

Simulating the Energy Management on Smartphones using Hybrid Modeling Techniques

Ibrahim Alagöz, Christoffer Löffler, Vitali Schneider, and Reinhard German

Department of Computer Science 7,
University of Erlangen-Nuremberg, Germany
{german,vitali.schneider@cs.fau.de,
ibrahim.alagoez@eei.stud.uni-erlangen.de,
christoffer.loeffler@informatik.stud.uni-erlangen.de
<http://www7.cs.fau.de>

Abstract. With the global growth of the market for smartphones new business ideas and applications are developed continuously. These often utilize the resources of a mobile device to a considerable extent and reach the limits of these. In this work we focus on the simulation of an on-demand music service on a modern smartphone. Our simulation model includes higher level descriptions of the necessary hardware components' behavior and their energy consumption. Thereby, the detailed simulation of battery plays a key role in the project. With this simulation study we find optimal parameters for the users of the examined application to maximize playback time, improve its battery life and reduce costly data transmissions.

Keywords: smartphone, energy management, streaming music on-demand, battery lifetime, kinetic battery model, simulation study

1 Introduction

With the global growth of the market for smartphones new business ideas and applications are continuously developed. These often exhaustively utilize the potential capabilities and resources of the devices. The users of these services often face the problem that some apps deplete the battery life too much. Regarding the complex relations between different apps, their settings, system software and hardware in such a device, it is hardly possible to make exact statements about the resource utilization and the resulting lifetime in advance. Nevertheless, using simulation approaches it is feasible to predict the expected system behaviour and to test various scenarios.

In this paper we focus on simulating a music streaming application on a smartphone with the deeper analysis of the battery performance and power consumption of the involved hardware modules. Our simulation model enables us to detect the bottlenecks of the application as well as of the mobile device and lets us predict the system's behavior for different parameter configurations. Thereby, we optimize the parameters of a music streaming app that the user can set in order to maximize the playback time.

The system we simulate consists of an Android smartphone *HTC Incredible S* and the music streaming app *Ubuntu One Music*. It enables the user to stream

music over the smartphone’s data connection, which in our scenario can be either the WiFi or the 3G network. The smartphone is set to continuously play music and is exclusively used for that task. While a data connection is active the device pre-fetches music to a local storage. The device plays the stored music continuously until either the storage or the battery is empty.

The simulated system consists of software and hardware components like chipsets and battery. Accordingly, our smartphone model consists of three parts, each describing an aspect of the smartphone. We model the app’s behavior, which controls the hardware. The hardware model implements the device’s different chipsets, which determine the system’s performance and power consumption. Finally, the battery model simulates the performance of the smartphone’s battery. Thereby, the integration of power consumption as an input unit into the non-linear kinetic battery model (*KiBaM*) [1] and the modeling of correlated WiFi data rates using a linear time invariant system are key methods presented in this paper.

Due to overall complexity of the system as well as of the user’s interaction with the smartphone we had to simplify certain aspects of the system. Thus, in our scenarios the user does not skip tracks and never activates the screen. Furthermore the granularity of the simulation model of certain hardware modules is limited and focuses strictly on the behavior for the use-case which can be validated.

The parameters for the optimization are limited to the controls provided to the user by the music streaming client and by the usage scenario. A scenario is defined by the time the user spends in WiFi and 3G networks. If the user switches off the data connection, the battery lasts much longer. Another important parameter is the size of the device-local storage to buffer music files. We show that the difference between 100MB and 5GB is significant in terms of power efficiency.

The rest of the paper is organized as follows. Section 2 illustrates some related work. In section 3 we describe the simulation model which consists of software, hardware and battery components. There we explain the integration of power consumption into the kinetic battery model. In section 4 we present techniques used for input modeling. The sampling of correlated random variates for the WiFi bitrate is a key topic. In section 5 we present the validation of the simulation model. We show our optimization results in section 6 and our conclusions in section 7.

2 Related Work

In this section we provide a brief summary on some of the related work on battery modeling, power estimation and signal processing.

The existing battery models have been classified by Jongerden in [2] into four categories: electrochemical, analytical, stochastic, and electrical circuit models. The author evaluates these different battery models and concludes the analytical *KiBaM* model, which was proposed in [1], to be most suitable for battery performance modeling. The authors of [3], [4], and [5] describe hybrid approaches based on the *KiBaM* model and extended by an electrical circuit model in order to predict the dynamic circuit characteristics of the battery additionally to its non-linear capacity effects and runtime.

In [6] the authors describe a modeling technique to monitor the power consumption of hardware modules of a smartphone using the power estimation and logging tool *PowerTutor* [7].

The basics of the signal processing and filtering algorithms applied in this work can be found in [8] and [9].

3 Simulation Model

Our simulation model consists of three layers: software, hardware and battery. Thereby, we employed a hybrid modeling approach with combined discrete-event and continuous simulation techniques. Thus, the software and the hardware layers are modeled following the discrete event simulation paradigm [10] with the use of finite state machines (FSMs), which interact by message passing. Furthermore, to represent the continuous performance of the battery, which is based on the *KiBaM* model, we are using system dynamics (SD) approach. The simulation model is completely realized in the simulation framework AnyLogic [11] that is capable of hybrid simulations.

Figure 1 shows the main view of the simulation model in AnyLogic. The software layer models the behavior of the *Ubuntu One Music* streaming client. The hardware layer implements relevant physical components of the smartphone like CPU, sound, memory, and network modules. In our work we focus strictly on the necessary parts for our scenario. Components like the screen, which are not involved for music playback and which usage patterns cannot be correctly validated, are excluded. The modules of software and hardware components communicate with connected components by exchanging messages over ports. Furthermore, the hardware components consume power continuously and need, therefore, to be connected to the battery model via so-called auxiliary variables from the system dynamics package of AnyLogic. The battery layer simulates the discharge of battery and causes stopping of the simulation if its state-of-charge reaches 0%.

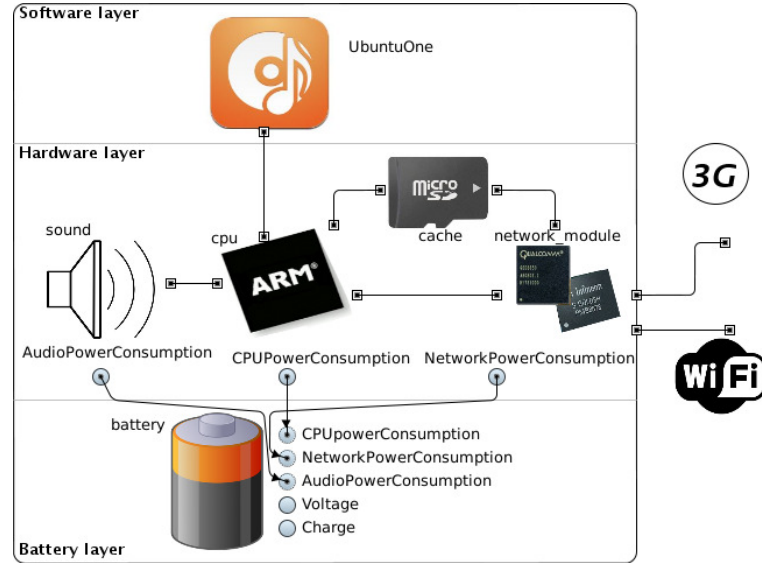


Fig. 1. Main view of the simulation model

3.1 Application / Software Module

We simplify the *Ubuntu One Music* app on the smartphone to two central functionalities. The first one is downloading music from the cloud servers to memory. The second is playing music from memory. As shown in Figure 2 we modeled the behavior of the app as two concurrent FSMs.

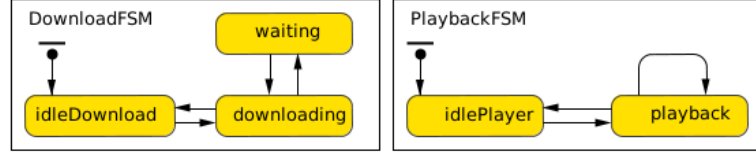


Fig. 2. Parallel state machines of the application module

The *DownloadFSM* controls the downloading task and initiates the network module to download a music track by sending a message to it. Thereby, it switches to the downloading state where it waits for a message from the network module. Such messages indicate either a finished download, upon which the FSM returns to *idleDownload*, or an interruption caused by a full memory or a network failure. In case of interruption the FSM switches to waiting state. Then, after a timeout, it sends a restart message and returns to the downloading state.

The *PlaybackFSM* controls the playback of downloaded music. It starts in *idlePlayer* and takes the transition to *playback* when there is enough data available in the app's local buffer, which is refilled with the data from the memory module in every simulated second. During the playback this buffer is decremented by the amount of data which equals to 1s of music depending on the specified music bit rate. When that local buffer is drained the FSM returns to *idlePlayer*. During the simulation we measure the total playback time which is central to the optimization of parameters.

3.2 Hardware Modules

Like in the real scenario, our hardware model is controlled by the software model. Thus the current state of both app FSMs has influence on several hardware modules and finally on their power consumption.

CPU. In our simulation model the CPU is primarily used to forward control messages between the app and hardware components, since the application logic is simulated at the software layer. Additionally, the CPU module also provides the power consumption for the different application steps that are triggered by messages from other components. As shown in Figure 3 the FSM of the CPU consists of a four distinct states which represent tuples of two overall system states: downloading and playback. Each state consumes thereby power at a different scale. In the section on input modeling we will explain this in more detail.

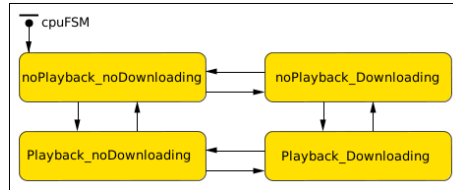


Fig. 3. State machine of the CPU module

Network Module. The network module implements the downloading of data over the available network connections: WiFi or 3G. In this module we measure the accumulated times spent in WiFi or 3G, and the downloaded data for each connection type.

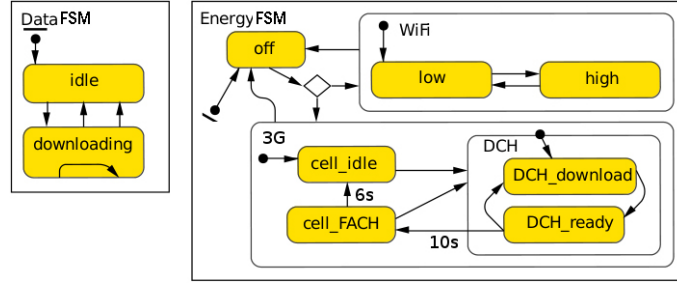


Fig. 4. Network module's finite state machines

The module consists of two state machines (see Figure 4), the *DataFSM* for downloading and the *EnergyFSM* for power consumption. The *DataFSM* contains only two states: *idle* and *downloading*. A download starts when a message with the file size arrives. Every second downloaded data is saved to the memory by sending a message to the memory module. The sizes of these data depend on the current bitrate of the available network, which can vary and needs to be sampled according to distributions shown in section 4.2. The downloading state is left either on the event of a finished or an interrupted download. The download can be interrupted due to full memory or if the current network becomes unavailable. In all cases the *DataFSM* switches back to *idle* and sends the corresponding message to the app component.

The *EnergyFSM* models the power consumption of the WiFi and the 3G network interfaces. Thereby, only one network is active at the same time, e.g., if WiFi is used, 3G is completely shut down and consumes no power. Our measurements were conducted with this simplification in mind. When the simulation starts, the *EnergyFSM* is in the *off* state. It switches to one of the sub FSM, either to WiFi or to 3G, depending on which network is available at the time. The implementation of both sub FSMs is based on the work presented in [6] and [12].

If the WiFi network is active, the state *low* is entered first. When the download is started, the real-world chipset switches into the state *high*, which consumes more power but provides higher bandwidth for downloads. When the number of transferred packages drops below a certain threshold, the chipset switches to the power conserving low state. We simplified this behavior in our model by switching from *high* to *low* only after a download task finishes, because the bit rates we have measured during the download were nearly always above this threshold.

If the 3G network is available, the low power state *cell_idle* is entered. Upon the start of a download, the chipset switches to the DCH-mode, in which it consumes most power. After the download is finished it returns to *cell_idle*, with a short stopover in the state *cell_FACH*. In the FACH mode the power consumption is in between that of DCH and idle.

Memory. The file storage memory, represented by a SDCard in Figure 1, is used to store the downloaded music. It serves as an adapter between the software layer and the network and does not consume power. We can parameterize the phone's local storage by limiting the memory size, which affects the playback time, as can be seen in section 6.

Audio. Our simulation model features the sound module as a separate hardware component with its own power consumption. Its FSM consists of the states *idle* and *playback*. Its transitions are triggered by messages from the software layer.

3.3 Battery Module

The battery, as shown in the Figure 5, is modeled according to the *KiBaM* model with the use of SD. As mentioned in [2] *KiBaM* is able to accurately simulate the non-linear behavior of a battery with regard to the recovery effect. Other models, e.g., the electrochemical model, were considered but excluded, because they offer marginal improvements to the precision at very high effort [2]. The modeling is based on a black-box view at real battery parameters which are deduced by testing of the battery's discharge behavior.

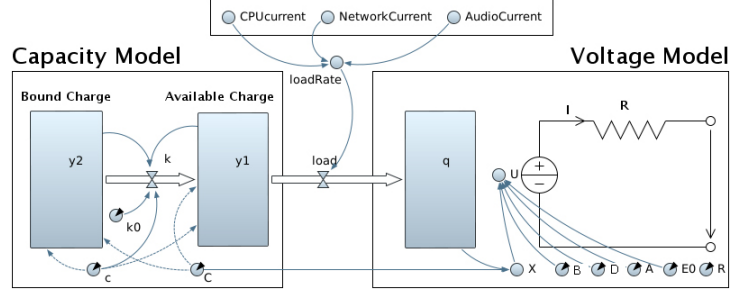


Fig. 5. SD model of battery

The capacity model of *KiBaM* consists of two charge wells, $y1$ and $y2$ representing the available energy and the chemically bound energy, as shown in Figure 5. These two wells are designed for modeling both the recovery and the rate-capacity effect of a battery in order to define its usable capacity. The discharge is the time-discrete value $i(t)$ depicted as *load* in Figure 5. It draws on $y1$ until its capacity is drained. The second well, $y2$, refills the first one continuously with the recovery rate k .

Since the *KiBaM* model requires a discharge current, but only power consumption of a hardware module can be accessed from the smartphone, we created an adapter to transform the overall power consumption from the discrete event model of the hardware layer into a discharge current. The adapter function is based on the voltage model (see Figure 5), which is taken from [5] and is required for the calculation of the instantaneous open circuit voltage U . In Equation 1 the expression for U is given, which depends on the variable $X(t)$ and is parametrized by E_0 , A , B , and D . Thereby, the variable X stands for the normalized charge at the time t . The calculation of it according to [5] is given in Equation 2. The estimation of parameters will be discussed later in the input modeling part.

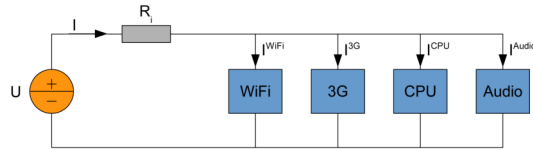


Fig. 6. Conceptual circuit design of hardware modules

For computing the overall discharge current we designed the circuit given in Figure 6, which consists of hardware modules with an internal battery resistance. We used the assumption in [6] that the overall power consumption of all hardware components can be calculated by summing up the individual power consumptions: $P = P^{WiFi} + P^{3G} + P^{CPU} + P^{Audio}$. Thus, we decided to place the components in parallel order to achieve parallel branches in which each com-

ponent is served by its individual current, whereby the same voltage is applied to each of them as shown in Equation 3.

For the conversion of the power consumption to current we first observed a relation between the branch currents depending on the individual power consumptions (see Equation 4) and then we used *Kirchhoff's circuit law* for the computations in Equation 5. We solve the unknown I^{WiFi} of this quadratic equation and compute the overall current I with Equations 3 and 4.

$$U(t) = E_0 + A \cdot X(t) + \frac{B \cdot X(t)}{D - X(t)} \quad (1)$$

$$X(T) = \frac{C - \int_0^T i(t) dt}{C} \quad (2)$$

$$\begin{aligned} I &= I^{WiFi} + I^{3G} + I^{CPU} + I^{Audio} \\ U^{WiFi} &= U^{3G} = U^{CPU} = U^{Audio} \end{aligned} \quad (3)$$

$$\Rightarrow I^{3G} = I^{WiFi} \frac{P^{3G}}{P^{WiFi}}; I^{CPU} = I^{WiFi} \frac{P^{CPU}}{P^{WiFi}}; I^{Audio} = I^{WiFi} \frac{P^{Audio}}{P^{WiFi}} \quad (4)$$

$$U = I \cdot R_i + U^{WiFi}$$

$$\Leftrightarrow U = [I^{WiFi} + I^{3G} + I^{CPU} + I^{Audio}] \cdot R_i + \frac{P^{WiFi}}{I^{WiFi}} \quad (5)$$

$$\Leftrightarrow 0 = [I^{WiFi}]^2 \cdot [1 + \frac{P^{3G}}{P^{WiFi}} + \frac{P^{CPU}}{P^{WiFi}} + \frac{P^{Audio}}{P^{WiFi}}] \cdot R_i - I^{WiFi} \cdot U + P^{WiFi}$$

4 Input Modeling

This section is segmented into the three different abstraction layers of the model: software, hardware, and battery.

4.1 Software

The only input parameters required for the software module are delays for timed transitions in *DownloadFSM*, which take impact on the average download rate and the recovery of the battery. The first parameter relates to the delay between downloads. For downloading of music tracks the app creates Java threads that are successively dispatched. But in between, different other threads of other applications and of the operating system are scheduled, which cause a delay between downloads. We measured these time delays with our device while downloading an amount of music files and modeled them in our model with the fixed mean delay of 3s. Another point is that downloads can be interrupted because of connection loss or full cache. Restarting downloading occurs after a fixed duration of 4s that can be extracted from the source code of the *Ubuntu One Music* application.

4.2 Hardware

The input modeling for the hardware layer consists of two major areas: the power consumption of hardware modules and the download data rates of 3G and WiFi networks.

For measuring the required parameters we logged the information from the smartphone with the tool *PowerTutor* [7]. Among others, it measures the data rates of network modules and approximates the power consumption of each hardware component of the smartphone within a configurable time step and saves it to log files. Furthermore, we used the distribution fitting tool *ExpertFit* [13] to analyze these logs and to look for suitable distribution functions.

CPU. The instantaneous power consumption of the CPU module depends on the tasks which are executed. For instance the download and the playback task have a main impact on the power consumption, next to the background tasks scheduled by other applications and also by the operating system.

So we decided to define 4 states depicted in Figure 3 whereby transitions between these states are triggered if enabling respectively disabling of the download and playback task occurs. In contrast to the remaining hardware modules we decided to model the power consumption of the CPU as a stochastic process because we had to deal with random background tasks.

For each state we conducted an independent experiment in which we logged every second the power consumption and estimated for that a distribution with the *ExpertFit* tool. For each state we obtained different *beta* distributions with the parameters listed in Table 1.

	p (lower shape)	q (upper shape)	min	max
$\neg\text{download} \wedge \neg\text{playback}$	1.71	5.77	3	430
$\text{download} \wedge \neg\text{playback}$	1.53	3.82	20	430
$\neg\text{download} \wedge \text{playback}$	2.01	6.14	36	414
$\text{download} \wedge \text{playback}$	1.72	1.93	20	417

Table 1. Parameters of *beta* distributions for power consumption of the CPU’s states

Power Consumption of Network Modules. The *PowerTutor* application estimates the power consumptions of the defined states in Figure 4 which is summarized in the following table:

WiFi		3G		
Low	High	IDLE	FACH	DCH
20mW	710mW	10 mW	401 mW	570 mW

Table 2. power consumption of the network states

WiFi Data Rates. We measured the download data rates achieved by WiFi network module with the help of the tool *PowerTutor*. The measurements were conducted with 1m proximity from a *WiFi 802.11n* router while downloading the music tracks using the *Ubuntu One Music* app.

In Figure 7a) we present the histogram and the lag correlation plot of the download data rates. We can observe that the data rates are correlated, which is primarily caused by bursts during the downloads. For our use-case it is important to model these bursts, because the use of fixed data rates leads to high inaccuracy, which we could state in validating the simulation’s download, playback and battery discharge times (see section 4). If the download bit rates are much higher than the playback rate, then the cache can overflow so that the energy used for downloading the music file packets is wasted. As a consequence less energy remains in the battery for playback. The correlation of download data rates is also important to determine the peak size of stored data when the cache size is assumed unlimited.

In order to generate the correlated data rates we use statistical methods for correlating and decorrelating of random variates. These methods are employed

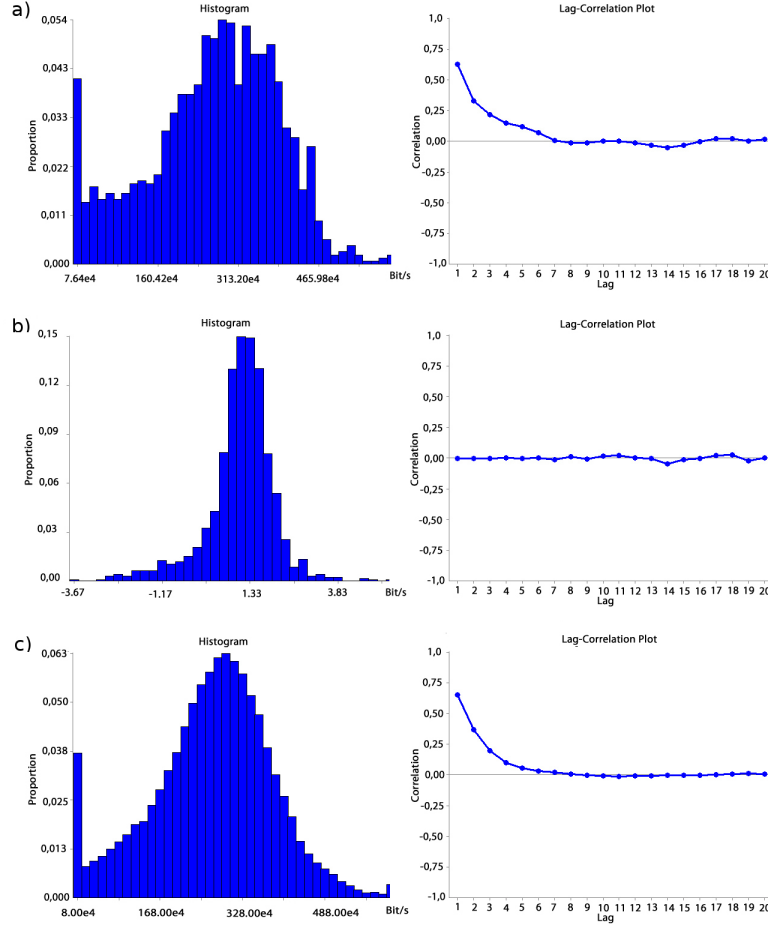


Fig. 7. Histogram and lag correlation plot of the a) measured, b) differential, and c) simulated WiFi download data rates

in the field of digital communications to reduce the transmission power for antennas [9]. Based on this, our approach is firstly decorrelating the download data rates with a pre-filter such that distribution fitting with *expertFit* can be processed. The fitted distribution is needed to generate random variates on runtime in *AnyLogic*. Using an inverse post-filter the correlation properties can be added to the generated random variates again in order to obtain similar correlated data rates during the simulation. In Figure 7 the histogram and the lag correlation plots after pre-filtering (b) and post-filtering (c) are also shown.

For decorrelating random variates, which are identically distributed but also correlated, a prediction filter has been designed. The output of this filter gives predictions of subsequent random variates. For further explanations the following definitions are needed:

- $b[k]$: download data rates $\hat{=}$ identically distributed, correlated random variates
- $\hat{b}[k]$: predictions of download data rates

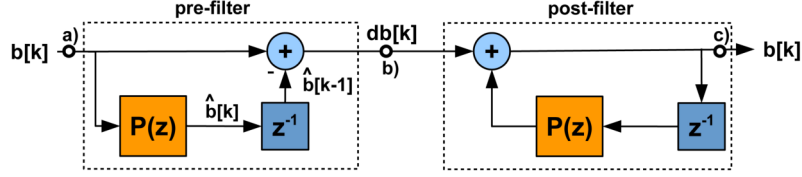


Fig. 8. Signal flow graph with relations between $b[k]$, $\hat{b}[k]$ and $db[k]$

The Figure 8 shows the signal flow graph of the prediction filter $P(z)$ which depicts the relations between $b[k]$, $\hat{b}[k]$ and $db[k]$. The mathematical relation between $b[k]$ and $\hat{b}[k]$ is given in Equation 6, where N is the degree of the finite prediction filter $P(z)$ and '*' stands for a convolution operation. The prediction filter $P(z)$ uses the statistical properties of the random variates $b[k]$ in order to generate $\hat{b}[k]$ in such a way that $\hat{b}[k-1]$ predicts $b[k]$. For achieving decorrelated random variates we use the operation in Equation 7.

$$\hat{b}[k] = \sum_{n=0}^N p[n] \cdot b[k-n] = p[k] * b[k] \quad (6)$$

$$db[k] = b[k] - \hat{b}[k-1] \quad (7)$$

In the following $db[k]$ is called *differential download data rates*. It can be observed in the Figure 7b) that the *random variates* $db[k]$ are decorrelated but are differently distributed than the random variates $b[k]$. It is possible to generate these (i.i.d) random variates $db[k]$ with AnyLogic. After $db[k]$ is generated a post-filter is implemented, which transforms the random variates $db[k]$ to the original random variates $b[k]$ (see Figure 8). With the knowledge of the lag correlation coefficients of $b[k]$ the prediction filter $P(z)$ can be constructed by solving the *Yule-Walker-Equations*. A matrix Φ_{bb} and a vector φ_{bb} are constructed as shown in Equation 8, where $\phi_{bb}[k]$ denotes the k -lag-correlation coefficient and N the filter degree of the prediction filter, which is calculated by Equation 9.

$$\Phi_{bb} = \begin{pmatrix} \phi_{bb}[0] & \phi_{bb}[1] & \cdots & \phi_{bb}[N] \\ \phi_{bb}[1] & \phi_{bb}[0] & \cdots & \phi_{bb}[N-1] \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{bb}[N] & \phi_{bb}[N-1] & \cdots & \phi_{bb}[0] \end{pmatrix} \quad (8)$$

$$\varphi_{bb} = (\phi_{bb}[1] \ \phi_{bb}[2] \ \cdots \ \phi_{bb}[N+1])$$

$$P = \Phi_{bb}^{-1} \cdot \varphi_{bb}, \text{ with } P = (p[0] \ p[1] \ \cdots \ p[N]) \quad (9)$$

In Figure 7a) it can be seen that the first 7 lag correlation coefficients are almost non-zero and would have the most impact in the solution of the *Yule-Walker-Equations*. So we set the filter degree N to 6 and get the filter shown in Figure 9 for generating $db[k]$. Overall, we achieve during our simulation data rates with the statistical properties given in Figure 7c).

3G Data Rates. For 3G we were not able to get a distribution of the data rates. There are numerous effects which influence the rate, many cannot be affected or are not known to the authors. One such impact is for example the number of users in the same 3G cell. Therefore we chose a triangular distribution with

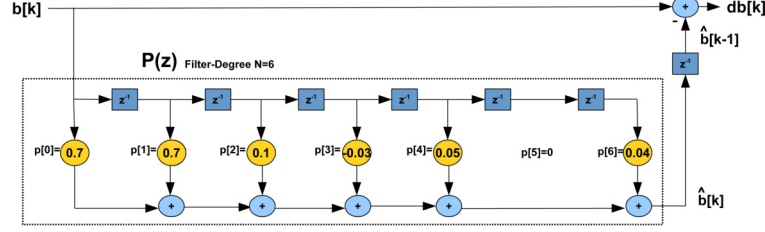


Fig. 9. Filter design to generate $db[k]$

a minimum zero, a mean 800KBit/s and a maximum 3.6MBit/s. These values were measured by a research group at chip.de [14].

Audio Interface. The power consumption for the audio interface is logged by the tool *PowerTutor*, but the accuracy is limited. The tool approximates it with the fixed value of 384mW. Since we have no other means to access the value, we are forced to use this data. The validation of the complete model shows no disadvantage which we can trace back to this value.

Memory. The internal storage of the phone is not modeled as a consumer of power, but as a logical entity setting parameters of the device and scenario. The storage size is crucial for the different finite state machines.

4.3 Battery

We calculated the parameters necessary for our capacity model based on three independent experiments. Each experiment discharged the battery at a different rate until the battery was empty. With the least-squares-algorithm we approximated the following parameters:

k0	c
0.0005 1/s	0.91

Table 3. Parameters of the capacity model

For estimating the parameters of the voltage model logs were recorded during discharging the battery in equidistant timestamps Δt . Among others, the current $i(\Delta t \cdot k)$ and the open circuit voltage $U(\Delta t \cdot k)$ have been recorded, where k is a scalar. Using these values we could calculate the short circuit voltage $V(\Delta t \cdot k) = U(\Delta t \cdot k) - i(\Delta t \cdot k) \cdot R_i$ as well as the normalized charge X according to Equation 2. To find the required parameters for the voltage model the constraint 10 has been derived from Equation 1. The resulting estimated parameters of the measurements are given in Table 4.

$$\min_{E_0, A, B, D, R_i} \sum_k \left[V(\Delta t \cdot k) - E_0 + A \cdot X(\Delta t \cdot k) + \frac{B \cdot X(\Delta t \cdot k)}{D - X(\Delta t \cdot k)} + i(\Delta t \cdot k) \cdot R_i \right]^2 \quad (10)$$

E_0	B	R_i	A	D
3.64V	0.09V	0.122 Ω	0.25V	1.29

Table 4. Parameters of the voltage model

5 Validation

In this section we present the validation of hardware and battery components and of the whole system.

5.1 Hardware

We validated the generation of the correlated WiFi bitrates. For this we downloaded about 425MB of data with our test device and logged the download bitrates. We calculated the mean and standard deviation of the measured bitrates and compared them with those generated during the simulation runs. As it is shown in Table 5 the relative error for the mean bitrate is about 1.1%.

	Real	Simulated	Relative error
Mean	2,688,717 bit/s	2,658,050 bit/s	1.1%
Deviation	1,167,606 bit/s	1,238,357 bit/s	6.0%

Table 5. Validation results of the simulated WiFi download rate.

Furthermore, we also calculated the squared error of 0.14 for the achieved correlation coefficients according to the expression given in Equation 11, where AKF stands for auto correlation function with a coefficient k .

$$\sum_k \left[AKF^{REAL}(k) - AKF^{SIMULATED}(k) \right]^2 = 0.14 \quad (11)$$

5.2 Battery

For the validation of the battery we conducted three experiments with different load levels for the battery. We set the smartphone's discharge current to a fixed discharge rate and logged the battery lifetime. Table 6 shows that our battery model has a relative error of only 2% for high discharge currents. For lower load levels the error grows. This is due to the difficulty of maintaining a constant discharge rate in the real system. Since we used no external discharge devices the Android OS' background tasks introduced noise into the power consumption.

Mean current	Lifetime real	Lifetime simulated	Relative error
148.72 mA	10h 2min 10s	9h 13min 1s	8 %
297.72 mA	3h 59min 18s	4h 19min 53s	9 %
484.54 mA	2h 29min 55s	2h 27min 13s	2 %

Table 6. Comparison and relative error of the simulated and the real batteries' lifetimes.

Furthermore, we validated the short circuit voltage $V(t) = U(t) - i(t) \cdot R_i$ by computing the relative error as follows:

$$\Phi(t) = \frac{|V^{REAL}(t) - V^{SIMULATED}(t)|}{V^{REAL}(t)} \quad (12)$$

The resulting relative error $\Phi(t)$ given in percent for three different discharge currents $i_1(t)$, $i_2(t)$ and $i_3(t)$ is presented in Table 7.

Mean current	Min $\{\Phi(t)\}$ in %	Max $\{\Phi(t)\}$ in %	Mean $\{\Phi(t)\}$ in %
mean $\{i_1(t)\}$ =148.72 mA	0.009 %	1.2 %	0.4 %
mean $\{i_2(t)\}$ =297.72 mA	0.5 %	6.2 %	2.6 %
mean $\{i_3(t)\}$ =484.54 mA	0.6 %	4.5 %	3 %

Table 7. Validation results of the short circuit voltage

5.3 System Wide

We use the system wide validation to validate the power consumption of the different hardware components as well as the software's correct modeling. We measure the system's data rate and power consumption for a test scenario, in which we download 425MB and compare the results with our simulation model.

The first experiment uses only the WiFi network interface and does not play music. The box plots in Figure 10 show how exact the WiFi interface is simulated based on 100 replications. The comparison shows a deviation of the battery discharge value of just 2%, while the relative error of download time is 5%. These relative errors would become much higher if we neglect the correlation of the WiFi download data rates. For instance, we obtained the relative errors of 54% for the battery discharge value and 53% for the download time, if just using an uncorrelated distribution of download rates shown in Figure 7a).

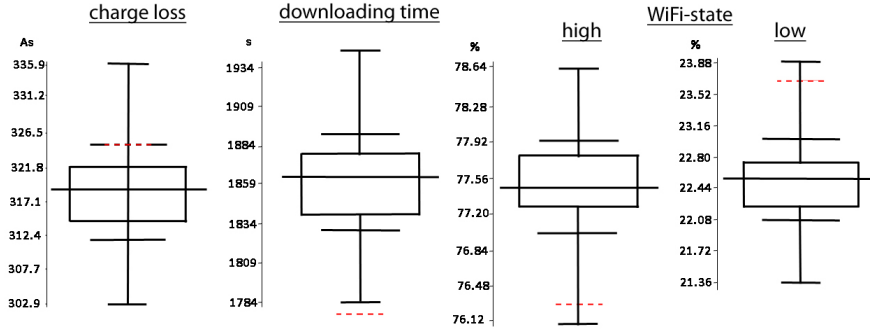


Fig. 10. Boxplots for charge loss, downloading time and proportion of state allocation for WiFi; dashed lines show measurement from real device.

Because of the difficulties in simulating the 3G network module as described in the previous sections, the validation of this module is much more complicated. Although, our simulation model produces good results for download bitrates that match the observations made in [14], the values only represent approximations of the very fluctuating rates in the real world. Due to the simplified modeling approach the overall power consumption of the simulated network module in 3G state was not more than 10% beyond the values we could measure on our hardware with a stable connection.

6 Results

The simulation study's goal is to provide users and developers with possible default parameters to maximize the user's playback time. For this we evaluated several practical scenarios and present the results in Table 8. The scenarios differ by the time the smartphone spends in WiFi, 3G or offline mode. The same cache size of 10GB was set for each use case.

For the first scenario we let the user stay in a WiFi network for 2 hours and in a 3G network for the rest of the simulation. The results in the first line of Table 8 show that on average 3110MB of music were uselessly downloaded and could not be played back because of a lack of energy. The battery was drained on average after already 6.57h.

Based on these results we limited the time for downloads to 2 hours, since 3110 MB of wasted downloads show space for improvement. In the second scenario the user stood in a WiFi network for 30m and in a 3G network for 90m.

Our simulation shows an extremely extended battery life and playback time on average, 22.26h and 17.98h respectively.

The difference in the battery lifetime and the actual playback time hinted to a drained storage due to a slow 3G connection. Therefore we set the time in the WiFi and 3G network to 60m each in the third scenario. The playback time increased on average to 20.6h and the wasted data accounted just to 56.4MB on average. This configuration yields the best results for continuous music playback.

n	Time in WiFi	Time in 3G	Network off	Wasted Downl.	Battery lifetime	Playback
1	2h	Rest	Never	3110MB	6.57h	6.57h
2	30M	90M	2h	0MB	22.26h	17.98h
3	60M	60M	2h	56.4MB	20.60h	20.6h

Table 8. Results for different scenarios

Table 9 shows the confidence interval half-length for mean values calculated for the confidence level fixed to 95%. The quite small confidence intervals after 100 replications indicate that the results seem to be accurate.

n	Wasted Downloads	Battery Lifetime	Playback
1	3.38MB	0.003h	0.003h
2	0.0MB	0.024h	0.032h
3	1.9MB	0.01h	0.01h

Table 9. The mean confidence interval half-length for the mean values from table 8

During the simulation the storage never reached values over 3.11GB. The results furthermore show that users should set the storage space on the device to that value in order to maximize their battery lifetime. This enables them to just switch off the data transmission and saves valuable power for an extended playback time.

Developers of music on demand services are well advised to analyze users' listening pattern and pre-fetch the most listened music tracks. Our simulations show that services which operate purely on demand have a high potential of frustrating customers by draining their smartphone's battery.

Our results show that a default storage size of 100MB is not fitted for power users who listen to more than 6 hours of music at a time. This results in a behavior very similar to ongoing streaming instead of power conserving pre-fetching.

7 Conclusions

We conducted a simulation study in which we analyze the optimal parameters for the use-case of on demand music streaming for modern smartphones. For the simulation we modeled a smartphone consisting of three layers, the software, hardware and battery layer. The paradigm for the software and hardware layer is discrete-event simulation using finite state machines and for the kinetic battery model is system dynamics.

For the input modeling we analyzed log files and used fitted distributions or analyzed source code of the application in order to deduce input parameters. We implemented a linear time invariant system to generate correlated random variates for WiFi download rates.

We conducted several experiments with the implemented model. Our primary interest was the optimization of the music application's playback time. The results show that the user can expect significant differences in playback times with certain settings and scenarios. On the other hand several improvements of the

music application are conceivable. Pre-fetching of data for music added to the user's playlists should be automatically done in WiFi networks and preferably while the device is recharging in order to extend the device's battery's lifetime.

Possible improvements are a more exact modeling of the 3G module's power consumption and the use of external measurement hardware for the device's battery. It may improve the battery model's parameters for lower discharge currents. Future work will include the use of a next-generation smartphone with more detailed logging capability and extended use-cases for better optimization results.

We created a flexible simulation model which executes in few minutes and which can be used to investigate other scenarios with different system configurations. With modifications to the software component the simulation model could possibly simulate other applications than music streaming.

References

1. Manwell, J.F., McGowan, J.G.: Lead Acid Battery Storage Model for Hybrid Energy Systems. *Solar Energy* **50**(5) (1993) 399 – 405
2. Jongerden, M.R.: Model-based Energy Analysis of Battery Powered Systems. PhD thesis, Enschede (December 2010)
3. Chen, M., Member, S., Rincón-mora, G.A., Member, S.: Accurate Electrical Battery Model Capable of Predicting Runtime and I-V Performance. *IEEE Transactions on Energy Conversion* (2006) 504–511
4. Kim, T., Qiao, W.: A Hybrid Battery Model Capable of Capturing Dynamic Circuit Characteristics and Nonlinear Capacity Effects. *IEEE Transactions on Energy Conversion* **26**(4) (2011)
5. Manwell, J., et al.: Improvements to the Hybrid2 Battery Model. American Wind Energy Association. Windpower 2005 Conference.
6. Zhang, L., et al: Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In: Proc. 8th IEEE/ACM/I-FIP int'l conf. on Hardware/Software codesign and system synthesis, ACM (2010)
7. PowerTutor: Version 1.5 (2013) UMich, Northern University, Google Inc. www.powertutor.org.
8. Prandoni, P., Vetterli, M.: Signal Processing for Communications. CRC Press (2008)
9. Kammeyer, K.D.: Nachrichtenübertragung. 5 edn. Vieweg+Teubner, Reihe Informations-/Kommunikationstechnik, Wiesbaden, Germany (Aug 2011)
10. Law, A.M.: Simulation Modeling and Analysis. 4 edn. McGraw-Hill Professional (Aug 2006)
11. AnyLogic: Version 6 (2013) The AnyLogic Company. www.anylogic.com.
12. Balasubramanian, N., Balasubramanian, A., Venkataramani, A.: Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In: Proc. 9th ACM SIGCOMM on Internet Measurement Conference. IMC '09, New York, USA, ACM (2009) 280–293
13. ExpertFit: Version 6.0.2 (2012) Flexsim Software Products, Inc. www.averill-law.com/distribution-fitting.
14. Pauler, W., Heinfling, B.: Das beste Netz aller Zeiten! Website Available online at www.chip.de/netztest; visited on September 3th 2013.