AALTO UNIVERSITY
School of Science
Department of Computer Science

Fine-grained Energy Profiling in Mobile Devices

Timo Lilja

Licentiate's thesis submitted in partial fulfilment of the requirements for the degree of Licentiate of Science in Technology

Supervisor Eljas Soisalon-Soininen
Instructor Vesa Hirvisalo

Espoo, August 10, 2016

`git-commit-id: 5fb0485`

Aalto University
School of Science

Abstract of Licentiate Thesis

Author
Timo Lilja

Title of Thesis
Fine-grained Energy Profiling in Mobile Devices

Abstract
Mobile phones have several use cases such as making a phone call, sending an SMS, browsing the Internet, or playing a game. Mobile phones are also equipped with a wide variety of hardware allowing these activities. The overall energy consumption is easily measurable but it does not explain how much power is dissipated on particular activities and which devices are mostly responsible for the consumption. Thus, this thesis defines the concept of fine-grained energy profiling where the total energy consumed is broken down into subsystems.

This thesis examines mobile phone power dissipation and energy consumption analysis. It describes how to measure overall total energy consumed by the device and develops methods that allow breaking down the energy consumption to components or subsystems. Specifically, the tools developed in this thesis allow studying the energy consumed by the CPU, GPU, display, WiFi, Cellular 3G and SSD disk. The device studied here is the Nokia N900 Maemo/Linux phone.

This thesis describes a full-system power measurement setup including a fake battery and DAQ acting as an electric power meter. The thesis also develops logging tools that monitor the subsystem load and provide a method for formulating a linear regression model between these two through a set of microbenchmarks. This model allows estimating the subsystem energy consumption and total energy consumption based on observed loads of the subsystems. The model achieved 80% accuracy when compared with measured total energy consumption to the total power dissipation predicted by model.

Research Field Computer Science	**Keywords** Energy, Regression, Phone
Supervising professor Eljas Soisalon-Soininen	**Pages** vii + 87
Thesis advisor Vesa Hirvisalo	**Language** English
Thesis examiner Sébastien Lafond	**Date** August 10, 2016

☒ The thesis can be read at `https://aaltodoc.aalto.fi/handle/123456789/27`

Aalto-yliopisto
Perustieteiden
korkeakoulu

Lisensiaatintutkimuksen tiivistelmä

Tekija
Timo Lilja

Lisensiaatintutkimuksen nimi
Mobiililaitteiden hienojakoinen energiaprofilointi

Tiivistelmä
Matkapuhelimia voi käyttää moniin tarkoituksiin kuten puheluihin, tekstiviestien lähettämiseen, Internet-selailuun tai pelien pelaamiseen. Tämän mahdollistamiseksi nykyaikaisissa matkapuhelimissa on paljon erilaisia laitteistokomponentteja eli alijärjestelmiä. Matkapuhelimen kokonaisvirrankulutus on helppo mitata, mutta se ei selitä sitä, kuinka paljon kukin alijärjestelmä kuluttaa energiaa. Tämän ongelman ratkaisemiseksi tässä työssä määritellään hienojakoisen energiaprofiloinnin käsite, jonka avulla kokonaisenergiankulutus voidaan jakaa alijärjestelmien energiankulutukseksi.

Tässä työssä tutkitaan matkapuhelimen tehon ja energiankulutuksen analysointia. Työssä kehitetään menetelmä matkapuhelimen kokonaisenergiankulutuksen mittaamiseen. Lisäksi esitellään menetelmä, jonka avulla energiankulutus jyvitetään matkapuhelimen alijärjestelmille. Työssä jyvitys tehdään seuraaville alijärjestelmille: suoritin (CPU), grafiikkasuoritin (GPU), näyttö, langaton verkko (WiFi), mobiilidatayhteys (3G) sekä tallennuslevy (SSD). Laitteena käytettiin Nokia N900 Maemo/Linux -puhelinta.

Työssä esitellään kokonaisenergiankulutuksen mittausjärjestely, joka sisältää valeakun ja virrankulutusmittarina toimivan DAQ-laitteen. Lisäksi työssä kehitetään työkalu, jonka avulla matkapuhelimen alijärjestelmien käyttöastetta voidaan tarkkailla ja tallentaa. Lopuksi työssä esitellään lineaariseen regressioon perustuva mallinnus, joka mahdollistaa mitatun tehon ja kokonaisenergiankulutuksen jyvittämisen yksittäisille alijärjestelmäkomponenteille. Työssä kehitetty malli saavutti noin 80 % selitysasteen, kun mallin ennustamaa tehonkulutusta verrattiin mitattuun tehonkulutukseen.

Tutkimusala tietotekniikka	**Avainsanat** energia, regressio, puhelin
Vastuuprofessori Eljas Soisalon-Soininen	**Sivumäärä** vii + 87
Ohjaaja Vesa Hirvisalo	**Kieli** englanti
Työn tarkastaja Sébastien Lafond	**Päiväys** 10. elokuuta 2016

☒ Luettavissa verkossa `https://aaltodoc.aalto.fi/handle/123456789/27`

Contents

Chapter 1

Introduction

Modern mobile phones have a plethora of *subsystems* that allow them to perform various tasks. These subsystems include CPU used for general computing, various radios such as WiFi or Cellular that are used for external communication, cameras and sensors used for observing, recording, and adapting to the environment, and GPS for location services. All of these subsystems consume energy when they are operated. Each of these subsystems have their individual and distinct energy consumption characteristics: a modern phone may be able to last for several hours of messaging but playing a game may drain the battery in less than an hour. Thus, it is a natural question to ask how much energy each of these subsystems consume during an operation.

This thesis develops a *tool* that allows giving individual subsystemwise energy consumption of a mobile device. The goal is to provide a method that allows explaining which subsystems are utilized by an application and how much energy they consume. The thesis studies the Nokia N900 mobile phone and provide a model that tells how much energy is consumed by the CPU, GPU, display, WiFi, Cellular 3G, and the solid-state disk subsystems when operating the device. While this thesis focuses on Nokia N900, most of the methods developed here are applicable to other platforms too.

1.1 Research Questions

The total energy consumption of a mobile device is the amount of energy drawn by the device from a battery or some other energy source. The problem of measuring the *total energy consumption* involves setting up an electric circuit that allows measuring the current drawn by a device with sufficient accuracy.

Measuring the *subsystemwise energy consumption* directly is difficult on most readily available hardware due to the lack of probe or measurement points that would allow directly observing the total energy consumed by a subsystem. Thus, this thesis develops an *indirect* measurement setup, where the total energy drawn by the entire system is observed and correlated it with individual subsystem utilizations or loads. Another aspect of subsystem energy consumption measuring is that the utilization and behavior of the subsystem usually

depends on some external factors (e.g., load) that cannot be observed by directly measuring the current drawn by the subsystem.

To develop a system that allows providing subsystemwise energy consumption estimates, the following research problems or questions are considered:

Q1 How to measure the total energy consumed by a mobile device.

Q2 How to measure the utilization of individual subsystems of a mobile device.

Q3 How to break down the total energy consumption to subsystem energy consumption.

Q4 How to verify that the subsystem energy consumption breakdown is accurate.

1.2 The Solution

This thesis develops a measurement framework called `eprof` that allows measuring the energy consumption of the system and individual subsystem utilization (e.g., CPU load). The total energy consumption is measured by a Data-Acquisition device (DAQ) acting as a multimeter and a fake battery setup and the subsystem utilization by a hand-written logging program that uses various utilization counters provided by the operating system of the mobile device. (Questions Q1 and Q2)

To correlate the total energy consumption and individual subsystem utilization, the thesis uses a *linear model* that is trained with microbenchmarks that utilize subsystems individually. Denote these microbenchmarks as the *calibration set* of the linear model. (Question Q3)

To verify the linear model's accuracy, the thesis framework uses another set of benchmarks that utilize multiple subsystems simultaneously. These benchmarks are denoted as the *verification set*. The linear model is used to predict subsystem and total energy consumption while concurrently measuring the actual power consumption with the fake battery setup. The thesis compares the actual measured and predicted energy consumption with a cross-correlation metric as the *accuracy criteria*. (Question Q4)

Overview of the research methodology can be seen in Figure 1.1.

1.3 Results

The main contributions of this thesis are a framework for mobile device subsystemwise energy consumption breakdown analysis through linear regression and a concrete application of the framework to Nokia N900 energy consumption analysis.

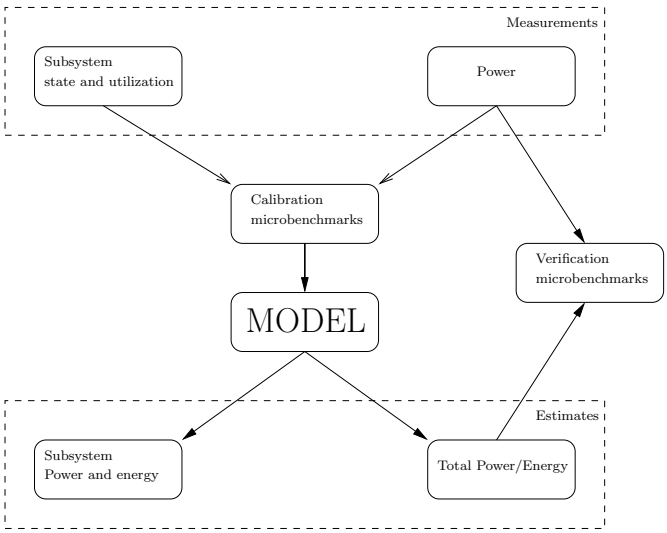


Figure 1.1: Overview of the methodology and solution.

1.4 Thesis Outline

The structure of the thesis is the following: Chapter 2 gives an introduction to power and energy measuring techniques and discusses the energy and power management in general. Related work is presented in Chapter 3. Mathematical background for statistics needed in this thesis is presented in Chapter 4. The Nokia N900 device is described in Chapter 6. These chapters may be omitted if the reader is familiar with these topics.

The main contributions of the author are in Chapters 5, 7, 8, and 9: Chapter 5 describes how to use the linear models to model subsystemwise energy consumption. Chapter 7 explains the measurement framework features and its implementation. Chapter 8 describes the training and verification of the experiment results for the linear model. Discussion, analysis of the results, and concluding remarks are given in Chapter 9.

Chapter 2

Background

This chapter explains how the concepts of energy and power are related. It also describes how electric power is measured in general and how energy and power management is done on integrated circuits and through various radio protocols.

2.1 Energy and Power

Energy and power are concepts often intermixed in a casual conversation, but they have rigorous physical definitions found in any physics book such as [60]. Energy is a quantity that expresses a physical system's ability to do work from a *source system* into a *destination system*. We denote this amount of transferred energy as ΔE. The SI unit of energy and work is *joule* (J).

Power is the *rate* at which work is done or energy is being transferred. The SI unit of power is *watt* (W). Given a quantity of work ΔE and a time interval Δt, we define the *average power* P_{av} as

$$P_{av} = \frac{\Delta E}{\Delta t}. \tag{2.1}$$

We define the *instantaneous power* P as the time derivative of energy:

$$P = \lim_{\Delta t \to 0} \frac{\Delta E}{\Delta t} = \frac{dE}{dt} \tag{2.2}$$

For the thesis, the focus is on *electromagnetic energy*. The energy source, *the battery*, transfers chemical energy into electrical energy. This energy is consumed by the destination system, *the load*. In this case, the load is a mobile phone, an electric circuit, that transfers electrical energy to various other forms of energy such as sound, light, radio signals, and—usually unwanted—heat.

To understand energy and power in electric circuits, we need the concepts of current, resistance, and voltage. See [60] for rigorous definitions. *Current* is denoted by I and its SI unit is *ampere* (A). Voltage or electric potential difference is usually denoted by ΔV but for brevity we use V. The SI unit of voltage is *volt* (V). Resistance is denoted by R and its SI unit is *ohm* (Ω).

For materials commonly used in electric circuits there is a relation between voltage, current and resistance called *Ohm's law* which states:

$$V = R \cdot I \tag{2.3}$$

To define the concept of *power* for electric circuits, we need to apply *Joule's law* which says that power is the product of voltage and current. We also apply Ohm's law from Equation (2.3) to obtain:

$$P = VI = R \cdot I^2 = \frac{V^2}{R}. \tag{2.4}$$

Now, we are ready to define the concepts of instantaneous power, energy and average power for electric circuits. We are given a circuit that draws current from a current source (battery) with a time-varying function denoted by $I(t)$. The instantaneous power is:

$$P(t) = V \cdot I(t) \tag{2.5}$$

where V denotes the supply voltage of the circuit.

The total energy E consumed by the system over time interval T is

$$E = \int_0^{T'} V \cdot I(t)dt. \tag{2.6}$$

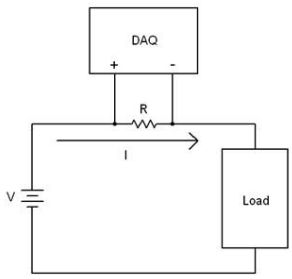
The average power P_{av} over T-length time interval is

$$P_{av} = \frac{E}{T} = \frac{1}{T}\int_0^{T} V \cdot I(t)dt \tag{2.7}$$

2.2 Measuring Electric Power and Energy

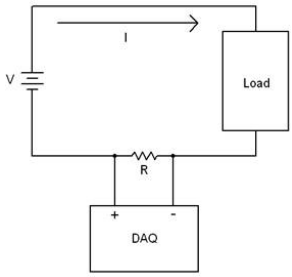
An *ammeter* is a measurement instrument that measures the electric current in a circuit. A *voltmeter* measures electric voltage between two probe points in an electric circuit. For the system, a DAQ, a digital voltmeter, that uses an analog-to-digital converter (ADC) to sample the voltage drop across the probe points is used. A *shunt* or *sense resistor* is connected in series between the battery and the load. Then the DAQ is used to measure the voltage drop across the shunt.

The shunt can be placed either *high-side* or *low-side* [5]. In high-side current measurement the shunt is placed between the battery and the load circuit.



The high-side connection provides the most accurate measurements, since all current that goes to the load flows through the shunt. The main disadvantage is that there might be high-common mode voltage across the shunt resistor which can damage the measurement device [5].

In the low-side connection, the shunt or sense resistor is placed between the load and ground:



This eliminates the power with high-common mode voltage but measures only the current that is directly returned to the battery. That is, leakage current to the chassis or the control circuits cannot be measured.

When the measurements are performed with the DAQ, the sampling period whose length is T and frequency f is used. Thus, $N = f \cdot T$ readings are obtained. We denote the set of measured supply and terminal voltages as $V[i]$ and $V_{\mathrm{BAT}}[i]$, $0 \leq i \leq N$.

Using the formula for instantaneous power from Equation (2.5) we get the formula for measured instantaneous power:

$$P[i] = V_{\mathrm{BAT}}[i]\frac{V[i]}{R}.$$

Here R is the resistance of the *shunt resistor*.

The formula for total measured energy consumption from Equation (2.6) is

$$E = \frac{1}{f}\sum_{i=0}^{N}\left[V_{\mathrm{BAT}}[i]\frac{V[i]}{R}\right]. \tag{2.8}$$

When computing the average power P_{av} we down-sample the signal to a time interval whose length is M, $M < N$ by taking a running average

$$P_{av}[j] = \frac{1}{M}\sum_{i=j\cdot M}^{(t+1)M}\left[V_{\mathrm{BAT}}[i]\frac{V[i]}{R}\right], \quad 0 \leq j \leq \lfloor N/M \rfloor.$$

For most of thhe studies, the averaging time interval is set to $M = f$ which gives:

$$P_{av}[t] = \frac{1}{f}\sum_{i=t\cdot f}^{(t+1)f}\left[V_{\mathrm{BAT}}[i]\frac{V[i]}{R}\right], \quad 0 \leq t \leq T. \tag{2.9}$$

This yields one second non-moving average.

2.3 Integrated Circuits and Power

Integrated circuits (IC) are miniaturized electronic circuits consisting of large number of transistors that are embedded into a single semi-conducting material. Their development began in the 1940s. One of the first patents regarding integrated circuits was filed in 1959 by Jack Kilby [32], a Texas Instruments employee.

Integrated circuits are made of transistors that can either switch state or amplify electric signals. Most of the integrated circuits used in computers and mobile devices use the transistors' state-switching capability to produce logic from which central processing units and other computing units can be constructed.

The power dissipation of an integrated circuit can be divided into *static loss* and *dynamic loss* [56]:

$$P_{\text{total}} = P_{\text{static}} + P_{\text{dynamic}}.$$

Static power loss is caused by leakage currents present in the transistors and wires. These leakages may be caused by transistors' sub-threshold conduction, tunneling currents, or leakage through junctions. Static dissipation caused by leakage can be considered as a product of the supply voltage V_{dd} and leakage current I_l:

$$P_{\text{static}} = V_{\text{dd}} I_l.$$

Dynamic power loss is caused by the *switching activity* of the circuit and *short-circuit currents*:

$$P_{\text{dynamic}} = P_{\text{switching}} + P_{\text{short}}.$$

Switching activity causes power loss by parasitic capacitance present in the integrated circuits. This capacitance is loaded and discharged when the transistors are active, i.e., switching states. This switching loss can be described with the formula:

$$P_{\text{switching}} = \alpha C V_{dd}^2 f \tag{2.10}$$

where α indicates the activity of the circuit (how many state switches there are per time unit), C is the capacitance of the circuit, V_{dd} is the supply voltage and f is the operating frequency.

Short-circuit loss occurs when a transistor is switching state and there is a brief period when the current flows from the voltage supply directly to the ground before the switching is completed. Short circuit power dissipation P_{sc} can be modeled with the formula:

$$P_{\text{short}} = V_{\text{dd}} I_{\text{short}}$$

where V_{dd} is the supply voltage and I_{short} is the short circuit current.

There are various ways to achieve energy consumption saving in integrated circuits: We can try to minimize the static or leakage loss by choosing a material with less resistance. Dynamic switching loss can be lowered by lowering the clock frequency f and the supply voltage V_{dd}. Dynamic short-circuit loss is lessened by using better materials [20]. Next we discuss these energy consumption saving techniques further.

2.3.1 Reducing Static Loss

Static power loss is always present in the system whether it is idle or active. To reduce it, we can use better materials that cause less resistance in the circuit wires and transistors. In addition, if the system is completely idle, we can use *power gating*. Power gating reduces the disabled block power dissipation to zero.

Another concept that relates to the power gating is the power saving modes or PSMs that describe how much of the system is switched off during idle periods.

Power gating requires parts of the system to be completely switched off. Thus, the state that is in those parts must be stored somewhere. This process is called *state retention* and it involves state retention registers that are kept online. When the system is woken up, the state is read from the retention registers and restored after which the system becomes fully operational.

Hardware designers must implement necessary functionality in order for the chip to support power gating. Namely, integrated circuits consist of *combinatorial* and *sequential logic*. Combinatorial logic circuits produce the same state given the same input while the sequential logic circuits' output depends on internal state. This state must be stored into the retention registers.

Power saving modes describe the level in which the system sleeps during an idle period. The deeper the sleep state, the less power is consumed. The downside is that the wakeup latency increases when a deeper sleep state is entered. Common sleep states that can be found from modern hardware include:

wait

- core clock is gated, i.e., activated only when needed
- normal operation is resumed on interrupt
- good for running I/O or other peripheral activity

doze

- core clock is gated
- peripherals can be switched off
- normal operation is resumed on interrupt

state retention

- all clocks are switched off
- external memory in low power self-refresh mode
- peripheral clocks are gated
- voltage can be dropped to minimum
- less power than doze but longer wakeup time
- no need to do data recovery after wakeup

deep sleep

- clocks are gated
- core platform power is gated

- register data must be saved before deep sleep

hibernate

- whole chip power is gated (switched off)
- operation resume equals cold boot
- data must be saved to external permanent memory

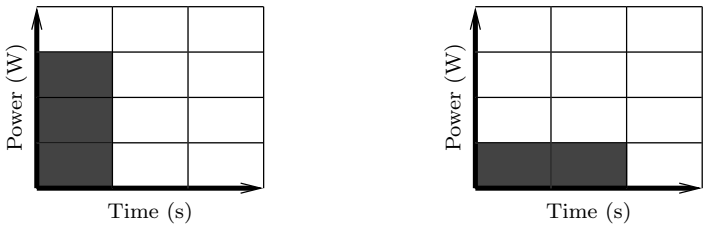
2.3.2 Reducing Dynamic Loss

Dynamic loss is mostly caused by the switching activity of the circuit. If the system is idle, the clock pulses are still emitted periodically even though no state switching occurs. Thus, we can save power by switching the clock off in a process called *clock gating*. This reduces dynamic power dissipation. The clock-gated transistors retain their state due to the supply voltage still being present.

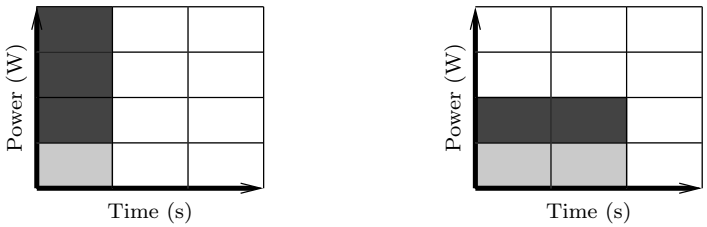
Clock gating is applicable if the system is completely idle. If it is not, we can reduce the dynamic power dissipation with a technique called *dynamic voltage and frequency scaling* or DVFS. From the Equation (2.10) we can see that the switching loss is quadratic to the supply voltage V_{dd} and linear to the operating frequency f. The operating voltage and frequency of an integrated circuit depend on each other and cannot be chosen arbitrarily. Thus, hardware designers define *operating performance points* or OPPs that are voltage–frequency pairs with which the system remains operational.

In DVFS, we have a predefined set of OPPs from which we must choose the suitable one for the current operation. Usually the OPP is determined so that some performance goal (high enough voltage/frequency) or power dissipation budget (low enough voltage/frequency) is met.

DVFS is beneficial if the static power loss is minimal compared to the dynamic power loss: consider a case where there is a task with a fixed duration. If we half the frequency, the task will take twice as much time to complete but consumes less energy as can be seen in the figure below:



If the idle consumption is not negligible (the light gray boxes in the figures below), then DVFS does not lead to power savings:



2.4 Radios and Power

Radio protocols include various power saving features that are distinct from the power saving used in CPUs. The basic principle of turning the device off when it is idle remains the same but due to the nature of radio communication some power management techniques (e.g., DVFS) do not apply. Here we focus only on the power management of the WiFi and Cellular devices as they are defined in their corresponding protocol specifications.

2.4.1 WiFi

WiFi is defined as wireless local network that is based on the Institute of Electrical and Electronic Engineers' 802.11 Standards. In WiFi there are mobile stations (STAs) and usually stationary access points (APs) that provide the connection to the Internet for the mobile station. IEEE Standard for 802.11 defines the power management features of the mobile station [29, Section 11.2].

According to the standard, a mobile station can be in either one of the following power states: *awake*: mobile station is fully powered, or *doze*: mobile station does no transmit or receive and consumes very little power. There are two power management modes specified in [29]:

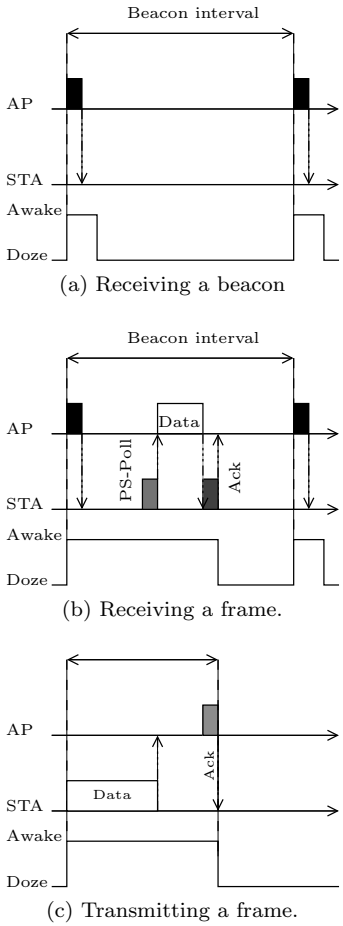
Active Mode In active mode (AM)the mobile station is in the Awake state and can receive frames at any time. The radio is always on and draws constant power from the system independent of the bandwidth utilization.

Power Save In Power Save (PS) mode the mobile station listens to *Beacon frames* with an interval specified by the remote access point (AP). If the access point has data for the mobile station, it turns a bit on in the beacon frame and buffers the data. The mobile station sends a poll request labeled PS-Poll that allows the AP to send the buffered data.

The mobile station is in doze state and enters the awake state only to listen to the beacon frames, to receive multicast or broadcast transmissions, to transmit, or to wait responses for PS-Poll frames.

There is no corresponding buffer on the mobile station when it transmits a frame to the access point. See Figure 2.1 for overall view of the power saving operations.

The above description is a simplification of the actual power saving and WiFi communication. For a detailed explanation on WiFi and power management, see [48, 49]. An analysis of the power saving feature can be found in [13].



(a) Receiving a beacon

(b) Receiving a frame.

(c) Transmitting a frame.

Figure 2.1: WiFi Power Save management. Simplified version.

2.4.2 Cellular

Cellular networks are the most commonly used radio networks for mobile hand-held devices like phones or tablets. Since their advent there has been a couple of revisions on the network standards used. For this study, the focus us on Cellular 3G standard mobile device power management features. Refer to the explanation given in [50].

A 3G network consists of mobile hand-held devices or phones called User Equipment (UE) which are connected to base stations called Node-Bs via a radio link. There are many components in 3G architecture which are omitted here. Instead, the focus is on the *radio resource control* (RRC) which associates a state machine for each UE and is the most relevant component with regards to power dissipation. There are three states in a typical RRC:

IDLE The UE is turned on but can receive and transmit only *control data* but no *user data*.

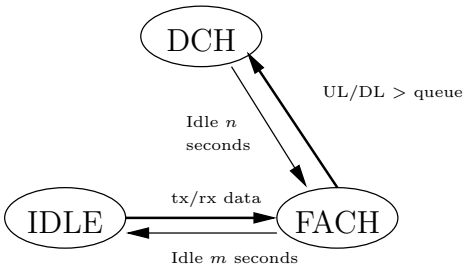
CELL_DCH The UE is turned on and can both send and receive data. The

UE is usually in a high-power state with a dedicated channel allowing it to fully utilize the available bandwidth.

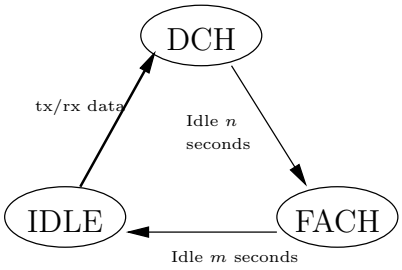
CELL_FACH The UE is turned on and connected but there is no dedicated channel allocated. Instead, the UE can transmit data through a shared low-speed channel. Since it consumes less radio resources than CELL_DCH it yields lower power dissipation.

State transitions can be either *promotions* or *demotions*. The state promotions occur when the UE moves from less resource-intensive state into a state consuming more resources and power. There are two methods for promotions. Promotions are triggered by network activity, namely the amount of data collected into the upload (UL) and download (DL) queues present in the base station. Demotions are triggered by timeouts that are controlled by the base station.

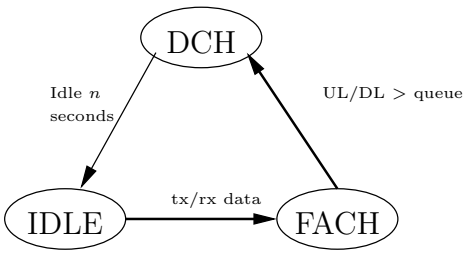
In the first method, the UE is first promoted into lower power CELL_FACH state if there is any data activity, and if the amount of data exceeds the configured UL/DL queue sizes, into the CELL_DCH state. Demotions are done by first dropping to the CELL_FACH state after a timeout and after another timeout the UE is dropped into the IDLE state:



In the second method, the UE is directly promoted to high power CELL_DCH state if there is any data transfer activity. After a timeout, demotion transfers UE into lower powered CELL_FACH state from which a subsequent timeout moves the system into IDLE state. There is usually some low data transfer rate which keeps the UE in the CELL_FACH state:



The third method is to have the system promote to CELL_FACH if there is some data transfer activity, and after exceeding a threshold, promote to CELL_DCH. After a single timeout the system drops directly to IDLE state:



2.5 Battery

A battery is an electrochemical cell that converts chemical energy into electrical energy. The battery is not a simple voltage source even though it is often modeled as such in circuit analysis. Unlike an ideal voltage source, the battery, or *terminal voltage*, depends on the level of charge, the amount of current flowing through the circuit, and the temperature.

Batteries can be constructed from various materials that have distinct characteristics. Nowadays, the most common battery type for mobile devices is lithium-ion, or Li-ion. It has an operating voltage of 3.6 volts. The energy density of a Li-ion battery is 175 Wh/kg or 500 Wh/l [56] which is rather good when compared to previously popular nickel cadmium battery's 35 Wh/kg or 70 Wh/l.

The cycle life, i.e., the number of charging times a Li-ion battery can hold is about 500. A Li-ion battery is considered "dead" if the the battery has dropped to 80% of its original capacity. Charging a Li-ion battery requires favorable conditions: the charging current and voltage must be within certain limits. The temperature of the environment must not be too low or too high. A Li-ion battery is permanently damaged if its voltage level drops below certain threshold and may explode if it is overcharged to voltage levels exceeding the maximum.

Most Li-ion batteries are equipped with a Battery Management Unit (BMU) that protects from overcharging and undercharging of the battery, monitors its temperature, and prevents short circuits. These BMUs usually have a digital interface from which the operating system can read battery statistics like voltage and current levels or charging status.

2.6 Operating Systems

For this discussion, refer to Advanced Configuration and Power Interface [12] (ACPI) power management standard for hardware and operating systems.

ACPI hardware interfaces are mostly used in the personal computer equipment or server equipment and not on mobile hardware but the operating system interfaces and the terminology is usually present on the mobile devices even though the low-level hardware implementation of power management is somewhat different.

The operating system's responsibility is to control the available resources of the

device. Previously, the resource control was mostly focused on multiplexing CPU and other peripherals among processes. The goal was to provide secure access to peripherals in a way that a single process or task could not crash the entire system. Nowadays the operating system resource control includes the concept of energy and power management, where the goal is to minimize energy consumption to provide extended battery life.

To develop an operating system energy and power management system we need to consider both hardware and software. Most of the hardware-enabled power saving features are controlled by hardware or BIOS interfaces such as ACPI [12]. Hardware provides a set of idle and active power states with varying level of performance and power consumption. It is the task of the operating system to decide which power state is suitable for each situation.

The main idea is to adjust the performance of the subsystems based on workloads utilizing these subsystems. An operating system energy and power management framework usually consists of three components:

- an *observer* that monitors the load of the system and its resource,
- a *controller* that performs state transitions (DVFS, gating, idle state changes),
- a *policy* that defines how/when these state transitions are performed [56]

State transitions consume time and energy and can involve long wakeup times. Thus, the policy manager must carefully determine when to perform transitions given certain performance requirements. Policy managers can be predictive or stochastic. In predictive policies, the history of the workload is used to deduce when to perform state transitions. In stochastic policies, a complex statistical model is used to predict the behavior of the workload which is then used to make power saving decisions.

2.6.1 CPU Power Management

ACPI standard [12, Chapter 8] defines the CPU power management as a state machine. A CPU is always in one *power state*: C0, C1, ..., or Cn. When the CPU is active it is in the C0 power state and in one *performance state*: P0, P1, ..., or Pn.

Power states C1, ..., Cn are processor sleeping states. The higher the sleeping state, the less power the processor consumes but the wakeup latency from the sleep state is higher. When the processor is active and in a higher performance state, the processor provides higher performance but consumes more power. Usually the higher performance states are implemented through DVFS. See Section 2.3.2 for similar techniques.

The operating system's policy management decides, based on the power and performance requirements, when to move from a power to a performance state or vice versa.

2.6.2 Device Power Management

For peripheral devices, the ACPI standard defines the concept of device power management [12, 3.3] and device power states [12, 2.3]. The overall goal of standardized interface is to provide the operating system a way to control the device power management.

The device power states that can be controlled by the operating system are:

D0	device is fully powered
D1	device is not fully operational
D2	device is not fully operational, consumes less than D1
D3hot	consumes less energy than D2, preserves device context
D3	device is off, the device context is lost

In addition, devices may implement performance or *P*-states for active operation power management similar to that of CPUs. The operating system device drivers offer interfaces which allow transition between the power states. The device driver may implement the power management by itself or it can be done in a centralized manner in the OS kernel.

2.6.3 System Power Management

Full system power management involves controlling the power management of all the peripherals and the CPU. That is, switching the whole device into deeper sleep states when the system is not active. ACPI defines the following sleep states:

S1	low wakeup latency, no context is lost
S2	low wakeup latency, CPU and cache context is lost
S3	low latency wakeup state, all context is lost except system memory
S4	longest wakeup latency sleep state, all devices powered off, context stored in memory
S5	soft off state, context saved to persistent media, long wakeup latency, requires boot to wake up

Sometimes the sleep states are described with the terminology: ON, STANDBY, SUSPEND, and HIBERNATE. In ON, the system is not in a sleep state. In STANDBY, the system is in one of S2–S3 ACPI power states depending on the implementation. In SUSPEND, the system is in S4 power state, and while in HIBERNATE, the system is in S5 state. It is typical that the wakeup latency for STANDBY is less than 1 second, SUSPEND is 3-5 seconds, and HIBERNATE may have a wakeup latency of up to 30 seconds [40].

Chapter 3

Related Work

This chapter gives a brief survey of literature relevant for the thesis. Especially, for comparison and discussion purposes, a more detailed overview of two articles whose work is similar to this thesis is given.

3.1 Modeling Power Consumption

In power modeling, we are trying to understand how a device (e.g., a mobile phone) consumes energy, especially, how much energy is consumed by a device or *peripheral*. Virtually all power models use some set of *input variables* which are fed into *a model* that produces power dissipation or energy consumption estimates. The choice of input variables, the training of the model, and the accuracy of produced estimates varies between different approaches. In this chapter we survey the approaches taken in the literature.

The *type* of the model describes whether we have a *device-specific model* or *full-system model*. In device-specific models, a single peripheral is studied and various aspects of it are modeled. For example, a CPU could be modeled so that we consider different kinds of instructions (arithmetic, control, floating points) and form a power dissipation model based on the instruction types [24]. In full-system models we consider the overall power dissipation of the entire device and ignore the intrinsic of a peripheral.

Device-specific models usually have plenty of input variables for a device whereas full-system models use only few input variables per device. Naturally, full-system models are more inaccurate but, on the other hand, incur less overhead than peripheral-specific models. For examples of device-specific models for WiFi, see [13, 52, 53, 58], for Bluetooth [37, 44], and for Cellular [50]. CPUs are studied in [24,30,31,35], GPUs in [21,23,28,41,57], and flash drives in [42,59].

Comparison studies among radio protocols such as WiFi, Cellular, and Bluetooth are a widely studied field in the literature as well [14,15,22,27]. Full-system models are compared in [54].

To provide power dissipation and energy consumption estimates, power dissipation measurements need to be obtained. These can be obtained from the

hardware specifications or by doing measurements. If the measurements are obtained through specifications, we denote the system as *specification-based* such as [14, 16] as opposed to *measurement-based*. The measurement-based approaches can be further divided into *direct* and *aggregate* measurements. In direct measurements, the system provides some kind of probe points that allow measuring the current flow between peripherals, like CPU and WiFi, directly. In aggregate measurements, some kind of *fake battery* construction that provides a measurement point between the battery terminals is used. Naturally, the aggregate measurement system produces single power measurements which have to somehow break down to subsystems to get peripheral-specific measurements.

The *approach* of the model can be either *sampling-based* or *event-based*. In the sampling-based approach, we collect *state variables* that describe the operation state (e.g., display on/off) or utilization (e.g., CPU load) of a single device by periodical sampling. For sampling-based approaches, monitoring the performance counters provided by the hardware has proven to be beneficial [33,36]. In event-based approaches annotate the system, it is beneficial to collect events that trigger a power change in the system (e.g., turning the radio on/off) and log them for analysis. According to Bellosa [17], event-based models incur less overhead and provide better accuracy than the sampling-based approaches. Martins [39] further argues that peripheral devices should directly expose their power state changes to make the event-based modeling easier. For an example of a full-system event-based modeling, see Pathak's article [47].

In power modeling, input data on various sources have to be collected, namely, measurements from the power meter and state variables describing the power states of the peripherals. These various data sources must be combined so that they are *synchronized*. One approach is to use *time-based synchronization* where both the power meter and the system keep their clocks in sync by calibrating their clocks through some means, for example, through the Network Time Protocol (NTP). Another approach is to trigger a predictable power dissipation spike (e.g., turning the device on/off) for the power measurements and use this to correlate the power measurements and power state input data. We denote this as *event-correlation synchronization*. For an example of an event-correlation synchronization, see [53].

To provide the estimates, a *model* which takes the state variables as input and produces power estimations as output ha to be used. The model usually involves some form of linear regression and is first trained with a set of microbenchmarks and later verified with an independent control set. Since most of the devices have a set of power states, and transitions between these states are inherently non-linear, a concept of *power-state machines* is used. Benini [18] describes a formalism that uses finite-state machines (FSMs) to model power states of peripherals.

Similar modeling for non-mobile devices has also been studied. For blade server environments, Economou [25] provides a linear model framework. Sagahyroon [55] and Mahesri [38] study power consumption of laptops.

The approach in this thesis is a full-system, sampling-based, time-correlated, linear model with power state machines and next a survey of the literature that uses similar approaches is provided.

3.2 Full-system Modeling

In this section, a survey of two of the most relevant articles to provide contrast to the work: The first article developed a power model for the OpenMoko Neo FreeRunner device [19] and the second article describes PowerTutor, an Android application that allows measuring individual subsystem power dissipation [61]. A brief description of the devices that are studied, measurement setup, applied methodology and the experiments is provided. Discussion and comparison to this thesis results is delayded to Chapter 9.

3.2.1 OpenMoko

OpenMoko Neo FreeRunner device was released on June 2008. It is an open source phone: the software is open and the hardware has specifications and documentation available for most components. The Android operating system was used together with a Google Nexus One, and an HTC Dream G1 phones in the experiments described by Carroll et al. in [19].

Below is the relevant hardware and software of the OpenMoko:

SoC	Samsung SS3C244 with ARM 920T 400 MHz
WiFi	Accton 3236AQ
Cellular	TI Calypso GSM+GPRS
Display	Topploy 480×640 TFT
RAM	128 MB
Flash	256 MiB NAND
OS	Android 1.5 with Linux 2.6.29 kernel

Carroll's measurement setup consists of National Instrument PCI-6229 DAQ acting as the electric power meter. The authors use shunt-resistors attached to power supply rail probe points present in the OpenMoko circuit board. This allows them to make direct power measurements of the subsystems. Their setup consists of a *host machine* that controls the execution: it interfaces with the OpenMoko device via SSH, controls the DAQ, and provides synchronization of the data. The sample rate was 400 ms yielding 2.5 Hz sampling frequency. A laboratory power supply is used to provide the power.

The authors measure the total power dissipated by the device and the power dissipated by individual subsystems. They define *total power* to be the total power dissipated by the device and the *aggregate power* as the sum of the power dissipated by all subsystems. Subsystems need different voltage levels than what is supplied by the main power source. Thus, regulators are needed to do the conversion. Since regulators are inefficient, this can cause up to 15-25% discrepancy between the total and aggregate power.

Carroll's research method is to make direct measurements on the following subsystems: CPU core, RAM, GSM, GPS, Bluetooth, LCD panel and touchscreen, LCD backlight, WiFi, audio, NAND flash, and SD card. For the Graphics subsystem, they use subtraction method since it had too many power rails for direct measurement. The authors have a set of *microbenchmarks* that is used to characterize independently individual component power dissipation. A set of

macrobenchmarks is used to utilize several components at the same time. They validate their results with indirect measurements for two Android smartphones: HTC Dream G1 and Google Nexus One.

In the experiments, the authors first measure the baseline power dissipation, then run the microbenchmarks, and finally the macrobenchmarks. The baseline power dissipation considers suspend and idle states and the backlight brightness. In the suspend state, the CPU is idle and the communication processor performs low-level activity. In the idle state, the system is fully awake but no application is active. Their measurements yielded the following results (all units are in milliwatts):

	CPU	RAM	GPU	WiFi	GSM	Audio	Rest	Total
suspend	13	3	11	6	30	4	4	68.6
idle	38	8	85	8	59	29	2	270

For the display brightness, they had the brightness levels available between 1 and 255. The Android's brightness control slider allowed them to set the value between 30 and 255, the centered slider corresponding to the level 143. The following measurements were obtained:

minimum	7.8 mW
centered	75 mW
maximum	414 mW

The content color had an effect on the power dissipation. The author's do not indicate which brightness level setting they were using but noted that a completely white screen consumed 33.1 mW and a completely black screen consumed 74.2 mW.

The microbenchmarks consist of CPU, Flash storage, and Network. In the CPU benchmark a part of SPEC CPU2000 benchmarks were run. The authors measure both CPU and RAM power dissipation considering CPU operating on 100 and 400 MHz. The Linux kernel on-demand frequency scaling was disabled during the test. The authors' results indicate that there is considerable difference based on the operating frequency and whether the running program is CPU-bound or memory-bound. The following results were obtained (results in milliwatts):

	CPU(100M)	RAM(100)	CPU(400)	RAM(400)
equake	60	10	180	400
vpr	60	20	160	40
gzip	60	30	140	80
crafty	55	55	90	80
mcf	50	70	80	90
idle	40	10	50	10

The flash storage microbenchmark consists of reading and writing for NAND flash storage. The read benchmark consist of reading a 64 MiB file with random data to /dev/null in 4 KiB blocks. The write test consist of writing 8 MiB random data in 4 KiB blocks. The page cache was flushed between each block write. The following results were obtained:

	NAND	CPU	RAM
READ	5	80	30
WRITE	5	100	30

In the network microbenchmark, a single file was downloaded with wget. The file size was 15 MiB for WiFi and 50 KiB for GPRS. The following power dissipation was measured:

	WiFi	GSM	CPU	RAM
WiFi	750	80	95	40
GPRS	10	650	50	10

The macrobenchmarks consist of a set of scenarios that utilized several subsystems at a time: audio playback, video playback, text messaging, phone call, emailing, and web browsing. We present the web browsing results here since they are used later when compared with the results. In the web browsing benchmark, the authors loaded the web browser and used it to fetch a BBC news site that was locally mirrored. They used both WiFi and GSM radios and obtained the following results:

	CPU	Graphics	WiFi	GSM	LCD	Rest
WiFi	50	90	60	90	30	40
GSM	50	90	10	210	30	40

In addition, they considered the backlight brightness with the following levels: 0: 0 mW, 33: 50 mW, 67: 150 mW, and 100: 420 mW.

The authors do not give accuracy estimates for their model or their measurements while they do give the accuracy of their multimeter.

3.2.2 PowerTutor

Zhang et al. discuss online power estimation in their article [61]. They present a method that allows manual construction of power models, propose a technique that automates power model creation, and introduce the PowerTutor tool that provides subsystem power dissipation breakdowns for Android devices. For the thesis, the relevant part is the manual power model creation which is discussed further and the rest of the contributions are omitted. The device used is HTC Dream which is an Android Development Phone (ADP) with the following specifications:

SoC	Qualcomm MSM7201A with ARM11 core
WiFi	Texas Instruments WL 1251B chipset
Cellular	Qualcomm RTR6285
Display	320×480 pixel TFT capacitive touch screen

The authors use Monsoon FTA22D combined power supply and meter as their measurement device. It provides a 5 kHz sampling rate. The power measurement trace is collected to a laptop computer. For the experiments, the authors provide *power exerciser* programs that cause power dissipation for individual subsystems and a logging tool that records the relevant state variables.

The research method is to use multivariate linear regression where the subsystem utilization is correlated with the power dissipation. The model is trained through a set of microbenchmarks utilizing the devices with varying levels. The authors call these microbenchmarks as *training suites*. To include the non-linear state variables, such as CPU frequency, in their models, the authors use dummy indicator variables. The components or subsystems the authors model are CPU, LCD, GPS, WiFi, cellular, and audio. Another set of benchmarks are run to validate the results. These benchmarks utilize several devices at the same time. To evaluate the accuracy, the authors use absolute average of the absolute value of errors:

$$\left|\frac{m-p}{m}\right|$$

where m is the measured power dissipation and p is the power consumption predicted by the model.

In the experimental section, the microbenchmarks used to train the model and the macrobenchmarks used to validate the model are presented. In the CPU microbenchmark the CPU utilization and frequency are the state variables. The training program controls the CPU frequency and duty cycle. The inferred model is:

Frequency	Model
246	$3.42 \cdot \mathrm{CPU_{load}} + 121.46 \cdot \mathrm{CPU_{on}}$ mW
385	$4.34 \cdot \mathrm{CPU_{load}} + 121.46 \cdot \mathrm{CPU_{on}}$ mW

In the LCD experiment, training program switched the backlight on, switched it off, and changed its brightness in 10 uniformly distributed brightness levels. The model:

BL	$2.40\ \mathrm{BL_{br}}$

In the WiFi, the underlying power model is rather complex. The observed system variables are the data rate and the channel rate. Their training program exchanges packets with a server by varying the delay between packets from 0 to 2 seconds in 0.1 second steps. The channel rate was varied in 11 Mbps, 36 Mbps, 48 Mbps, and 54 Mbps. Interestingly, the authors note that the packet size does not affect the power dissipation, which they verify by varying the packet size, nor does the transmission protocol whether it is TCP or UDP.

Their power model consists of four states: ltransit, htransit, low, and high. The device is in ltransit or htransit states only briefly when it transfers data. The transition from low to high occurs when more than 15 packets are transmitted, and the transition from high to low occurs when 8 or less packets are transmitted. The inferred power model:

State	Model
ltransit, htransit	1000 mW
low	20 mW
high	$710\ \mathrm{mW} + [(48 - 0.768) \cdot \mathrm{R_{channel}}]\mathrm{R_{data}}$

where $\mathrm{R_{channel}}$ is the channel data rate and $\mathrm{R_{data}}$ is the number of packets transmitted (sent and received).

The authors also modeled cellular 3G in their work. The system variables were UL/DL queue sizes, and state transition timeouts (see Section 2.4.2). The authors inferred the UL/DL queue sizes by sending a packet from the phone to a server with varying the packet size from 10 B to 1 KB apparently in 1 byte increments.

The timeouts were inferred by fetching a 80 kilobyte http file 30 times with a period that increased from 1 to 29 seconds in one second intervals. They recorded two round trip times. The first, RTT1, is the time between sending TCP SYN and receiving the corresponding SYN-ACK. The second, RTT2, is the time between sending HTTP GET and receiving the first data packet. These are used to deduce the timeouts for DCH-FACH and FACH-IDLE transitions. The authors do not consider that the system might have any other promotion and demotion strategies (see Section 2.4.2). The inferred model:

UL queue	151 bytes
DL queue	119 bytes
DCH $\rightarrow$ FACH timeout	6 s
FACH $\rightarrow$ IDLE timeout	4 s

For the validation of the model, the authors used a set of macrobenchmarks that utilize several subsystems at the same time. According to their measurements, their absolute average error was less than 10% over all benchmarks.

Chapter 4

Statistical Tools

This chapter describes the statistical tools and gives an overview on the mathematics behind it. Statistical methodology, linear regression analysis, and cross-correlation are introduced. This chapter follows the description given in [26].

4.1 Regression Analysis

Regression analysis is a technique for modeling the relationship between one or more *dependent variables* and one or more *independent variables*. In *single regression*, there is only one independent and one dependent variable. In *multiple regression*, the number of independent variables is more than one. Likewise, in *multiple multivariate regression*, the number of dependent variables is more than one. In general regression analysis, there is no assumption on the relationship (e.g., linear) between the independent variables and the dependent variables. For this work, multiple regression where there are more than one independent variable and a single dependent variable is applied. This chapter develops the notation according to this assumption.

We assume that we have n independent variables:

$$(x_1, \ldots, x_n).$$

and we assume that we have a single output variable denoted by y. In order to infer the model, we have to do *calibration*. That is, we take k samples of the dependent variable to form the dependent variable vector denoted by Y

$$Y = (y_1, \ldots, y_k)$$

and we also sample k samples of the n independent variables to form the *calibration set*

$$\{(y_i, x_{1i}, \ldots, x_{ni})\}_{i=1}^{k}.$$

With this calibration set, we can form the *model matrix* X whose single row consists of the independent variables of a sample and whose dimensions are

$n \times k$:

$$X = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{1k} & x_{2k} & \cdots & x_{nk} \end{pmatrix}.$$

In regression analysis, the goal is to find a set of *unknown parameters* denoted by B. Here, B is a k-element vector of the form

$$B = (\beta_1, \ldots, \beta_k)$$

where the elements β_i, $i \in [1, k]$ are called the *regression coefficients*. We assume some known relationship between the function f that relates the X and Y. That is, we solve the parameters B according to some "goodness criteria" from the equation

$$Y \approx f(X, B) \tag{4.1}$$

To solve the regression coefficients B, we have to consider the relation between the number of independent variables n and samples k we take. If $k < n$, the system is *underdetermined*, and there is usually no single solution for the regression coefficients. If $k = n$ and f is linear, we can find an exact solution for B by solving the set of linear equations given by Equation (4.1). If $k > n$ the system is *overdetermined*, and we cannot find an exact solution for B, thus we have to choose a "best fit" according to some criteria. If f is non-linear, there can be none, one, or more than one solution.

If we are given k observations of n unknowns, we denote $(n - k)$ as the *degrees of freedom* which describes how much freedom there is to do the fitting; the k samples are needed to provide the fitting. The $(n - k)$ remaining samples are used to evaluate the goodness of the fitting.

4.2 Linear Regression

In *linear regression*, we assume that the function f in Equation (4.1) is linear. Thus, the model is of the form

$$y_i = \beta_1 x_{1i} + \cdots + \beta_k x_{ni} + \epsilon_i, \quad i = 1, \ldots, k$$

where ϵ_i is the *error term*. The equation can be written equivalently in a matrix form

$$Y = X\beta + \epsilon \tag{4.2}$$

where ϵ is the *error vector*. The above equation can be expanded into:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{1k} & x_{2k} & \cdots & x_{nk} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \tag{4.3}$$

For the linear regression to be applicable, the behavior between the independent and dependent variables must be linear, the sample size must be greater or

equal to the number of unknowns, that is, $k > n$, and the measurements of the independent and dependent variables must be error free.

Since Equation (4.2) is over-determined, perfect fitting cannot be done. Thus, we need to choose some kind of an *estimation method* with which we can calculate the regression coefficient vector B. Here the ordinary least squares (OLS) method is presented.

4.2.1 Ordinary Least Squares

In the Ordinary Least Squares (OLS) estimation, we start with Equation (4.2):

$$Y = X\beta + \epsilon.$$

The goal is to find an estimate $\hat{\beta}$ that minimizes the *sum of least squared residuals*. That is, we are given k observations and we define the residual of the ith, $1 \leq i \leq k$ observation as

$$\epsilon_i = y_i - X_i\beta$$

where X_i denotes the ith row of the design matrix X. The sum of the least squared residuals S is defined as

$$S(\beta) = \sum_{i=1}^{k}(y_i - X_i\beta)^2 = \sum_{i=1}^{k}\epsilon_i^2 = \epsilon^2$$

The ordinary least squares estimate, denoted by $\hat{\beta}$, produces minimum value for the function for the sum of squared residuals S. To see this, observe that

$$\begin{aligned} S(\beta) = \epsilon^2 = \epsilon^T\epsilon &= (Y - X\beta)^T(Y - X\beta) \\ &= Y^TY - 2\beta X^TY + \beta^TX^TX\beta \end{aligned}$$

Function $S(\beta)$ is quadratic and $S(\beta) \geq 0$, so we can find the global minimum $\hat{\beta}$ by differentiating it with respect to β:

$$0 = \frac{dS}{d\beta}(\hat{\beta}) = \frac{d}{d\beta}\left(Y^TY - 2\beta X^TY + \beta^TX^TX\beta\right)\Big|_{\beta=\hat{\beta}} = -2X^TY + 2X^TX\hat{\beta}$$

Thus, we find that $\hat{\beta}$ satisfies

$$X^TX\hat{\beta} = X^TY$$

Now, assume that X^TX is invertible to obtain the least square estimate

$$\begin{aligned} \hat{\beta} &= (X^TX)^{-1}X^TY \\ X\hat{\beta} &= X(X^TX)^{-1}X^TY \\ &= HY \end{aligned}$$

where $H = X(X^TX)^{-1}X^T$ is the orthogonal projection onto the space spanned by X. The hat matrix H can be used to compute

- predicted values: $\hat{Y} = Hy = X\hat{\beta}$

- residuals: $\hat{\epsilon} = Y - X\hat{\beta} = Y - \hat{Y} = (I - H)Y$
- residual sum of squares: $\hat{\epsilon}^T\hat{\epsilon} = Y^T(I - H)(I - H)Y = Y^T(I - H)Y$

where I denotes the *identity matrix*.

To explain why $\hat{\beta}$ is a good estimate, we must consider the fact that it is a projection of the independent variable vector Y onto the *model space*, that is, the space spanned by X. The geometrical interpretation for the residual vector $\hat{\epsilon}$ is the vector difference $\hat{\beta} - Y$. In other words, it is the difference between the model space projection and actual measurement. For more analytical considerations, β is the maximum likelihood estimate, and according to the Gauss-Markov theorem it is the best linear unbiased estimate. Any introductory statistical book such as [26] provides the details.

According to [26], the OLS estimates are a good choice if the errors are uncorrelated and have equal variance. Otherwise, other estimation methods should be considered. For this thesis, without detailed knowledge of the system, it seems reasonable to assume the preconditions of the OLS to be valid.

4.3 Variable Categorizations

Regression variables can be divided into *categorical* and *quantitative variables*. Categorical variables are variables that have a discrete number of possible values. If a categorical variable can hold only two values, it is labeled as a *binary variable*. If the variable can have more than two values, it is called an *ordinal variable*.

Quantitative variables are variables where arithmetical relations can be considered. Quantitative variables can have a bounded or an unbounded domain. If the domain is bounded, we denote the variable as an *interval*. Quantitative variables can be either discrete or continuous. Discrete quantitative variables differ from categorical variables in that respect that they have an ordering, and making arithmetical operations (e.g., taking average) can be done to them while it does not make sense to take an average of a categorical variable.

Dummy variables are categorical binary variables that are not direct input variables of the model. Instead, they describe some external event that cannot be directly measured.

4.4 Segmented Regression

Segmented regression is a concept where the relation between the independent and dependent variables is piecewise linear. That is, the relation can be described with two or more straight lines connecting at unknown breakpoints. Segmented regression tools infer both the straight line coefficients and the breakpoint values. For this thesis, the segmented regression implementation provided by Muggeo in [43] is used.

4.5 Cross-correlation

For verifying the model accuracy, a metric that allows us to estimate how good the linear estimation is is needed. For this method, the *cross-correlation* was chosen. In cross-correlation, we are given two discrete finite time-series of data $y_1, \ldots, y_n$ and $y'_1, \ldots, y'_n$, and we calculate the cross correlation r as follows:

$$r = \text{cross}(y, y') = \frac{1}{n-1} \cdot \frac{\sum_i [(y_i - \overline{y}) \cdot (y'_i - \overline{y'})]}{\sqrt{\sum_i (y_i - \overline{y})^2} \sqrt{\sum_i (y'_i - \overline{y'})^2}},$$

where $\overline{y}$ and $\overline{y'}$ denote the averages of the corresponding series.

Chapter 5

Power Modeling

In power modeling, the goal is to form a correlation between a subsystem's *utilization* and its *power dissipation*. A subsystem consists of a set of operating states that have distinct power dissipation behavior. The device can be in an *idle state* such as wait, sleep, or retention (see Section 2.3.1) or an *active state* that has some utilization variable. The utilization variable can be binary (subsystem on/off), limited domain (e.g., CPU load: 0-100%), or unlimited (e.g., number of packets transferred). In addition, the subsystem can have some active state power management feature on such as DVFS (see Section 2.3.2) or power and clock gating (see Section 2.3.1).

All idle and active states are considered as the *operating states* of the subsystem. Each operating state has its characteristic power dissipation that is either constant or depends on one or more utilization variables. In power modeling, assume that the power consumption of a state is linear with regards to the operating or utilization state. Linear regression (see Section 4.2) is used to infer this relationship.

In this thesis, a measurement setup that allows us to observe the total power dissipation of the studied device (see Chapter 7) was created. The constructed power model breaks down the total power consumption into subsystem-specific power dissipation. To do this, first the model needs to be *calibrated* with microbenchmarks that utilize individual subsystems while keeping the other subsystems idle. Both the power state of the subsystem and the aggregate power consumption of the device are recoreded and correlated. This is done to all subsystems. To verify the model accuracy, another set of benchmarks is used. These verification benchmarks utilize multiple subsystems and use the model to predict the total power dissipation by summing the individual subsystem dissipations. This sum is correlated with the measured power dissipation to provide a goodness of the fit metric.

5.1 Calibration of Subsystems

We begin by first considering how to calibrate the model for a single subsystem. We are given a set of subsystems $C = \{CPU, WiFi, cellular, \cdots\}$, and for

each subsystem $c \in C$ we have n operating states that have individual power dissipation characteristics. We define a function u that describes a subsystem utilization as follows:

$$u_{kt}(c) = \begin{cases} 0 & \text{device is not in } k\text{th utilization state at time } t \\ U_C^k(t) & \text{device is in } k\text{th utilization state at time } t \end{cases} \tag{5.1}$$

Here the operating function $U_C^K(t)$ domain can be either constant (e.g., 1), limited domain (e.g., [0,100]), or unlimited domain (e.g., [0,∞]). This domain depends on the subsystem in question.

We denote the total aggregate power dissipation of the device at time t as $P_t(C)$ and individual subsystem power dissipation at time t as $p_t(c)$, $c \in C$.

Assumption 1 *For calibration, we assume that the measured total aggregate power dissipation $P_t(C)$ equals to the subsystem-specific power dissipation $p_t(c)$. That is,*

$$P_t(C) = p_t(c), \quad 0 \leq t \leq k$$

and all the other systems are switched off or their power dissipation is negligible compared to the systems under test, since the training set stresses heavily the devices under test and keeps the other systems idle. Thus,

$$P_t(d) = 0, \forall d \in C \setminus c.$$

When we calibrate the model, we take k samples of the device operating states:

$$\{u_{1i}(c), u_{2i}(c), \ldots, u_{ni}(c)\}_{i=1}^{k} \tag{5.2}$$

and k samples of the total power dissipation:

$$\{P_i(C)\}_{i=0}^{k} \tag{5.3}$$

We use the regression model and notation of Section 4.2. We denote the set of regression coefficients for the device c by

$$\beta_1(c), \ldots, \beta_n(c)$$

and the set of error terms

$$\epsilon_1(c), \ldots, \epsilon_n(c)$$

Now, given Equations (5.2), (5.3), and Assumption 1, we can form the linear equation similar to that of 4.2:

$$P(c) = U(c)\beta(c) + \epsilon(c) \tag{5.4}$$

or in expanded form:

$$\begin{pmatrix} p_1(c) \\ p_2(c) \\ \vdots \\ p_k(c) \end{pmatrix} = \begin{pmatrix} u_{11}(c) & \cdots & u_{n1}(c) \\ u_{12}(c) & \cdots & u_{n2}(c) \\ \vdots & \ddots & \vdots \\ u_{1k}(c) & \cdots & u_{nk}(c) \end{pmatrix} \begin{pmatrix} \beta_1(c) \\ \beta_2(c) \\ \vdots \\ \beta_n(c) \end{pmatrix} + \begin{pmatrix} \epsilon_1(c) \\ \epsilon_2(c) \\ \vdots \\ \epsilon_n(c) \end{pmatrix}$$

5.2 Calibrating Full-system Model

Now that we have a method to calibrate a single subsystem, we must consider the interactions between subsystems when performing the calibration: Some devices cannot be used without utilizing other devices. Thus, when we utilize a single device that needs another device for its load, we must subtract the consumption caused by the other subsystem in order to properly calibrate the linear model for the new subsystem. For example, to utilize the graphics card, we need to utilize the CPU in order to generate a load for the graphics card (GPU). Thus, we must first calibrate the model for the CPU, perform the microbenchmark that utilizes the GPU, subtract the CPU power dissipation, and after that form the linear model for the GPU device.

For each device $c \in C$, we define a set of nuisance subsystems $N(c) \subseteq C \setminus c$. When measuring the total power dissipation $P(C)$ we define the *real power dissipation* P' as the difference between the measured total power dissipation $P(C)$ and subtracted nuisance power dissipation

$$P'(c) = P(C) - \sum_{d \in N(c)} p(d).$$

where $p(d)$ denotes the modeled power dissipation of the device $d \in C$. Then we apply Equation (5.4):

$$P'(c) = U(c)\beta(c) + \epsilon(c).$$

5.3 Model Verification

To verify the model accuracy, a cross-correlation metric (see Section 4.5) is used. Since the calibration was done by utilizing only a single subsystem if possible, one has have to verify that the model gives precise results even when more than one subsystem is simultaneously active. Thus, another set of benchmarks that utilize multiple subsystems simultaneously was runm the total power dissipation of the device was measured, and the model was used to predict the individual subsystemwise power dissipation $p(c)$, $c \in C'$, for a subset $C' \subset C$. Then the cross-correlation

$$\text{cross}\left[P(C), \sum_{c \in C'} p(c)\right]$$

was calculated and used as the metric for the goodness of the fit.

5.4 Warm-up and Cool-down

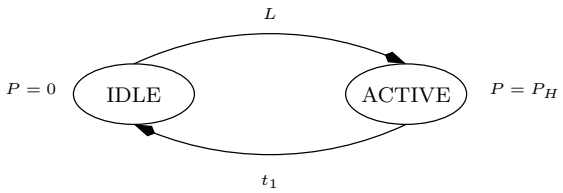
A subsystem usually has some time delay before it enters into an operating state. Likewise, when it leaves the operating state, some delay is exhibited. When the calibration is performed, these warm-up and cool-down periods must be filtered out of the measurements. In the system, start first utilizing the subsystem with a microbenchmark, and wait for a subsystem-specific *grace period* until

the system starts the utilization and power dissipation. Similarly, when the benchmarking ends, stop the measuring, and after another *grace period* drop the device utilization. See Section 7.5 for an example of linear fitting and warm-up/cool-down periods.

Since these transition periods consume little time compared to the constant utilization periods we ignore them in the model. Some accuracy is lost but the overall energy consumption accuracy should still be reasonable.

5.5 Power Transitions

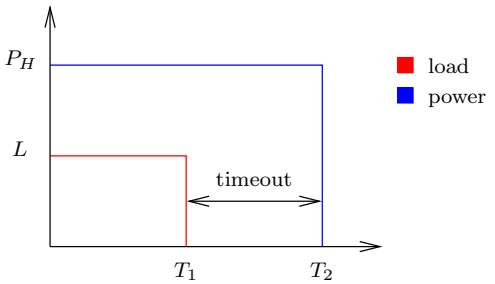
All devices can be considered state machines. The simplest model involves only *idle* and *active* states. In addition to the power states, the device has the concept of *power transition* between the power states. The simplest model can be described as a state machine graph:



Here the device is first in the idle state consuming no power, $P = 0$. When the device is utilized with a *load* L, it moves into a high power state where the power consumption is $P = P_H$. See also the discussion in Section 2.4 on the "statefulness" of peripheral subsystems.

5.5.1 Timeouts

Power transitions can be caused by a timeout. This usually occurs when the utilization of a subsystem has ended. Due to the fact that the subsystem utilization periods tend to cluster, it is beneficial not to switch the device off immediately, especially if the setup of the device consumes lots of power. This is especially typical for the 3G modem (see Section 2.4.2).

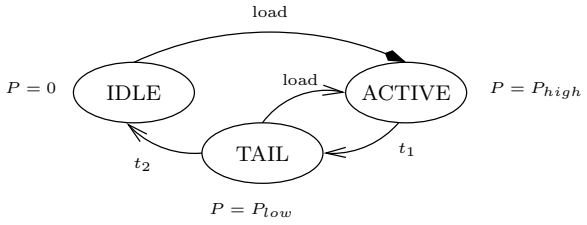


Here, the subsystem has utilization L which ends at T_1 but the subsystem is not switched off until at T_2 giving the timeout $T = T_2 - T_1$. In the above example,

if it was observed that the load drops at T_2, a dummy variable would be set to the value 1 during the timeout period.

5.5.2 Tail States

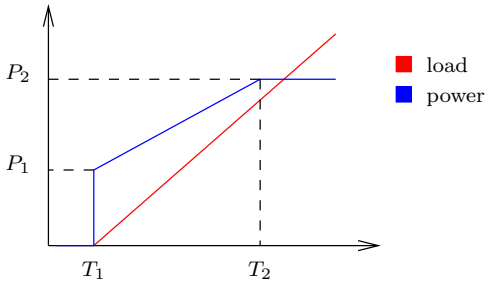
If the transition from idle to active state requires a laborious setup and initialization, one can employ some tail state where the system remains somewhat active but consumes less than in the active state and from which the transition to the active state is less consuming. These *tail states* are used in, for example, Cellular 3G modems, due to the high initial setup time required when moving from idle to active. A tail state can also involve some power or clock gating (see Section 2.3.1) which can require data retention to the main memory, for example.



Dummy variables similar to timeouts presented in the previous section are used to encode transitions to tail states.

5.5.3 Device Utilization

Device utilization can be bounded or unbounded. Bounded utilization is when the utilization has a limited interval and unbounded when there is no upper limit for the utilization. For example, CPU utilization has a bounded interval consumption having the range from 0 to 100 percent, while WiFi card has an unbounded utilization: the number of packets transferred per second can range from 0 to ∞[1].



Above is an example of unbounded utilization. The load starts at T_1 and increases indefinitely. At T_2, a saturation point for power dissipation is reached. At T_1, the power dissipation immediately rises to level P_1 and increases linearly

[1] In practice, there is always an upper limit, however.

until the saturation point, reaching P_2. It is typical for most of the devices to have a constant power dissipation subsystem and a linearly dependent subsystem. The constant dissipation is always present when the device is on, and the linearly dependent subsystem depends on the utilization value. To infer unbounded utilization with bounded power dissipation segmented regression is used (see Section 4.4).

5.5.4 Operating Modes

In addition to utilization, the devices can have several operating modes that have an effect on the power dissipation, e.g., the CPU clock frequency or WiFi power saving mode. It is typical that these operating modes affect the baseline and maximum power intake. Thus, these must be considered in the model for the system when it is inferred.

For example, to encode the operating modes for CPU where we have two operating frequencies, f_L and f_H, we could define the utilization functions as follows:

$$f_{L_{kt}} = \begin{cases} 1 & \text{CPU frequency is } f_L \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

and a similar definition for $f_{H_{kt}}$. Now we define two utilization variables and two regression coefficients, $u_{L_{kt}}(c) = U_C^k(t) \cdot f_{L_{kt}}$ and $u_{H_{kt}}(c) = U_C^k(t) \cdot f_{H_{kt}}$.

Chapter 6

Nokia N900

This chapter surveys the Nokia N900 mobile phone hardware subsystems and relevant power saving features to be used as a reference when validating the model later. The chapter also provides the power dissipation specifications if obtainable from the specifications.

Nokia N900 is a smartphone that consists of several subsystems providing access to phone calls, taking photographs, SMS, navigation, and connecting to the Internet, among others. In order to achieve this, the phone is equipped with a global OMAP3 System-on-a-Chip acting as the central processor and a number of peripherals providing various hardware features such as WiFi, Bluetooth, Cellular, Display, GPS, FM receiver and transceiver, audio subsystem, flash eMMC, and NAND for persistent storage, see Figure 6.1.

6.1 OMAP3430

The main component of the Nokia N900 is the Texas Instruments OMAP 3430 System-on-a-Chip (SoC) [3, 10]. It is equipped with several subsystems: ARM Cortex A8 CPU, PowerVR SGX 530 GPU, and Imaging, Video, and Audio (IVA) accelerator based on TI C64x VLIW DSP Core. The IVA accelerator can be used to encode and decode MPEG/H.264 video up to HD resolutions. It has several I/O interfaces for camera, display, and other sensors. The chip is manufactured with 65 nm process.

6.1.1 Power Management Techniques

Several OMAP subsystems support DVFS (see Section 2.3.1) to reduce dynamic power dissipation. In addition, OMAP supports Dynamic Power Switching (DPS) which allows the subsystem to switch to a lower power higher wakeup-latency state when the system is waiting for an event. Another similar technique is Standby Leakage Management (SLM) which switches the system to deeper sleep states when the system is completely idle. The difference between DPS and SLM is that DPS is used when the system is active, and DPS requires the system to predict when a wakeup is going to occur while in SLM the system

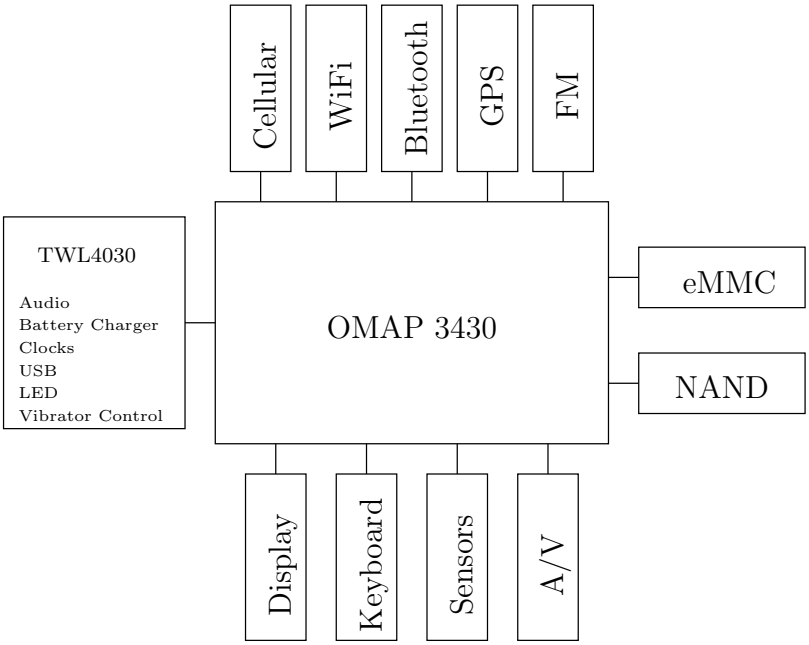


Figure 6.1: Nokia N900 overall schematics. This figure is a logical view of the architecture and does not reflect the physical chip layout.

is completely idle and is only waken up by an external event such as a user touching the keyboard.

OMAP has two kinds of clocks [11, 4.1.4.1]: *interface* and *functional*. The interface clocks are used to synchronize the communication between the subsystems while the functional clocks are used to synchronize the computation occurring inside a subsystem. DVFS has an impact on the latter kind of clocks.

6.1.2 Power Management Architecture

OMAP power management architecture is based on *domains*. There are several domains that are used to implement clock and power gating (see Sections 2.3.1 and 2.3.2) and the DVFS.

Clock gating minimizes the dynamic power dissipation and is implemented with a set of *clock domains*. By gating the clocks of a domain, all switching activity in the domain is stopped.

Power gating is implemented with *power domains* [11, 4.1.3.2]. Each power domain has its own independent power rails which allow toggling the power supply on or off. In clock gating, the subsystem retains its state indefinitely but in power gating all state is lost when the supply power is cut. Thus, the system needs to store the state to some retention registers where it can be restored when the power-gated subsystem is woken up. To implement this, OMAP defines *power states* for each power domain [11, 4.6.1.4]:

ACTIVE	fully operational
INACTIVE	clocks are off
RETENTION	context is stored in memory
OFF	logic is off, clocks are off, no context is stored

OMAP 3430 has several power domains described below. Not all of them implement all power states. That is, for certain power domains, retention state is not required. The WKUP power domain is always active. Some of the domains are controllable by the operating system, whereas others are controlled by the OMAP chip itself.

name	explanation	control
MPU	microprocessor domain	HW
IVA2	audio/video DSP	HW
NEON	SIMD multimedia Co-processor domain	HW
CORE	interconnect, memory, peripherals, clock	HW
SGX	GPU	SW
DSS	low power peripherals	SW
CAM	camera controller	SW
EFUSE	eFuses	HW
WKUP	wakeup domain (always on)	HW
USBHOST	USB host	SW
EMU	emulation domain (for debugging)	HW
PLL1..5	clock generators	HW

Voltage domains [11, 4.10.2] are used to implement dynamic voltage and frequency scaling (DVFS). OMAP implements them by allowing the operating system to define up to 6 operating performance points (OPPs) which are voltage/frequency pairs. There are two freely scalable (OS definable) voltage domains: VDD1 and VDD2. In addition, there are three memory controlled voltage domains: VDD3, VDD4, and VDD5. The available domains [11, 4.10.1] and their operating performance points are:

VDD	Domain	Available OPPs	control
1	CPU	OFF/RET/OPP1-6	HW/SW
2	CORE	OFF/RET/OPP1-3	HW/SW
3	wakeup	Low/Normal/Emulation	HW
4	CPU SRAM	MEM-OFF/RET/VDD1$\in$OPP1-5	HW
5	CORE SRAM	MEM-OFF/RET/VDD2$\in$OPP1-3	HW

Controlling of the power and clock gating is done by the Power, Reset, and Clock Management Module (PRCM) [11, 4.2]. PRCM is divided into two modules: the Power Reset Manager (PRM) which is responsible for handling the power domains, voltage domains, subsystem resets, wakeups from sleep states and subsystem clock source control. The second module is the clock manager (CM) which handles the clock signal generation and distribution through the clock tree. The interface for the PRCM module is implemented in the Maemo kernel (Nokia's vendor-modified Linux kernel) in the file `arch/arm/mach-omap2/prcm.c`.

See Figure 6.2 for an overview of available subsystems, power and voltage domains.

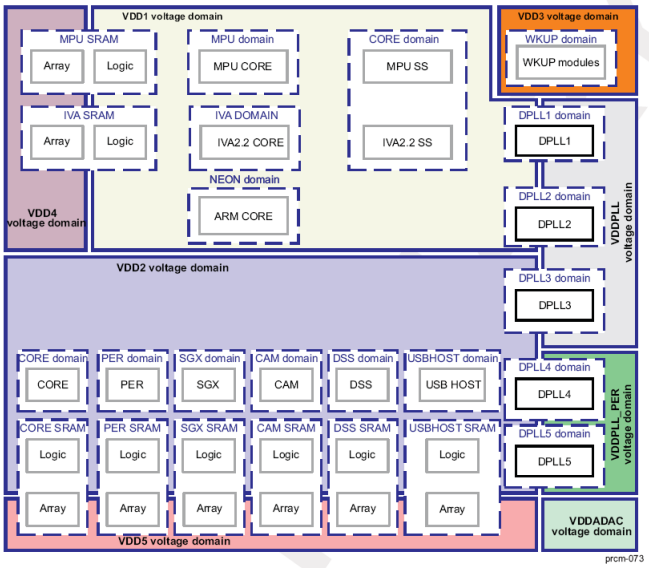


Figure 6.2: OMAP 3430 SoC Subsystems, Power and Voltage domains. (Image from [11].)

6.2 CPU

The CPU of the Nokia N900 device and its OMAP3430 board is a 32-bit ARM Cortex A8 with ARM version 7 ISA + Thumb-2^{TM}, Jazelle Java accelerator. The core includes a NEON extension which is a SIMD coprocessor providing support for multimedia handling. The CPU has 16 32-bit registers, 16 kB L1 instruction and data cache, and 256 kB unified L2 cache [11, 3.1.2]. The ARM subsystem is included in the MPU power domain that supports all power states (OFF, INA, RET, and ACT), and the NEON coprocessor (OFF, ON) has its own power domain. According to [6], the worst case power dissipation for the integrated OMAP 3 ARM CPU is 750 mW at 800 MHz.

6.2.1 Power Management

Nokia N900 comes with a Nokia-modified Linux kernel. The kernel provides the CPU idle and active power management through an interface consisting of a driver and a policy manager. The driver provides low-level access to the power management hardware, and the policy manager decides when to transfer between the operating states. The idle management interface is called *cpuidle* and the active (frequency scaling, DVFS) management is *cpufreq*.

The idle power management [45] low-level driver for the Maemo Linux kernel is implemented in `arch/arm/mach-omap2/cpuidle34xx.c`. The driver defines 7 sleep states but the `sys/devices/system/cpu/cpu0/cpuidle` interface only reports 4 of them. Here we describe them as reported by the sysfs interface:

state	target_residency	exit_latency	MPU	CORE	BM
C1	5	272	ON	ON	-
C2	309	286	ON	ON	+
C3	46507	2001	RET	RET	+
C4	484329	22779	OFF	OFF	+

Here `target_residency` indicates the amount of time in microseconds that is needed to be spent in the state in order to make the transition worthwhile. The variable `exit_latency` indicates the amount of time in microseconds to bring the system back into a fully functional state. BM stands for `CPUIDLE_FLAG_CHECK_BM` that indicates the DMA or other bus activity is not correctly updated during the sleep state and the system should perform necessary tasks during wakeup to make the system consistent. The MPU and CORE columns indicate which power state the subsystems enter during the sleep state.

The Linux kernel supports various idle policy management algorithms. The idle power management policy side is not customized by Nokia. The default policy (or *governor* in the kernel terminology) for the Maemo kernel is the *menu governor*. The policy implementation is in `drivers/cpuidle/governors/menu.c`. According to [45], the menu governor considers various parameters including

- processes' expected sleep time
- latency requirements
- how much time is spent in previous C-states
- bus activity

and chooses the sleep state that has the maximum power advantage with the least performance impact.

The active power management has similar two-tier implementation to the idle power management. The low-level OMAP-specific driver interface is scattered in various files in `arch/arm/mach-omap2` and in the file `arch/arm/plat-omap/cpu-omap.c`. The policy side is uncustomized Linux kernel stock on-demand CPU frequency governor.

The on-demand governor [46] algorithm that decides the operating frequency is implemented in the file `drivers/cpufreq/cpufreq_ondemand.c` and basically scales the CPU operating frequency based on the system load.

Naturally, the operating frequencies are a discrete set of values. For the Nokia N900, the available CPU operating frequencies and their corresponding operating voltages are:

Frequency (MHz)	Core Voltage (V)
250	1.075
500	1.2
550	1.275
600	1.35

The parameters for the cpufreq on-demand governor are accessible through the sysfs interface: `/sys/devices/system/cpu/cpu0/cpufreq/ondemand` and are the following:

ignore_nice_load	0
sampling_rate	300000 μS (0.3 s)
sampling_rate_max	150000000 μS (150 s)
sampling_rate_min	150000 μS (0.150 s)
up_threshold	95%

The parameter `ignore_nice_load`, if set to 0, indicates that all processes should be included when counting the CPU load. If it is set to 1, processes running with a nice value are not counted to the load.

6.3 Interconnect

The interconnect [11, 5.1] is responsible for connecting the subsystems to each other. It provides access control through configurable firewalls that limit the access between the subsystems and has a 4-level hierarchy:

L1	internal to CPUs
L2	IVA and MPU domain
L3	chip-level interconnect connects the subsystems
L4	external peripheral interconnect

L3 interconnect [11, 5.2], which is the main bus for the OMAP chip, is in the CORE power domain and is controlled by PRCM. It has a *smart-idle* mode which makes the interconnect go to a sleep mode once all requests have been served.

6.4 GPU

PowerVR SGX 530 is a programmable GPU with a universal shader model. It supports DirectX 10.1 and OpenGL ES 2.0. According to [34], the PowerVR GPU is clocked at 110 MHz on N900.

SGX 530 has its own power domain and it supports the ON and OFF power states [11, Table 4-26], lacking the support for retention states. The SGX Subsystem has both the interface and functional clocks [11, 13.2.1.1], both of which can be gated. The functional clock can be either a division of the interface clock with ratios 1:3, 1:4, or 1:6 applied. Alternatively, a constant 96 MHz clock can be issued from the DPLL4 clock generator.

The SGX subsystem supports the following clock gating features [11, 13.2.1.3]

- deep power sleep (all clocks are gated)
- idle (2D and 3D clocks are gated)
- 3D (no clocks are gated)

The driver for the SGX subsystem is located in the `drivers/gpu/pvr` directory in the kernel sources. There is support for some dynamic voltage and frequency

scaling based on the utilization of the device but due to the lack of specification, it is not clear from the sources whether this scaling is implemented by the OMAP hardware or the kernel.

6.5 Memory

Nokia N900 has a single 256 MB low-power LPDDR1 RAM chip that is clocked to 166 MHz [34]. There are no specifications available on the power requirements of the memory chip.

6.6 Display

Nokia N900 has a Sony ACX565AKM 3.5-inch resistive touch-screen display with an 800x480 pixel resolution and 16-bit color depth. The display has a Content Adaptive Backlight Control (CABC) power saving feature that according to [34] has four available modes:

- off
- UI
- still-image
- moving-image

The CABC allows reducing the backlight brightness level according to the image shown. E.g., if the image contains only dark pixels, the backlight brightness can be lowered. There seems to be no specification for power dissipation available.

The driver for the panel provides access for the backlight brightness through a sysfs interface and is in `drivers/video/omap2/displays/panel-acx565akm.c`.

According to [7], the panel power dissipation is 61.13 mW and the typical voltage and current for backlight are 9.6 V and 15 mA, respectively, yielding a typical power dissipation of 144 mW.

6.7 WiFi

The WiFi chip-set is a Texas Instruments WL1251 that supports the 802.11b (1, 2, 5.5, and 11 MB/s) and 802.11g (6, 9, 12, 18, 24, 36, 48, and 54 MB/s) standards and the following extensions:

- 802.11i WPA2 or RSN security extension
- 802.11d regulatory domain extension
- 802.11k radio resource management extension
- 802.11e QoS extensions

According to the FCC Compliance Test Reports for the WiFi [1], the Nokia N900 WiFi module has the following transmit power characteristics:

Mode	Rate	Power
802.11b	11 Mbps	239 mW (398 mW)
802.11g	6 Mbps	156 mW (260 mW)

The power dissipation in parenthesis indicates the lower limit for the power dissipation assuming a 60% efficiency ratio of printed surface-mounted WiFi antenna [34]. See Section 2.4.1 for generic WiFi power saving features.

The kernel driver for the WiFi chip-set is in `drivers/net/wireless/wl12xx/` directory.

6.8 Cellular

Nokia N900 has a Nokia-proprietary cellular modem labeled Rapuyama in the hardware specifications. It supports the 850/900/1800/1900 MHz GSM bands and the 900/1700/2100 MHz WDCMA bands. The chip supports the 3GPP Release 5 with WCDMA/HSDPA and GPRS data bearers. The maximum speeds according to [34] are:

Network	Type	DL (kB/s)	UL (kB/s)
2G	GPRS multi-slot class 32	107	64.2
2.5G	EDGE class A	296	177.6
3G	3GPP release 5/PS	384	384
3G	3GPP release 5/HSDPA	10000	5800

The chip supports the Dual Transfer Mode (DTM) that allows simultaneous voice and packet data connection in GSM/EDGE networks. According to [34], the transmit power is adjustable according to the following table:

Frequency	min	max	Number of steps
GSM 900 MHz	3.2 mW	1800 mW	15
GSM 1800 MHz	1.0 mW	900 mW	16
WCDMA all	0.01 μW	180 mW	75

These adjustments are not user-controllable. See Section 2.4.2 for the generic Cellular modem power management features.

The FCC Compliance Test Report [2][1] results for the average transmit power for the cellular networks are:

Frequency	Power
GSM 850	437 mW
EGPRS 850	58 mW
GSM 1900	1445 mW
EGPRS 1900	407 mW
WCDMA 1700	166 mW

[1]This applies to frequencies used in the US but the difference to European frequencies should not be large.

6.9 Solid State Disks

Nokia N900 has one NAND and one eMMC chip [34]:

Type	NAND	eMMC
Manufacturer	Numonyx	Toshiba
Voltage (read ops)	1.8 V	2.7 V
Maximum Current	?	3.5 mA
SoC bus size	16-bit	8-bit
Size	256 MB	32 GB
Sequential R/W	?	33.38/11.8 MB/s
Random R/W	?	16.7/0.42 kB/s

The 256-MB NAND is formatted with UBIFS file system which is a file system written specifically for flash drives. The mount point for the NAND flash is `/` (root). The 32-GB eMMC is split into three partitions:

file system	size	mount point
swap	768 MB	-
ext3	2 GB	/home
VFAT	25 GB	/home/user/MyDocs

There is no indication on what is the exact model number for the NAND flash used in the N900, which makes finding the data sheet and thus deducing the power draw impossible.

6.10 Battery and Charger

The Nokia N900 has a 1320 mAh 3.7 V Li-Ion battery labeled BL-5J. According to Nokia, it provides 9 hours of GSM and 5 hours of WCDMA talk time. The stand-by time is given as 10 days for both GSM and WCDMA.

The charger is labeled AC-10 and it has the following specifications:

	Voltage	Current
Input	AC 100/240 V / 50-60 Hz	
Output	DC 5.0 V	1200 mA

The charger is connected to Nokia N900 with a micro-USB connector.

6.11 Other Peripherals

Nokia N900 includes several other peripherals: a touch-screen controller, the TWL4030 Audio and Power Management chip, Flash Torch, Led, USB, A/V Connector, Accelerometers, Bluetooth, FM Receiver and Transmitter, GPS, Vibrator, Front and Back Cameras, Ambient Light Sensor, Proximity Sensor, Keyboard, and Fuel Gauge. These devices were ommitted from the model to limit the scope of this thesis.

Chapter 7

`eprof` Framework

This chapter describes the experiment framework, `eprof`. The framework is responsible for measuring the power dissipation and device utilization of the system. This information is collected and used to produce plots or perform statistical analysis such as linear regression. The framework also provides access to a serial port present in the Nokia N900 device. In addition, the system has a software-controlled power switch functionality allowing us to do hard power-offs during the experiments. This chapter introduces the hardware and software needed to implement the framework.

7.1 Measuring Electric Power

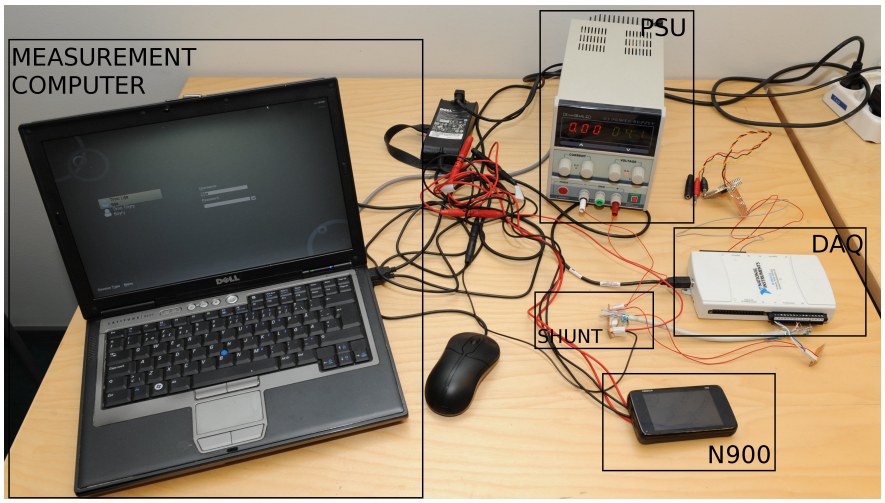


Figure 7.1: Part of the measurement setup. The control computer, power toggle, and serial port interface are not shown.

The electric power measurement setup allows measuring the power dissipation of the device, provides an interfacing capability for connection to the Nokia

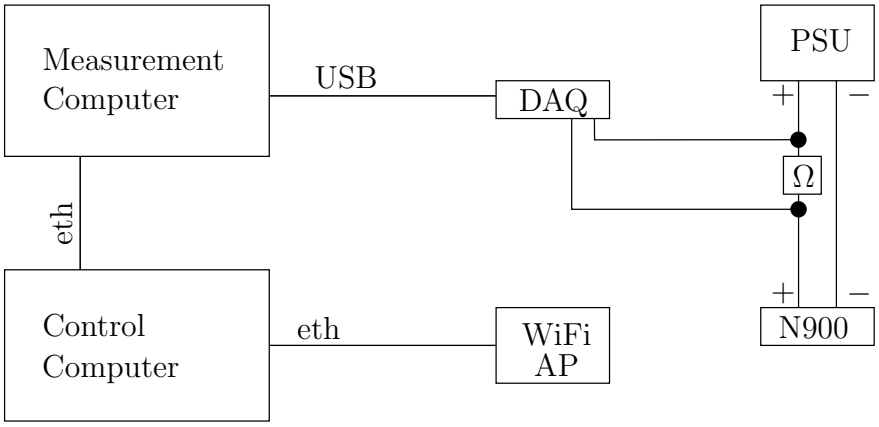


Figure 7.2: `eprof` overall system architecture. Access Point is labeled as AP and Ω indicates the shunt-resistor. The serial port interface, power toggle, and ground connections are omitted from the diagram.

N900, debug serial port, and a software-controlled power switch. The setup can be seen in Figure 7.1 and the overall architecture of the power measurement system in Figure 7.2.

The power measurement system consists of the following subsystems:

Nokia N900 Unmodified Nokia N900 running standard Maemo 5.0PR1.3 operating system is used as the test hardware. The following software packages are installed in order to run the framework: strace, tcpdump, openntpd, bash, python, python-dbus, and cpufrequtils.

DAQ The data-acquisition device is National Instruments NI-USB 6216 DAQ with the following specifications [8]:

Model	USB-6216
Input channels	8 differential/16 single-ended
Input ranges	± 0.2 V, ± 1 V, ± 5 V, ± 10 V
ADC resolution	16 bits
Max sampling rate	400 kS/s
Timing accuracy	50 ppm of sample rate
Timing resolution	50 ns
Sensitivity	4.8 μV @ ± 0.2 V range 10.4 μV @ ± 1 V range 47.2 μV @ ± 5 V range 91.6 μV @ ± 10 V range

For the accuracy, the NI DAQ specifications define the following limits:

range	accuracy	R=max
±0.2 V	$192R + 50$	90 μV
±1 V	$152R + 160$	310 μV
±5 V	$142R + 705$	1420 μV
±10 V	$132R + 1300$	2710 μV

where R is the DAQ multimeter reading. The last column indicates the absolute error when the measurement reading is on the end of the scale.

The input range $\pm$ 0.2 V was chosen for the voltage supply measurements and $\pm$ 5 V for the battery terminal voltage measurements (see Section 2.2).

For the voltage supply measurements, the 16-bit ADC produces a nominal resolution of 6.4 μV assuming 5% over range, and for the terminal voltage measurements we get the nominal resolution 160 μV assuming 5% over range [9, pp. 4-3].

The sampling rate was chosen to be 1000 Hz. We use the differential ground reference setting with floating signal sources [9, pp 4-24] when performing both the load and terminal voltage measurements.

PSU Laboratory power supply unit DC-17/32SB-3-ALED was used to provide 4.1 V DC terminal voltage for the test system.

Fake Battery A fake battery setup that fits into the Nokia N900 battery case allowing to feed current from an external power source. The fake battery setup allows using a laboratory PSU or normal battery as the power source but for the tests it was chosen to use only the PSU since it provides a stable voltage source and allows ignoring the effects of battery draining.

The fake-battery allows connecting the Nokia N900 serial port TX pin into a serial port but it turned out that the serial port causes considerable power dissipation (around 200 mW constantly), so it was not used in the tests. See Figure 7.3 for a picture of the fake battery setup.

The fake-battery is just a connector with no active or passive components.

Shunt-Resistor A shunt-resistor with a 0.1 $\Omega \pm 5\%$ resistance was used. The shunt is connected to the positive/high-side of the current supply (see Section 2.2).

Power Toggle An IRLZ44N MOSFET transistor that is connected to the DAQ Digital Output (DO) source and the high current side of the circuit as the power toggle switch was used. This provides a programmable power switch for the testing system.

Measurement Computer A laptop that interfaces with the DAQ through a USB interface and has the receiving side of the fake battery serial port interface. The computer has openSUSE 11.2 Linux operatins system with 2.6.3.1.14-0.1-desktop kernel and the NI-DAQmx Base 3.4.0 software installed.

The measurement computer instructs the DAQ to start and stop measurements and logs the measurement samples into a text file.

Control Computer a Dell Optiplex 755 running Debian GNU/Linux 6.0. The control computer is used to run the tests and perform the analysis of the results. The use of a separate measurement and control computer is more of a convenience: all the tests and analysis could as well be done in the measurement computer.

Access Point Linksys WRT54GL v1.1 wireless router is used as the WiFi Access Point for performing the WiFi tests. The AP is connected to a private network with the Control Computer.

Given the DAQ specifications and the 0.1 $\Omega \pm 5\%$ shunt-resistor, 4.1 V terminal voltage, and 1000 Hz sampling rate, the measurements can be done with the following specifications:

maximum observable power	8.2 W
observable power resolution	0.260 mW
minimum observable signal length	1 ms

The power measurement system produces a time-indexed text file that is stored on the measurement computer. The measurement system is implemented by hand-written C code and uses an NI DAQmx Base 3.40 application programming interface provided by the National Instruments.

7.2 Measuring Device State and Utilization

To formulate a model between the electric power dissipation and subsystem utilization, both the power and the subsystem operating state (off, on, active, power saving) and the subsystem utilization need to be observed. For this, it was chosen to rely on the data provided by the Nokia N900 Linux kernel. A logging tool called `eprof-log` was developed. The tools observes the following system variables:

- system load through /proc/stat
- OMAP3430 subsystem operating modes
- WiFi packets and bytes transferred/received
- cellular packets and bytes transferred/received
- display brightness
- disk sectors read/written
- device operating mode: flight or normal

The logging tool allows a fine-grained configuration of the state variables and utilization counters that are to be monitored. For example, the tool can be configured to only log the WiFi utilization. To estimate the baseline overhead, the tool supports *none* mode where only a time stamp is periodically written into a file. In the *estimate* mode, all system variables relevant to this thesis are logged into the file.

The device utilization logging system produces a time-indexed text file that is stored on the Nokia N900 SSD disk. The sampling rate of the tool is set to 1 Hz. The logging tool is implemented in C, and it uses various Nokia Maemo interfaces to gather the desired information.

7.3 Control Scripts

The experiment framework control scripts are responsible for configuring both the DAQ and the Nokia N900 utilization measurement systems. The framework is responsible for setting up the system so that a microbenchmark program can be executed in the system. This microbenchmark creates workload for an individual device or subsystem (e.g., CPU) while keeping all other devices idle. This allows us to study the power behavior of a single device.

The framework allows direct controlling of various subsystems and peripherals of the Nokia N900 device:

- toggle WiFi on/off
- turn vibrate on/off
- turn FM radio on/off
- toggle cell radio on/off
- switch 2G, 3G, or 2G/3G modes
- take a picture with the front and back cameras
- play videos with the N900 built-in media player
- suspend/wake up OMAP power domains
- turn the loudspeaker on/off and set its volume
- set the CPU frequency
- turn the display on/off and set its brightness

Certain experiments (especially network) require an external server that sends or receives data. The test scripts allow setting up a server software run in the measurement computer. Nokia N900 can connect to this server software through WiFi or cellular connection, The WiFi connection is achieved by having the measurement computer and N900 share the same WiFi access point, and the Cellular connection is achieved by connecting the measurement computer to the Internet and using a cellular data connection.

The major design goal of the test framework is to have support for automatic non-interactive repeatable batch experiments. After each experiment, the data is collected from the power measurement (Section 7.1) and utilization (Section 7.2) systems for further analysis (Section 7.5). The control scripts are partly written in C and partly in Python. They rely on the Nokia Maemo interfaces to perform the activities.

7.4 Synchronization

The experiment framework (Section 7.3) provides both the power and the device utilization data. The power data is sampled with 1000 Hz and the device

utilization data with 1 Hz. To combine these different frequency signals, we down-sample the power measurements by taking an average (see Section 2.2) and combine the signals into a single multivariate time-series.

The test framework produces two sources of time-indexed data: the DAQ measurements and the subsystem utilization data records. There is no global clock available in the DAQ and N900 setup, and the DAQ system has a buffer and some latency due to the USB interface connecting the DAQ to the measurement computer. Thus, for the synchronization NTPD is used both in the N900 and the measurement computer to make both Nokia N900 and the measurement computer clocks synchronized. Since the granularity between the samples is 1 second, this coarse-grained synchronization is sufficient.

However, if there was a need for more precise synchronization, the serial port interface couldbe used as a signal trigger. The framework supports this but it is not usable since the debug serial port induces a 200 mW constant power dissipation to the system while the idle consumption without the serial port enabled is around 50 mW. See also the discussion in Section 3.1.

7.5 Statistical Analysis

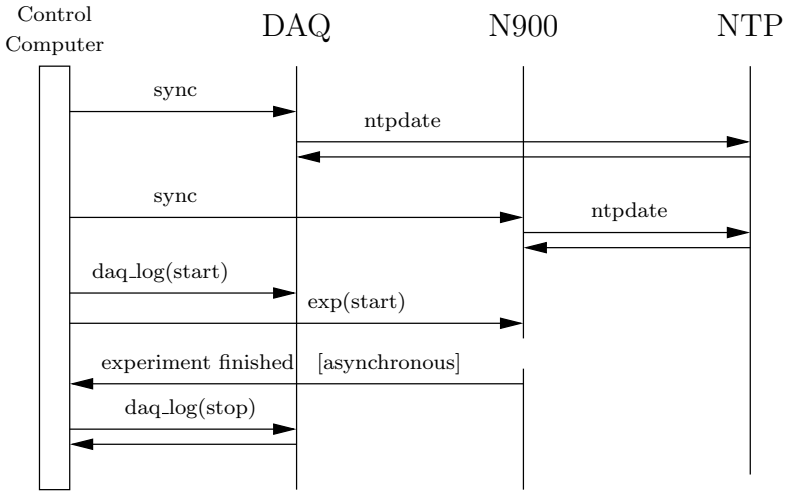
Statistical analysis involves calibrating the linear model. After the linear model has been calibrated, the analysis phase can be used to produce subsystemwise predictions on how the power is consumed.

Before an experiment is run, there is usually a *warm-up period* that involves setting up the system so that the device can be utilized. After the experiment has been run, there is a *cool-down period* where the device moves back to an idle state. When the linear regression analysis is performed, the warm-up and cool-down periods are not interesting. Thus, these uninteresting transition periods are filtered out before the actual regression analysis takes place, see Figure 7.4.

The linear regression and prediction are implemented with the R [51] statistical analysis software.

7.6 Experiment Sequence Flow

Below is a figure describing the sequence flow of an experiment. First, both the DAQ and N900 clocks are synchronized by the control computer sending a command that makes them execute ntpdate respectively. Then the DAQ logging is started after which the experiment is executed.

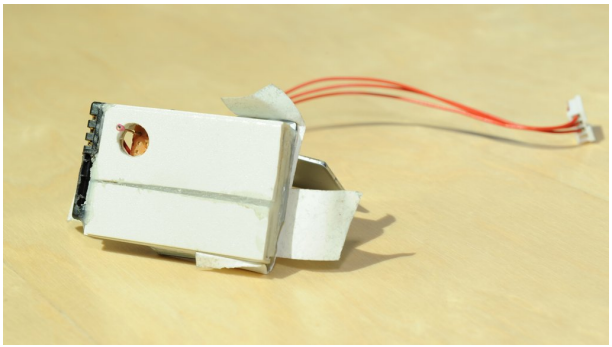


Executing the experiment involves the following steps:

- copy workload scripts and binaries into the N900 host
- start the utilization logging
- execute the actual experiment that utilizes a device
- after the experiment is terminated, send the logs back to the control computer asynchronously

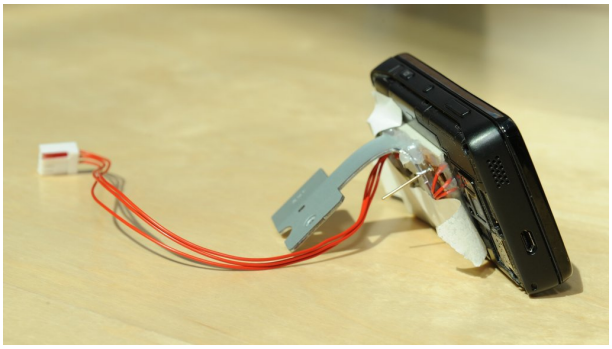
After the experiment has been completed, the control computer stops the DAQ and collects the log from there.

The analysis phase is done offline, and even though it is part of the `eprof` framework, it is not part of the automatic batch execution part so it is not depicted here.



(a) The custom-built fake battery setup with a serial port pin on the upper left corner.

(b) Nokia N900 backside where the debug pins are visible. The TX pin for the serial port is on the right bottom pad's left upper corner. The bottom pad's left lower corner is the RX pin.



(c) Fake battery mounted into a Nokia N900.

Figure 7.3: Fake battery.

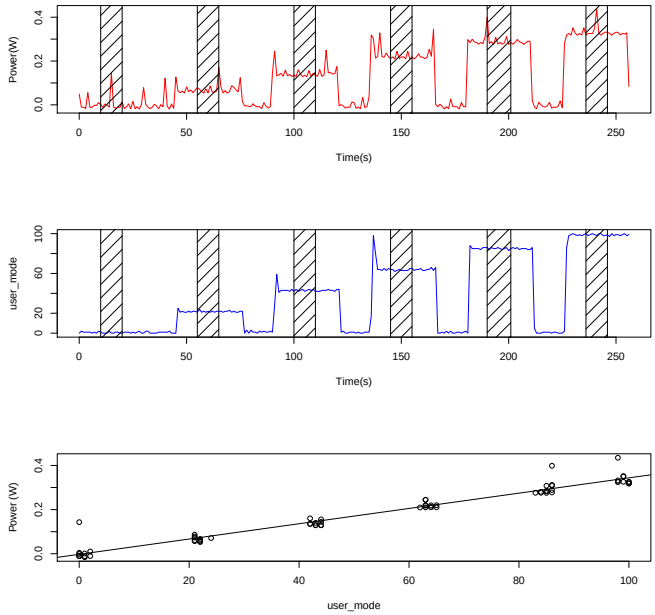


Figure 7.4: An experiment with varying the utilization (middle figure) levels causes varying power dissipation (upper figure). The framework allows tuning of the warm-up and cool-down periods so that only the relevant samples (striped area) are considered when performing the fitting.

Chapter 8

Results

This chapter gives the results of the experiments used to train the linear model. Both the calibration and the verification set results are given here. Discussion and analysis of the results is delayed to the next chapter. The experiments were repeated three times and the results are reported here. All of the experiments yield piecewise linear models with coefficients, constant variables and the points where the behavior of the function changes. One of the experiment results was chosen as the model. The methodology and the experiment framework are described in Chapters 5 and 7, respectively.

8.1 Calibration

In the calibration phase, the linear model regression coefficients are inferred one variable at a time. See Section 5.1 for explanation of the methodology.

8.1.1 Idle

In the idle experiment, both the normal and flight mode (all radios off) was measured. The experiment had 10-second settling periods before starting and after finishing it. Each operating mode was kept active for 180 seconds, after which there was a 20-second idle period before the next mode was switched on. The experiment was repeated three times. The results are presented in Table 8.1.

Table 8.1: Results for the idle experiment. The units are milliwatts.

	#1	#2	#3
flight	18.8	18.7	19.2
normal	39.28	39.7	39.1

From the three repetitions, it is clear that the accuracy of the measurements is clearly in the range of one milliwatt. Thus, it was assumed that the idle

consumption is $P_{\text{idle}} = 40$ mW. The idle consumption is subtracted from further measurements in order to avoid it to be counted more than once.

8.1.2 Log Overhead

In the logging overhead experiment, the overhead of the logging tool flags *none* and *estimate* (see Section 7.2) was measured.

The experiment had 10-second settling periods before starting and after finishing it. Each logging mode was kept active for 180 seconds, after which there was a 20-second idle period before the next logging mode was started. The experiment was repeated three times. The obtained results are presented in Table 8.2.

Table 8.2: Results for the logging overhead experiment. The idle consumption $P_{\text{idle}} = 40$ mW is subtracted from the results. The units are milliwatts.

	#1	#2	#3
none	5.9	4.5	6.3
estimate	20.4	19.9	20.4

For further benchmarks where the logging tool is used, it was assumed that the logging overhead is $P_{\text{log}} = 20$ mW. Thus, the combined overhead of idle and logging is around $P_{\text{baseline}} = P_{\text{idle}} + P_{\text{log}} = 60$ mW, which is subtracted from the following experiments.

8.1.3 CPU

In the CPU experiment, the kernel CPU frequency governor was switched off and the CPU frequency was manually locked to each valid value 250 MHz, 500 MHz, 550 MHz, and 600 MHz. The CPU microbenchmark consists of a program that has a duty cycle corresponding to the desired load level. For each second, the program executed n instructions based on the clock frequency and slept the rest of the second so that the load was in the required level. The load was varied between 0 and 100 in 20 percentage point increments.

The system was kept idle for 5 seconds before and after each experiment. Between each load level the system was kept idle for 15 seconds. The system was kept active for 30 seconds on each load level. The frequency was kept constant while the load was varied. The experiment was repeated three times. When performing the linear regression, 10 seconds from both ends of the active period was cut out (see Section 7.5). The results are in Table 8.3 and Figure 8.1.

Table 8.3: The CPU experiment results. The system CPU load is CPU_{load}. The units are milliwatts.

Frequency	#1	#2	#3
250	$4.2\text{CPU}_{\text{load}}$	$4.2\text{CPU}_{\text{load}}$	$4.1\text{CPU}_{\text{load}}$
500	$7.4\text{CPU}_{\text{load}}$	$7.3\text{CPU}_{\text{load}}$	$7.3\text{CPU}_{\text{load}}$
550	$8.4\text{CPU}_{\text{load}}$	$8.4\text{CPU}_{\text{load}}$	$8.3\text{CPU}_{\text{load}}$
600	$9.9\text{CPU}_{\text{load}}$	$9.8\text{CPU}_{\text{load}}$	$9.7\text{CPU}_{\text{load}}$

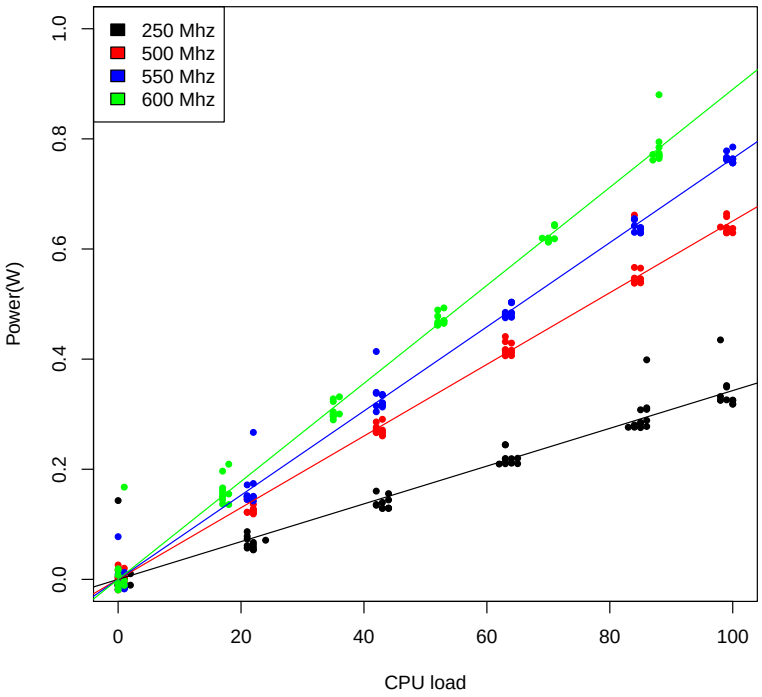


Figure 8.1: The CPU experiment #1 fitting results.

8.1.4 GPU

In the GPU experiment, the PowerVR Insider SDK [4] examples were run and the power consumption was measured. After which the baseline and CPU power dissipation were subtracted, and the average power dissipation was calculated. The method was to observe the overall power consumption and subtract the CPU power dissipation, and also observe the tail states and manually measure their duration. See Figure 8.2.

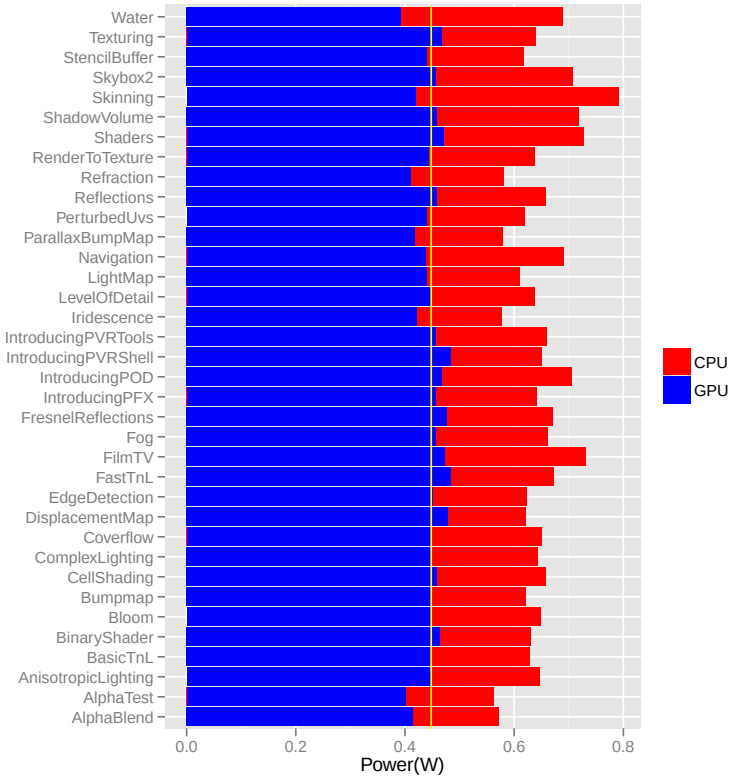


Figure 8.2: The GPU experiment results. The examples are from the PowerVR Insider SDK. The total power dissipation is the sum of both the red and blue bars. The red bar is the CPU power dissipation deduced from the total power dissipation. The remaining blue bar indicates the GPU power dissipation. The yellow line is the average over all experiments. The observed average power dissipation is 448 ± 50 mW.

The tail states were deduced by observing the power measurement plot of all the experiments. Figure 8.3 presents one sample but the rest had similar tail state behavior. The tail states are presented in Table 8.4.

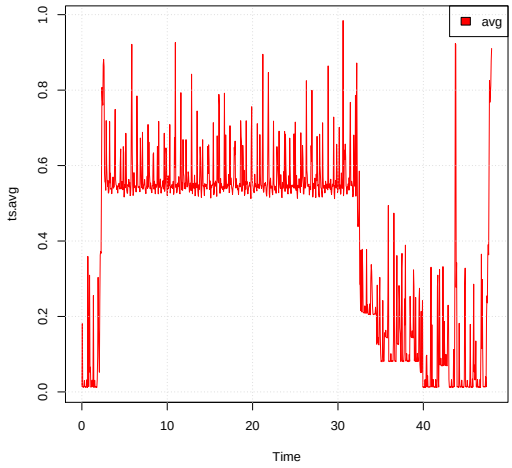


Figure 8.3: Running the Alpha Blend example from PowerVR SDK. The tail states are clearly visible in the Figure.

Table 8.4: The GPU experiment results. GPU ON indicates the GPU being active as reported by the OMAP driver in the Maemo Linux kernel. The tail states are not reported as active states by the kernel.

state	power (mW)	duration
GPU ON	448	-
tail-1	448	3 s
tail-2	266	2 s
tail-3	81.1	5 s

8.1.5 Display

In the display experiment, the backlight brightness was varied between 0 and 255 in 25-step increments. The screen displayed Maemo default background. The state of the display (on or off) was also recorded. The dim timeout of the screen was set to 40 seconds.

The system was kept idle for 5 seconds before and after the experiment. Between each brightness level the system was kept idle for 10 seconds. The backlight was turned on for 40 seconds on each level. The experiment was repeated three times. When performing the linear regression, 10 seconds was cut from both ends (see Section 7.5). The results are in Table 8.5 and Figure 8.4.

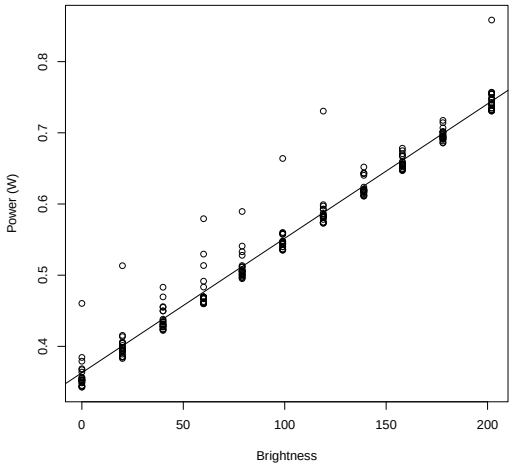


Figure 8.4: The results of the display #1 experiment.

Table 8.5: The display experiment results. The P_{baseline} and CPU power dissipation has been subtracted. BL_{br} is in range 0–255 and BL_{on} has the value 0 if the display is off and 1 if on. The experiment was repeated three times.

N	mW
#1	$1.9\text{BL}_{\text{br}} + 330\text{BL}_{\text{on}}$
#2	$1.9\text{BL}_{\text{br}} + 330\text{BL}_{\text{on}}$
#3	$1.9\text{BL}_{\text{br}} + 330\text{BL}_{\text{on}}$

8.1.6 WiFi

The WiFi experiment began by first measuring the maximum transfer capacity between Nokia N900 and a WiFi access point. It was discovered that the maximum upload and download rate was ca. 600 KB/s. The WiFi was set to 2.437 GHz frequency and the bit rate was 54 MBit/s.

The setup consists of three experiments: *send*, *receive*, and *reply*. In the send experiment, a 1024 byte UDP packet was sent from the device into a local server. In the receive experiment, a 1024 byte packet was sent from the local server into the phone. In the reply experiment, a 1024 byte was sent to a local server and immediately echoed back by the server. The observed state variable was *npackets*, the number of packets transferred (sent and received) through the N900 WiFi interface.

The microbenchmark varied the number of packets sent (received) between 1–10 in 1 pkt/s increments, between 20-100 in 20 pkt/s increments, and between 100–700 in 100 pkt/s increments. The active period lasted 15 seconds, and the system was kept idle for 10 seconds between the increments. Before doing the regression fitting, 5 seconds from both ends of the active period was removed because the experiments had 5 second warm-up and cool-down periods.

The power measurement results here have the baseline and CPU power dissipation subtracted. The experiment was repeated three times, and for each iteration, the linear model was inferred with segmented regression (see Section 4.4). Results for the regression are in Tables 8.6 and 8.7. Figures 8.5 and 8.6 show the measurements and the regression fitting for send and receive experiments; the reply experiment yielded similar results. There is a lot of variance in the experiment but when the device only receives packets, the power dissipation is considerably low.

For the maximum power intake, it was noted that the WiFi device saturated when receiving ca. 800 pkt/s or receiving and sending 1200 pkt/s which yield ca. 1 W power dissipation. When receiving more than 400 pkt/s, the device power dissipation remained constant around 200 mW.

Table 8.6: The inferred models for send and reply experiments. The variable T_{wlan} indicates the number of packets transferred. The experiment was repeated three times.

send			
	#1	#2	#3
$1 \leq T_{wlan} < 4$	$171T_{wlan} + 19.6$	$194T_{wlan} - 32.5$	$189T_{wlan} - 30.3$
$4 \leq T_{wlan} \leq 800$	$0.4T_{wlan} + 773$	$0.3T_{wlan} + 762$	$0.4T_{wlan} + 761$
$T_{wlan} \geq 800$	$1,000$ mW	$1,000$ mW	$1,000$ mW
reply			
	#1	#2	#3
$1 \leq T_{wlan} < 8$	$87T_{wlan} - 13$	$83T_{wlan} + 19$	$86T_{wlan} + 11$
$9 \leq T_{wlan} \leq 1200$	$0.2T_{wlan} + 763$	$0.2T_{wlan} + 763$	$0.2T_{wlan} + 766$
$T_{wlan} \geq 1200$	$1,000$ mW	$1,000$ mW	$1,000$ mW

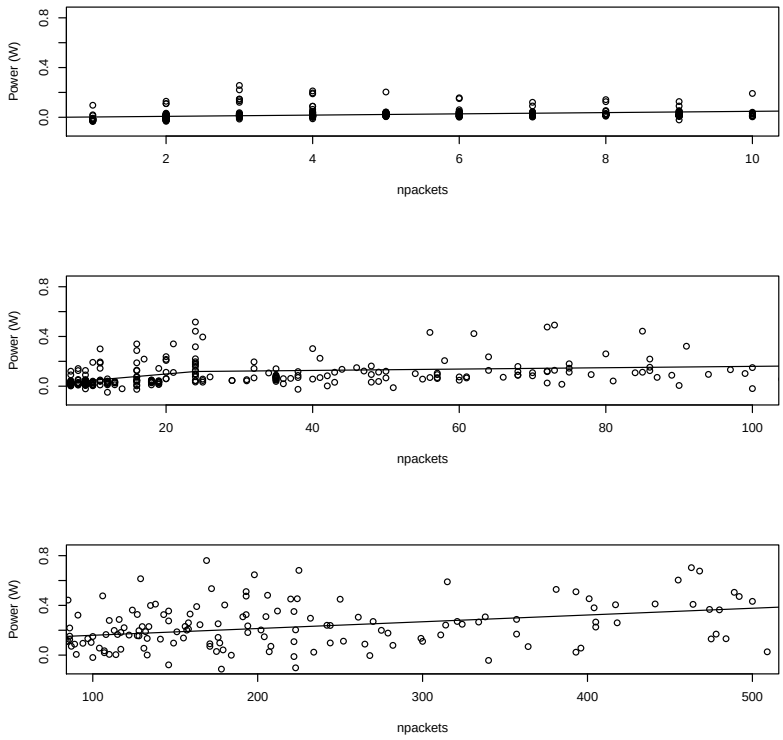


Figure 8.5: The WiFi receive experiment. The variance is considerably high.

Table 8.7: The inferred model for the receive experiment. The variable T_{wlan} indicates the number of packets transferred.

#1	
$1 \leq T_{wlan} < 55$	$3.3T_{wlan} + 13$
$55 \leq T_{wlan} < 400$	$0.3T_{wlan} + 154$
$T_{wlan} \geq 400$	200
#2	
$1 \leq T_{wlan} < 85$	$2.3T_{wlan} + 20$
$85 \leq T_{wlan} < 400$	$0.1T_{wlan} + 190$
$T_{wlan} \geq 400$	200
#3	
$1 \leq T_{wlan} < 23$	$5.5T_{wlan} + 2.1$
$23 \leq T_{wlan} < 400$	$0.54T_{wlan} + 116$
$T_{wlan} \geq 400$	200

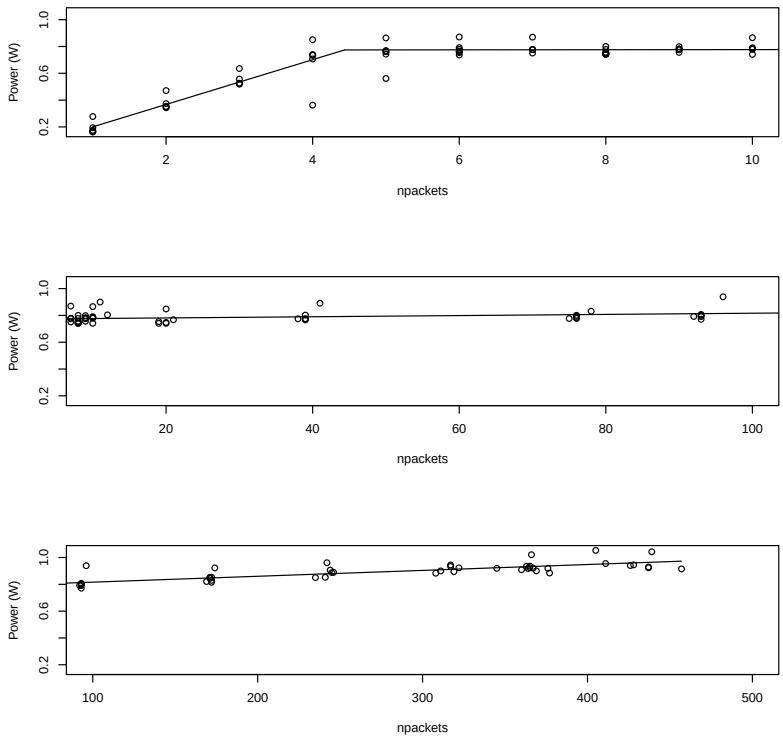


Figure 8.6: The WiFi send experiment. The upper figure shows the power dissipation from 0 to 10 pkt/s. The middle figure displays the power dissipation when the transfer rate is from 10 to 100 pkt/s. The bottom figure displays the power dissipation from 100 pkt/s to 450 pkt/s.

8.1.7 Cellular 3G

In the cellular 3G experiment, it was needed to deduce the radio resource control (RRC) parameters: promotion strategy, demotion strategy, state transition timeouts, upload and download queue sizes, and the power dissipation in the relevant operating states. See Section 2.4.2 for an explanation of the 3G operating states and radio resource control. For inferring the parameters, the method described in [50] was followed. In this description, denote the Nokia N900 as the User Equipment (UE).

Promotion strategy To infer the promotion strategy, use substantial delay in round-trip times of a packet if a state transition occurs. In addition, assume that when the system has been idle for 30 seconds, it drops to the IDLE state. The key idea is first to send a small packet and immediately after that a large packet and measure their round-trip times. The large packet forces the system to move into the DCH state, while the small packet may transfer it either to the FACH or DCH state. If the promotion strategy is two-step, then there is a difference between the round-trip times of the first and second packet, otherwise not. The length of the packet has a negligible effect on the overall round-trip time and can be ignored in this experiment.

A 48-byte packet was sent to a server that echoed it back, and the round-trip time was recorded (RTT1). Immediately after this, a 1024-byte packet was sent to a server which echoed it back. Its round-trip time was recorded (RTT2). The experiment was repeated three times, and between each iteration, the system was kept idle for 30 seconds in order to force the 3G to IDLE state.

	#1	#2	#3
RTT1	280 ms	684 ms	684 ms
RTT2	917 ms	1963 ms	900 ms

Since the RTT2 is substantially higher than the RTT1, the promotion occurs by first switching the system to FACH state and after that to DCH state when there is enough traffic.

Demotion strategy The demotion was inferred by first sending a 1024-byte packet to a server which echoed a 48-byte packet back. This forces the UE to enter the DCH state. After this, the system was kept idle for n seconds, where n varied between 1 and 30 seconds in 1-second increments. Then, a 48-byte packet was sent, which the server echoed back. Depending on the demotion strategy, the UE moves either into the DCH or FACH state. The round-trip time of the latter packet was recorded and plotted as a function of the idle time in Figure 8.7.

Next, a 1024-byte packet was sent to a server, which echoed a 48-byte packet back. The UE is now in the DCH state. Once again, the system was kept idle for n seconds, where n varied between 1 and 30 seconds in 1 second increments. Then, a 1024-byte packet was sent to the server, which echoed a 48-byte packet back. Now, the UE is in the DCH state.

The round-trip times of the latter packet were recorded and the results are depicted in Figure 8.8 as the function of the idle time.

To infer the demotions strategy, one must consider whether the strategy is DCH → IDLE or DCH → FACH → IDLE. If the demotion strategy is DCH → IDLE, one should see only one level change in Figures 8.7 and 8.8. If the demotion goes first to FACH and after that to IDLE, there should be two levels present in the figures.

From the Figure 8.9, it is clearly visible that the demotion strategy is DCH→ FACH → IDLE. In addition, the difference between round-trip times between 5 s and 13 s allows deducing the FACH → DCH promotion delay.

Timeouts The transition timeout for demotion DCH → FACH and FACH → IDLE can be seen from both Figures 8.7 and 8.8. Clearly, in the period 1–4 s, UE is in the DCH state, it drops to FACH state when the delays are in 5–14 s and to IDLE when the delay is 15 seconds or more. Thus, the timeout for DCH → FACH is 5 seconds, and FACH → IDLE is 10 seconds.

Uplink and downlink queue sizes To infer the queue sizes, a binary search was used with the FACH → DCH promotion delay as the indicator. The uplink queue size was 480 bytes and the downlink queue size was 400 bytes.

Power consumption To DCH and FACH state power dissipation was observed from the DAQ measurements. The DCH power dissipation was found to be 880 mW and FACH to be 380 mW.

A summary of the RRC parameters for Elisa's network (a Finnish mobile carrier):

Network	Elisa
promotion strategy	IDLE → CELL_FACH → CELL_DCH
demotion strategy	CELL_DCH → CELL_FACH → IDLE
α: CELL_DCH → CELL_FACH	5 s
β: CELL_FACH → IDLE	10 s
UL queue	480 B
DL queue	400 B
CELL_FACH	380 mW
CELL_DCH	880 mW

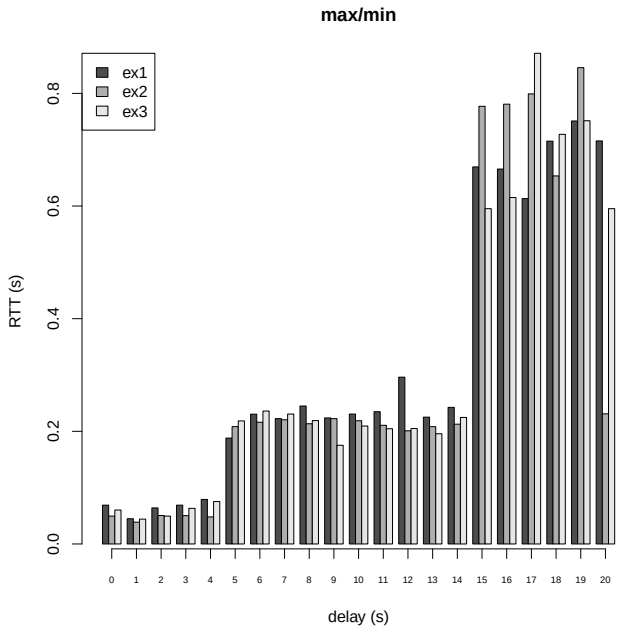


Figure 8.7: The 3G demotion strategy inferring. The x-axis indicates the number of seconds slept between sending a 48-byte and a 1024-byte packet. The y-axis measures the round-trip time of the 1024-byte packet.

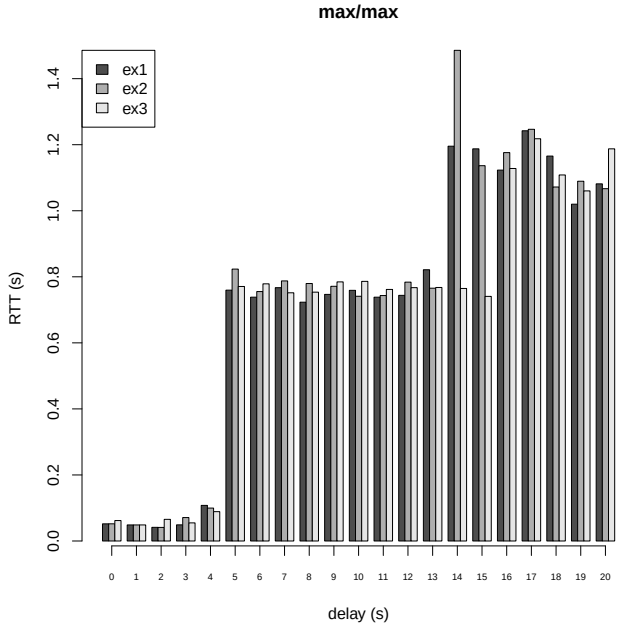


Figure 8.8: The 3G demotion strategy inferring. The x-axis indicates the number of seconds slept between sending the first 1024-byte and the second 1024-byte packet. The y-axis measures the round-trip time of the latter 1024-byte packet.

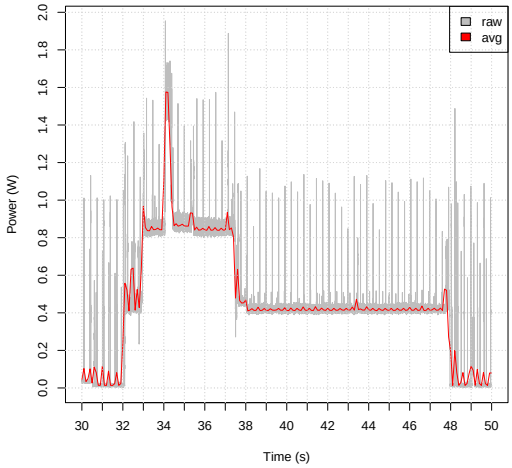


Figure 8.9: The measured power dissipation during the promotion strategy experiment. From the figure, the promotion strategy IDLE→FACH→DCH is clearly visible at 32–33 s. The system is in the high-power DCH state between 33 and 37 seconds, and drops to FACH state where it stays for 10 seconds between 38 s and 48 s.

8.1.8 Solid State Disk

In the solid state disk experiment, the focus was only on the eMMC solid state disk that is used for N900 applications (see Section 6.9). The number of sectors read and written as reported by the kernel was measured. The experiment consists of two parts: read and write.

In the read experiment, a 35 MB file was sequentially read varying the frequency of the read system calls. All reads were done in 4096-byte blocks, and the caches are dropped between each read operation. The microbenchmark issued 1, 10, 20, 30, 40, 50, 100, 500, 800, 1000, and 2000 read operations per second.

The write benchmark consists of writing null data into an initally empty file in 4096-byte blocks. The write operations were varied in in 10, 100, 200, 500, 800, 1000, 3000, 4000, and 5000 operations per second.

The limits for the maximum number of read and write operations were chosen by examining the maximum number of operations achievable when the caches were flushed between each read operation and the data was synced between each write operation. This was done so that it was possible to observe the actual number of maximum disk operations that could be performed without kernel cache interference.

The system was kept idle for 5 seconds before and after each experiment. The system was active for 15 seconds when the disk operations were performed. Between each level, there was a 10-second cool-down period. The linear regression inference results are in Table 8.8 for the read experiment and in Table 8.9 for the write experiment. The standard regression was used for the read experiment and the segmented regression for the write.

In the read experiment model, assume that the negative coefficient is a measurement error and the proper model is a constant, thus yielding the read consumption to 111 mW. For the write experiment, the maximum power dissipation was around 600 mW which was reached when there was more than 2100 sectors written per second.

Table 8.8: The results for the read experiment. The variable R_{SSD} is the number of sectors read per second.

#1	#2	#3
$-0.004R_{SSD} + 110$	$-0.003R_{SSD} + 110$	$0.001R_{SSD} + 113$

Table 8.9: The results for the write experiment. The variable W_{SSD} is the number of sectors written per second.

#1	
$1 \leq W_{SSD} < 2069$	$0.2W_{SSD} + 144$
$2069 \leq W_{SSD} < 2100$	$-0.06W_{SSD} + 593$
$W_{SSD} > 2100$	600
#2	
$1 \leq W_{SSD} < 2069$	$0.2W_{SSD} + 120$
$2069 \leq W_{SSD} < 2100$	$0.004W_{SSD} + 430$
$W_{SSD} > 2100$	600
#3	
$1 \leq W_{SSD} < 2069$	$0.2W_{SSD} + 150$
$2069 \leq W_{SSD} < 2100$	$0.01W_{SSD} + 430$
$W_{SSD} > 2100$	600

8.2 The Model

To construct the model based on experiments presented in Section 8.1, take the average of three measurements performed for each individual subsystem or device and use it in the next section to validate the model and measure its accuracy. The GPU experiment was not repeated, thus use the average over all SDK entries (see Section 8.1.4) as the model. For most of the subsystems, all repeated experiments gave similar results but there was a rather large discrepancy when performing the *receive* experiment in the WiFi (Section 8.1.6) and the write experiment in the SSD (Section 8.1.8). The exact reasons for these large discrepancies remain unclear but may be caused some other subsystems remaining active during the measurements. There, we choose one of the inferred models arbitrarily for WiFi and SSD.

For the WiFi model, it was chosen to use the model inferred in the *reply* experiment (Table 8.6) if there were packets being transmitted and received. For the SSD, if there were only reads, use the model inferred in the *read* experiment (Table 8.8), otherwise use the model for *write* (Table 8.9). For other systems, there is no need to consider the access type of the subsystem when defining the power model.

The model is given in Table 8.10. The table describes the power dissipation for each subsystem that is modeled in this work and gives the maximum power the subsystem can draw. Naturally, not all of the subsystems can be active at the same time due to the current limitation features of the phone.

From the figure it is obvious that the most power-hungry subsystems are the cellular modems, display, and the CPU when operating in the high performance states. On the other hand, the lower performance states reduce their power dissipation significantly.

Further discussion and analysis of the model is delayed to the next chapter.

Device	State	Predicted Power Consumption	Parameter Range	Max Power
IDLE	normal	39.3 mW		39.3 mW
	flight	18.9 mW		18.9 mW
LOG	none	5.6 mW		5.6 mW
	estimate	20.2 mW		20.2 mW
CPU	$f = 250$ MHz	4.1 $\mathrm{CPU_{load}}$ mW	$\mathrm{CPU_{load}} \in [0, 100]$	410 mW
	$f = 500$ MHz	7.3 $\cdot\mathrm{CPU_{load}}$ mW		730 mW
	$f = 550$ MHz	8.4 $\cdot\mathrm{CPU_{load}}$ mW		840 mW
	$f = 600$ MHz	9.8 $\cdot\mathrm{CPU_{load}}$ mW		980 mW
GPU	GPU ON	448 mW		448 mW
	tail-1 (3s)	448 mW		448 mW
	tail-2 (2s)	266 mW		266 mW
	tail-3 (5s)	81 mW		81 mW
DISPLAY		1.89 $\mathrm{BL_{br}}$ + 363 $\mathrm{BL_{on}}$ mW	$\mathrm{BL_{br}} \in [0, 255]$ $\mathrm{BL_{on}} \in [0, 1]$	845 mW
WIFI	$1 \leq \mathrm{T_{wlan}} < 9$	$85 \cdot \mathrm{T_{wlan}} + 6$ mW	$\mathrm{T_{wlan}} > 0$	1000 mW
	$9 \leq \mathrm{T_{wlan}} < 1200$	$0.2 \cdot \mathrm{T_{wlan}} + 764$ mW		
	$n \geq 1200$	1000 mW		
	$1 \geq \mathrm{T_{wlan}} < 85$	$2.3\mathrm{T_{wlan}} + 20$ mW	$\mathrm{T_{wlan}} = 0$	200 mW
	$85 \leq \mathrm{T_{wlan}} < 400$	$0.1\mathrm{T_{wlan}} + 190$ mW		
	$\mathrm{T_{wlan}} \geq 400$	200 mW		
CELLULAR 3G	FACH	380 mW		380 mW
	DCH	880 mW		880 mW
SSD	$1 \leq \mathrm{R_{SSD}} < 2000$	110 mW	$\mathrm{R_{SSD}} > 0$, $\mathrm{W_{SSD}} = 0$	110 mW
	$1 \leq \mathrm{W_{SSD}} < 2069$	$0.2 \cdot \mathrm{W_{SSD}} + 144$ mW	$\mathrm{W_{SSD}} > 0$	600 mW
	$2069 \leq \mathrm{W_{SSD}} < 2100$	$0.43 \cdot \mathrm{W_{SSD}} + 164$ mW		
	$\mathrm{W_{SSD}} \geq 2100$	600 mW		

Table 8.10: The summary of the Nokia N900 power model.

8.3 Verification with Microbenchmarks

In the verification phase, the model's accuracy is verified by using the calibrated model to predict the total power dissipation over the time axis and correlate this with the measured power dissipation provided by the measurement setup.

8.3.1 CPU

In the CPU experiment, three microbenchmarks were run that fully utilized the CPU: integer, float, and memory. In the integer experiment, a C program was run that performed a few arithmetic operations in a busy loop. The float program was analogous to the floating point operations. The memory operation copied 1 MB of memory in a busy loop from a memory buffer into another. All experiments were run for 60 seconds with 5 second warm-up and cool-down periods before and after.

The cross-correlation results are presented below. The experiment was repeated three times.

	#1	#2	#3
integer	0.98	0.95	0.95
float	0.97	0.99	0.99
memory	0.98	0.98	0.96

8.3.2 Network

In the network experiment, a 1 MB file was downloaded from a web server with the `wget` command. The file was written to */dev/null*. The experiment was run on both WiFi and 3G radios. The results for the cross-correlation are shown below. The experiment was repeated three times.

	#1	#2	#3
WiFi	0.86	0.94	0.93
3G	0.88	0.94	0.90

8.3.3 SSD

In the SSD experiment, a 1 MB file was downloaded from a web server with the `wget` command. The file was downloaded with a WiFi radio and was written to the eMMC SSD disk. The results for the cross-correlation are presented below. The experiment was repeated three times.

	#1	#2	#3
SSD	0.88	0.95	0.88

8.4 Verification with Applications

This section demonstrates the subsystem breakdowns.

8.4.1 Browser and Radios

In this experiment, the Wikipedia article `http://en.wikipedia.org/wiki/Caesar_cipher` was downloaded with the Nokia N900 native browser by using 3G and WiFi radios. The measured power dissipation can be seen in Figures 8.10 and 8.11. The variables used for the subsystem power dissipation breakdown are described in Section 7.2. The correlation of the measured power dissipation and the consumption estimated by the model is presented in Table 8.11.

Table 8.11: The cross-correlation of the measured and estimated power dissipation of the browser and radios experiment.

mode	#1	#2	#3
3G	0.73	0.72	0.73
WiFi	0.72	0.73	0.72

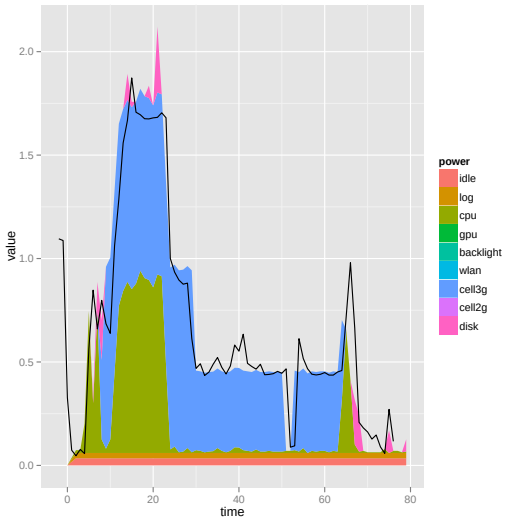


Figure 8.10: The cellular 3G subsystemwise power breakdown of the browser and radios experiment. The black line indicates the measured power dissipation.

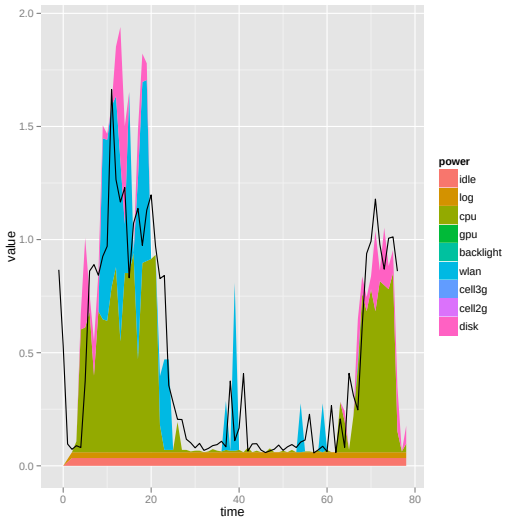


Figure 8.11: The subsystemwise power breakdown of the browser and radios experiment using WiFi. The black line indicates the measured power dissipation.

8.4.2 Angry Birds

In the Angry Birds experiment, the game was played for 60 seconds and the system variables were observed and fed to the linear model. The correlation between the measured and the model's predicted power dissipation was found to be 0.75. The results can be seen in Figures 8.12 and 8.13.

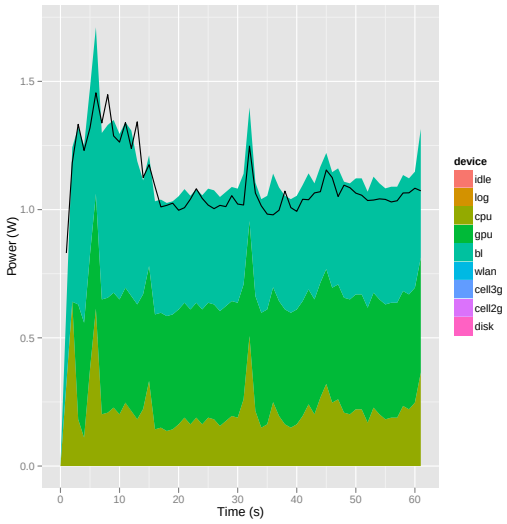


Figure 8.12: The overall power dissipation of playing 60 seconds of the Angry Birds game. Stacked plot indicates the per-device estimated power dissipation and the black line the measured power consumption.

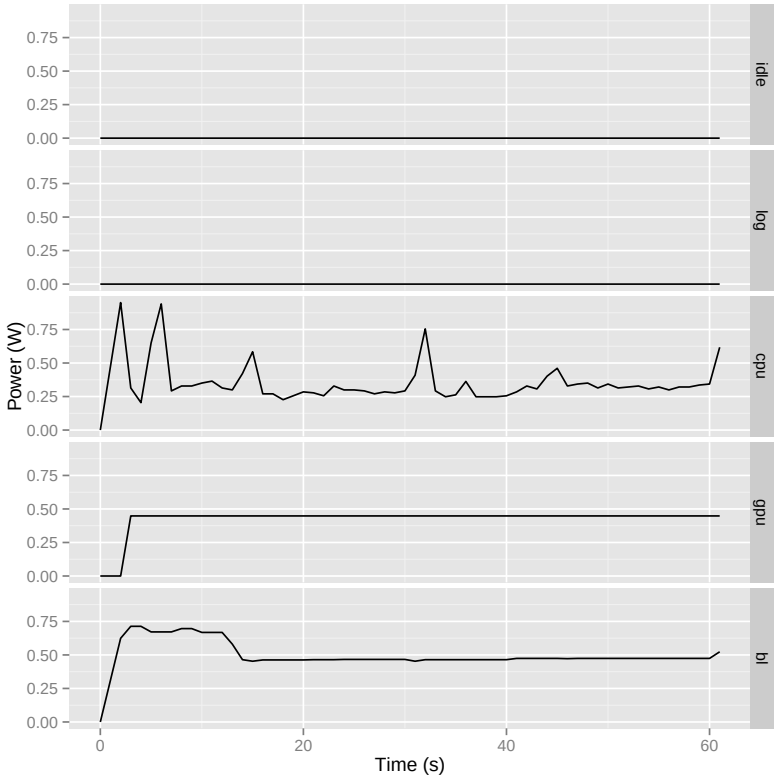


Figure 8.13: The per-subsystem predicted plots for playing the Angry Birds game.

8.4.3 Summary

Below is the summary of the verification experiments, which devices they utilize, and what is the correlation between the measured and the predicted total power dissipation:

Experiment	Devices	Correlation
Browser and radios	CPU, 3G	0.73
	CPU, WiFi	0.72
Angry Birds	CPU, GPU, DISPLAY	0.75

Chapter 9

Discussion and Conclusion

This chapter summarizes the work and offers a critical review of it. It gives detailed discussion on power modeling in general and the subsystem modeling in particular. First, the measuring and modeling aspects are discussed, then each modeled subsystem is reviewed and discussed.

9.1 Measuring

Measuring involves both observing the power dissipated by the device and the utilization of the subsystems. The overall goal is to explain the connection between the operating state of the subsystems and their power dissipation.

All measurement systems have a limited accuracy. The DAQ electric power measurement system's accuracy is around 0.2 μW (Section 7.1) which is more than suitable for this work, since even the pure idle consumption with no logging is around 35 mW (Section 8.1.1).

The sampling rate is another important aspect when performing the measurements. The sampling frequency was chosen to be 1000 Hz and the subsystem utilization logging to be 1 Hz. These values allow observing power changes that last longer than 2 ms and subsystem state changes that last more than 2 seconds. It can be argued that having a higher sampling frequency would allow us observing phenomena whose duration is considerably shorter: at least in radios and CPUs, state changes occurring in the microsecond range are common. But having a higher sampling rate causes overhead, especially for the subsystem logging. At a 1 Hz rate, it has a 24 mW overhead: a 60% increase to the idle consumption (see Section 8.1.2). Having a higher sampling rate would increase the logging overhead even more. On the other hand, a lower sampling rate would decrease the overhead but might hide some interesting events.

The overall goal of this work is to provide overall energy consumption estimates. Overall energy consumption does not care much abourt short duration power dissipation pulses, thus the 1 Hz subsystem logging sampling rate seems reasonable since it misses only events that are short duration. The power measurement sampling rate of 1000 Hz, however, is somewhat arbitrary and we

could use higher frequencies. For example, in [61] the authors use 5000 Hz sampling but give no reasoning behind it.

Another alternative to the sampling-based subsystem logging is to drop it altogether. Pathak [47] considers a system call tracing based approach, where all power state changing system calls are traced, and whenever a system call that has an effect on the power dissipation is used, the event is logged. This eliminates the need for constant polling and virtually all overhead caused by it.

Synchronization (Section 7.4) is required so that the power measurements and subsystem utilization logs can be merged together. In this work, it was chosen to use `ntpdate` on both the power measurement computer and the Nokia N900 device. This does not achieve a high synchronization accuracy but since we are operating with rather low sampling rates, namely 1 Hz, this is sufficient.

The DAQ system supports signal-triggered measurements which could be an alternative form of synchronization, if it were possible to connect a general purpose I/O pin to the DAQ and use some software as a trigger for the DAQ to start and stop measuring. This, however, was not done due to the lack of documentation for the Nokia N900 debug pads (see the middle picture in Figure 7.3).

The second alternative would be to use direct power measurements. That is, if the hardware supported *probe points* between subsystems (e.g. a WiFi chip), we could insert the power measurement setup there and directly observe the dissipated power. This was the approach taken by Carroll in [19] (see Section 3.2). Unfortunately, there is no information available on whether the Nokia N900 device has these kind of probe points between the subsystems.

The third alternative would be to utilize some device in a way that it generates a *synchronization pulse* in the power measurement signal and use this synchronization point. This is the approach taken by Rice [53]. This would drop the requirement for doing NTP synchronization before the experiment starts. But given the rather low frequency subsystem logging sampling rate of 1 Hz, this was deemed unnecessary.

In general, understanding the power dissipated by a subsystem involves more than just measuring the current drawn by the subsystem and correlating it with the load of the subsystem: a subsystem usually consists of several components that have different power usage characteristics that depend on the operating state and the load of the system. While direct measurements would allow observing the current drawn by the subsystem, it would not explain the operating state and the load-level of the system. Thus, the load and performance state based logging approach seems reasonable.

9.2 Modeling

The goal was to produce a model that gives the subsystem power dissipation breakdown given the total power measurements from the battery. Author had a hypothesis that the relation between the subsystem utilization and the power dissipation is linear. The experiments seem to support this claim, and the approach is widely used in the literature [24, 25, 33, 53].

Virtually all subsystems can be described having a set of operating and idle states. The idle states involve no operating activity (Section 2.3). To minimize the subsystem idle power dissipation, the clocks or the whole subsystem power off can be switched off. This will yield less or no static power dissipation but causes increased latencies when waking up. It seems reasonable to assume that all of these sleep states have a constant power dissipation when a subsystem is operating on the state.

When a subsystem is active, it operates on one of several active power states. Each active power state or operating performance point (OPP) involves a trade off between the power dissipation and performance: the more performance is required, the more power must be fed to the subsystem. In active states, the subsystems' electrical components exhibit switching activity that causes dynamic power dissipation (Section 2.3.2). The transition between these OPPs are inherently nonlinear but their duration is short, thus their overall power dissipation is negligible. While a subsystem is in an OPP, its power dissipation behavior is essentially linear w.r.t. the subsystem's load.

For some devices, there is a saturation point after which the power dissipation does not increase even though the load increases. For example, the WiFi in the experiments behaves this way. For those, it was chosen to use the segmented regression (Section 4.4) modeling to infer a segmented linear model explaining their load and power correlation.

9.3 Calibration

To discuss the calibration phase of the model construction, there are several things to consider. For each subsystem that was chosen to be modeled, it should be considered, whether the correct state variables that accurately describe the power dissipation behavior of the subsystem were chosen. It should also considered, whether the sampling frequencies are sufficiently high to capture the behavior of the device, and finally, is the linear model the correct way to model the behavior of the device.

9.3.1 Idle

In the idle experiment, the system is in an idle mode. Thus, there should be no switching activity. The study did not include the full-system sleep states such as the hibernate mode. These were not modeled because with the default configuration the Nokia N900 does not enter these deeper sleep states.

However, the OMAP in general and the CPU in particular do enter the deep sleep states (Section 6.2), thus studying these states would yield a more accurate power model of general use.

Another aspect of the model is to subtract the idle consumption from the other measured consumption. This is based on the assumption that the idle consumption is always present, and it would be included twice if it was not subtracted from the measurement before performing the regression. This assumption may not be entirely valid, which might partly explain the low correlation results in

the verification experiments (Section 8.3).

9.3.2 Log Overhead

The state variable sampling causes a 60% overhead when compared to the idle consumption (Section 8.1.2) which is a rather large overhead if this logging software was to be constantly run, for example, by an end user. Since this was not the case, and the goal was to do logging and measurements in a laboratory environment only, this overhead was deemed acceptable.

Further development and power savings could be achieved by optimizing the logging software so that it does no unnecessary polling or changing it to an event-based approach such as [47].

9.3.3 CPU

The CPU model considers only the CPU frequency and the load as the input variables for the model. The type of instructions executed (e.g., logic, control, memory) altogether were omitted. Isci [30] took the approach of analyzing the instruction type power dissipation. These kind of inside-CPU models would yield a better accuracy but would incur larger overheads because it would require sampling a larger set of utilization counters.

This work does not model the actual behavior of the idle and the active power management in the Nokia N900 Maemo kernel (Section 6.2) because this would require accessing the in-kernel process ready queue in a similar fashion that the Linux kernel's idle and active power management governors do. Since the sampling approach used here is userland only, this was deemed unfeasible.

The CPU model differs from the model presented by [61] (Section 3.2). They have a constant factor present while the one presentd does not. The lack of the constant coefficient in this work's model may be caused by Nokia N900's OMAP SoC having less idle power dissipation than the Qualcomm SoC used in their experiments. To verify this claim, more experiments should be performed.

9.3.4 GPU

The GPU is a device with sophisticated power saving features including clock gating, power gating, and some form of DVFS (Section 6.4) based on the load of the GPU. Unfortunately, there is no load counter exported by the GPU driver, thus, forming a similar load-based model between the GPU and the power dissipation proved to be difficult.

The experiment (Section 8.1.4) demonstrates that the GPU power dissipation is somewhat constant once the CPU consumption is removed even though the programs utilizing the GPU vary from simple triangle drawing to complex shader programs.

Thus, it was decided to model the GPU as a device that has a constant power dissipation depending only on whether the GPU subsystem is on or off. When performing the experiments, it was noticed that there are few tail states visible

in the power measurements. Just by observing these tail states from the graph, we deduced their duration.

The GPU driver in the Maemo kernel has the functionality to observe the frequency scaling, which would suggest that a more accurate model could be formed if this GPU clock was exported to userland so that the logging tool could observe it.

9.3.5 Display

The display experiment clearly demonstrates a linear behavior w.r.t the brightness level (Section 8.1.5). Carroll's experiment [19] (see Section 3.2) exhibited a slight non-linear behavior while, Zhang's [61] has a similar linear model (see Section 3.2) to the one presented here except they lack the constant coefficient. This is probably due to the fact that their display device is different from the one used in Nokia N900.

9.3.6 WiFi

During the WiFi experiment, the maximum power dissipation was 880 mW when the experiment was transmitting data. Receiving with no transmission yielded only a 50 mW power dissipation. According to the FCC (Section 6.7) the maximum radiated power is 239 mW. Based on the results in Section 8.1.6, this will yield ca. 70% overhead for the chip operation.

The experiments consider only rather low transfer rates. Further study would be needed to infer the model for all data rates. In addition, the effect of the WiFi channel was ignored.

Another factor to consider is the CAM and PSM modes (Section 2.4.1) and the QoS features present in the WiFi chip.

In the receive-only experiment (Figure 8.5), the power usage remained rather low which is probably due to the WiFi base station buffering the received packets allowing the WiFi to go to sleep between the packets.

In the both send-only (Figure 8.6) experiments, the power usage first increased rapidly until it reached a saturation point around the 7 packets/s rate. It seems that both receiving and receive-with-reply cause similar power usage.

From the figures and analysis, it seems that the packet size does not affect the power usage. Thus, we consider only the packet transmission frequency and whether the packets are received only or transmitted in the power model.

9.3.7 Cellular 3G

Cellular 3G has a rather consistent power behavior. It is a simple state machine, as explained in Section 2.4.2. The long tail states are clearly visible. Inferring the state machine changes by observing the packet latencies is cumbersome at best. A better approach would be to instrument the kernel to report the power state which it is in, and report it to the subsystem logging tool.

The 3G mobile device transmit power is controlled by the base station and is varied by factors such as signal strength. The experiments were done in a stationary location, thus, we expect the model to be imprecise if the device is moved to some other location. From the experiments (Section 8.1.7), the 3G operating states are clearly visible and have a constant power dissipation.

9.3.8 Solid State Disk

The experiment focused only on the EMMC disk (Section 6.9). There was no product bulletin or specifications available for the EMMC disk to verify the results. Carroll [19] experimented on the EMMC SSD in their experiments and noticed that a considerable amount of energy is consumed by the RAM and the CPU subsystems. Since the model presented here considers only the utilization of the SSD subsystem, it may be possible to improve the accuracy substantially if RAM and CPU were considered during the calibration.

9.4 Verification

Currently, the OMAP interconnect (Section 6.3) or the memory subsystem (Section 6.5) are not modelled. According to [19], they have a considerable impact on the power dissipation in a wide variety of usage scenarios. Thus, it can be argued that most of the model inaccuracies visible in the verification experiments (Section 8.3) may be caused by this.

Another aspect that causes inaccuracy is the 1 Hz sampling rate which masks very short duration events that can have substantial power draw. This could be improved by increasing the sampling rate of the logging system but, as discussed before, this would cause substantial overhead.

The goal was not to provide accurate power breakdowns but instead provide a method that would allow us to explain which subsystems are utilized by an application and how much power they consume. Thus, even though ca. 70% coefficient of determination was achieved (Section 8.4.3) when performing power modeling, it can be argued that the total energy consumption modeling is accurate enough to provide the user an understanding of which components are mostly responsible for the power dissipation. That is, the "guilt" of energy consumption can be assigned to the correct subsystems with sufficient accuracy.

9.5 Review of Research Questions

The first research question Q1 asked how to measure the power dissipation. The result is that the custom measurement setup with a fake battery was required. In the future, it would be beneficial if the device was equipped with a current or a power measurement hardware allowing direct measurements of the power dissipated by the subsystems. Ideally, thus the current/power measurement hardware could be deployed on both the subsystems inside and outside the SoC.

The research question Q2 asked how to measure the subsystemwise utilization. The sampling approach incurred rather large overhead. An event-based or a kernel-level performance measurement framework would be more applicable and would incur less overhead. If the system had support for both a hardware power measurement and a kernel level power awareness, the operating system could do a better resource allocation and confine the system to given energy budgets or try to maximize the battery life.

The research question Q3 asked how to do the subsystemwise breakdown of the power measurements given the total power consumption and the subsystemwise utilization. As long as the hardware supporting direct subsystemwise power measurements is missing, the linear regression seems to be a viable method. Training the model can be done in a laboratory environment, and the results for power breakdowns can be used on a unmodified mobile hardware. These kind of products already exist in the market. (See, for example, Android PowerTutor [61].)

The research question Q4 asked how to verify the accuracy of the subsystem power breakdown. Yet again, as long as there is no direct power measurement hardware between the subsystems available, statistical methods must be relied on. That is, train the model with sufficient data and perform enough control experiments so that the model's accuracy can be verified.

9.6 Conclusion and Future Work

The goal was to study the methods with which to do power measurements and provide tools which allow a subsystemwise breakdown of the power usage of a mobile device. Based on the results, it seems feasible that the subsystemwise power dissipation can be deduced from the aggregate.

The problem of understanding the subsystem power dissipation involves not only directly observing the current drawn by the circuitry forming the subsystem but also observing the operating state and the load of the subsystem.

This thesis provides a further basis for the observation that measuring the total power dissipation and using a linear model calibrated with microbenchmarks can be used to provide subsystemwise power dissipation breakdowns. For the validation of the linear model, a 70% cross-correlation was obtained.

CPU, GPU, display, WiFi, 3G, and SSD subsystems were observed. The measurements indicate that the power dissipation of the individual subsystems vary greatly based on the type of the system load, e.g., whether the application uses the network or a graphics card.

In conclusion, it seems reasonable that the system can be used with an application, observe the utilization or load of the individual subsystems, and use a linear model to attribute the total energy consumed by the application.

For future work, there are many aspects to consider. The model considers only a part of the subsystems present in the Nokia N900 phone. Modeling the rest of them to understand the overall power dissipation of the device in various usage scenarios would be reasonable.

More complex models for the subsystems could also be considered. Namely, it should considered whether each subsystem should be modeled with more input variable to provide better accuracy for the model.

Other hardware and software modeling could be considered. This could be done by introducing new, currently used mobile phone environments like Android or iOS. In addition, the models could be further developed for tablets and laptop computers as well.

To minimize the overhead caused by the logging system, an event-based logging system could be developed. An interesting further study subject would be to see if this could be done automatically by some form of machine learning.

Bibliography

[1] FCC PART 15 Compliance Test Report (WLAN: 2412.0–2462.0 MHz). Nokia Corporation. FCC ID: LJPRX-51, IC: 661E-RX51.

[2] FCC PART 22/24/27 Compliance Test Report: Part 1 (Cellular). Nokia Corporation. FCC ID: LJPRX-51, IC: 661E-RX51.

[3] Nokia Corporation N900 Technical Specifications. `http://maemo.nokia.com/n900/specifications/`.

[4] PowerVR Insider SDK. `http://www.imgtec.com/powervr/insider/sdkdownloads/index.asp`.

[5] Make Accurate Power Measurements with NI Tools. National Instruments, 2008.

[6] OMAP TM 3 architecture from Texas Instruments opens new horizons for mobile Internet devices, 2008.

[7] ACX565AKM Data Sheet. Sony, 2009.

[8] NI USB-621x Specifications. National Instruments, 2009.

[9] NI USB-621x User Manual. National Instruments, 2009.

[10] OMAP3 Product Bulletin. Texas Instruments, 2009.

[11] OMAP3430 Technical Reference Manual. Texas Instruments, 2010.

[12] Advanced Configuration and Power Interface Specification Revision 5.0, 2011.

[13] G. Anastasi, M. Conti, E. Gregori, and A. Passarella. 802.11 power-saving mode for mobile computing in Wi-Fi hotspots: limitations, enhancements and open issues. *Wireless Networks*, 14:745–768, 2008.

[14] R. Balani. Energy Consumption Analysis for Bluetooth, WiFi and Cellular Networks. Technical report, University of California at Los Angeles, 2007.

[15] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, IMC '09, pp. 280–293. ACM, New York, NY, USA, 2009.

[16] K. Banerjee and E. Agu. PowerSpy: fine-grained software energy profiling for mobile devices. In *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, volume 2, pp. 1136 – 1141. 2005.

[17] F. Bellosa. The benefits of event driven energy accounting in power-sensitive systems. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pp. 37–42. ACM, New York, NY, USA, 2000.

[18] L. Benini and G. D. Micheli. System-Level Power Optimization: Techniques and Tools. *ACM Transactions on Design Automation of Electronic Systems*, 5(2):115–192, 2000.

[19] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pp. 21–21. USENIX Association, Berkeley, CA, USA, 2010.

[20] A. Chandrakasan and R. Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceedings of the IEEE*, 83(4):498–523, 1995.

[21] J. Chen, B. Li, Y. Zhang, L. Peng, and J.-K. Peir. Tree structured analysis on GPU power study. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*. 2011.

[22] J.-C. Chen, K. Sivalingam, P. Agrawal, and S. Kishore. A comparison of MAC protocols for wireless local networks based on battery power consumption. In *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pp. 150–157. 1998.

[23] S. Collange, D. Defour, and A. Tisserand. Power Consumption of GPUs from a Software Perspective. In *Proceedings of the 9th International Conference on Computational Science: Part I*, ICCS '09. Springer-Verlag, Berlin, Heidelberg, 2009.

[24] G. Contreras and M. Martonosi. Power prediction for Intel XScale processors using performance monitoring unit events. In *Proceedings of the 2005 international symposium on Low power electronics and design*, ISLPED '05, pp. 221–226. ACM, New York, NY, USA, 2005.

[25] D. Economou, S. Rivoire, and C. Kozyrakis. Full-system power analysis and modeling for server environments. In *Workshop on Modeling Benchmarking and Simulation (MOBS)*. 2006.

[26] J. J. Faraway. Practical Regression and Anova using R, 2002.

[27] R. Friedman, A. Kogan, and Y. Krivolapov. On power and throughput tradeoffs of WiFi and Bluetooth in smartphones. In *INFOCOM, 2011 Proceedings IEEE*, pp. 900–908. 2011.

[28] S. Hong and H. Kim. An integrated GPU power and performance model. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10. ACM, 2010.

[29] IEEE. Standard 802.11-2007 Part 11: Wireless LAN MAC and PHY Specifications.

[30] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pp. 93–. IEEE Computer Society, Washington, DC, USA, 2003.

[31] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, pp. 135–140. ACM, New York, NY, USA, 2001.

[32] J. S. Kilby. Miniaturized Electronic Circuits, US Patent 3,138,743, 1964.

[33] Y. Kim, Y. Cho, S. Park, and N. Chang. System-level power estimation using an on-chip bus performance monitoring unit. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '08, pp. 149–154. IEEE Press, Piscataway, NJ, USA, 2008.

[34] J. Kurtto. *Mapping and improving energy efficiency of the Nokia N900*. Master's thesis, University of Helsinki, Department of Computer Science, 2011.

[35] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, pp. 160–171. ACM, New York, NY, USA, 2003.

[36] M. Y. Lim, A. Porterfield, and R. Fowler. SoftPower: fine-grain power estimations using performance counters. In *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 308–311. ACM, New York, NY, USA, 2010.

[37] D. Macii and D. Petri. An Effective Power Consumption Measurement Procedure for Bluetooth Wireless Modules. *Instrumentation and Measurement, IEEE Transactions on*, 56(4):1355–1364, 2007.

[38] A. Mahesri and V. Vardhan. Power Consumption Breakdown on a Modern Laptop. In B. Falsafi and T. N. Vijaykumar, editors, *Power-Aware Computer Systems*, volume 3471 of *Lecture Notes in Computer Science*, pp. 165–180. Springer, 2004.

[39] M. Martins and R. Fonseca. The Case for Device Power States. Technical report, Brown University, 2011.

[40] P. Mochel. Linux Kernel Power Management. In *Proceedings of Ottawa Linux Symposium*. 2003.

[41] B. Mochocki, K. Lahiri, and S. Cadambi. Power analysis of mobile 3D graphics. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, DATE '06, pp. 502–507. European Design and Automation Association, 3001 Leuven, Belgium, 2006.

[42] V. Mohan, S. Gurumurthi, and M. Stan. FlashPower: A detailed power model for NAND flash memory. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pp. 502 –507. 2010.

[43] V. M. R. Muggeo. Segmented: An R package to Fit Regression Models with Broken-Line Relationships. *R News*, 8(1):20–25, 2008.

[44] L. Negri, M. Sami, D. Macii, and A. Terranegra. FSM-based power modeling of wireless protocols: the case of Bluetooth. In *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, pp. 369 – 374. 2004.

[45] V. Pallipadi and A. Belay. cpuidle—Do nothing, efficiently.... In *Proceedings of the Linux Symposium*. 2007.

[46] V. Pallipadi and A. Starikovskiy. The Ondemand Governor. In *Proceedings of the Linux Symposium*. 2006.

[47] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pp. 153–168. ACM, New York, NY, USA, 2011.

[48] X. Perez-Costa and D. Camps-Mur. IEEE 802.11E QoS and power saving features overview and analysis of combined performance. *Wireless Communications, IEEE*, 17(4):88–96, 2010.

[49] X. Perez-Costa, D. Camps-Mur, and T. Sashihara. Analysis of the integration of IEEE 802.11e capabilities in battery limited mobile devices. *Wireless Communications, IEEE*, 12(6):26–32, 2005.

[50] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3G networks. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pp. 137–150. ACM, New York, NY, USA, 2010.

[51] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[52] E. Rantala, A. Karppanen, S. Granlund, and P. Sarolahti. Modeling energy efficiency in wireless internet communication. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, MobiHeld '09, pp. 67–68. ACM, New York, NY, USA, 2009.

[53] A. Rice and S. Hay. Decomposing power measurements for mobile devices. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pp. 70–78. 2010.

[54] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pp. 3–3. USENIX Association, Berkeley, CA, USA, 2008.

[55] A. Sagahyroon. Power Consumption in Handheld Computers. In *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on*, pp. 1721–1724. 2006.

[56] F. Shearer. *Power Management in Mobile Devices*. Communications engineering series. Newnes, 2007.

[57] R. Suda and D. Q. Ren. Accurate Measurements and Precise Modeling of Power Dissipation of CUDA Kernels toward Power Optimized High Performance CPU-GPU Computing. In *Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on*. 2009.

[58] Y. Xiao, P. Savolainen, A. Karppinen, M. Siekkinen, and A. Ylä-Jääski. Practical Power Modeling of Data Transmission over 802.11g for Wireless Applications. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pp. 75–84. ACM, New York, NY, USA, 2010.

[59] B. Yoo, Y. Won, J. Choi, S. Yoon, S. Cho, and S. Kang. SSD characterization: from energy consumption's perspective. In *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*, HotStorage'11, pp. 3–3. USENIX Association, Berkeley, CA, USA, 2011.

[60] H. D. Young and R. A. Freedman. *University Physics*. Addison-Wesley Publishing Compnay, Inc., 9th edition, 1996.

[61] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, pp. 105–114. ACM, New York, NY, USA, 2010.