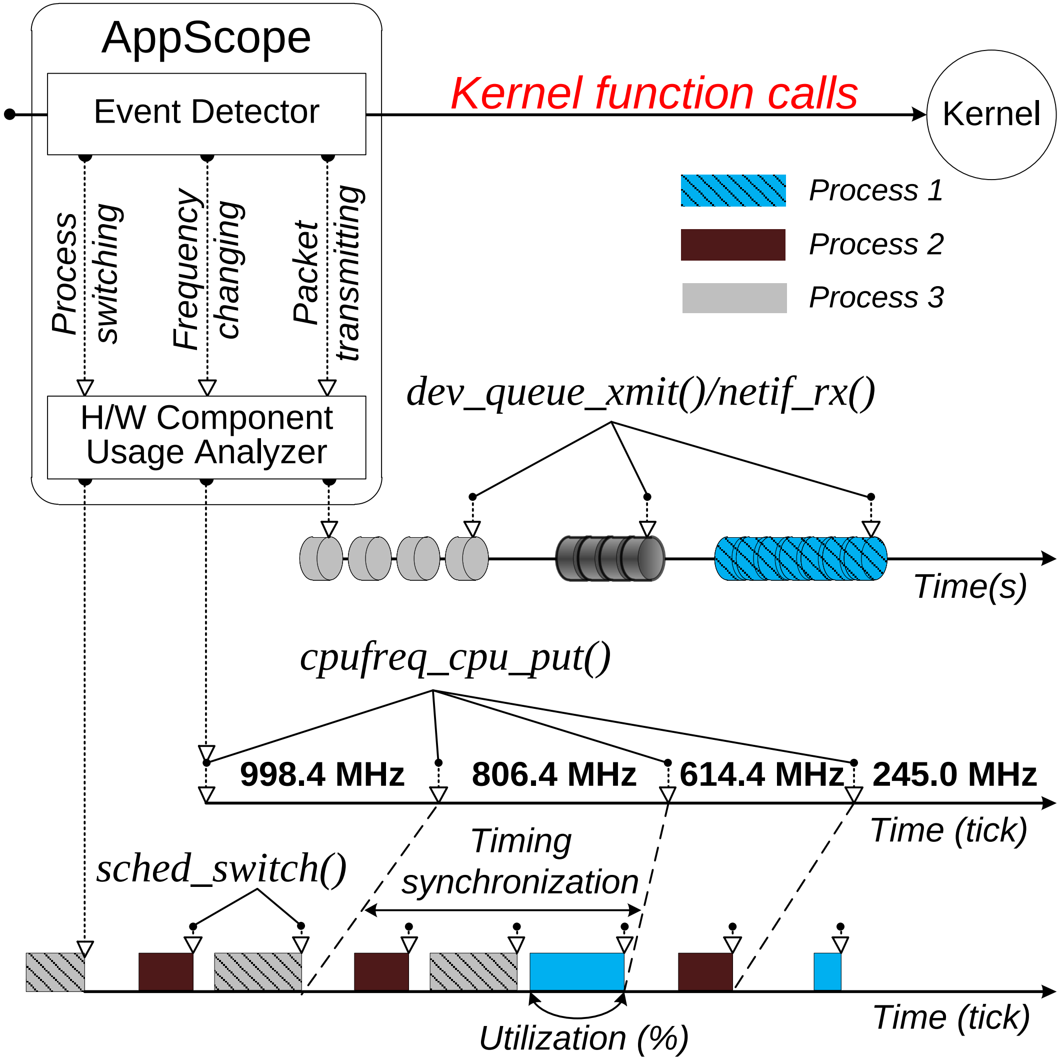
HARDWARE OPERATION

```
#define BC_TRANSACTION
#define BC_REPLY
#define BR_TRANSACTION
#define BR_REPLY

struct binder_transaction_data {
    union {
        size_t          handle;
        void            *ptr;
    } target;
    void            *cookie;
    unsigned int    code;
    unsigned int    flags;
    pid_t           sender_pid;
    uid_t           sender_euid;
    size_t          data_size;
    size_t          offsets_size;
    union {
        struct {
            const void  *buffer;
            const void  *offsets;
        } ptr;
        uint8_t         buf[8];
    } data;
};
```



Hardware Component Usage Analysis (1)



WiFi

- DAI layer

$$P^{WIFI} = \begin{cases} \beta_l^{WIFI} \times p + \beta_l^{base}, & if\ p \le t \\ \beta_h^{WIFI} \times p + \beta_h^{base}, & if\ p > t \end{cases}$$

p: packet rate, t: threshold

CPU

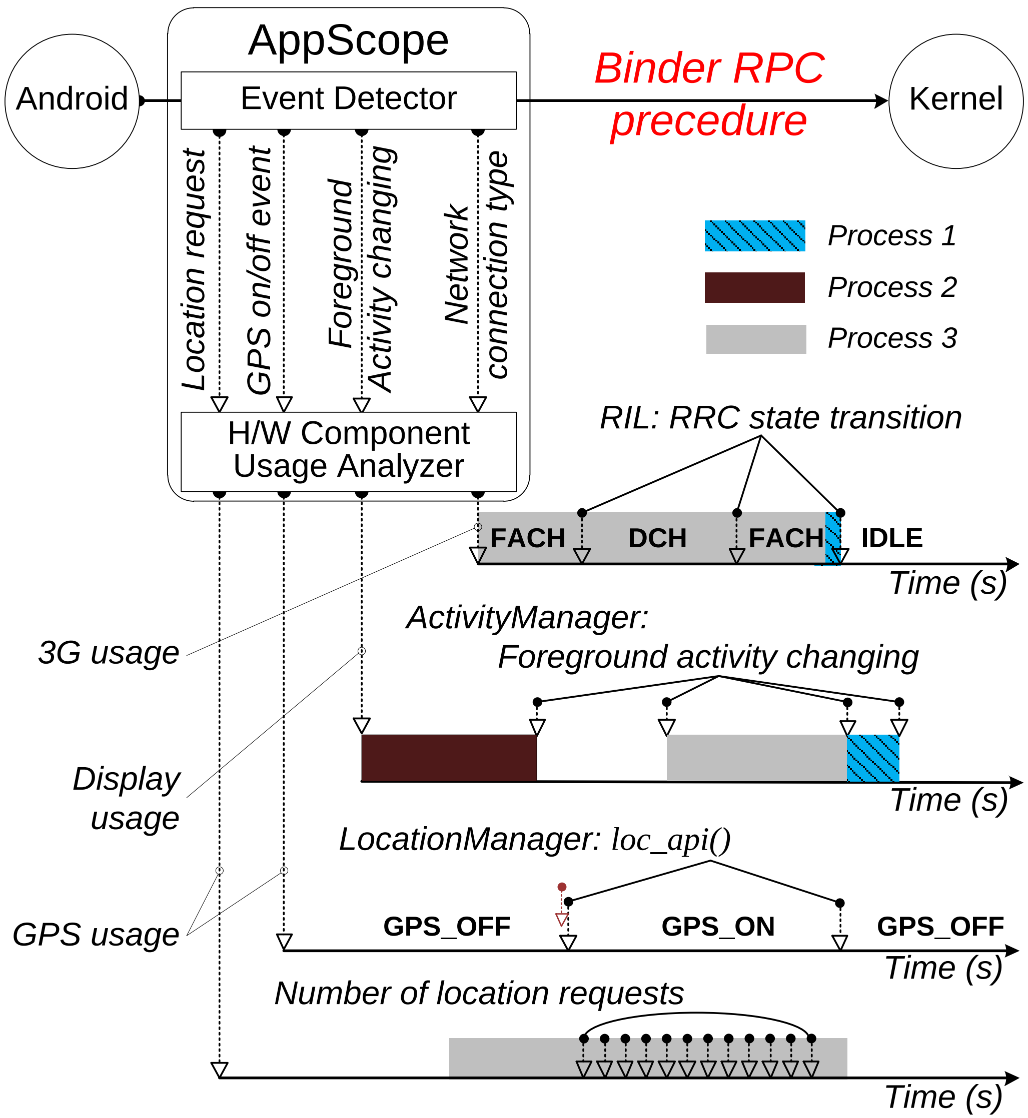
- DFVS governor

$$P^{CPU} = \beta_{freq}^{CPU} \times u + \beta_{freq}^{idle}$$

u: utilization, $0 \le u \le 100$

$freq$: frequency index, $freq = 0,1,2 \cdots, n$

Hardware Component Usage Analysis (2)



3G

- Network connection type
  → RRC state transition

$$P^{3G} = \begin{cases} \beta^{3G}_{IDLE}, & \text{if RRC state is IDLE} \\ \beta^{3G}_{FACH}, & \text{if RRC state is FACH} \\ \beta^{3G}_{DCH}, & \text{if RRC state is DCH} \end{cases}$$

Display

- *Activity Manager* IPC data

$$P^{LCD} = \beta^{LCD}_b$$

b: brightness level,
$\mathrm{MIN}(level) \le b \le \mathrm{MAX}(level)$

GPS

- *loc_api()*
- *LocationManager* IPC data

$$P^{GPS} = \beta^{GPS}_{on}, \text{ if GPS is on}$$

Evaluation (1)

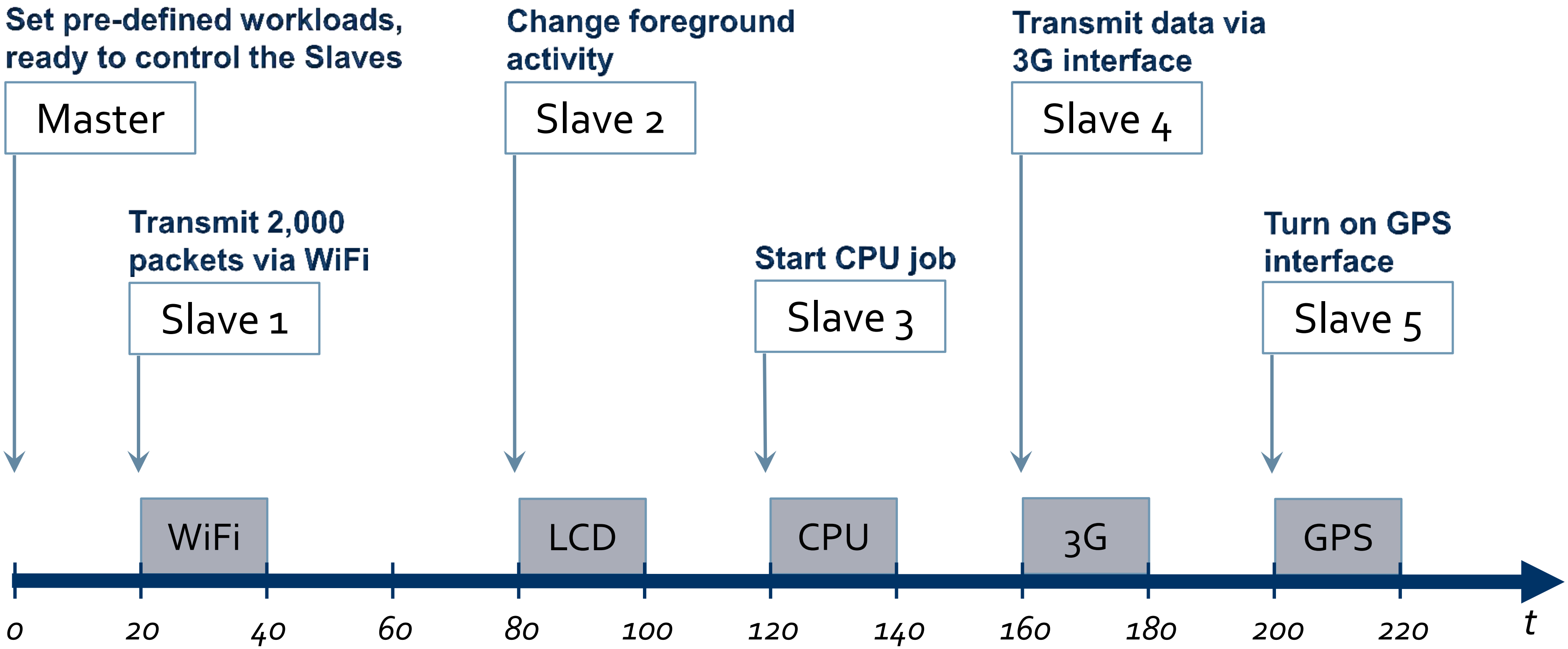
Component Usage Monitoring	Energy Metering Validation	Overhead Analysis	Real Application Energy Metering
• Hardware event detection • Hardware usage statistics	• Granularity of information • Accuracy of power metering	• CPU overhead • Power overhead	• Case Study • Error analysis
• 6 test apps • Pre-defined workload • Workload scheduling	• DevScope power model • Per UID • Per Component • Vs. Monsoon	• Loaded case • Unloaded case	• Angry Birds • Skype (WiFi) • Browser (WiFi) • Browser (3G) • Google Maps

Evaluation (2)

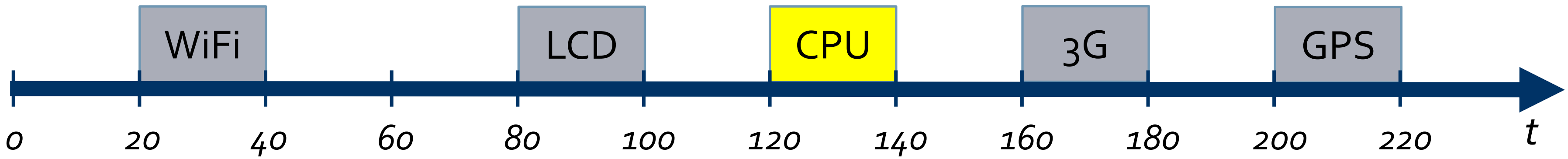
- Development environments
  - Linux kernel 2.6.35.7
  - SystemTap 1.3 (also uses Kprobes)
  - Android platform 2.3
- Device
  - HTC Google Nexus One
    - Qualcomm QSD 8250 Snapdragon 1GHz
    - 3.7-inch Super LCD display
    - MAXIM DS2784 Fuel-gauge-IC
  - External measurement device
    - The Monsoon Power Monitor

Test Scenario

- 1 "Master" and 5 "Slave" applications

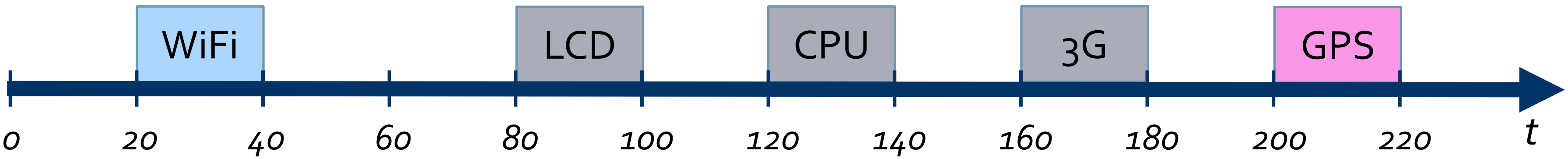


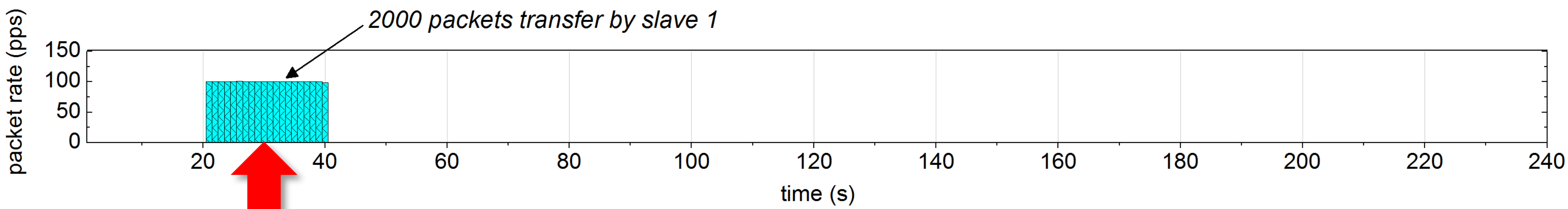
Component Usage Monitoring (1)



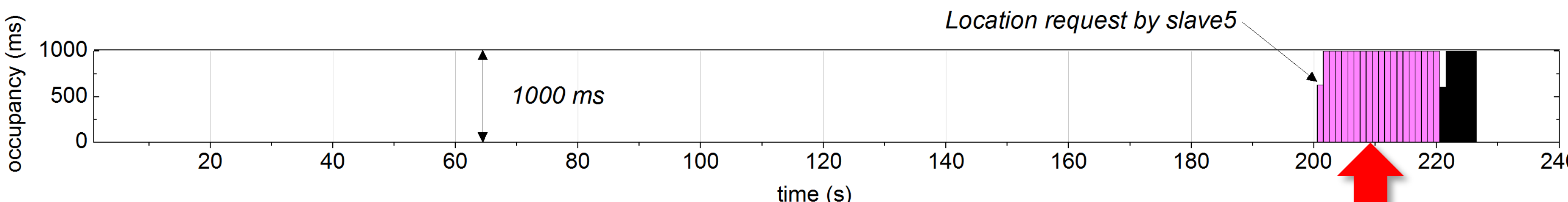
Medium frequency (766/576 Mhz) due to OnDemand policy

Component Usage Monitoring (2)



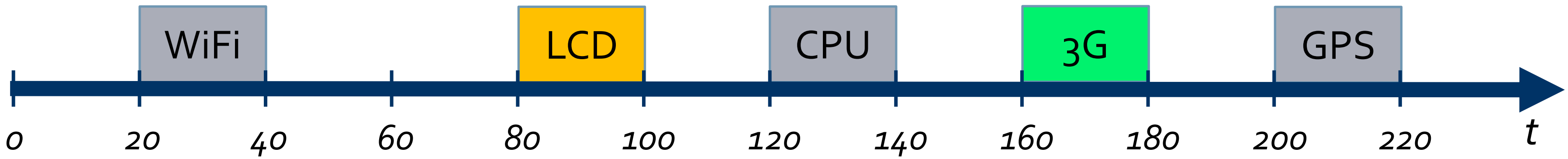


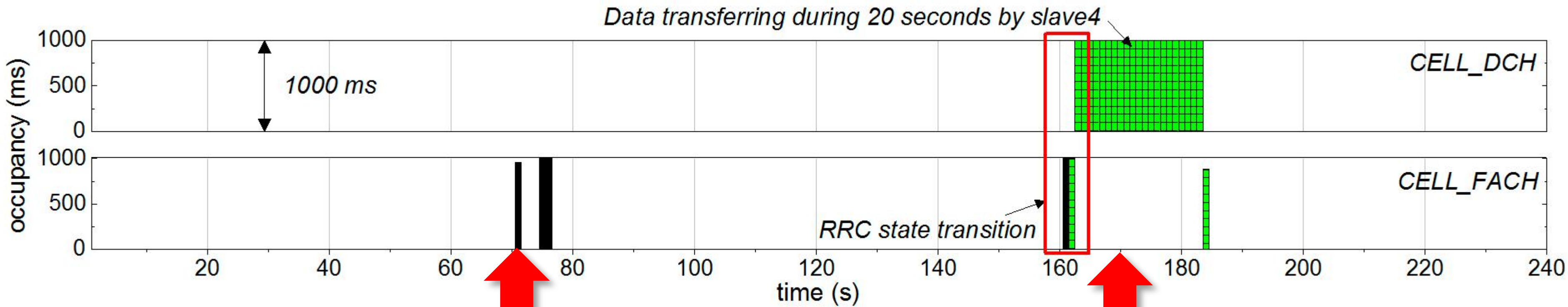
Packet rate is 100 pps for 20 secs



"System" uses GPS for 2 ~ 4.5 sec

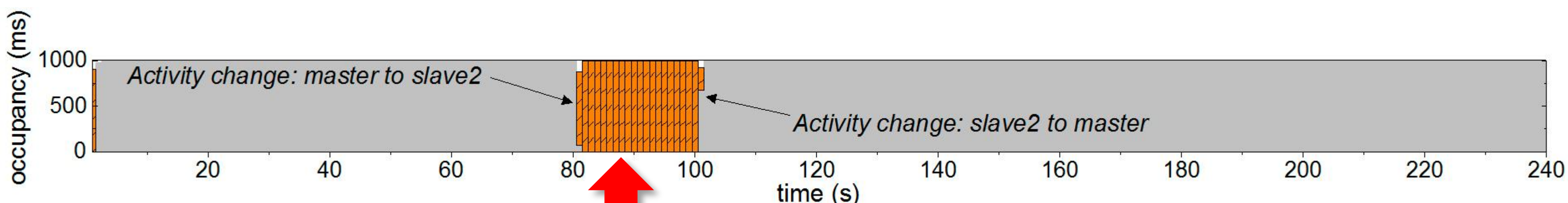
Component Usage Monitoring (3)





Turned on

"System" transmits some data



Activity change occurs in *ActivityManager*

Power Model for Google Nexus One (N1)

Component-specific
DevScope Power Models

nexus one

Component	Model
CPU	$P^{CPU} = \beta^{CPU}_{freq} \times u + \beta^{idle}_{freq}$ u: utilization, $0 \le u \le 100$ $freq$: frequency index, $freq = 0,1,2 \cdots, n$
LCD	$P^{LCD} = \beta^{LCD}_{b}$ b: brightness level, $\mathrm{MIN}(level) \le b \le \mathrm{MAX}(level)$
WiFi	$P^{WIFI} = \begin{cases} \beta^{WIFI}_{l} \times p + \beta^{base}_{l}, & if\ p \le t \\ \beta^{WIFI}_{h} \times p + \beta^{base}_{h}, & if\ p > t \end{cases}$ p: packet rate, t: threshold
cellular(3G)	$P^{3G} = \begin{cases} \beta^{3G}_{IDLE}, & if\ RRC\ state\ is\ IDLE \\ \beta^{3G}_{FACH}, & if\ RRC\ state\ is\ FACH \\ \beta^{3G}_{DCH}, & if\ RRC\ state\ is\ DCH \end{cases}$
GPS	$P^{GPS} = \beta^{GPS}_{on},\ if\ GPS\ is\ on$

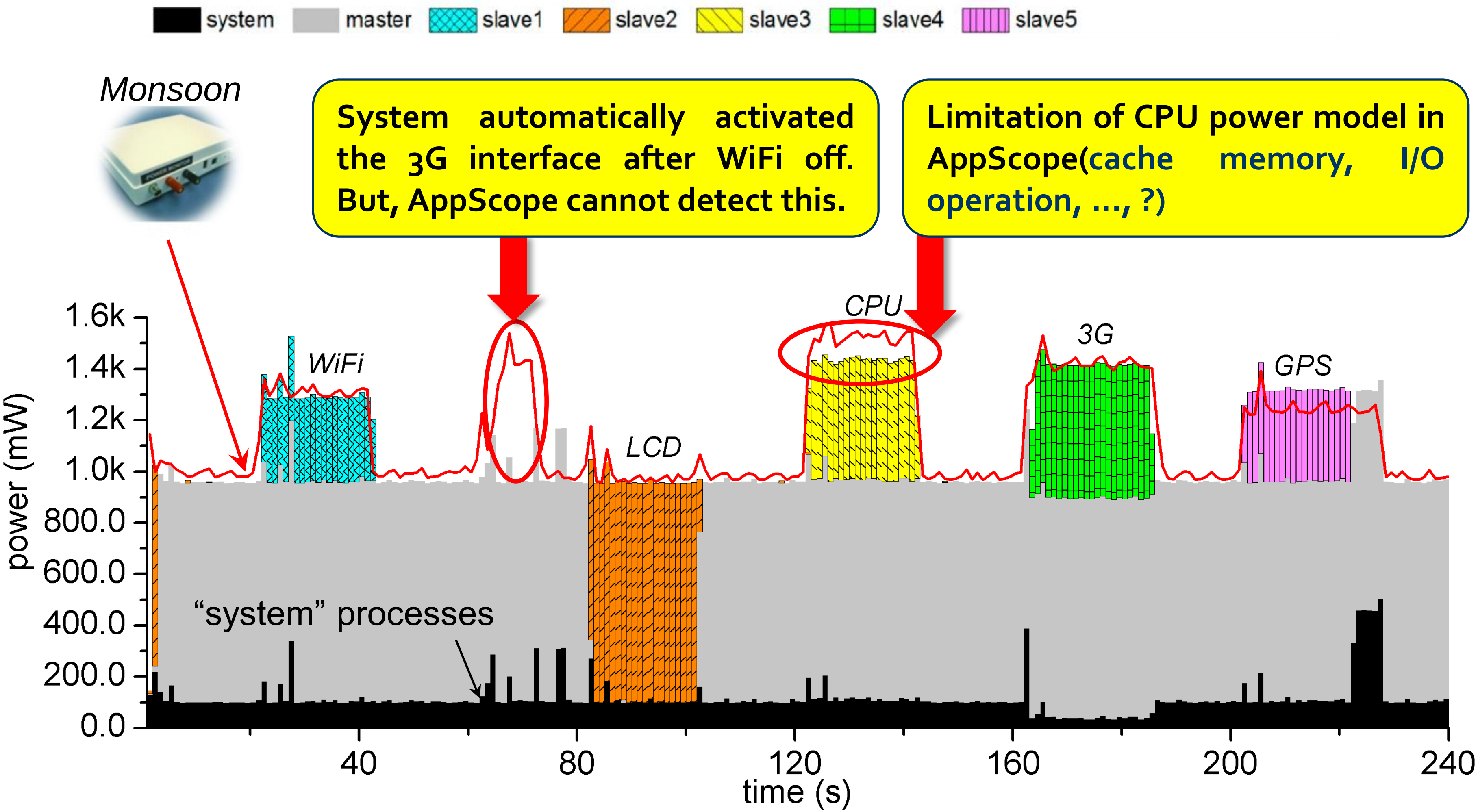
Comp.	Index	Coefficient	
CPU	$freq$ (Mhz)	β^{freq}_{i}	β^{idle}_{i}
	245.0	201.0	35.1
	384.0	257.2	39.5
	460.8	286.0	35.2
	499.2	303.7	36.5
	576.0	332.7	39.5
	614.4	356.3	38.5
	652.8	378.4	36.7
	691.2	400.3	39.6
	768.0	443.4	40.2
	806.4	470.7	38.4
	844.8	493.1	43.5
	998.4	559.5	45.6
GPS		β^{gps}	
	ON	354.7	

Comp.	Index	Coefficient	
LCD	b	$\beta^{brightness}_{b}$	
	5	367.8	
	55	451.5	
	105	631.1	
	155	697.9	
	205	775.4	
	255	854.0	
3G	rrc	β^{rrc}	
	IDLE	63.9	
	FACH	267.9	
	DCH	519.3	
WiFi		β_l	β_h
	Transmit	1.2	0.8
	Base	238.7	247.0
	Threshold	25pps	

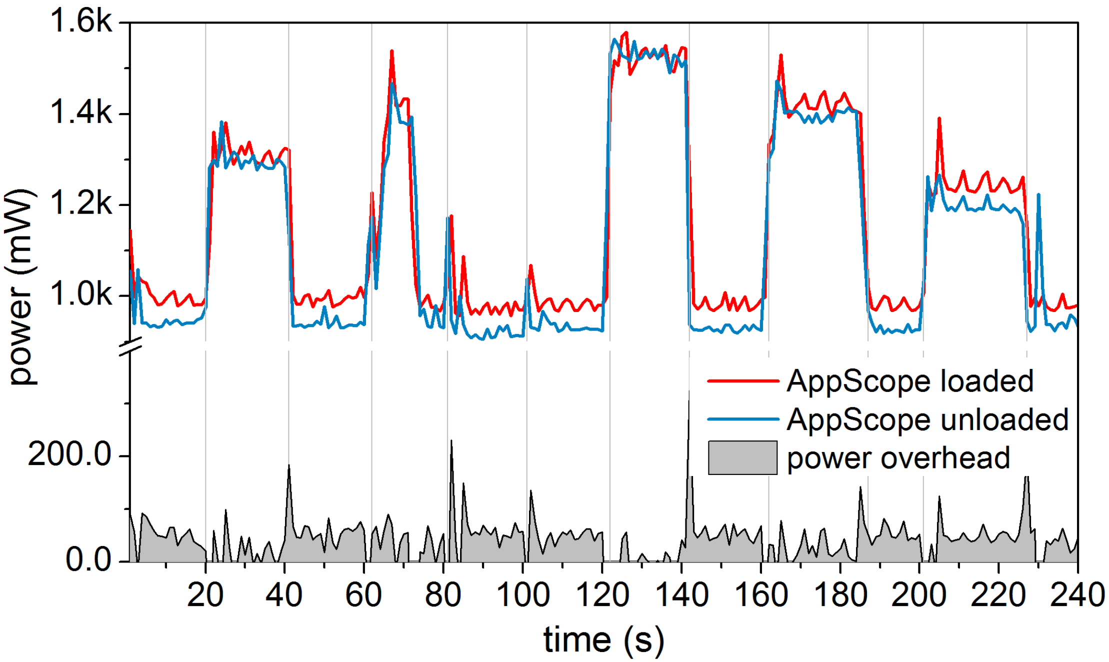
Energy Metering Validation (1)

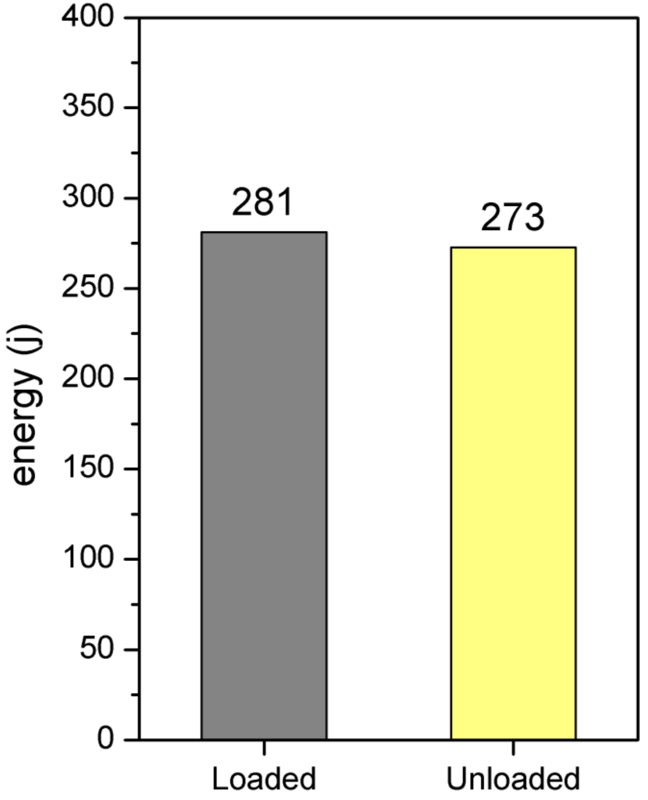
System automatically activated the 3G interface after WiFi off. But, AppScope cannot detect this.

Limitation of CPU power model in AppScope(cache memory, I/O operation, …, ?)



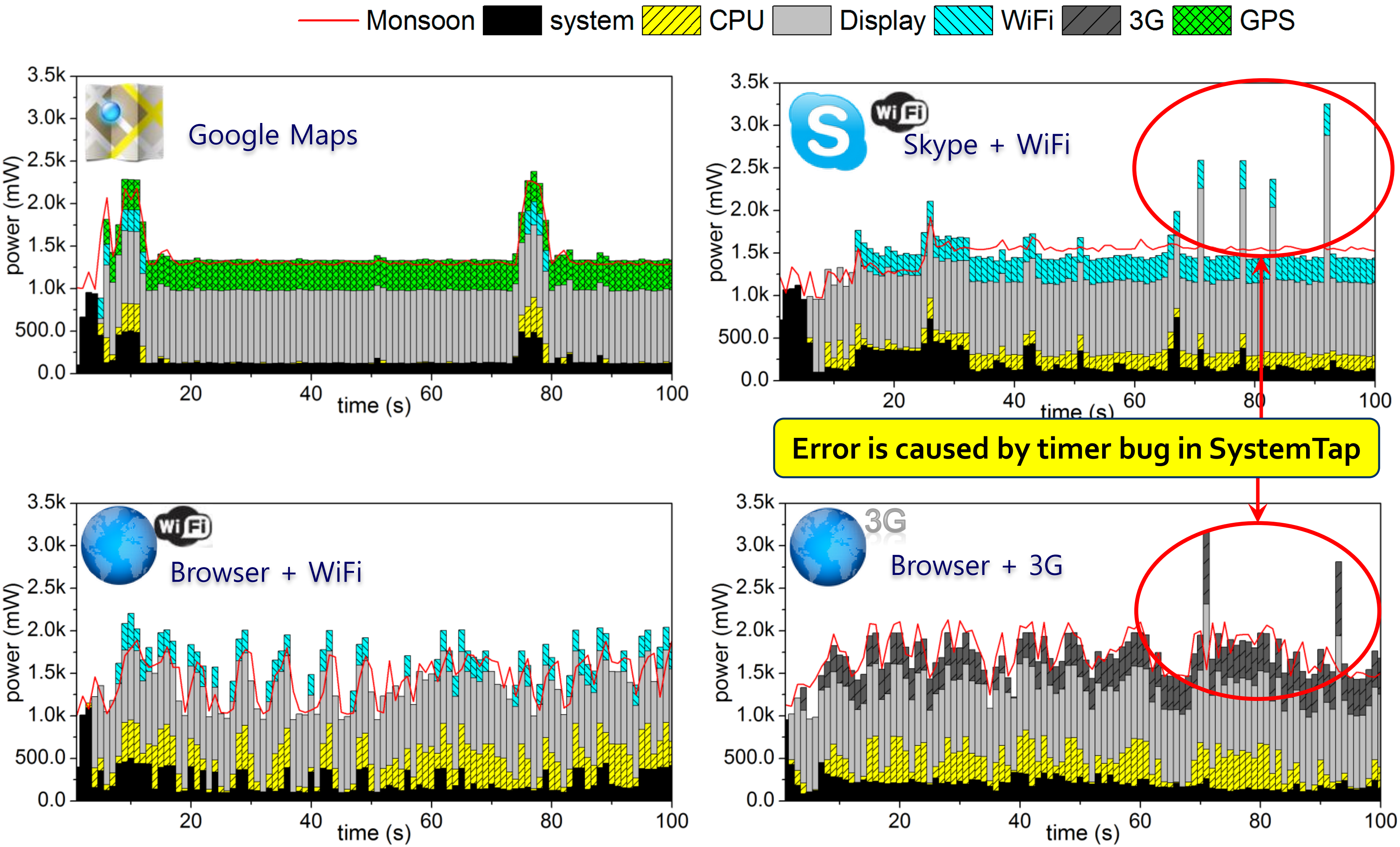
Overhead Analysis





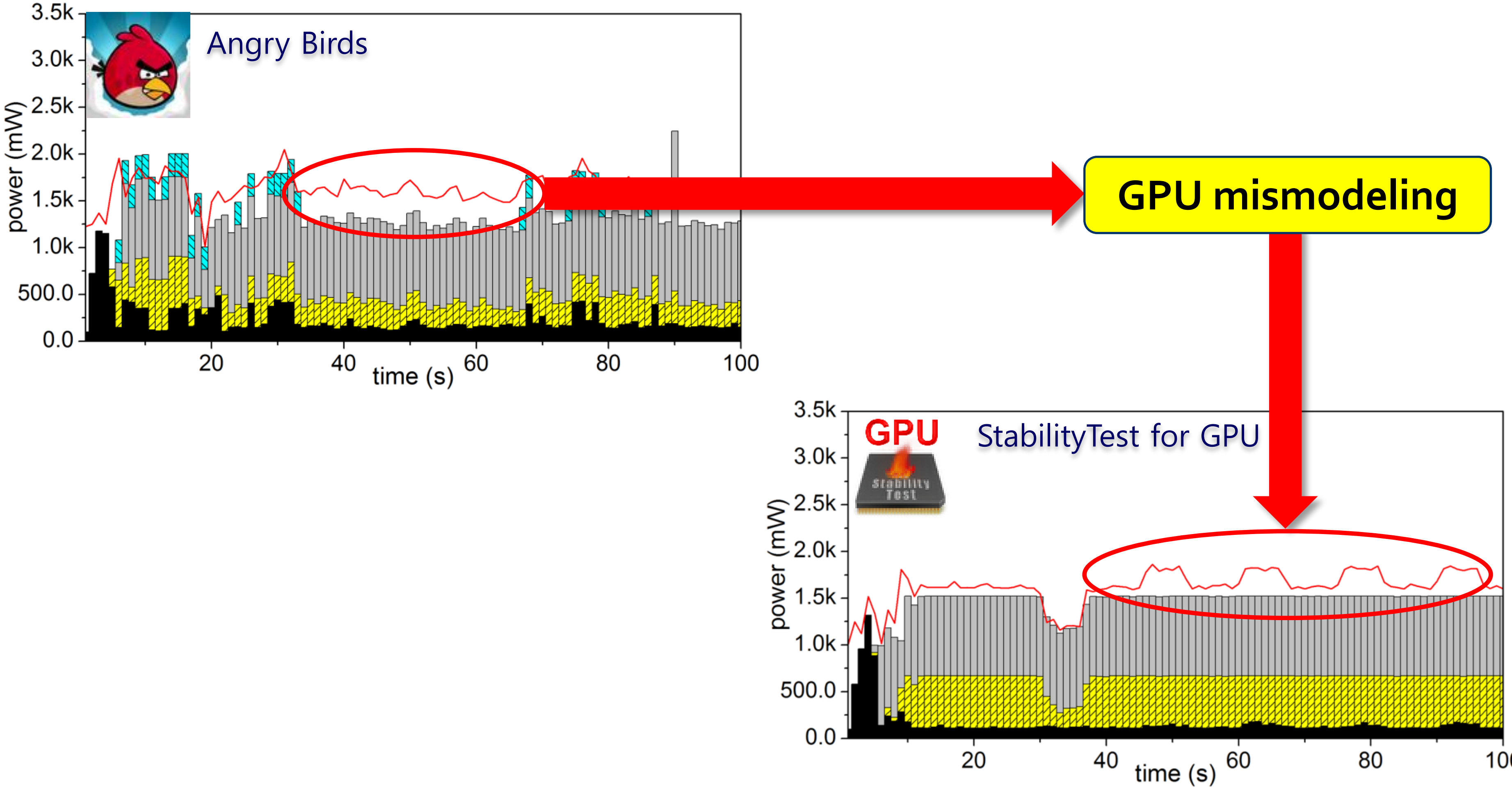
- **8.4J energy overhead for 240 seconds**
- **About 2.1% CPU overhead on average**

Real Application Energy Metering (1)



Real Application Energy Metering (2)

- GPU error



Limitations

- Processor power modeling
  - No consideration on GPU
  - Do not cope with multi-core processor architecture
  - No consideration on memory component
    - CPU-bound job vs. Memory-bound job
- Tail-state energy estimation
  - Limitation of linear power model (c.f. FSM power model)
- Hardware components
  - OLED display
  - Sensors: INS, MIC, Camera, …

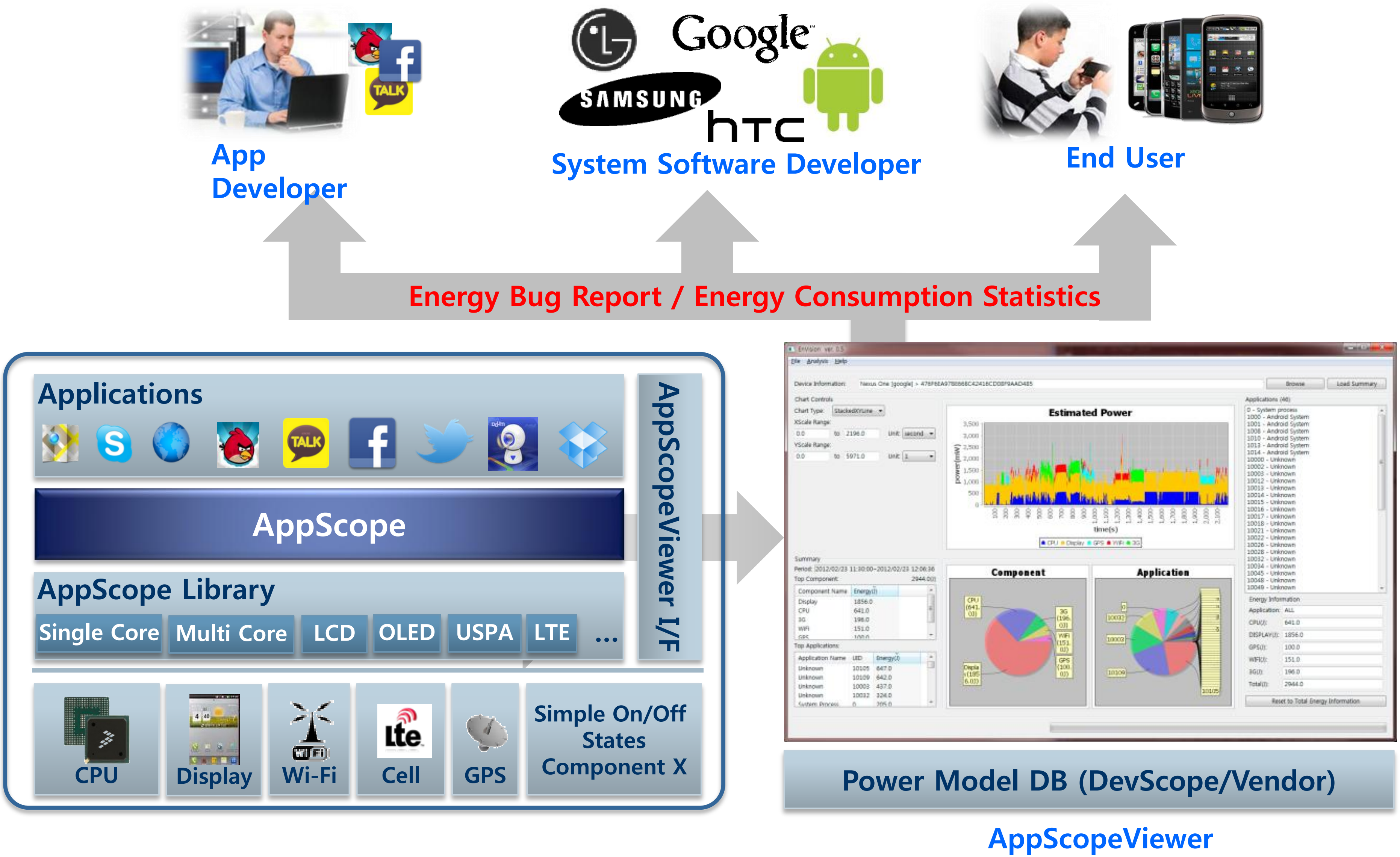
Related Tools

Tool	Description
Android Built-in Battery Info	• This does not provide fine-grained power profile
PowerTop	• This is not available for smartphones
Trepn Profiler [Qualcomm]	• Hardware sensor-based power profiler • This is only available on Snapdragon MDP
Energy Profiler [Nokia]	• Device power consumption • External APIs for testing a application • Developer's solution
PowerTutor	• State of the ART

AppScope Vs. PowerTutor

PowerTutor	AppScope
An Android application	Linux kernel module (+ External power profilers)
Polling using Android *BatteryStat*	Event-driven using Linux *Kprobes*
Application(UID) level	Process level
Reading */proc and /sys*, Using Android API, Using modified Android framework	Monitoring kernel function call
CPU, LCD/OLED, WIFI , 3G data, GPS, AUDIO	CPU, LCD, WIFI, 3G data + voice call, GPS

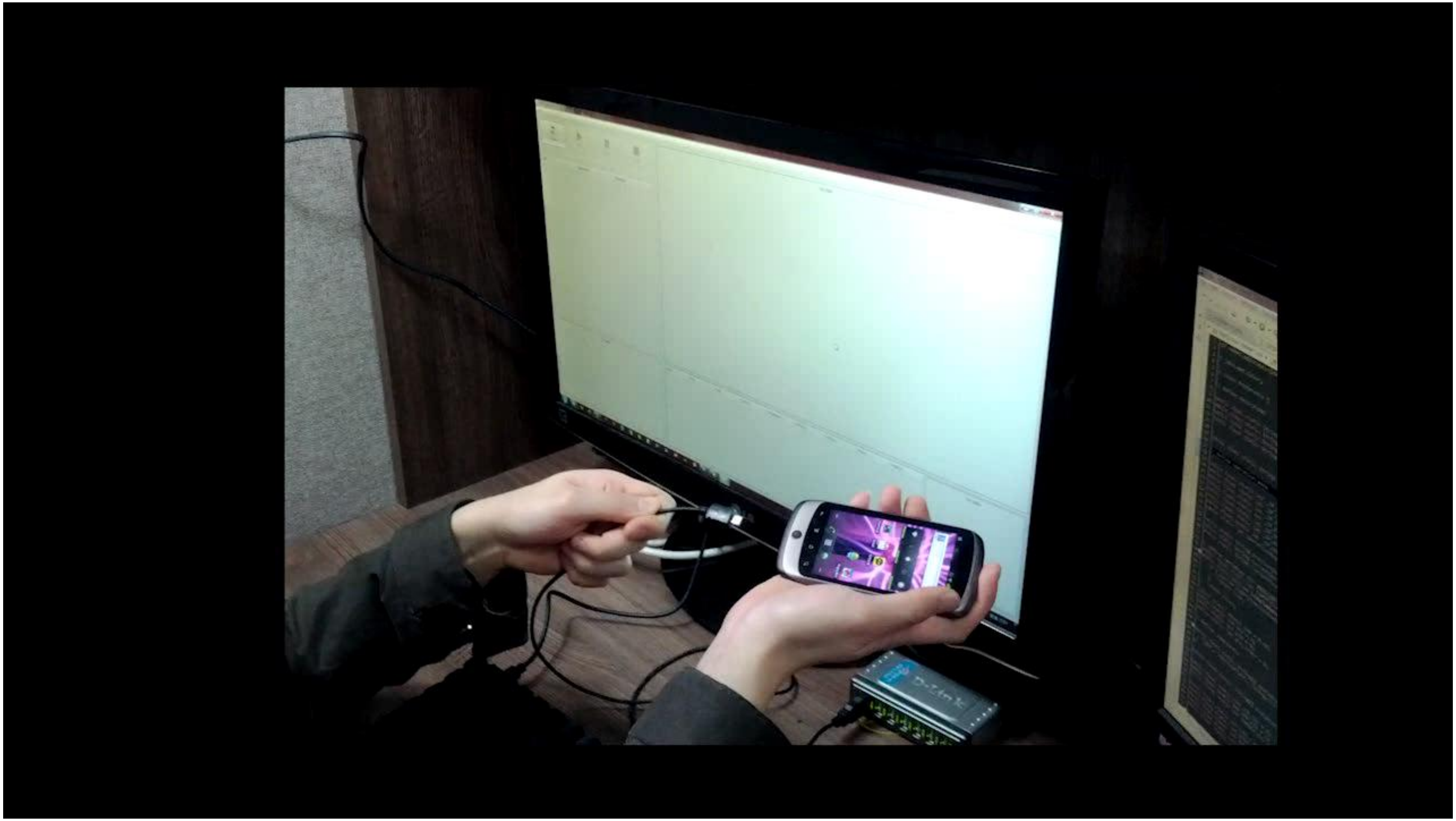
The AppScope Project



AppScope Suite

- AppScopeViewer: Real-time Android power profiler
  - An Java application providing device's power profile graphically
  - Interacts with AppScope in target device
  - Easy to use without any external measurement device
- Visit our project homepage
  - http://mobed.yonsei.ac.kr/~appscope
  - Our release includes binaries of AppScope, kernel image, and AppScopeViewer.
  - Currently, AppScope supports Google Nexus One
    - CPU, 3G, WiFi, LCD, GPS.

Demo



Conclusion

- Contributions
  - Provide energy consumption of Android application, being customized to the underlying system software and hardware components in device
  - Accurately estimates in real-time (with AppScopeViewer)
  - Implemented using module programming to improve portability
- Future work (*in progress*)
  - Supporting diverse hardware components:
    - OLED display, various sensors, …
  - Supporting multi-core processor architecture
  - GPU power modeling

Thank You

http://mobed.yonsei.ac.kr/~appscope