Studiengang:	Informatik
Prüfer:	Prof. Dr. K. Rothermel
Betreuer:	Dipl. Inf. Daniel Herrscher Dipl. Inf. Gregor Schiele
begonnen am:	1. August 2003
beendet am:	31. Januar 2004
CR-Klassifikation:	C.2.1, C.2.4, I.6.4

Studienarbeit Nr. 1917

Emulation mobiler Geräte: Integration eines Batteriemodells

Andreas Störzbach

Institut für Parallele und
Verteilte Systeme (IPVS)
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Inhaltsverzeichnis

1 Einleitung **8**

1.1 Motivation 8

1.2 Aufgabenstellung 9

1.3 Aufbau der Arbeit 10

2 Grundlagen **12**

2.1 Aufbau einer Batterie 12

2.2 Akkutechnologien 13

2.2.1 Nickel-Cadmium-Akkus 14

2.2.2 Nickel-Metall-Hydrid-Akkus 14

2.2.3 Lithium-Ionen-Akkus 15

2.3 Batterieeffekte 16

2.3.1 Rate-Capacity-Effekt 16

2.3.2 Recovery-Effekt 17

3 Batteriemodelle **18**

3.1 Bewertungskriterien 18

3.2 Analytische Modelle 19

3.2.1 Modell von Peukert 19

3.2.2 Modell von Rakhmatov und Vrundhula 21

3.3 „Electrical Circuit“-Modelle, am Beispiel vom Modell von Hageman 24

3.4 Stochastische Batteriemodelle 26

3.4.1 Modell von Panigrahi 26

3.4.2 Modell von Sarkar und Adamou 28

3.5 Elektrochemische Batteriemodelle am Beispiel vom Modell von Gu u.a. 30

3.6 Fazit 31

4 Energieverbrauchermodelle **32**

4.1 Energieverbrauch einer 802.11 Netzwerkkarte 32

4.2 Modell von Feeney und Nilsson 33

4.2.1 Messumgebung 34

4.2.2 Broadcast-Kommunikation 35

4.2.3 Punkt-zu-Punkt-Kommunikation 35

4.2.4 Verwerfen von Paketen 36

4.2.5 Promiskuitiver Betriebsmodus 37

4.2.6 Messergebnisse 37

4.3 Messexperimente von Ebert, Burns und Wolisz 38

4.3.1 Messumgebung 39

4.3.2 Messergebnisse 40

4.4 Fazit 41

5 Realisierung eines Batteriemodells **42**

5.1 Systemumgebung 42

5.1.1 PROC-Filesystem 42

5.1.2 Netzwerkstatistik 42

5.1.3 Batterieschnittstelle 43

5.2 Systementwurf 45

5.2.1 Batterieentwurf 45

5.2.2 Integration der Batteriesoftware in Linux 48

5.2.3 Verbraucherentwurf 48

5.2.4 Programmablauf 50

5.3 Implementierung 51

5.3.1 Threadsynchronisation 51

5.3.2 Taktung 51

5.3.3 Batteriezellenkapazitätsberechnung 52

5.3.4 Benutzerparameter 53

6 Ausblick **56**

6.1 Zusammenfassung 56

6.2 Zukünftige Arbeiten 56

7 Literatur **58**

A Bedienung der Batterieemulationssoftware **62**

A.1 Proc File Simulator 62

A.2 Die Batterieemulationssoftware 62

B Beispielskonfigurationsdateien **64**

Abbildungsverzeichnis

1 Batterielücke 9
2 Batterieaufbau 13
3 Energiedichte unterschiedlicher Akkutechnologien 15
4 Verhältnis zwischen verschiedenen Peukert-Zahlen 20
5 Eindimensionale Diffusion im Modell 22
6 n-stufige Treppenfunktion 23
7 Spice Modell 25
8 Stochastischer Entladeprozess 26
9 Stochastischer Entladeprozess 29
10 Versuchsaufbau Teil 1 39
11 Versuchsaufbau Teil 2 39
12 Verhältnis zw. Energieverbrauch, Ü-Rate und Sendestärke 40
13 E_{bit_good} für div. Übertragungsraten bei 50mW Sendestärke . . . 41
14 Programmübersicht 45
15 ein Batterietakt 51

Tabellenverzeichnis

1 Aufstellung der Ergebnisse 31

2 Messergebnisse (2Mbps) für die verschiedenen Koeffizienten aus [FN01] 36

3 Messergebnisse (2Mbps) für die verschiedenen Koeffizienten aus [FN01] 38

4 Aufbau einer Batteriezelle 46

5 Aufbau der Batterie 47

6 abstrakte Verbraucherklasse 49

7 konstanter Verbraucher 49

8 Netzwerkverbraucher 50

Zusammenfassung

Das „Network Emulation Testbed“ (kurz *NET*) der Uni Stuttgart, Institut für Parallele und Verteilte Systeme bietet eine Umgebung für die Analyse von verteilten Anwendungen und Netzwerkprotokollen. Ein in der Zukunft immer wichtiger werdendes Gebiet sind mobile Netze mit mobilen Endgeräten. Die meisten mobilen Endgeräte sind während ihrem Betrieb auf eine Batterie zur Energieversorgung angewiesen. Damit Anwendungen und Protokolle auf ihren Energiebedarf hin in *NET* untersuchen werde können, wird eine Batterie für die einzelnen Clients benötigt. In der Literatur gibt es verschiedene Batteriemodelle, welche sich mehr oder weniger für die Emulation einer Batterie eignen. Die Studienarbeit: „Emulation mobiler Geräte: Integration eines Batteriemodells“ beschäftigt sich mit der Untersuchung verschiedener theoretischer Batteriemodelle. Neben der Untersuchung der Batteriemodelle beschäftigt sich diese Studienarbeit mit der Untersuchung des Energieverbrauchs von 802.11 Geräten und wie diese für die emulierte Batterie als Verbraucher modelliert werden können. Den Abschluss bildet der Entwurf und Realisierung eines ausgewählten Batteriemodells in *NET* mit Verbrauchern.

1 Einleitung

1.1 Motivation

Die wachsende Anzahl unterschiedlichster mobiler Geräte hat zu einem zunehmenden Bedarf an mobilen Energiespeichern wie Batterien und Akkus geführt. Die erhöhte Leistungsfähigkeit mobiler Geräte wie PDAs, Notebooks, Mobilfunktelefone etc., führt zu einem erhöhten Energiebedarf und zu höheren Anforderungen an die Batterie. Die Entwicklung des Energieverbrauchs und der verfügbaren Energie in mobilen Geräten verläuft unterschiedlich schnell und führt zu einer immer größer werdenden Lücke zwischen der verfügbaren Energie und der benötigten Energie (siehe Abbildung 1 Batterielücke aus [HBZ+03]).

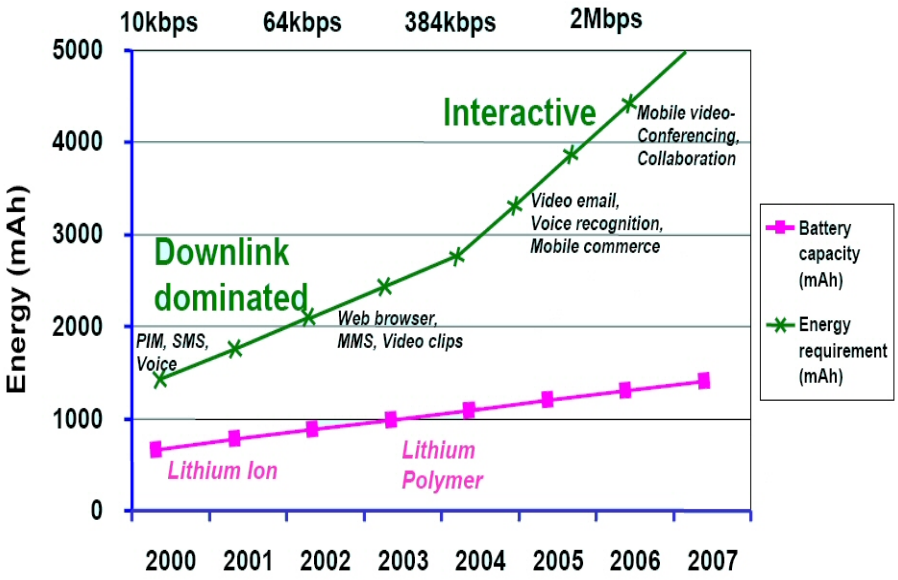

Abbildung 1: Batterielücke, in Anlehnung an [HBZ+03]

Die Verbesserung der verfügbaren Kapazität in der Batterietechnologie bewegt sich pro Jahr bei ungefähr 6% [Buc03], während sich die Leistungsfähigkeit der Geräte nach dem Moorschen Gesetz alle 18 Monate verdoppelt. Bisher entwickelte Hard- und Software hat den Energieverbrauch nur unzureichend in Betracht bezogen. Die verfügbare Energie steht zentral im Mittelpunkt für mobile Geräte und der Bedarf an akkuraten Batteriemodellen für die Entwicklung von Hard- und Software ist sehr hoch. Das ,,Network Emulation Testbed“ bietet eine Umgebung für die Analyse von verteilten Anwendungen und Netzwerkprotokollen auf einem emulierten Netzwerk. Damit diese Anwendungen und Protokolle unter anderem auf ihren Energieverbrauch hin getestet werden können, benötigen die Clients eine Batterie. Ziel der Studienarbeit ist es, eine Batterie auf jedem Client zu emulieren, damit der Energieverbrauch analysiert werden kann.

1.2 Aufgabenstellung

Im Rahmen dieser Studienarbeit sind zunächst existierende Modelle für den Zusammenhang von Batterieentladevorgängen und verbleibender Kapazität zu untersuchen. Ein geeignetes Modell ist für die Implementierung auszuwählen. Als zweiter Schritt ist zu prüfen, wie bestimmte Aktionen auf dem emulierten Gerät Entladevorgänge auslösen können. Von einer einfachen Schnittstelle zur Batterie (Anwendung A entlädt die Batterie explizit um x Einheiten) bis zu einem völlig transparenten Modell (bei jedem Sendevorgang wird automatisch die Batterie um x Einheiten entladen) ist ein breites Spektrum an Lösungsmöglichkeiten denkbar. Schließlich ist das entwickelte Batteriemodell zu implementieren

und in das Emulationssystem *NET* zu integrieren. Dabei ist insbesonders darauf zu achten, dass sich die emulierte Batterie im Linux-basierten *NET* über dieselbe Betriebssystemschnittstelle wie eine tatsächliche Batterie darstellt.

1.3 Aufbau der Arbeit

Die Studienarbeit ist wie folgt gegliedert: In Kapitel 2 wird zunächst ein Überblick über die Grundlagen gegeben. Die unterschiedlichen theoretischen Batteriemodelle werden in Kapitel 3 näher untersucht und bewertet. In Kapitel 4 werden unterschiedliche Energieverbrauchermodelle näher beleuchtet. Die Realisierung des Batteriemodells wird in Kapitel 5 beschrieben. Das Kapitel 6 fasst die Ergebnisse der Studienarbeit zusammen und gibt einen Ausblick. Kapitel 7 beinhaltet das Literaturverzeichnis. Im Anhang wird auf die Bedienung der Batterieemulationssoftware näher eingegangen und Beispielskonfigurationsdateien abgebildet.

2 Grundlagen

In diesem Kapitel werden die Grundlagen zu Batterien und Akkus vorgestellt, welche für die Betrachtung der theoretischen Batteriemodelle in Kapitel 3 notwendig sind.

2.1 Aufbau einer Batterie

Batterien lassen sich in zwei unterschiedliche Kategorien einteilen:

- Primär-Batterien und
- Sekundär-Batterien.

Primär-Batterien sind in der Regel Einwegbatterien und können nach der vollständigen Abgabe der gespeicherten Energie nicht wieder aufgeladen werden. Eine Ausnahme ist die Alkalibatterie, welche eine bedingte Regeneration der Batterie erlaubt [Zin 96]. In der Batterie findet eine elektrochemische Reaktion statt, welche die gespeicherte Energie an den Stromkreislauf abgibt. Nach vollständiger Entladung ist eine Batterie entsprechend umweltgerecht zu entsorgen. Die Sekundär-Batterien oder sogenannte Akkumulatoren, kurz Akku, erlauben im Gegensatz zu Primär-Batterien ein Wiederaufladen der Kapazität innerhalb des Akkus. Die Anzahl der Lade- und Entladezyklen ist je nach Akkutyp unterschiedlich. Die Funktionsweise einer Batterie wird im Folgenden kurz vorgestellt:

Eine Batterie besteht in der Regel aus ein, oder mehreren galvanischen Zellen. Jede dieser Zellen besteht prinzipiell aus folgenden Komponenten:

- zwei Elektroden (Anode und Kathode)
- Elektrolyt

Die zwei Elektroden bestehen aus unterschiedlichen Metallen [Zin01][Häb03]. Die Elektrode, welche aus einem unedlen Metall besteht, wird Anode genannt und ist der Minus-Pol der Batterie. Ein unedles Metall (beispielsweise Zink) neigt dazu Ionen ins Elektrolyt freizusetzten (Oxidation).
Die andere Elektrode wird Kathode genannt und besteht aus einem edlen Metall (beispielsweise Kupfer), d.h. es können Ionen aufgenommen werden (Reduktion). Wenn die beiden Elektroden in das Elektrolyt getaucht werden, dann kann zwischen den beiden eine elektrische Spannung gemessen werden. Über die externe Verbindung der zwei Elektroden fließen die Elektronen von dem Minus-Pol zum Plus-Pol, und im Elektrolyt findet eine Redoxreaktion statt, d.h. positiv geladene Teilchen bewegen sich von der Anode (Oxidation) zur Kathode und werden dort reduziert, siehe Abbildung 2.

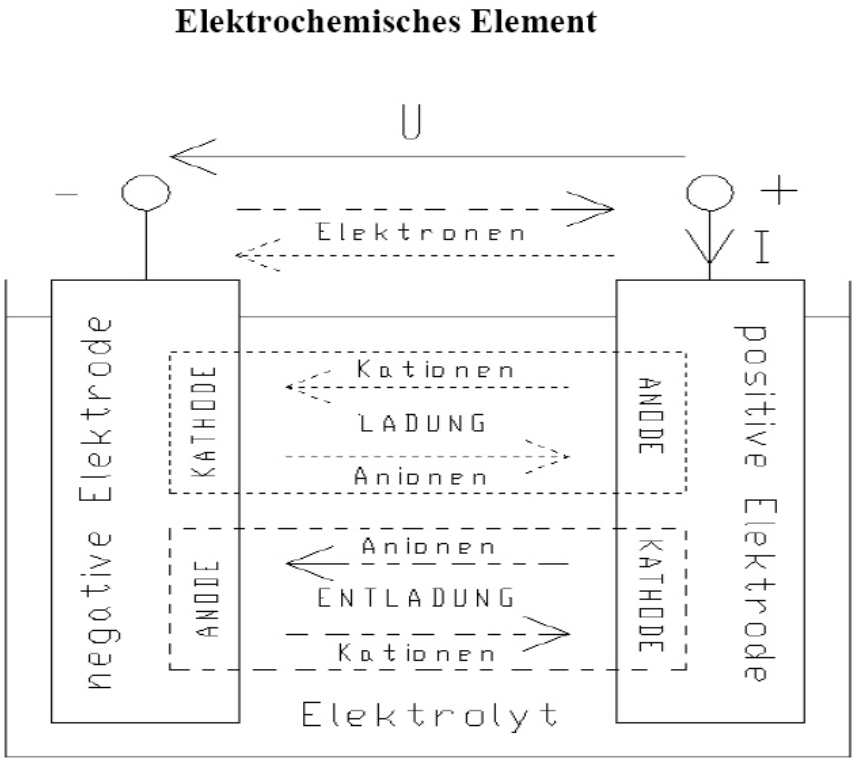

Abbildung 2: Batterieaufbau [Zin01]

Der Vorteil von Akkus ist, dass die elektrochemische Reaktion innerhalb der Zellen umgekehrt werden kann und der Akku so wieder geladen werden kann, siehe Abbildung 2.

2.2 Akkutechnologien

Im Folgenden wird ein kurzer Überblick über die momentan am Markt verbreitetsten Akkutechnologien:

- Nickel-Cadmium-Akku
- Nickel-Metall-Hydrid-Akku
- Lithium-Ionen-Akku

Die verschiedenen Akkutechnologien werden unter unterschiedlichen Gesichtspunkten betrachtet und bewertet:

- **Lade- und Entladecharakteristika:** Das Verhalten der Akkutechnologie während dem Lade- und Entladevorgang. Je nach Technologie verhält sich der Akku unterschiedlich für verschiedene Belastungen.

- **Energiedichte:** Die Menge der Energie, welche pro Gewicht in der Akkutechnologie gespeichert werden kann. Je höher die Energiedichte (in Wh/kg) ist, desto besser ist die Akkutechnologie.
- **Lebenszyklus:** Die Anzahl der theoretisch möglichen Lade- und Entladevorgänge. Je höher diese Anzahl ist, desto rentabler ist die Akkutechnologie für einen Langzeiteinsatz.
- **Kosten:** Die Akkutechnologien unterscheiden sich unter anderem erheblich in den Herstellungskosten.

2.2.1 Nickel-Cadmium-Akkus

Nickel-Cadmium ist die älteste der hier vorgestellten Technologien und ist mittlerweile sehr gut erforscht [Buc01]. In der Zukunft wird diese Technologie keine große Rolle mehr spielen, da giftige Metalle in dem Akku benutzt werden (in einigen Ländern sind diese Akkus mittlerweile verboten) und die anderen Technologien bereits dabei sind, die Nickel-Cadmium-Akkus zu ersetzen.

Lade- und Entladecharakteristika: Nickel-Cadmium-Akkus besitzen ein sehr gutes Lade- und Entladeverhalten. Sie sind sowohl für niedrige als auch für hohe Strombelastungen geeignet und verfügt beispielsweise auch über gute Leistung bei niedrigen Temperaturen [Buc03]. Nickel-Cadmium-Akkus leiden allerdings unter einer sehr starken Selbstentladung und neigen zum Memory-Effekt, d.h. die Akkus müssen regelmäßig geladen und entladen werden.

Energiedichte: Die Energiedichte ist im Vergleich zu anderen Technologien sehr niedrig, siehe Abbildung 3, was mittlerweile ein großer Nachteil im Vergleich zu anderen Technologien ist.

Lebenszyklus: Nickel Cadmium Akkus verfügen über den längsten Lebenszyklus der hier vorgestellten Technologien. Sie liefern bei weit über 1000 Lade- und Entladezyklen sehr gute Leistung und können über mehrere Jahre hinweg eingesetzt werden.

Kosten: Die Technologie ist mittlerweile sehr kostengünstig in der Herstellung und verfügt über das beste Kosten-Lebenszyklus-Verhältnis.

2.2.2 Nickel-Metall-Hydrid-Akkus

Nickel-Metall-Hydrid ist eine der Nachfolgetechnologien von Nickel-Cadmium. Es werden umweltfreundlichere Materialien verwendet und die Technologie hat sich am Markt etabliert.

Lade- und Entladecharakteristika: NiMh-Akkus bieten mittlerweile fast gleich gute Leistungen beim Lade- und Entladeverhalten wie NiCd-Akkus. Der

Memory-Effekt ist weiterhin vorhanden, allerdings wirkt sich der Effekt nicht so stark aus wie bei der NiCd-Technologie und er kann durch entsprechende Behandlung wieder rückgängig gemacht werden [Dur03].

Energiedichte: Die Energiedichte ist ca. 40% höher als bei NiCd, siehe Abbildung 3.

Lebenszyklus: NiMh-Akkus bieteb zwischen 300–500 Lade- und Entladezyklen eine gute Leistung und können über mehrere Jahre hinweg eingesetzt werden.

Kosten: Die Kosten für die NiMh-Technologie sind im Vergleich zu NiCd relativ teuer.

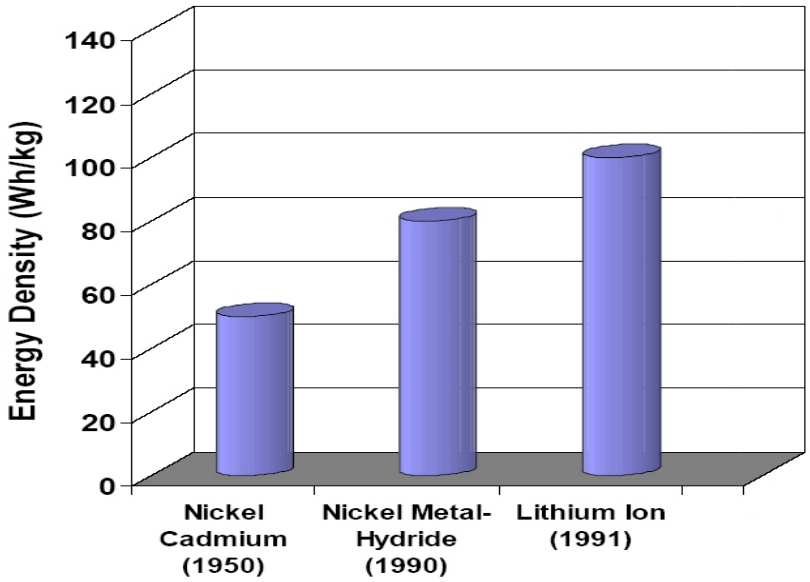

Abbildung 3: Vergleich der Energiedichte unterschiedlicher Akkutechnologien, in Anlehnung an [LRD02]

2.2.3 Lithium-Ionen-Akkus

Die Lithium-Ionen-Technologie ist die jüngste der hier vorgestellten Technologien. Sie unterscheidet sich teilweise sehr stark von den anderen. Beispielsweise ist das Elektrolyt in Lithium-Ionen-Akkus nicht flüssig und die Akkus können deshalb nicht auslaufen.

Lade- und Entladecharakteristika: Li-Ionen-Akkus bieten nur gute Leistungen für geringe und moderate Entladeströme. Hohe Strombelastung ist nicht möglich, da Li-Ionen-Akkus durch einen Schutzschaltkreis innerhalb des Akkus

gesichert sind. Der Memory-Effekt tritt bei Li-Ionen-Akkus so gut wie nicht auf.

Energiedichte: Hier wird die höchste Energiedichte der drei Technologien geboten, was unter anderem ein Grund dafür ist, dass mittlerweile in vielen tragbaren Computern Li-Ionen-Akkus verwendet werden.

Lebenszyklus: Li-Ionen-Akkus altern unabhängig davon, ob sie benutzt werden oder nicht. Die Lebensdauer ist im Vergleich zu anderen Technologien sehr begrenzt (max. 2–3 Jahre) und bereits nach einem Jahr kann der Kapazitätsverlust bis zu 40% betragen [Buc03]. Li-Ionen-Akkus vertragen zwischen 300–500 Ladezyklen.

Kosten: Die Herstellung von Li-Ionen-Akkus ist am teuersten im Vergleich zu den anderen Technologien.

2.3 Batterieeffekte

Das Entladeverhalten einer Batterie wird von unterschiedlichen Batterieeffekten beeinflusst. Im Folgenden werden die zwei wichtigsten Batterieeffekte vorgestellt. Die Folge dieser Batterieeffekte ist es, dass das Entladeverhalten einer Batterie nicht linear verläuft. Um das Entladeverhalten einer Batterie akkurat zu modellieren, müssen diese Batterieeffekte miteinbezogen werden.

2.3.1 Rate-Capacity-Effekt

Jede Batterie hat eine vom Hersteller spezifizierte Angabe, bei welcher Belastung die Batterie die angegebene Kapazität zur Verfügung stellt. Wenn höhere Entladeströme von der Batterie benötigt werden, dann kann die Batterie nicht mehr die volle Kapazität zur Verfügung stellen. Je höher die Batterie belastet wird, desto niedriger ist die verfügbare Kapazität in der Batterie. Dieser Effekt bei Batterien wird Rate-Capacity-Effekt genannt. Die Folge dieses Effekts ist, dass das Entladeverhalten einer Batterie stark nichtlinear verläuft. Der Grund für den Rate-Capacity-Effekt liegt im Inneren der Batterie. Wenn die Belastung über dem vom Hersteller angegeben Wert ist, dann reagieren nur die äußeren Bereiche an der Kathode mit den positiv geladenen Teilchen (Ionen), welche sich von der Anode zur Kathode bewegen (siehe Kapitel 2.1). Im Inneren der Kathode sind noch Bereiche verfügbar, welche für eine chemische Reaktion mit den Ionen geeignet wären, allerdings sind diese Bereiche nicht mehr erreichbar. Das bedeutet also, dass die Batterie theoretisch noch Kapazität zur Verfügung hätte, diese aber nicht nutzen kann. Je größer die Belastung ist, desto mehr Energie wird innerhalb der Batterie verschwendet.

2.3.2 Recovery-Effekt

Ein weiterer Effekt von Batterien ist der Recovery-Effekt. Wenn eine Batterie nicht belastet wird, dann sind die Ionen im Elektrolyt zwischen Anode und Kathode gleichmäßig verteilt. Wenn die Batterie nun stark belastet wird, dann reagieren sehr viele Ionen an der Kathode. Möglicherweise oxidieren nicht genug Ionen an der Anode im selben Zeitraum nach und es entsteht ein Ungleichgewicht. Was wiederum bedeutet, dass ein Mangel an Ionen im Elektrolyt herrscht und die Kapazität der Batterie sinkt. Wenn die Batterie allerdings kurze Zeit nicht belastet wird, dann kann sich dieses Ungleichgewicht im Elektrolyt möglicherweise wieder durch Diffusion ausgleichen. Man spricht dann vom Recovery-Effekt. Die Folge dieses Effekts ist, dass die Kapazität der Batterie wieder steigt. Eine gepulste Entladung mit einer gewissen Ruhezeit zwischen den einzelnen Pulsen kann bedingt durch den Recovery-Effekt zu einer effektiv längeren Batterielaufzeit führen.

3 Batteriemodelle

In diesem Kapitel werden unterschiedliche theoretische Batteriemodelle vorgestellt und näher untersucht. Unter einem Batteriemodell versteht man ein mathematisches oder physikalisches Modell, welches das reale Verhalten einer Batterie widerspiegelt. Es gibt unterschiedliche Modelle, welche die Realität mit unterschiedlicher Abstraktion und Genauigkeit modellieren.

Die hier untersuchten Batteriemodelle lassen sich in vier Kategorien einteilen:

- Analytische Modelle
- „Electrical Circuit“ Modelle
- Stochastische Modelle
- Elektrochemische Modelle

Die Modelle besitzen einen unterschiedlichen Grad an Abstraktion und modellieren das Verhalten einer Batterie mit verschiedenen Techniken. Für die Realisierung eines Batteriemodells im Rahmen dieser Studienarbeit ist neben der möglichst realistischen Modellierung der Batterie auch beispielsweise die technische Realisierbarkeit ein Kriterium. Im Folgenden werden Bewertungskriterien für die einzelnen Batteriemodelle aufgestellt und die Modelle danach bewertet.

3.1 Bewertungskriterien

Die folgenden Batteriemodelle werden nach unterschiedlichen Bewertungskriterien beurteilt. Jedes Modell hat spezifische Vor- und Nachteile und eignet sich daher mehr oder aber auch weniger für eine Implementierung im Rahmen dieser Studienarbeit. Die Bewertungskriterien sind wie folgt:

- **Anzahl der modellierbaren Batterieeffekte:** Eine Batterie verhält sich während der Entladung stark nichtlinear. Um eine Batterie möglichst exakt nachzubilden, muss das Batteriemodell unterschiedliche Batterieeffekte in die Berechnung mit einbeziehen. In der Regel ist ein Batteriemodell exakter, je mehr Batterieeffekte es simulieren kann. Das Maß für das Kriterium ist die Anzahl der modellierbaren Batterieeffekte, wobei die wichtigsten Batterieeffekte der Rate-Capacity-Effekt und der Recovery-Effekt sind. Neben diesen beiden Effekten gibt es noch andere Effekte wie z.B. thermale Effekte oder Alterseffekte, welche ein Batteriemodell vervollständigen, aber schwierig zu modellieren sind.
- **Umsetzbarkeit des Batteriemodells zur Implementierung unter Linux:** Die verschiedenen Batteriemodelle besitzen eine unterschiedliche

Berechnungskomplexität und eignen sich daher unterschiedlich gut für eine Implementierung unter Linux. Voraussetzung für eine Umsetzung sind außerdem, dass das Batteriemodell mit den vom System zur Verfügung gestellten Mitteln und Werkzeugen implementiert werden kann. Das Kriterium wird mit Hilfe einer Ordinalskala von 1 bis 4 eingeordnet, wobei 1 bedeutet, dass sich das Modell sehr gut umsetzen lässt und 4, dass das Modell nicht implementierbar ist.

- **Realitätsnähe des Batteriemodells:** Je nach untersuchtem Batteriemodell wird das Entladeverhalten einer Batterie mehr oder weniger realistisch modelliert. Eine hundertprozentige Nachbildung des Verhaltens einer Batterie wird durch keines der Modelle erreicht. Modelle mit einem hohen Grad an Realitätsnähe besitzen einen geringen Grad an Abstraktion und bilden unter anderem die elektrochemischen Vorgänge innerhalb einer bestimmten Batterietechnologie ab. Als Skala zur Bewertung der Realitätsnähe bietet sich eine Angabe in Prozent an, welche angibt, wie realitätsnah das Modell ist. Die Skala reicht von 100%, für eine perfekte Nachbildung einer Batterie, bis zu 0%.

- **Parametrisierbarkeit des Batteriemodells:** Die Batteriemodelle sind unterschiedlich gut an die verschiedenen Batterietechnologien anpassbar. Die Spannweite reicht von Batteriemodellen, die genau für einen bestimmten Batterietyp entworfen sind, bis zu Batteriemodellen, welche für eine Vielzahl an unterschiedlichen Batterietypen geeignet sind. Die unterschiedlichen Freiheitsgrade in der Parametrisierung eines Modells wird mit Hilfe einer Ordinalskala von 1 bis 4 eingeordnet, wobei 1 für ein frei parametrisierbares Modell steht und 4 für ein Modell mit sehr wenig Freiheitsgraden.

3.2 Analytische Modelle

Analytische Batteriemodelle sind die erste Klasse der Batteriemodelle, welche hier näher untersucht werden. Diese Modelle sind im Bereich der High-Level-Modelle einzuordnen, da von vielen technischen Details abstrahiert wird. In den letzten Jahren wurde eine Vielzahl an verschiedenen analytischen Batteriemodellen entwickelt, welche verschiedene Stärken und Schwächen besitzen. Alle diese Modelle haben gemeinsam, dass sie versuchen, durch eine oder mehrere analytische Gleichungen eine Batterie zu modellieren. Im Folgenden wird eine kleine Auswahl an analytischen Batteriemodellen vorgestellt und die Modelle werden nach den Bewertungskriterien aus Kapitel 3.1 bewertet.

3.2.1 Modell von Peukert

Eines der ersten und einfachsten analytischen Batteriemodelle ist die Formel von Peukert, welche die Entladezeit oder auch tatsächlich nutzbare Kapazität

einer Batterie unter einer konstanten Last berechnen kann [MS96] [Joh03]. Die Formel von Peukert ist wie folgt:

$$T = \frac{C}{I^n}$$

Wobei C für die theoretische Kapazität (in A(mpere)h) der Batterie steht, d.h. das ist die Kapazität, welche als Optimum von der Batterie geliefert werden kann. I steht für den Strom (in Ampere). Der Parameter n steht für die Peukert-Zahl von der verwendeten Batterie. Die Peukert-Zahl spiegelt wider, wie stark der Rate-Capacity-Effekt auf die Batterie bei hoher Strombelastung einwirkt, siehe Abbildung 4. Werte für n sind in dem Intervall [1.1 , 1.3] üblich. T steht für die Zeit (in Stunden) bis die Batterie entladen ist. Über die Entladezeit kann die tatsächliche Kapazität der Batterie berechnet werden, welche je nach Last und Peukert-Zahl von der theoretischen Kapazität stark abweicht.

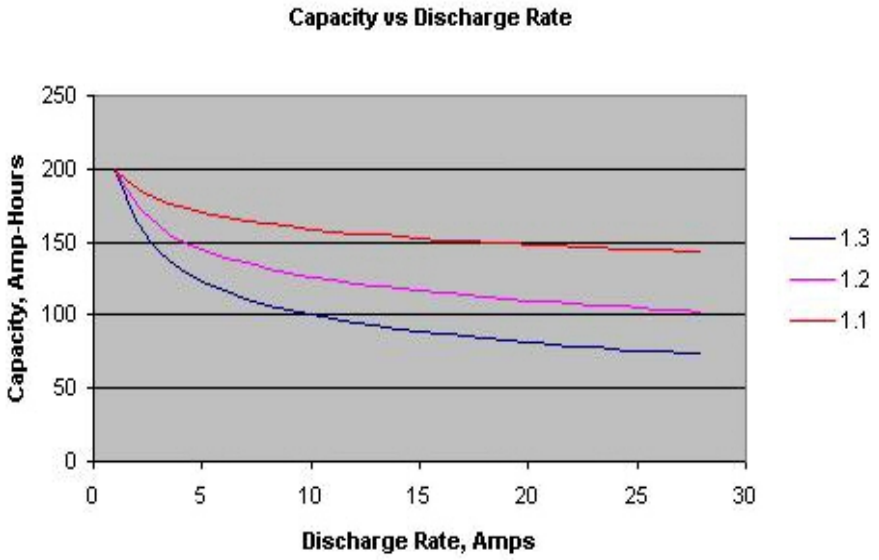

Abbildung 4: Verhältnis zwischen Kapazität und Last für verschiedene Peukert-Zahlen [Joh03].

Betrachtet man das Peukert-Modell unter Berücksichtigung der Bewertungskriterien aus Kapitel 3.1 ergibt sich folgende Situation:

Anzahl der modellierbaren Batterieeffekte: Das Peukert-Modell unterstützt nur den Rate-Capacity-Effekt.

Umsetzbarkeit des Batteriemodells zur Implementierung unter Linux: Das Modell von Peukert besitzt eine sehr geringe Berechnungskomplexität, da ,,nur“ eine einzige lineare Gleichung auszuwerten ist. Insofern eignet

sich das Modell sehr gut, da der Batteriezustand in Echtzeit berechnet werden kann. Für eine Implementierung unter Linux sind keine zusätzlichen Werkzeuge notwendig. Das Modell wird daher mit 1 bewertet.

Realitätsnähe des Batteriemodells: Das Modell von Peukert bietet einen hohen Grad an Abstraktion und lässt sich für verschiedene Batterietechnologien einsetzen. Die Realität wird durch das Modell nur unter gewissen Bedingungen gut abgebildet. Das Modell setzt voraus, dass die Batterie mit einer konstanten Last entladen wird, was aber in vielen Fällen nicht der Realität in mobilen Umgebungen entspricht. Aus diesen Gründen wird das Peukert-Modell mit 30% bewertet.

Parametrisierbarkeit des Batteriemodells: Das Modell kann über den Parameter n an verschiedene Batterien angepasst werden. Die Peukert-Zahl n wird entweder durch Messungen ermittelt oder vom Batteriehersteller mitgeliefert. Das Modell wird mit 1 bewertet.

3.2.2 Modell von Rakhmatov und Vrundhula

Da sich das Peukert-Batteriemodell nur für konstante Batteriebelastung eignet, wurden für variable Batteriebelastung andere Modelle entwickelt. Das folgende Modell von Rakhmatov und Vrundhula [RV02],[RV03] entwickelt die analytischen Gleichungen aus elektrochemischen Gesetzen und ermittelt unbekannte Parameter mit Hilfe von Messungen. Ziel des Modells ist es, die Entladezeit für die Batterie zu bestimmen. Eine Batterie besteht aus Anode, Kathode und Elektrolyt, wie in der Einleitung beschrieben. Das Modell geht davon aus, dass Lade- und Entladevorgänge der Batterie symmetrisch sind. An der Anode oxidieren während der Entladung Ladungsteilchen und an der Kathode werden entsprechend Ladungsteilchen reduziert. Im geladenem Zustand sind die Ladungsteilchen an den Elektroden gleichmäßig verteilt. Nach [BF01 Kapitel 1.4] kann man die Reaktionen an der Elektrode als analytische Gleichung darstellen. Diese analytische Gleichung basiert auf einer eindimensionalen Diffusion über die feste Länge w wobei w der Abstand zwischen der Anode und der Kathode im Elektrolyt ist (siehe Abbildung 5).

Der Zustand einer vollständig geladenen Batterie wird in Abbildung 5(a) dargestellt. Die Ladungsteilchen im Elektrolyt sind gleich verteilt. Durch die Belastung der Batterie werden an der Kathode die Ladungsteilchen reduziert und es entsteht ein Ungleichgewicht 5(b), da nicht genug Ladungsteilchen an der Anode oxidieren. Wenn die Batterie kurzzeitig nicht belastet wird, wird das Ungleichgewicht durch Diffusion ausgeglichen (Recovery-Effekt) 5(c). Sobald die Anzahl der Ladungsteilchen an der Kathode unter einen gewissen Schwellwert fallen kann keine elektrochemische Reaktion mehr stattfinden und die Batterie ist damit entladen 5(d).

Mathematisch wird die Diffusion folgendermaßen modelliert:

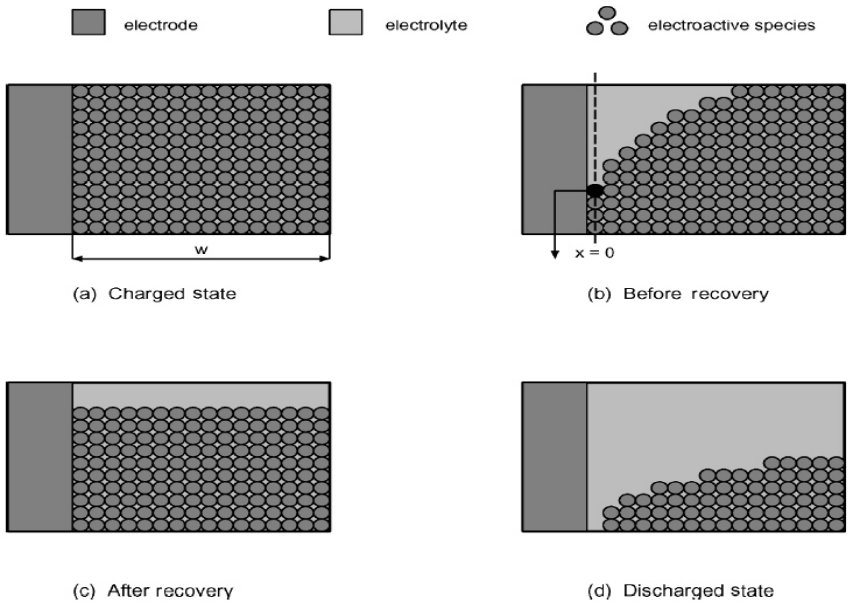

Abbildung 5: Veranschaulichung der eindimensionallen Diffusion im Elektrolyt aus [RV03]

Sei $C(x,t)$ die Konzentration der Teilchen im Elektrolyt zur Zeit $t \in [0,L]$ mit $x \in [0,w]$. Für das Modell ist im Grunde nur die Konzentration der Teilchen am äußeren Rand der Elektrode (Kathode) interessant, da ab diesem Zeitpunkt, wo keine Ladung mehr am Rand der Elektrode ist, die Kapazität der Batterie erschöpft ist.
Sei die Konzentration am Anfang C^*, und sei $\rho(t) = 1 - \frac{C(0,t)}{C^*}$. Wenn $C(0,t)$ für $t = L$ unter den C_{cutoff} Level fällt, es also keine Ladungsteilchen mehr am äußereren Rand der Elektrode gibt, dann ist zu $\rho(L) = 1 - \frac{C_{cuttoff}}{C^*}$ die Batterie entladen. Für die Funktion $\rho(t)$ ist eine geeigenete analytische Gleichung zu entwerfen. Die genaue Umformung und Berechnung für die Formel $\rho(t)$ ist hier nicht von Interesse und kann in [BF01][RV02][RV03] nachgeschlagen werden.
Für $\rho(t)$ ergibt sich nun folgende Gleichung:

$$\rho(t) = \frac{1}{\nu F A w C^*} \left[\int_0^t i(\tau) d\tau + 2 \sum_{m=1}^{\infty} \int_0^t i(\tau) e^{-\frac{\pi^2 D(t-\tau)m^2}{w^2}} d\tau \right]. \qquad (1)$$

A ist die Fläche der Elektrode, ν ist die Anzahl der reagierenden Elektronen und F bezeichnet die Faradaysche Konstante (Ladung von einem mol Elektronen). Sei $\alpha = \nu F A w C^* \rho(L)$ und $\beta = \frac{\pi\sqrt{D}}{w}$ und $t = L$, dann ergibt sich aus (1) folgende Gleichung:

$$\alpha = \int_0^L i(\tau) d\tau + 2 \sum_{m=1}^{\infty} \int_0^L i(\tau) e^{-\beta^2 m^2 (L-\tau)} d\tau. \qquad (2)$$

Die Entladezeit L steht also in Zusammenhang mit dem Entladeprofil $i(t)$ und den zwei Parameter α und β . Der Parameter α repräsentiert die Batterieka-

pazität (in Coulomb) und β modelliert die Nichtlinearität der Batterie. Die Parameter α und β können durch einfache Messversuche ermittelt werden.
Für eine konstante Strombelastung reduziert sich die Gleichung (2) auf folgende Form:

$$\alpha = IL \left[1 + 2 \sum_{m=1}^{\infty} \frac{1 - e^{-\beta^2 m^2 L}}{\beta^2 m^2 L} \right] \tag{3}$$

Nach n Versuchen mit $I_{(1)},..,I_{(n)}$ hat man n-mal Werte für L. α und β können nun bestimmt werden, indem man die folgende Summe von Quadraten minimiert: $\sum |I_{(k)} - \hat{I}_{(k)}|^2$, wobei $\hat{I}_{(k)}$ durch folgende Gleichung bestimmt wird:

$$\hat{I}_{(k)} = \frac{\alpha}{L_{(k)} + 2 \sum_{m=1}^{\infty} \frac{1 - e^{-\beta^2 m^2 L_{(k)}}}{\beta^2 m^2}}$$

Für variable Strombelastungen kann man das Modell über eine Treppenfunktion an die variable Last annähern, siehe Abbildung 6.

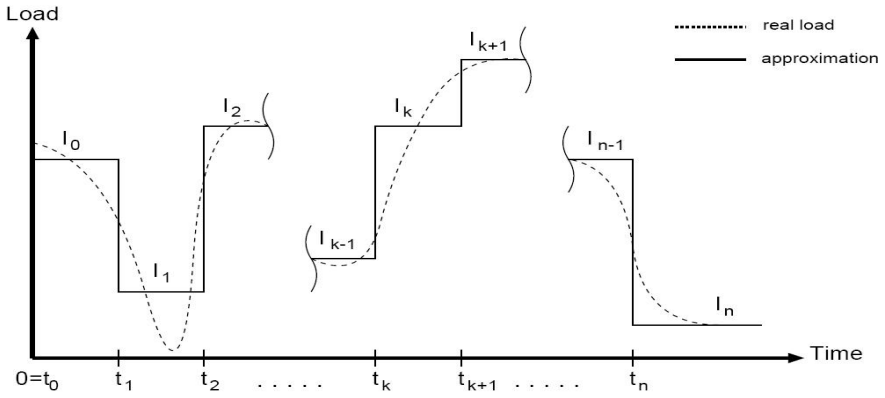

Abbildung 6: n-stufige Treppenfunktion [BF01]

In [RV03] wurden verschiedene Messexperimente durchgeführt und für die Parameter α und β beispielsweise folgende Werte ermittelt: α=39668 und β= 0.574. Das Modell wurde experimentell mit verschiedenen Strombelastungen und Akkutechnologien verglichen und der Fehler lag bei maximal 5%.

Betrachtet man das Modell unter Berücksichtigung der Bewertungskriterien ergibt sich folgende Situation:

Anzahl der modellierbaren Batterieeffekte: Das Modell von Rakhmatov und Vrundhula unterstützt Rate-Capacity- und Recovery-Effekte.

Umsetzbarkeit des Batteriemodells zur Implementierung unter Linux: Das Modell besitzt eine moderate Berechnungskomplexität. Allerdings setzt das Modell voraus, dass das Entladeprofil für die Batterie im Voraus bekannt ist. Eine Modifikation des Modells für den Einsatz in Echtzeit unter Linux

benötigt zusätzlichen Aufwand [HT03]. Für eine Implementierung unter Linux sind keine zusätzlichen Werkzeuge notwendig. Das Modell wird mit 3 bewertet.

Realitätsnähe des Batteriemodells: Das Modell bildet das Entladeverhalten einer Batterie gut nach, unter der Voraussetzung, dass die Batterie für Messversuche vorhanden ist (Parameterermittlung). Es wird konstante und variable Strombelastung modelliert und die Fehlerquote des Modells liegt in Tests bei maximal 5%. Aus diesen Gründen wird das Modell von Rakhmatov und Vrundhula mit 85% bewertet.

Parametrisierbarkeit des Batteriemodells: Das Modell ist gut an verschiedene Batterietechnologien über die zwei Parameter α und β anpassbar. Die Parameter werden durch Messexperimente bestimmt und steuern das Verhalten der zu modellierenden Batterie. Das Modell wird mit 1 bewertet.

3.3 „Electrical Circuit“-Modelle, am Beispiel vom Modell von Hageman

„Electrical Circuit“-Modelle modellieren das Entladeverhalten einer Batterie durch einen elektrischen Schaltkreis. Im Allgemeinen wird die Batterie durch ein SPICE- (Simulation Program with Integrated Circuit Emphasis) oder VHDL-(Very high speed integrated circuit Hardware Description Language) Modell repräsentiert. In SPICE können elektrische Schaltungen auf unterschiedlichste Weise simuliert werden (Kennlinien, Einschwingvorgänge...). VHDL wird zum Hardwaredesign und Chipentwurf verwendet und erlaubt auch die Modellierung von Schaltkreisen. Im Folgenden wird ein (SPICE)-Modell von Hageman vorgestellt.

Das Modell ist speziell für NiCd-Batterien entwickelt worden [Hag93]. Die grafische Veranschaulichung des (P)SPICE-Modells ist in Abbildung 7 abgebildet. Die Ladung der Batterie wird durch einen Kondensator modelliert, im Modell C_Capasitor, siehe Abbildung 7. Die momentane Spannung E_bat wird über eine Lookup-Tabelle ermittelt, welche den Ladestand des Kondensators mit berücksichtigt. Weiterhin wird der Rate-Capacity-Effekt über E_Lostrate modelliert, siehe Abbildung 7. Hier wird ebenfalls in einer Lookup-Tabelle nachgeschaut wie stark der Effekt in Erscheinung tritt, abhängig vom Ladestand des Kondensators und der Strombelastung.

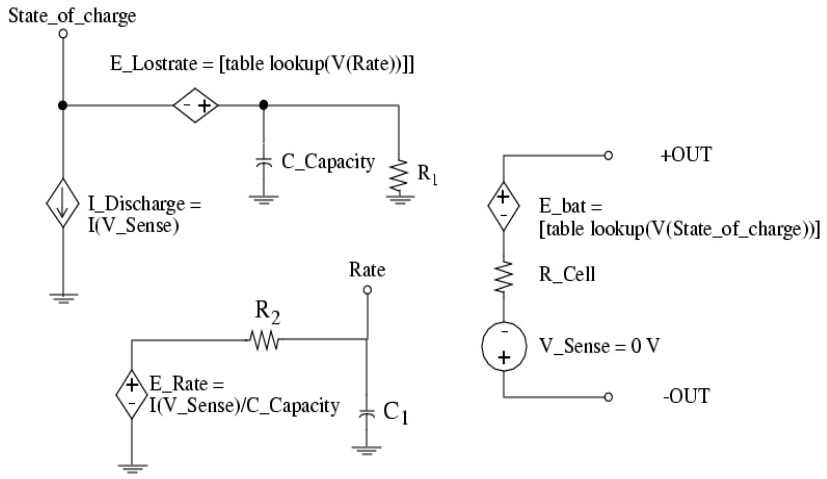

Abbildung 7: Spice Modell für NiCd Batterien [Mar99]

Betrachtet man die elektrischen Schaltkreismodelle unter Berücksichtigung der Bewertungskriterien ergibt sich folgende Situation:

Anzahl der modellierbaren Batterieeffekte: Das Modell unterstützt nur den Rate-Capacity-Effekt. Es gibt allerdings komplexere SPICE-Modelle, welche unter anderem auch thermale Effekte modellieren [LRD+02]. Allerdings unterstützt keines der elektrischen Schaltkreismodelle den Recovery-Effekt.

Umsetzbarkeit des Batteriemodells zur Implementierung unter Linux: Die Berechnungskomplexität des Batteriemodells ist unterschiedlich, je nach eingestellter Genauigkeit (via Parameter) im SPICE-Simulator. Da im Linux Cluster NET kein (P)SPICE (oder VHDL) verfügbar ist, können „Electrical Circuit“-Modelle nicht implementiert werden. Aus diesem Grund wird das Modell mit 4 bewertet und auf Grund der fehlenden Software nicht näher betrachtet.

Realitätsnähe des Batteriemodells: Das Modell von Hageman modelliert das Verhalten eines bestimmten Battertyps sehr detailliert. Es wird konstante und variable Batteriebelastung modelliert. Da keine Recovery-Effekte modelliert werden, wird das Modell mit 75% bewertet.

Parametrisierbarkeit des Batteriemodells: Das Modell kann über unterschiedliche Parameter an eine bestimmte Batterietechnologie (NiCd) eingestellt werden. Für andere Batterietechnologien wird ein anderes SPICE-Modell benötigt. Das Modell wird mit 3 bewertet.

3.4 Stochastische Batteriemodelle

Stochastische Batteriemodelle modellieren das Entladeverhalten einer Batterie über einen stochastischen Prozess. Die Modelle sind den analytischen Modellen sehr ähnlich. Die im Folgenden vorgestellten Batteriemodelle modellieren das Verhalten der Batterie über eine Markov-Kette und beherrschen Rate-Capacity- und Recovery-Effekte [PCD+01].

3.4.1 Modell von Panigrahi

Das Batteriemodell besteht aus folgenden Komponenten:

- eine Batteriezelle
- ein stochastischer Entladeprozess
- eine Recoveryfunktion
- eine Lookup-Tabelle für den Rate-Capacity-Effekt
- Batterieparamater

Eine Batterie besteht in dem Modell aus einer Batteriezelle, welche durch eine endliche, diskrete Anzahl von Elementarladungen repräsentiert wird. Im Modell wird zwischen der theoretischen Kapazität T und der tatsächlich verfügbaren Kapazität N der Batteriezelle unterschieden. Die Kapazität N ist unter einer konstanten, vom Batteriehersteller empfohlenen, Entladung effektiv verfügbar und weicht unter Umständen stark von der theoretischen Kapazität der Batteriezelle ab.
Der Entladevorgang des Batteriemodells ist in einzelne, diskrete Zeitscheiben unterteilt. Die Batterie befindet sich zum Start des Entladevorgangs im Zustand N. Die folgende Abbildung 8 veranschaulicht den Entladevorgang.

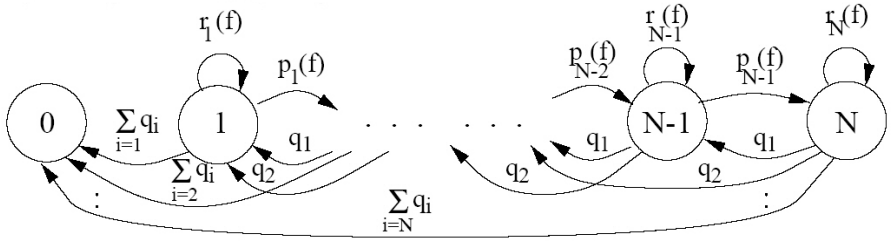

Abbildung 8: Stochastischer Entladeprozess [PCD+01]

Die Batterie ist entladen, falls der Zustand 0 erreicht wird. Ein Zustandsübergang findet in dem Modell statt, wenn in einer Zeitscheibe die Batterie im Zustand z ist und von dem System n Energieeinheiten angefordert werden. Wenn in einer Zeitscheibe keine Energie angefordert wird, dann erholt sich die Batterie

mit einer gewissen Wahrscheinlichkeit. Sei q_i mit $i > 0$ die Wahrscheinlichkeit, dass in einer Zeitscheibe i Energieeinheiten angefordert werden, dann wechselt die Batterie vom Zustand z in den Zustand $z - i$. Sei q_0 die Wahrscheinlichkeit, dass in einer Zeitscheibe keine Energieeinheiten angefordert werden, dann ergibt sich ein Wechsel vom Zustand z in den Zustand $z + 1$. Für das Modell können unterschiedliche Wahrscheinlichkeitsverteilungen benutzt werden. Zur Validation wurde in [PCD+01] mit einem Bernoulli-Prozess gearbeitet.

Das ursprüngliche Modell simuliert den Rate-Capacity-Effekt nicht, kann aber leicht erweitert werden [PCD+01 Kapitel 4.4]. Dazu wird eine Lookup-Tabelle angelegt, in welcher der Entladeprozess jedes Mal nachschaut, um wie viel sich die Batterie bei einer bestimmten Anforderung entlädt. Diese Lookup-Tabelle muss an verschiedene Batterietypen angepasst werden und spiegelt die Effizienz der Batterie wider. Der stochastische Entladeprozess setzt voraus, dass die Wahrscheinlichkeiten für die Energieanforderungen im Voraus bekannt sind. Im Weiteren wird aber auch eine Modifikationsmöglichkeit vorgestellt, um das Modell dynamisch auf den Energiebedarf reagieren zu lassen. Der Recovery-Effekt ist je nach Zustand der Batterie unterschiedlich. Der gesamte Entladeprozess kann in unterschiedliche Phasen unterteilt werden f mit $(f = 0, .., f_{max})$. Jede Phase ist d_f Elementarladungen lang und geht in die nächste Phase über, wenn d_{f+1} Elementarladungen angefordert werden. Die Wahrscheinlichkeit p für den Recovery-Effekt (Wiedergewinnung einer Elementarladung im Idlezustand) ist folgendermaßen definiert:

$$p_j(f) = \begin{cases} q_0 e^{-(N-j)g_N - g_C(f)} & j = 1, .., N-1 \wedge f = 1 \\ q_0 e^{-(N-j)g_N - d_f g_C(f)} & j = 1, .., N-1 \wedge f = 2, ..f_{max} \end{cases}$$

Wobei j für den aktuellen Zustand repräsentiert und die Parameter g_N und g_C die Recoveryeigenschaften der Batterie modellieren. Die Wahrscheinlichkeit, dass keine Ladung im Idlezustand in einer Phase f zurückgewonnen wird ist folgendermaßen definiert:

$$\begin{aligned} r_j(f) &= q_0 - p_j(f) \quad j = 1, .., N-1 \\ r_N(f) &= q_0 \end{aligned}$$

Damit das Modell dynamisch auf die Energieanforderungen reagieren kann, wird in [PCD+01 Kapitel 4.3] eine Erweiterung vorgeschlagen. Der stochastische Entladeprozess entfällt und stattdessen reagiert das Modell dynamisch auf die Anforderungen. In jeder Zeitscheibe wird dem Batteriemodell von den Verbrauchern gemeldet, wie viel Energie verbraucht wurde.

Für ein möglichst realistisches Nachbilden einer Batterie müssen die Parameter des Batteriemodells entsprechend gesetzt werden. In [PCD+01] wurde dazu eine Lithium-Ionen-Batterie als Referenz genommen. Die Lithium-Ionen-Batterie wird durch ein elektrochemisches Modell modelliert, welches das Verhalten der Batterie durch eine Reihe von partiellen Differentialgleichungen beschreibt [DFN93]. Zu diesem Modell existiert ein Programm mit dem Namen Dualfoil [New03], welches zur Simulation des Modells benutzt werden kann. Die

Validation des stochastischen Modells in [PCD+01] im Vergleich zum elektrochemischen Modell hat ergeben, dass der maximale Fehler bei 4% liegt und der durchschnittliche Fehler bei 1%. Die Laufzeit des stochastischen Modells liegt im zweistelligen Sekundenbereich, wohingegen die Simulationszeit des elektrochemischen Modells zwischen ein und zwei Tagen liegt.

Betrachtet man das Modell von Panigrahi unter Berücksichtigung der Bewertungskriterien ergibt sich folgende Situation:

Anzahl der modellierbaren Batterieeffekte: Das Modell von Panigrahi unterstützt Rate-Capacity- und Recovery-Effekte.

Umsetzbarkeit des Batteriemodells zur Implementierung unter Linux: Das Modell besitzt eine geringe Berechnungskomplexität. Der stochastische Prozess entfällt für eine Implementierung unter Linux, damit das Batteriemodell in Echtzeit auf die Verbraucher reagieren kann. Die Implementierung unter Linux benötigt keine zusätzlichen Werkzeuge. Das Modell wird mit 1 bewertet.

Realitätsnähe des Batteriemodells: Das Modell lässt sich gut an verschiedene Batterietechnologien anpassen. Das Entladeverhalten der Batterie wird sehr genau abgebildet (Fehler max 5%). Es wird konstante und variable Batteriebelastung modelliert. Das Modell wird mit 85% bewertet.

Parametrisierbarkeit des Batteriemodells: Das Modell ist gut an verschiedene Batterietechnologien anpassbar. Der Rate-Capacity-Effekt wird über eine Lookup Tabelle eingestellt und der Recovery-Effekt über eine Funktion modelliert. Das Modell wird mit einer 1 bewertet.

3.4.2 Modell von Sarkar und Adamou

Sarkar und Adamou haben in [SA03] das Modell von Panigrahi aufgenommen und für mehrere Batteriezellen innerhalb einer Batterie erweitert. Eine Batterie besteht aus einer Gruppe von L Zellen und jede dieser Zellen hat wie in [PCD+01] eine theoretische (in [SA03] mit C bezeichnet) und eine tatsächlich verfügbare Kapazität. Der stochastische Entladeprozess einer Batteriezelle wird hier über einen zweidimensionalen Zustandsraum modelliert, siehe Abbildung 9.

Die Zelle startet in dem Zustand (N,C) und jeder Folgezustand (n_i,c_i) repräsentiert den aktuellen Kapazitätsstand zum Zeitpunkt i. Da eine Batterie aus L Zellen besteht, wird die Batterie in dem Modell zum Zeitpunkt k als $2L$-Tupel $x_k = (n_{1k}, c_{1k}, ..., n_{Lk}, c_{Lk})$ dargestellt. Die Berechnung des Recovery-Effekts erfolgt analog zu Panigrahi. Der Rate-Capacity-Effekt wird interessanterweise ignoriert in dem Modell. Ziel des Modells von Sarkar und Adamou ist es eine Strategie zu entwickeln, wie die Batteriezellen in der Batterie zu belasten

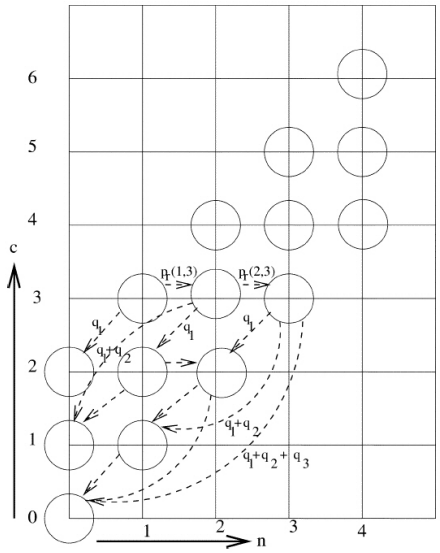

Abbildung 9: Stochastischer Entladeprozess aus [SA03]

sind. Wird beispielsweise in einer Batterie mit zwei Zellen zuerst die erste Zelle für die Energieversorgung benutzt und danach die zweite, dann liefert die Batterie eine maximale Kapazität von $2N$ (unter der Annahme, dass in jedem Zeittakt die Batterie entladen wird). Wenn die zwei Zellen sich der Reihe nach abwechseln würden, wäre die erreichbare Kapazität durch den Recovery-Effekt zwischen $2N$ und $2C$. Sarkar und Adamou entwickeln in [SA03] eine Strategie, welche versucht die maximal erreichbare theoretische Kapazität jeder einzelnen Zelle auszuschöpfen.

Betrachtet man das Modell von Sarkar und Adamou unter Berücksichtigung der Bewertungskriterien ergibt sich folgende Situation:

Anzahl der modellierbaren Batterieeffekte: Das Modell von Sarkar und Adamou unterstützt nur den Recovery-Effekt, allerdings sollte eine Modellierung des Rate-Capacity-Effekts in Anlehnung an das Modell von Panigrahi problemlos möglich sein.

Umsetzbarkeit des Batteriemodells zur Implementierung unter Linux: Das Modell besitzt eine geringe Berechnungskomplexität. Der stochastische Prozess entfällt für eine Implementierung unter Linux, damit das Batteriemodell in Echtzeit auf die Verbraucher reagieren kann. Die Implementierung unter Linux benötigt keine zusätzlichen Werkzeuge. Das Modell wird mit 1 bewertet.

Realitätsnähe des Batteriemodells: Das Modell lässt sich gut an verschiedene Batterietechnologien anpassen. Das Entladeverhalten der Batterie wird durch mehrere Batteriezellen sehr genau abgebildet (Fehler max 5%). Es wird

konstante und variable Batteriebelastung modelliert. Das Modell wird mit 85% bewertet.

Parametrisierbarkeit des Batteriemodells: Das Modell ist gut an verschiedene Batterietechnologien anpassbar. Der Recovery-Effekt wird analog zu dem Modell von Panigrahi über eine Funktion modelliert. Das Modell wird mit einer 1 bewertet.

3.5 Elektrochemische Batteriemodelle am Beispiel vom Modell von Gu u.a.

Elektrochemische Batteriemodelle bilden die letzte Klasse der Batteriemodelle. Diese Modelle lassen sich im Bereich der Low-Level-Modelle einordnen und beziehen sich sehr detailgenau auf einzelne Batterietechnologien und sind sehr aufwendig. Elektrochemische Modelle bilden die elektrochemische Raktion innerhalb einer Batterie möglichst genau ab. Die Simulation eines elektrochemischen Modells benötigt mehrere Stunden. Stellvertretend für die verschiedenen Modelle wird im Folgenden kurz das Modell von Gu und Wang vorgestellt.

Das Modell ist eine Kopplung von einem elektrochemischen und einem thermalen Modell. Das Modell verfügt über 21 Parameter, welche die genauen Eigenschaften (Diffusionskoeffizienten, Massentransferkoeffizienten, geometrische Parameter...) der Batterie widerspiegeln. Das Modell besteht aus einer Reihe von Differentialgleichungen. Für die Batteriezellentemperatur ergibt sich beispielsweise folgende Gleichung:

$$< q >= -\frac{1}{V_c}\int_{V_c}\sum_j a_{sj}\bar{i}_{nj}\left(U_j - T\frac{\partial U_j}{\partial T}\right)dv - IV + \frac{1}{V_c}\int_{V_c}\left(\Delta h^*_{hyd}a_{s3}\bar{i}_{n3}\right)dv$$

Die vollständige Herleitung des Modells kann [GW00] entnommen werden.

Unter Berücksichtigung der Bewertungskriterien ergibt sich folgende Situation:
Anzahl der modellierbaren Batterieeffekte: Das Modell unterstützt alle Batterieeffekte.

Umsetzbarkeit des Batteriemodells zur Implementierung unter Linux: Die Berechnungkomplexität des Modells ist sehr hoch. Eine Implementierung unter Linux für den Einsatz in Echtzeit ist auf Grund der Komplexität nicht möglich. Das Modell wird mit 4 bewertet und wurde auf Grund der Komplexität nicht näher betrachtet.

Realitätsnähe des Batteriemodells: Das Modell bietet die höchste Realitätsnähe für einen ganz bestimmen Batterietyp. Es wird konstante und variable Strombelastung modelliert. Das Modell wird mit 99,9% bewertet.

Parametrisierbarkeit des Batteriemodells: Das Modell ist für einen be-

stimmen Batterietyp entworfen und modelliert diesen möglichst genau. Es existieren keine Freiheitsgrade in Bezug auf andere Batterietechnologien. Das Modell wird mit 4 bewertet.

3.6 Fazit

Die folgende Tabelle fasst noch einmal die Ergebnisse zusammen:

Batteriemodell	Effekte	Umsetzbarkeit	Realitätsnähe	Parametr.
Peukert	Rate-Cap.-Effekt	1	30%	1
Rakhmatov und Vrundhula	Rate-Cap.-Effekt Recovery-Effekt	3	85%	1
Hageman	Rate-Cap.-Effekt	4	75%	3
Panigrahi	Rate-Cap.-Effekt Recovery-Effekt	1	85%	1
Sarkar	Rate-Cap.-Effekt Recovery-Effekt	1	85%	1
Gu und Wang	Rate-Cap.-Effekt Recovery-Effekt Thermale-Effekte	4	99,9%	4

Tabelle 1: Aufstellung der Ergebnisse

Das Modell von Peukert bietet eine einfache Möglichkeit eine Batterie zu modellieren. Leider hat das Modell große Defizite in der Realitätsnähe und modelliert nur den Rate-Capacity-Effekt und konstante Strombelastungen. Das Modell von Rakmahtov und Vrundhula bietet im Gegensatz dazu eine viel bessere Realitätsnähe, allerdings unter der Voraussetzung, dass das Entladeprofil für die zu modellierende Batterie im Voraus bekannt ist. Eine Implementierung unter Linux für die Modellierung des Batterieverhaltens in Echtzeit gestaltet sich unter diesen Umständen als schwierig. Das Modell von Hageman ist nur für einen bestimmten Batterietyp geeignet und setzt die Verfügbarkeit von einem (P)SPICE-Simulator voraus, welcher im Linux Cluster *NET* nicht zur Verfügung steht. Damit scheidet dieses Modell für eine Implementierung im Rahmen dieser Studienarbeit aus. Das Modell von Gu und Wang scheidet für eine Implementierung im Rahmen dieser Studienarbeit auch aus, da die Komplexität dieses Batteriemodells für die Emulation einer Batterie zu hoch ist. Das Modell bietet allerdings für einen festgelegten Batterietyp die größtmögliche Realitätsnähe und eignet sich vor allem zur Simulation. Die Modelle von Panigrahi und Sarkar bieten eine sehr gute Realitätsnähe in Zusammenspiel mit einer sehr guten Umsatzbarkeit für die Echtzeitanforderungen zur Implementierung des Modells unter Linux. Die wichtigsten Batterieeffekte werden von den Modellen modelliert und die Modelle sind nicht auf eine bestimmte Batterietechnologie festgelegt.

4 Energieverbrauchermodelle

In diesem Kapitel wird der Energieverbrauch unterschiedlicher Komponenten näher untersucht. Der Energieverbrauch in mobilen Geräten ist ein kritischer Faktor, da der Fortschritt der Batterietechnologie nur sehr langsam verläuft. Der Energieverbrauch eines Systems und seiner einzelnen Komponenten spielt mehr und mehr eine wichtige Rolle im Systementwurf. Der Verbrauch an Energie lässt sich nur sehr schwierig berechnen, da Energieverbrauch in erster Linie nur gemessen werden kann. Über Messexperimente kann ein Modell zu einem Verbraucher erstellt werden, welches allerdings nur für diesen speziellen Verbraucher korrekt ist. In [FS99] wird ein Werkzeug vorgestellt, mit dem der Energieverbrauch eines Systems analysiert werden kann. Für einzelne Verbraucher wie beispielsweise eine CPU existieren mittlerweile unterschiedliche Modelle wie Instruction Level Modeling [TMW+96] mit denen der Verbrauch berechnet werden kann. Die im Folgenden betrachteten Modelle untersuchen den Energieverbrauch für 802.11 Netzwerkkarten, da diese in mobilen Geräten eine zunehmend wichtige Rolle einnehmen. In mobilen Geräten machen die Energiekosten für Kommunikation bis zu 50% des Gesamtverbrauchs aus.

4.1 Energieverbrauch einer 802.11 Netzwerkkarte

Im Folgenden werden verschiedene Modelle zur Ermittelung des Energiebedarfs von 802.11 Netzwerkkarten untersucht. Der Energieverbrauch einer 802.11 Netzwerkkarte wird von unterschiedlichen Parametern beeinflusst. Je nach verwendetem Kartenmodell ändert sich der Energiebedarf erheblich. In IEEE 802.11 sind zwei unterschiedliche Betriebsmodi spezifiziert: Infrastrukturmodus und Ad-Hoc Modus. Diese beiden Modi unterscheiden sich hinsichtlich des Energiebedarfs. Im Infrastrukturmodus realisiert ein Access Point zentrale Funktionen des Netzwerks wie z.B. Uhrensynchronisation oder Pufferung von Paketen. Da sich die einzelnen Teilnehmer auf die feste Infrastruktur verlassen können, können einzelne Teilnehmer problemlos für eine gewisse Zeit in den stromsparenden Sleep-Modus umschalten um Energie zu sparen. Da die Teilnehmer eines 802.11 Netzwerks sich möglicherweise nicht alle im gleichen Empfangsbereich befinden, gibt es unter anderem Probleme mit der Kollisionserkennung und das 802.3 Protokoll CSMA/CD kann nicht eingesetzt werden. Als Zugriffsverfahren auf das Medium wurden in 802.11 folgende Verfahren definiert:

- CSMA/CA
- CSMA/CA mit RTS/CTS
- Point Coordinaiton Function (PCF)

Nur das erste Verfahren ist verbindlich für 802.11 Geräte, die anderen zwei Verfahren sind optional. Je nach eingesetztem Zugriffsverfahren variiert der Energieverbrauch, da z.B. mehr Kontrollnachrichten verschickt werden müssen. Im

Ad-Hoc-Modus gibt es keine feste Infrastruktur und die Geräte können sich nicht darauf verlassen, dass sie nicht Pakete verlieren, wenn in einen Energiesparmodus gewechselt wird.
Eine 802.11 Netzwerkkarte hat vier verschiedene Operationsmodi:

- Transmit (versenden von Daten)
- Receive (empfangen von Daten)
- Idle (warten auf Kommunikation)
- Sleep (energiesparender Modus)

Der Energieverbrauch ist je nach Modus unterschiedlich. Der Transmit-Modus ist der Modus mit dem höchsten Energiebedarf, da Daten an andere Teilnehmer mit einer gewissen Sendestärke gesendet werden müssen. Weiterhin ist die verwendete Sendestärke (RF) für den Energieverbrauch von großer Bedeutung. In [EBW02] wurde experimentell festgestellt, dass eine Erhöhung der Sendestärke von 1mW auf 50 mW den Energieverbrauch um ungefähr 500mW erhöht. Der Receive- und Idle-Modus sind vom Energieverbrauch her sehr ähnlich. Der Receive-Modus braucht etwas mehr Energie als der Idle-Modus, da die empfangenen Rahmen, welche an die Netzwerkkarte adressiert sind, an die nächste Schicht weitergereicht werden müssen. Im Sleep-Modus werden die Empfangs- und Sendeteile abgeschalten und daher ist hier der Energieverbrauch am geringsten, allerdings ist in diesem Modus keine Kommunikation möglich und es besteht die Gefahr, dass Daten verloren gehen. Die verwendete Übertragungsgeschwindigkeit beeinflusst ebenfalls den Energieverbrauch, allerdings ist der endgültige Verbrauch auch von der Qualität der Verbindung abhängig. Die Paketgröße ist ein weiterer Faktor, welcher den Energiebedarf beeinflusst. Je nach Situation ist es effizienter kurze oder lange Pakete zu verschicken. Kurze Pakete bedeuten mehr Aufwand, da mehrere Sendevorgänge benötigt werden, allerdings kann das unter Umständen günstiger sein, falls die Fehlerrate im Medium, bedingt durch Störungen, sehr hoch ist. In einem Medium mit geringer Fehlerrate sind größere Pakete zu bevorzugen. Die Verwendung unterschiedlicher Transportprotokolle, wie z.B. TCP oder UDP, trägt ebenfalls zum Energiebedarf bei. TCP baut eine zuverlässige Verbindung auf, benötigt allerdings mehrere kleinere Kontrollnachrichten, welche den Energieverbrauch erhöhen. Wiederholtes Senden im Fehlerfall und Broadcast im Gegensatz zu Punkt-zu-Punkt-Kommunikation haben unterschiedliche Energieanforderungen. In den folgenden zwei Teilkapiteln werden zwei experimentelle Modelle für den Energieverbrauch von 802.11 Netzwerkkarten vorgestellt. Ein theoretisches Modell für den Energieverbrauch kann unter [CSA+98] nachgeschlagen werden.

4.2 Modell von Feeney und Nilsson

In [Fee01] und [FN01] wird der Energieverbrauch über eine Serie von Experimenten ermittelt. Mit Hilfe der Messergebnisse wird der Energieverbrauch durch

lineare Gleichungen modelliert. Der Vorteil ist, dass von vielen, oben beschriebenen, Effekten abstrahiert werden kann und der Energiebedarf in Relation zu der Paketgröße berechnet werden kann. Die Ergebnisse dieser Experimente sind natürlich beeinflusst von der verwendeten Hardware und den Messbedingungen, allerdings können trotzdem wichtige Erkenntnisse über den Energieverbrauch gewonnen werden. Im Folgenden wird nur der Energieverbrauch von 802.11 Netzwerkkarten betrachtet, welche sich im Ad-Hoc-Betriebsmodus befinden. Es wird hierbei angenommen, dass sich die 802.11 Netzwerkkarte die meiste Zeit im Idle-Zustand befindet, da keine Basisstation vorhanden ist, welche die Kommunikation regelt. Routing wurde in dem Netzwerk nicht untersucht (Single Hop Netzwerk).

Der Energieverbrauch wird als lineare Gleichung pro Paket modelliert.

$$Energie = m \times size + b$$

Die Gleichung kann für unterschiedliche Fälle angepasst werden: Empfang von Paketen, Versenden von Paketen, Wegwerfen von Paketen und promiskuitiver Empfang von Paketen. Die Koeffizienten m und b werden über die Experimente ermittelt, wobei b einen festen und m den inkrementellen Energiebedarf repräsentiert. In dem festen Anteil b sind unter anderem der Energieverbrauch für Rahmenheader, Kontrollnachrichten und Prüfbits eingerechnet. Dieses paketorientierte Modell betrachtet folgende Effekte nicht:

- Der Energieverbrauch durch einen fehlgeschlagenen Versuch, den Kanal zu belegen, da ein anderer Teilnehmer schon den Kanal belegt hat.
- Der Energieverbrauch durch verloren gegangene Pakete.
- Der Energieverbrauch durch Bitfehler während der Übertragung.
- Der Energieverbrauch durch Verbindungsverluste.
- Der Energieverbrauch durch unterschiedliche Sendestärken.
- Der Energieverbrauch durch Link-Layer-Fragmentation.

Diese Fälle wurden nicht modelliert, da sie schwer in kontrollierten Experimenten reproduzierbar sind. Da sie dennoch den Energieverbrauch einer 802.11 Netzwerkkarte mit beeinflussen, kann der Energieverbrauch probabilistisch modelliert werden, siehe [CSA+98].

4.2.1 Messumgebung

Die Experimente wurden auf einem IBM Thinkpad 560 mit FreeBSD 4.0 als Betriebssystem durchgeführt. Die Testnetzwerkkarten waren Lucent IEEE 802.11 WaveLAN PC „Bronze“ (2Mbps) und „Silver“ (11Mbps). Die Energiesparmodi des Notebook wurden deaktiviert und es wurde versucht, die Experimente vor

Interferenzen durch andere Geräte zu schützen. Der Energieverbrauch wurde mit Hilfe eines kleinen Widerstandes mit 0.5Ω gemessen, welcher zwischen Notebook und Netzwerkkarte eingefügt wurde. Der Strom wurde mit Hilfe eines digitalen Oszilloskops, welches die Spannung über den Widerstand gemessen hat, berechnet. Details zum Versuchsaufbau sind in [FN01] und [GHS96]. Als Traffic wurden UDP-Pakete verschickt.

4.2.2 Broadcast-Kommunikation

Die Broadcast-Kommunikation wird über die folgenden Gleichungen modelliert.

$$E_{broadcast-send} = m_{send} \times size + b_{send(bcast)}$$

$$E_{broadcast-recv} = m_{recv} \times size + b_{recv(bcast)}$$

Die experimentell ermittelten Werte für die Koeffizienten sind in den Tabellen 2 und 3 eingetragen. Broadcast-Kommunikation benötigt i.d.R. mehr Energie als Punkt-zu-Punkt-Kommunikation, da aus Kompatibilitätsgründen mit der langsamsten Übertragungsrate kommuniziert wird, da sonst eventuell nicht alle Empfänger die Daten empfangen können. Durch die geringere Geschwindigkeit befindet sich die Netzwerkkarte länger im Transmit-Modus und benötigt daher mehr Energie.

4.2.3 Punkt-zu-Punkt-Kommunikation

Die Punkt-zu-Punkt-Kommunikation wird getrennt betrachtet, da sich in den Experimenten gezeigt hat, dass der Energieverbrauch für die Punkt-zu-Punkt-Kommunikation im Gegensatz zur Broadcast-Kommunikation je nach Übertragungsrate unterschiedlich ist.
Im Fall von 2 Mbit als Übertragungsrate ist der Energiebedarf für Punkt-zu-Punkt-Kommunikation höher als bei Broadcast-Kommunikation. Der Grund für den höheren Energiebedarf liegt darin, dass der Sender zuerst eine RTS (request to send) Kontrollnachricht als Broadcast verschickt und danach auf die Antwort CTS (clear to send) vom Empfänger wartet. Dieser höhere Aufwand spiegelt sich in den Koeffizienten $b_{send(p2p)}$ und $b_{recv(p2p)}$ wider.

$$E_{p2p-send} = m_{send} \times size + b_{send(p2p)}$$

$$E_{p2p-recv} = m_{recv} \times size + b_{recv(p2p)}$$

Im Fall von 11 Mbit als Übertragungsrate ist der Energiebedarf für Punkt-zu-Punkt-Kommunikation i.d.R. niedriger als für Broadcast-Kommunikation, da der Wert für m_{send} im Fall von 11Mbit als Übertragungsrate viel niedriger ist als bei 2 MBit, vgl. Tabelle 2 und 3.

	$\mu W \cdot sec/byte$ $\mu W \cdot sec$
point-to-point send (a)	$1.9 \times size + 454$
broadcast send (b)	$1.9 \times size + 265$
point-to-point recv (c)	$0.50 \times size + 356$
broadcast recv (d)	$0.50 \times size + 56$
	non-destination $n \in S, D$
promiscuous recv (e)	$0.39 \times size + 140$
discard (f)	$-0.61 \times size + 70$
	non-destination $n \in S, n \notin D$
promiscuous recv (g)	$0.54 \times size + 66$
discard (h)	$-0.58 \times size + 24$
	non-destication $n \notin S, n \in D$
promiscuous recv (i)	$0.0 \times size + 63$
discard (j)	$0 \times size + 56$
idle (ad hoc)(k)	$843mW$
idle (BSS)	$66mW$

Tabelle 2: Messergebnisse (2Mbps) für die verschiedenen Koeffizienten aus [FN01]

4.2.4 Verwerfen von Paketen

Die Teilnehmer im Ad-Hoc-Netzwerk warten im Idle-Zustand auf Kommunikation mit den anderen Teilnehmern. Falls ein Paket bei einem Teilnehmer ankommt und es nicht für ihn bestimmt ist, dann kann das Paket weggeworfen werden. Allerdings kostet die Überprüfung, ob das Paket an den Teilnehmer adressiert ist, auch Energie. Im Folgenden werden drei Fälle unterschieden:

- Der Teilnehmer ist in Reichweite von Sender und Empfänger.
- Der Teilnehmer ist nur in Reichweite des Senders.
- Der Teilnehmer ist nur in Reichweite des Empfängers.

Im Fall, dass der Teilnehmer in der Reichweite des Senders und Empfängers ist, wird der Energieverbrauch mit folgender Formel modelliert:

$$E = m_{disc} \times size + b_{non-dest(S,D)}$$

m_{disc} kann in diesem Fall unterschiedliche Werte annehmen. Für $m_{disc} \in (0, m_{recv}]$ wird vor der Entscheidung das Paket wegzuwerfen, ein Teil oder das ganze Paket von dem Teilnehmer empfangen. Für $m_{disc} = 0$ wird das Paket von dem Teilnehmer ignoriert und er verbleibt im Idle-Zustand. Für $m_{disc} < 0$ wird weniger Energie verbraucht als im Idle-Zustand, da der Teilnehmer, für die Dauer der Kommunikation zwischen Sender und Empfänger, in einen Energiesparzustand (Sleep) wechselt.

Der zweite Fall, dass der Teilnehmer nur in Reichweite des Senders ist, unterscheidet sich nur in dem Fixkostenanteil b, da hier vom Teilnehmer unter anderem nur die RTS-Kontrollnachricht mit empfangen wird.

$$E = m_{disc} \times size + b_{non-dest(S,\not{D})}$$

Im letzten Fall ist die verbrauchte Energie eines Teilnehmers nur in Reichweite des Empfängers:

$$E = m_{none} \times size + b_{non-dest(\not{S},D)}$$

Der Wert von m_{none} ist 0, da der Teilnehmer nicht in der Reichweite des Sender ist. Da allerdings die CTS-Kontrollnachricht empfangen wird, sind prinzipiell auch Werte $m_{none} < 0$ möglich. Dies wäre der Fall, wenn kein anderer Sender in der Reichweite des Teilnehmers ist und der Teilnehmer in einen Energiesparzustand wechselt.

4.2.5 Promiskuitiver Betriebsmodus

Ein Teilnehmer im Ad-Hoc Netzwerk kann sich auch im promiskuitiven Betriebsmodus befinden, d.h. er empfängt jede Punkt-zu-Punkt-Kommunikation, welche in seinem Empfangsbereich gesendet wird. Die empfangenen Pakete werden ohne Prüfung an die nächste Schicht weitergegeben. Es gibt analog zum Verwerfen von Paketen wieder drei Fälle, falls der Teilnehmer nicht der eigentliche Empfänger von dem Paket ist. Die Gleichungen für den Energieverbrauch sind wie folgt:

$$E = m_{recv} \times size + b_{non-dest(S,D)}$$
$$E = m_{recv} \times size + b_{non-dest(S,\not{D})}$$
$$E = m_{none} \times size + b_{non-dest(\not{S},D)}$$

4.2.6 Messergebnisse

Die Tabellen 2 und 3 enthalten die Messergebnisse mit einer Lucent 802.11 WaveLan PC Card mit 2 Mbps und 11 Mbps als Übertragungsgeschwindigkeit. Es wurden auch Experimente mit einer Übertragungsgeschwindigkeit von 11Mbps durchgeführt, wobei sich gezeigt hat, dass der Energieverbrauch nicht linear mit der Übertragungsgeschwindigkeit sinkt. Das hat mehrere Gründe:

- Damit sich Netzwerkkarten mit unterschiedlichen Geschwindigkeiten im gleichen Netzwerk zusammenarbeiten können, werden RTS- und CTS-Kontrollnachrichten mit der langsameren Übertragungsgeschwindigkeit übertragen, d.h. die fixen Energiekosten b sind fast identisch bei 11Mbps.
- Broadcastpakete können in einem gemischten Netzwerk auch nur mit der langsameren Geschwindigkeit verschickt werden.
- Die schnellere Kommunikationsgeschwindigkeit macht sich in kleineren Werten für die Koeffizienten m bemerkbar. Weiterhin wurde festgestellt, dass im Gegensatz zu 2 Mbps Karten die 11Mbps Karten beim Verwerfen von Paketen nicht in einen Energiesparzustand wechseln, d.h. $m \geq 0$.

	$\mu W \cdot sec/byte\ \mu W \cdot sec$
point-to-point send (a)	$0.48 \times size + 431$
broadcast send (b)	$2.1 \times size + 272$
point-to-point recv (c)	$0.12 \times size + 316$
broadcast recv (d)	$0.26 \times size + 50$
	non-destination $n \in S, D$
promiscuous recv (e)	$0.14 \times size + 97$
discard (f)	$0.11 \times size + 66$
	non-destination $n \in S, n \notin D$
promiscuous recv (g)	$0.10 \times size + 70$
discard (h)	$0.11 \times size + 42$
	non-destication $n \notin S, n \in D$
promiscuous recv (i)	$0.0 \times size + 32$
discard (j)	$0 \times size + 38$
idle (ad hoc)(k)	$741mW$
idle (BSS)	$48mW$

Tabelle 3: Messergebnisse (2Mbps) für die verschiedenen Koeffizienten aus [FN01]

4.3 Messexperimente von Ebert, Burns und Wolisz

In [EBW02] wird ebenfalls, wie bei Feeney und Nilsson, der Energieverbrauch pro übertragenem Bit mit Hilfe von Messexperimenten ermittelt. Der Versuchsaufbau ist ähnlich zu dem Aufbau bei [FN01] und besondere Aufmerksamkeit wird auf die Signalstärke RF und verschiedene Übertragungsgeschwindigkeiten gelegt. Die Messexperimente wurden mit 802.11b Netzwerkkarten durchgeführt, welche ebenfalls wie bei [FN01] im Ad-Hoc-Modus betrieben wurden. Das Modell betrachtet den Energieverbrauch (pro übertragenem Bit) in Abhängigkeit von:

- der Übertragungsgeschwindigkeit (1-11 Mbps)
- dem Zustand der Netzwerkkarte (receive, transmit)
- der Sendestärke RF (1, 5, 20 und 50mW)

Die folgenden Effekte werden nicht betrachtet:

- Der Energieverbrauch durch einen fehlgeschlagenen Versuch den Kanal zu belegen, da ein anderer Teilnehmer schon den Kanal belegt hat.
- Der Energieverbrauch durch verloren gegangene Pakete.
- Der Energieverbrauch durch Bitfehler während der Übertragung.
- Der Energieverbrauch durch Verbindungsverluste.
- Der Energieverbrauch durch Link-Layer-Fragmentation.

4.3.1 Messumgebung

Die Experimente wurden ähnlich zu den Experimenten in [FN01] durchgeführt. Es wurden zwei Notebooks mit 802.11b Netzwerkkarten ausgestattet. Beide Notebooks sind mit einem PC verbunden, welcher die beiden Laptops steuert, z.B. Traffic erzeugt. Der Stromverbrauch wird wie in [FN01] über einen kleinen Widerstand zwischen Netzwerkkarte und Notebook gemessen, siehe Abbildungen 10 und 11. Als Traffic wurden UDP Pakete verschickt. Der Abstand zwischen

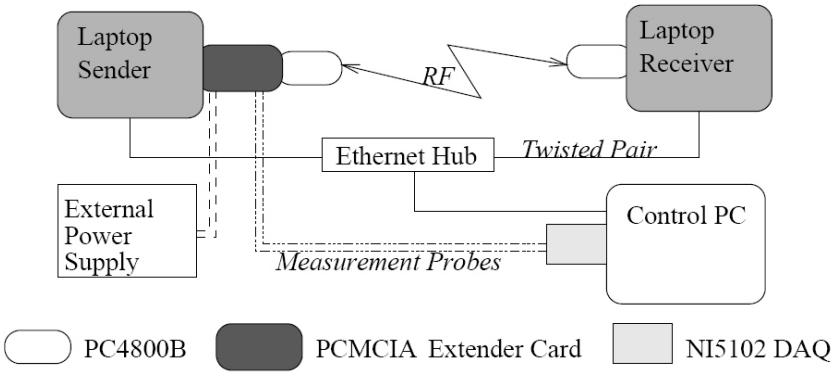

Abbildung 10: Versuchsaufbau Teil 1 in [EBW02]

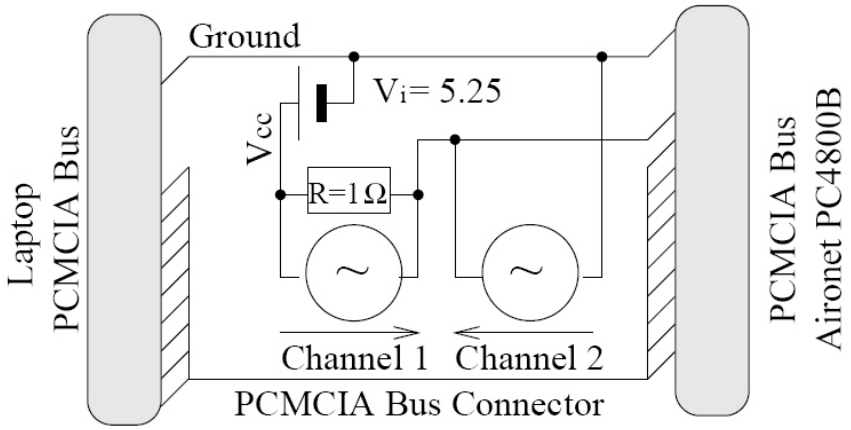

Abbildung 11: Versuchsaufbau Teil 2 in [EBW02]

den zwei Notebooks beträgt fünf Meter, es wurden keine Sendewiederholungen erlaubt und die RTS- und CTS-Kontrollnachrichten wurden deaktiviert.

4.3.2 Messergebnisse

Ein Ziel der Experimente war die Untersuchung, inwiefern sich die Sendestärke beim Verschicken von Paketen auf den Energieverbrauch auswirkt. In Abbildung 12 wird die Abhängigkeit zwischen Übertragungsrate und Sendestärke dargestellt. Es fällt auf, dass die Energie in diesem Fall weniger von der Übert-

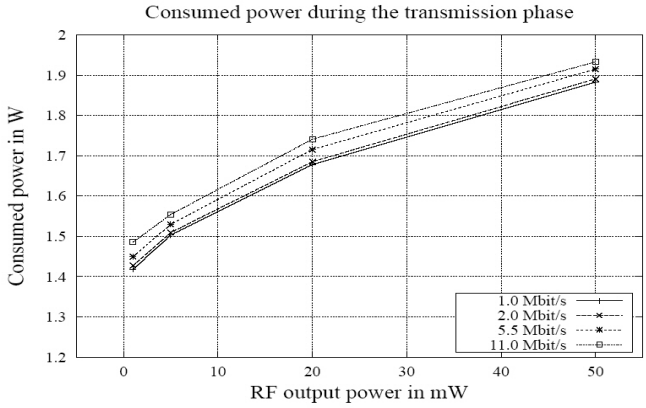

Abbildung 12: Zusammenhang zwischen Energieverbrauch, Übertragungsrate und Sendestärke in [EBW02]

ragungsgeschwindigkeit und mehr von der Sendestärke beeinflusst wird. Eine Erhöhung der Sendestärke von 1mW um 49mW auf 50 mW verursacht einen ungefähr 500mW höheren Energiebedarf. Weiterhin wurde untersucht, inwiefern sich die Paketlänge, Sendestärke und Übertragungsrate auf das Versenden und Empfangen von Paketen auswirkt. Je größer das zu versendende Paket ist, desto größer ist auch der Energiebedarf. Die Experimente haben weiterhin ergeben, dass möglichst mit der geringsten Sendestärke die Pakete verschickt werden sollen und dass der Empfang kleiner Pakete mehr Energie benötigt als der Empfang größerer Pakete, was gegen eine Fragmentierung auf der Mac-Ebene spricht. Der Grund in dem höheren Energieverbrauch für den Empfang kleinerer Pakete liegt darin, dass Acknowlegdments mit höchster Sendestärke verschickt werden. Abschließend wird noch betrachtet, wie viel die Kommunikation eines Bits an Energie kostet.

$$E_{bit_good}[J/Bit] = \frac{Average_Consumed_Power[W]}{Goodput[Bits/s]}$$

Die Gleichung setzt die durchschnittlich verbrauchte Energie (mit allen Acknowledgments, Idle-Zeiten und sonstige Energiekosten) ins Verhältnis zu den übertragenen oder korrekt empfangenen Bits. Siehe Abbildung 13 für E_{bit_good} beim Verschicken von Paketen mit 50mW Sendestärke. Es wird deutlich, dass größere Pakete weniger Energie verbrauchen, falls der Kanal weitgehend fehlerfrei ist.

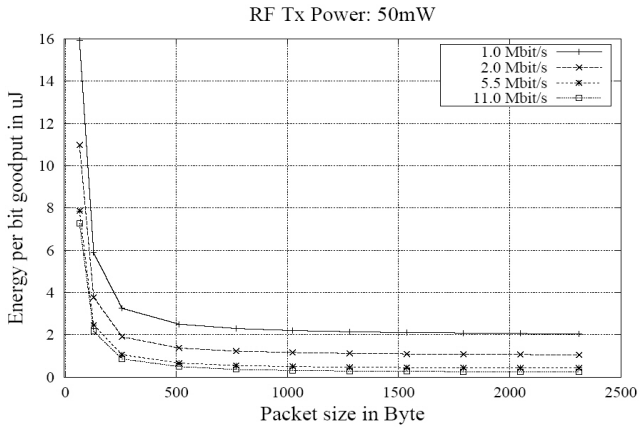

Abbildung 13: Energie pro Bit für verschiedene Übertragungsraten bei 50mW Sendestärke in [EBW02]

4.4 Fazit

Der Energieverbrauch einer 802.11 Netzwerkkarte lässt sich nur schwer berechnen. Die hier vorgestellen Modelle untersuchen den Verbrauch von 802.11 Netzwerkkarten über Messexperimente. Der Energieverbrauch ist größtenteils abhängig von der verwendeten Netzwerkkarte und deren Betriebsmodi. Im Modell von Feeney werden mehrere Formeln für den Energieverbrauch aufgestellt, welche sich sehr flexibel für andere Netzwerkkarten anpassen lassen. Die Messexperimente von Ebert u.a. betrachten zusätzlich den Energeiverbrauch unter verschiedenen Sendestärken. Leider lassen sich nicht alle Effekte, welche den Energieverbrauch einer 802.11 Netzwerkkarte beeinflussen, in den Messexperimenten messen.

5 Realisierung eines Batteriemodells

In diesem Kapitel wird zuerst ein Überblick über die Systemumgebung für die Batterieemulationssoftware gegeben. Danach folgen der Entwurf und die Realisierung der Batterieemulationssoftware.

5.1 Systemumgebung

Die Batterieemulationssoftware wird im Cluster *NET* in der Sprache Java2 SE 1.41 implementiert. Der *NET*-Cluster besteht aus 64 Rechnern mit Linux als Betriebssystem, wobei jeder Rechner für sich einen virtuelle Knoten im emulierten Netzwerk repräsentiert. Die virtuellen Knoten werden durch so genannten virtuelle LANs untereinander vernetzt und bilden das zu emulierende Netzwerk. Einen Überblick über die Architektur der Hard- und Software von *NET* ist in [HM03] verfügbar.

Die entwickelte Batterieemulationssoftware kann auf jedem einzelnen Knoten gestartet werden, falls für diesen Knoten eine Batterie zur Emulation benötigt wird. Die emulierten Batterien der einzelnen Knoten sind unabhängig voneinander konfigurierbar.
Es folgt ein Überblick über die einzelnen Bereiche der Systemumgebung, welche mit der Batterieemulationssoftware in Berührung kommen.

5.1.1 PROC-Filesystem

Das PROC-Filesystem von Linux ist ein virtuelles Dateisystem, welches Informationen über den aktuellen Systemzustand (z.B. der aktuelle Kapazitätsstand der Batterie) durch virtuelle Dateien zur Verfügung stellt. Der Inhalt der virtuellen Dateien wird durch den Kernel in Echtzeit erstellt, wenn ein lesender Zugriff auf die Dateien erfolgt. Der Inhalt der Dateien ist also immer aktuell und spiegelt den momentanen Zustand des Systems (Kernel) wider. In `/proc` befinden sich beispielsweise Informationen über CPU, Speicher und die einzelne Prozesse. Die komplette Struktur von `/proc` kann in [Red01] nachgeschlagen werden.

5.1.2 Netzwerkstatistik

Unter Linux werden detaillierte Statistiken über das Netzwerk und den Traffic unter `/proc/net` gespeichert [Red01]. Informationen über die Netzwerkgeräte des Systems werden in `/proc/net/dev` abgelegt. In `/proc/net/dev` wird für die einzelnen Geräte unter anderem aufgelistet, wie viel Pakete und Bytes jeweils empfangen und verschickt wurden. Die Modellierung eines Verbrauchers für die emulierte Batterie wird sich auf die Informationen aus dieser Statistikdatei stützen.

5.1.3 Batterieschnittstelle

Damit die Implementierung des Batteriemodells sich für andere Anwendungen wie eine physisch vorhandene Batterie präsentiert, muss das Batteriemodell die Batterieschnittstelle von Linux mit den entsprechenden Werten ansteuern. Linux unterstützt auf x86er Systemen zwei unterschiedliche Power-Management-Technologien: APM (Advanced Power Management) und ACPI (Advanced Configuration and Power Interface), welche nicht gleichzeitig benutzt werden können. Jede dieser Technologien besitzt eine Batterieschnittstelle, welche die Informationen einer (eventuell) physisch vorhandenen Batterie auswerten und anderen Anwendungen zur Verfügung stellen.

- **APM** ist die ältere der beiden Technologien und wurde von Intel und Microsoft entwickelt [Apm01]. Bei APM wird das Power-Management durch das Mainboard BIOS kontrolliert. Unter Linux ist APM ausgereifter, bietet aber weniger Flexibilität als ACPI. In APM werden unterschiedliche Energiezustände (für einzelne Geräte oder das Gesamtsystem) definiert:

  - Ready: In diesem Zustand ist das System oder Gerät voll betriebsbereit.
  - Stand-by: Das System oder Gerät geht in einen energiesparenden Zustand über, bis z.B. ein Gerät einen Interrupt auslöst.
  - Suspended: Das System fährt in einen hohen Energiesparmodus und unterbricht die Arbeit.
  - Hibernation: Hier wird der komplette Systemzustand gesichert (auf persistenten Speicher) und das System abgeschaltet.
  - Off: Das System oder Gerät wird komplett abgeschaltet (keine Sicherung von Daten oder Zuständen).

  Mit dem Kommandozeilenprogramm „apm“ kann unter anderem der aktuelle Batteriezustand ausgegeben werden. Die aktuellen Daten werden im virtuellen Dateisystem PROC unter `/proc/apm` gespeichert. Die Datei `/proc/apm` hat beispielsweise folgenden Inhalt:

  ```
  1.16 1.2 0x03 0x00 0x00 0x01 99% 1792 min
  ```

  Das Programm apm gibt dann folgende Ausgabe aus:

  ```
  APM Bios 1.2 (kernel driver 1.16)
  AC offline, battery status high: 99% (1 day, 5:52).
  ```

  Der Aufbau der Datei `/proc/apm` kann unter [Rot01] eingesehen werden.

- **ACPI** wurde in Zusammenarbeit von Intel, Microsoft und Toshiba entwickelt [Acp]. Im Gegensatz zu APM besitzt bei ACPI das Betriebssystem die Kontrolle über das Power-Management. Die Implementierung unter

Linux ist im Vergleich zu APM noch nicht in gleicher Weise ausgereift und wird daher noch seltener eingesetzt. In ACPI werden unterschiedliche Energiezustände definiert [Hog03]:

- S0=On
- S1-3=Sleeping (verschieden Schlafzustände)
- S4=Soft Off
- S5=Off (mechanisch ausgeschalten)

Mit dem Programm „acpi“ kann das Power-Management kontrolliert und überwacht werden. Informationen über die Batterie (Kapazität, Spannung, Batterietyp und Restlaufzeit) wird in
`/proc/acpi/battery/BAT0/state` und
`/proc/acpi/battery/BAT0/info` gespeichert.

Die Datei `/proc/acpi/batter/Bat0/state` hat beispielsweise folgenden Inhalt:

```
present: yes
capacity state: ok
charging state: discharging
present rate: unknown
remaining capacity: 3840 mAh
present voltage: 14800 mV
```

5.2 Systementwurf

In diesem Teilkapitel wird der Systementwurf für die Batterieemulation beschrieben.

Die folgende Abbildung 14 gibt einen Überblick über die entworfene Software.

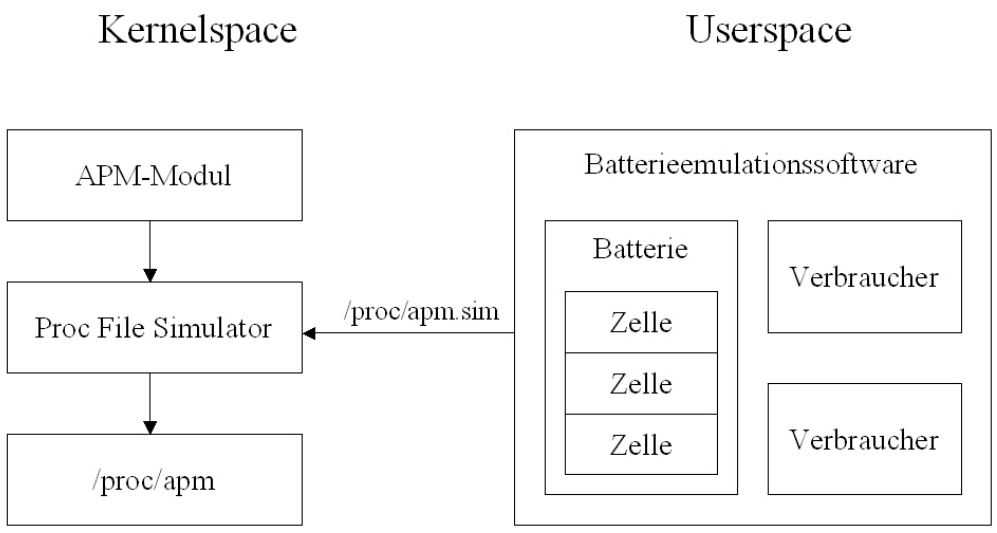

Abbildung 14: Programmübersicht

Die Batterieemulationssoftware läuft im Userspace. Damit die Batterieschnittstelle von der Batterieemulationssoftware angesteuert werden kann, wird eine zusätzliche Softwarekomponente (Proc File Simulator) im Kernelspace benötigt, siehe Kapitel 5.2.2 Integration der Batteriesoftware in Linux.

Die zu entwerfende Batterieemulationssoftware besteht aus zwei Teilen:

- Die emulierte Batterie
- Die Verbraucher, welche die Batterie belasten

Die beiden Teile werden im Folgenden genauer vorgestellt.

5.2.1 Batterieentwurf

Für die zu entwerfende Software habe ich mich für eine Implementierung des Batteriemodells von Panigrahi entschieden, da dieses Modell sehr flexibel ist und sich auf Grund der Bewertungskriterien aus Kapitel 3 sehr gut für eine Implementierung eignet. Das Batteriemodell von Sarkar erweitert das Modell

von Panigrahi am verschiedenen Stellen und einige dieser Erweiterungen fließen in die Implementierung mit ein. Eine Batterie besteht nach dem Modell von Sarkar aus mehreren Batteriezellen, welche folgenden Aufbau haben (angelehnt an UML-Diagrammsyntax), siehe Tabelle 4.

Batteriezelle
theoretische Kapazität normale Kapazität aktuelle Kapazität
berechne_Kapazität()

Tabelle 4: Aufbau einer Batteriezelle

Jede Batteriezelle hat eine theoretische und eine normale Kapazität. Die theoretische Kapazität stellt die maximal verfügbare Kapazität unter Berücksichtigung von Recoveryeffekten dar. Die normale Kapazität ist die unter normalen Belastungen (konstante Belastung ohne Rate-Capacity-Effekt) verfügbare Kapazität. Die aktuelle Kapazität einer Batteriezelle spiegelt den aktuellen Kapazitätsstand der Batteriezelle wider. Theoretische und normale Kapazität sind vom Benutzer festgelegte Fixwerte. Die aktuelle Kapazität wird über die Methode `berechne_Kapazität()` berechnet. Der Kapazitätsstand einer Batteriezelle ist unabhängig von den anderen Zellen. Die Berechnung der aktuellen Kapazität schließt den Rate-Capacity- und Recovery-Effekt mit ein.

Die folgende Batterieklasse fasst mehrere Batteriezellen zu einer Batterie zusammen, siehe Tabelle 5. Die Aufgabe der Batterie ist es, die verschiedenen Batteriezellen zu instanziieren und anzusteuern.

Die Gesamtkapazität errechnet sich aus der Summe der einzelnen Batteriezellkapazitäten. In der Rate-Effekt-Lookup-Tabelle werden die vom Benutzer definierten Werte für den Rate-Capacity-Effekt gespeichert. Der Rate-Capacity-Effekt tritt dann in Erscheinung, wenn die Batterie in einem Takt zu stark belastet wird, d.h. die angeforderte Energie liegt über dem Grenzwert für den Rate-Capacity-Effekt. Die Rate-Capacity-Effekt-Lookup-Tabelle besteht aus mehreren 2-Tupeln, welche aus einem Schwellwert und einem Multiplikationsfaktor bestehen. Liegt die angeforderte Kapazität beispielsweise zwischen dem ersten und zweiten Schwellwert aus der Lookup-Tabelle, dann wird die angeforderte Energie mit dem Multiplikationsfaktor des ersten Schwellwerts multipliziert und an die zu belastende Batteriezelle übergeben. Die Auswirkung des Rate-Capacity-Effekts ist, dass in der Batterie mehr Energie verbraucht wird, als angefordert wurde. In der Recovery-Effekt-Lookup-Tabelle werden die Reco-

Batterie
Batteriezellen Gesamtapazität Rate-Effekt-Lookup-Tabelle Recovery-Effekt-Lookup-Tabelle angeforderte Energie
initialisiere_Batterie() berechne_Kapazität() übermittle_Verbrauch() angeforderte_Energie() schreibe_Batterieschnittstelle() schreibe_Statistik()

Tabelle 5: Aufbau der Batterie

verywahrscheinlichkeiten der verschiedenen Recoveryphasen gespeichert. Eine Batteriezelle hat wie in Kapitel 3.4.1 und 2.4.2 beschrieben mehrere Recoveryphasen und je nach Recoveryphase besteht eine bestimmte Wahrscheinlichkeit für den Recoveryeffekt. Zur einfacheren Konfiguration des Recovery-Effekts kann die Recoverywahrscheinlichkeit einer Phase direkt in Prozent angegeben werden. In der Variable angeforderte Energie wird der Energieverbrauch der verschiedenen Verbraucher pro Takt gesammelt. Der Zugriff auf diese Variable erfolgt synchronisiert.
Das benutzte Batteriemodell sieht vor, dass die Batterie getaktet wird und die einzelnen Batteriezellen werden der Reihe nach belastet (pro Takt eine Batteriezelle). In jedem Takt wird der aktuelle Batteriestand berechnet (Summe der einzelnen Zellkapazitäten).
Die Batterie hat über die Methode `übermittle_Verbrauch()` eine Schnittstelle nach außen zu den Verbrauchern. Pro Takt wird die von den Verbrauchern angeforderte Energie in der Batterie gesammelt und dann über die Methode `angeforderte_Energie()` an die Batteriezelle übermittelt, welche an der Reihe ist. Falls diese Batteriezelle bereits erschöpft ist, dann ist die nächste, noch nicht erschöpfte, Batteriezelle zu belasten. Die Aufgabe den Verbrauch pro Takt abzuholen und der nächsten Batteriezelle zuzuführen übernimmt die Methode `berechne_Kapazität()`.
Bei der Realisierung der Methoden `übermittle_Verbrauch()` und `angeforderte_Energie()` ist darauf zu achten, dass die Methoden nicht gleichzeitig gestartet werden. Das Hauptprogramm hat die Aufgabe die Batterie zu

initialisieren und zu takten. Auf einem Linux-basierten Java-System kann nicht schneller als $10ms$ getaktet werden. Die Batterie läuft als Thread parallel zu den Verbrauchern.

5.2.2 Integration der Batteriesoftware in Linux

Die im Rahmen dieser Studienarbeit entwickelte Batterieemulationssoftware verwendet die APM-Batterieschnittstelle, da im *NET*-Cluster APM installiert ist.
Prinzipiell ist eine Portierung der Software für die ACPI-Batterieschnittstelle möglich, allerdings ist die aktuelle Integration von ACPI unter Linux (Kernel 2.4.22) noch unausgereift. Das Schreiben in die Datei `/proc/apm` ist für das Batterieemulationsprogramm direkt nicht möglich, da das APM-Modul des Kernels diese Datei bei Bedarf erstellt. Damit das Batterieemulationsprogramm trotzdem die Batterieschnittstelle ansteuern kann, muss zusätzlich zu dem Programm auch das Kernelmodule Proc File Simulator gestartet werden. Dieses Modul leitet die ursprüngliche Ausgabe des APM-Moduls um und schreibt die Ausgabe des Batterieemulationsprogramm in die Datei `/proc/apm`.
Nach jeder Gesamtkapazitätsberechnung wird über die Methode `schreibe_Batterieschnittstelle()` die Ausgabedatei für die Batterieschnittstelle erzeugt. Die Bedienung des Proc File Simulator ist zusammen mit der Bedienung und Konfiguration der Batterieemulationssoftware im Anhang beschrieben.

5.2.3 Verbraucherentwurf

Zum jetzigen Zeitpunkt ist der Entwurf eines 802.11 Verbrauchers schwierig, da in *NET* noch keine 802.11 Geräte emuliert werden können (eine Diplomarbeit zur Integration von 802.11 läuft). Wenn eine 802.11 Karte emuliert wird, dann könnten unterschiedliche Parameter wie Sendestärke, Übertragungsrate oder Betriebsmodus der Karte ausgelesen werden und mit in die Verbrauchsberechnung einfließen.

Der Verbraucher wird im Batterieemulationsprogramm als abstrakte Klasse entworfen. Von dieser Klasse erben dann eine Klasse konstanter Verbraucher und Netzwerkverbraucher siehe Tabelle 6. Jedem Verbraucherobjekt wird eine Referenz auf das Batterieobjekt übergeben, damit das Verbraucherobjekt die Methode zur Übermittlung des Verbrauchs aufrufen kann.
Die Methode `übermittle_Verbrauch(Batterie)` wird an die konkreten Verbraucher vererbt. Über diese Methode wird eine einheitliche Schnittstelle zu der Batterie hergestellt.

Verbraucher *abstrakt*
Batterie
initialisiere_Verbraucher()$_{abstrakt}$ berechne_Verbrauch()$_{abstrakt}$ schreibe_Statistik()$_{abstrakt}$ übermittle_Verbrauch(Batterie, Verbrauch)

Tabelle 6: abstrakte Verbraucherklasse

In den einzelnen von der abstrakten Verbraucherklasse abgeleiteten Klassen muss daher nur die Initialisierung des Verbrauchers, die Berechnung des Energieverbrauchs und das Erzeugen einer Statistik spezifiziert werden.
In dem Programm gibt es zwei unterschiedlich spezifizierte Verbraucherklassen, welche beide von der abstrakten Verbraucherklasse erben. Die Klasse konstanter Verbraucher beschreibt einen Verbraucher mit konstantem Verbrauch, siehe Tabelle 7. In der Variable Verbrauch wird die Energie gespeichert, welche der

konstanter Verbraucher
Verbrauch
initialisiere_Verbraucher() berechne_Verbrauch() schreibe_Statistik()

Tabelle 7: konstanter Verbraucher

Verbraucher im nächsten Batterietakt von der Batterie anfordert.
Die Methode `initialisiere_Verbraucher()` konfiguriert den Verbraucher mit den Benutzervorgaben. Mit der Methode `berechne_Verbrauch()` wird der Energieverbrauch pro Takt berechnet. In dem Fall eines konstanten Verbrauchers wird diese Methode mit einem Fixwert realisiert. Die Statistik für den konstanten Verbraucher gibt an, wie viel Energieeinheiten während der Programmlaufzeit insgesamt angefordert wurden.

Die Klasse Netzwerkverbraucher modelliert den Energieverbrauch durch das Modell von Feeney und den Netzwerkverkehr in NET, siehe Tabelle 8. Der Netzwerkverkehr wird in `/proc/net/dev` protokolliert und der Netzwerkverbraucher

Netzwerkverbraucher
Verbrauch
initialisiere_Verbraucher() berechne_Verbrauch() schreibe_Statistik()

Tabelle 8: Netzwerkverbraucher

analysiert periodisch die Statistikdatei und berechnet den angefallenen Energieverbrauch. Die Initialisierung des Netzwerkverbrauchers wird mit der Methode `initialisiere_Verbraucher()` durchgeführt. Die Energieverbrauchsformeln und welches Geräte aus `/proc/net/dev` für den Energieverbrauch analysiert werden soll kann über eine Konfigurationsdatei vom Benutzer spezifiziert werden, siehe Anhang.

Das Hauptprogramm erzeugt mehrere Verbraucherobjekte und taktet diese Objekte. Jeder Verbraucher läuft als Thread parallel zu den anderen Verbraucherobjekten und dem Batterieobjekt, siehe Programmübersicht.

5.2.4 Programmablauf

Die Programmablauf der Batterieemulation lässt sich in drei Phasen einteilen:

- **Startup-Phase:** Das Programm instanziiert das Batterieobjekt (inklusive der Batteriezellen) und die Verbraucherobjekte.
- **Emulations-Phase:** Die Batterie und Verbraucher laufen parallel als einzelne Threads und führen die Emulationsberechnungen durch. Die Batterie und Verbraucher sind getaktet, siehe Abbildung 15.
- **Shutdown-Phase:** Die Batterie und Verbraucher erzeugen nach einer vom Benutzer spezifizierten Programmlaufzeit eine Statistik und beenden die Emulation der Batterie.

Es besteht die Möglichkeit, die Verbraucher unterschiedlich schnell zu takten. Eine Taktung schneller als der Batterietakt ist nicht sinnvoll, da sich der Verbrauch innerhalb der Batterie bis zur nächsten Batteriestandsberechnung „nur“ aufaddiert. Die Batterie ist per default auf die kleinste Zeitgranularität ($10ms$) eingestellt. Andere Zeitintervalle sind auch möglich, allerdings verschlechtert sich die Realitätsnähe des Batteriemodells je gröber die Taktung der Batterie erfolgt.

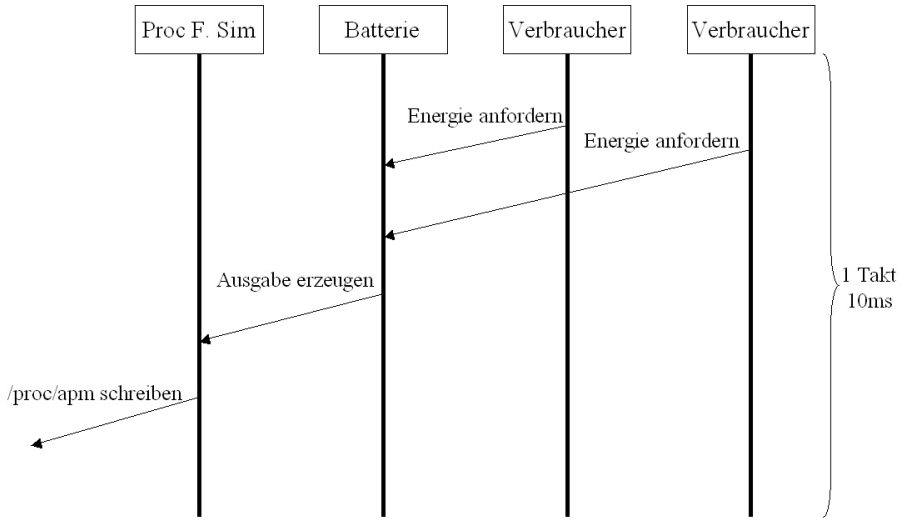

Abbildung 15: ein Batterietakt

5.3 Implementierung

Diese Kapitel beschreibt Teile der Implementierung der Batterieemulationssoftware. Der komplette Sourcecode ist dokumentiert und im *NET*-Cluster verfügbar.

5.3.1 Threadsynchronisation

Damit die verschiedenen Verbraucherthreads und der Batteriethread nicht gleichzeitig auf dem gleichen Objekt arbeiten, müssen die Threads synchronisiert werden, d.h. ein Thread sperrt bei Benutzung das Objekt für die anderen Threads. Die folgenden zwei Methoden müssen synchronisiert werden:

- übermittle_Verbrauch()
- angeforderte_Energie()

In Java2 werden Methoden über das Schlüsselwort `synchronized` synchronisiert. Dadurch wird sichergestellt, dass immer nur ein Thread Zugriff auf das benötigte Objekt hat.

5.3.2 Taktung

Die Taktung der Batterie und der Verbraucher erfolgt über ein Timerobjekt aus der Klasse `java.util.Timer`, welches einzelne Threads über TimerTaskobjekte

aus der Klasse `java.util.TimerTask` erzeugt.
Die Klasse `java.util.TimerTask` hat unter anderem die Methode
`scheduleAtFixedRate(TimerTask` *`task`*`, long` *`delay`*`, long` *`period`*`)`,
welche eine wiederholte Ausführung nach der angegebenen Periode ausführt.

5.3.3 Batteriezellenkapazitätsberechnung

Die Batteriezellenkapazitätsberechnung gliedert sich in 3 Phasen:

- Rate-Capacity-Effekt berechnen
- Batteriestand berechnen
- Recovery-Effekt berechnen

In der ersten Phase wird der angeforderte Verbrauch für die Batteriezelle mit der Lookup-Tabelle verglichen und festgestellt, ob der Rate-Capacity-Effekt in Kraft tritt. Das Ergebnis dieser Berechnung ist der aktuelle Verbrauch für diese Zelle (evtl. höher als der angeforderte Verbrauch).

In der zweiten Phase wird der neue Batteriestand abzüglich des aktuellen Verbrauchs berechnet. Wenn die Zelle daraufhin entladen ist, dann scheidet sie für zukünftige Berechnungen aus.

Die dritte Phase wird gestartet, falls der aktuelle Verbrauch unter dem spezifizierten Schwellwert für den Recoveryeffekt liegt. Die Berechnung des Recoveryeffefekts gliedert sich in zwei Phasen:

- Die aktuelle Recoveryphase der Batteriezelle berechnen: Der Recoveryeffekt ist abhängig vom Kapazitätsstand. Je nachdem wie der Benutzer die Batterie konfiguriert hat befindet sich die Batterie in einer bestimmten Recoveryphase.
- Recovery-Effekt bestimmen: Der Recoveryeffekt hängt von der Recoverywahrscheinlichkeit für die aktuelle Recoveryphase ab. Es wird eine Zufallszahl (zwischen 0 und 100) erstellt und falls diese Zahl kleiner als die Recoverywahrscheinlichkeit ist tritt der Recovery-Effekt in Kraft.

Pro Batterietakt wird eine der Batteriezellen mit der angeforderten Energie belastet. Die anderen Zellen können sich während dieser Zeit erholen, falls sie noch nicht vollständig entladen wurden. Die anderen Batteriezellen werden daher mit einem Verbrauch von 0 belastet, d.h. es wird nur die dritte Phase der Batteriezellenkapazitätsberechnung durchgeführt.

5.3.4 Benutzerparameter

Das Programm benötigt eine Reihe von Parametern vom Benutzer. Die Batterie und Verbraucher werden über Konfigurationsdateien eingestellt. Der Aufbau einer Konfigurationsdatei ist wie folgt:

- in jeder Zeile steht entweder ein Schlüsselwort
- oder ein (oder mehrere) Wert(e) als Parameter

Der einem Schlüsselwort zugehörige Wert muss in der darauffolgenden Zeile stehen.

Im Folgenden folgt eine Aufstellung der Schlüsselwörter von der Batteriekonfigurationsdatei und der erlaubten Werte:

- Die Anzahl der Bateriezellen: Schlüsselwort **cells**
  Erlaubte Werte sind natürliche Zahlen ohne die 0.
- Die theoretische Kapazität einer Batteriezelle: Schlüsselwort **t-capacity**
  Erlaubte Werte sind natürliche Zahlen ohne die 0. Die Einheit ist Ladungseinheiten.
- Die normale Kapazität einer Batteriezelle: Schlüsselwort **n-capacity**
  Erlaubte Werte sind natürliche Zahlen ohne die 0. Die Einheit ist Ladungseinheiten.
- Die Rate-Cap.-Effekt-Lookup-Tabelle: Schlüsselwörter sind **rate-start** und **end** . Das Schlüsselwort **rate-start** leitet eine Reihe von Wertepaaren ein. Jedes Wertepaar besteht aus einer natürlichen Zahl ohne die 0 (Schwellwert) und einer positiven reellen Zahl (Multiplikationsfaktor). Jedes dieser Wertpaare steht für sich in einer Zeile. Das Schlüsselwort **end** beendet die Lookup-Tabelle.
- Die Recovery-Lookup-Tabelle: Schlüsselwörter sind **rec-start** und **end** . Das Schlüsselwort **rec-start** leitet eine Reihe von Werten ein, welche die Recoverywahrscheinlichkeit für die unterschiedlichen Phasen festlegt. Je nach gewünschter Anzahl der Phasen müssen Werte zwischen 0.0 und 100.0 eingetragen werden. Jeder Wert muss in einer extra Zeile stehen. Der erste Wert steht für die erste Phase und der letzte Wert für die n-te Phase. Eine volle Batterie startet in Phase n und endet in Phase 1. Das Schlüsselwort **end** beendet die Lookup-Tabelle.
- Der Recoverywert: Schlüsselwort **rec-value**
  Erlaubte Werte sind natürliche Zahlen. Dieser Wert gibt an, wieviel Energieeinheiten bei erfolgreichem Recovery zurückgewonnen werden
- Der Recoveryschwellwert: Schlüsselwort **rec-limit**
  Erlaubte Werte sind natürliche Zahlen. Dieser Wert gibt an, bis zu welchem Wert (pro Takt) Recovery stattfindet.

Beispielskonfigurationsdateien werden mit dem Programm mitgeliefert und sind im Anhang B abgebildet.

Der Verbraucher wird wie die Batterie durch eine Konfigurationsdatei eingestellt und der Aufbau ist analog. Zur leichteren Konfiguration sind beide Verbraucher in der gleichen Konfigurationsdatei zusammengefasst. Falls beispielsweise kein konstanter Verbraucher erzeugt werden soll, dann ist der entsprechende Wert mit 0 zu konfigurieren.

- Das Netzwerkdevice: Schlüsselwort **device**
  Das Gerät, welches analysiert werden soll, beispielsweise eth0.
- Der Parameter m_{recv}: Schlüsselwort: **m-receive**
  Erlaubte Werte sind positive reelle Zahlen. Dieser Parameter wird nach dem Modell von Feeney mit der Anzahl der empfangenen Bytes multipliziert.
- Der Parameter m_{send} Schlüsselwort: **m-send**
  Erlaubte Werte sind positive reelle Zahlen. Dieser Parameter wird nach dem Modell von Feeney mit der Anzahl der gesendeten Bytes multipliziert.
- Der Parameter b_{recv} Schlüsselwort: **b-receive**
  Erlaubte Werte sind natürliche Zahlen. Dieser Parameter wird nach dem Modell von Feeney mit der Anzahl der empfangenen Pakete multipliziert.
- Der Parameter b_{send} Schlüsselwort: **b-send**
  Erlaubte Werte sind natürliche Zahlen. Dieser Parameter wird nach dem Modell von Feeney mit der Anzahl der gesendeten Pakete multipliziert.
- Der Verbrauch eines konstanten Verbrauchers: Schlüsselwort **constant**
  Erlaubte Werte sind natürliche Zahlen. Einheit in Ladungseinheiten.

Neben den Konfigurationsdateien benötigt der Start des Batterieemulationsprogramms folgende Parameter:

- Programmlaufzeit in Sekunden
- Name der Batteriekonfigurationsdatei
- Name der Verbraucherkonfigurationsdatei
- Name der (zu erzeugenden) Ausgabedatei
- Name der (zu erzeugenden) Statistikdatei

Weitere Informationen zur Bedienung und Start der Batterieemulationssoftware befinden sich im Anhang A.

6 Ausblick

In diesem Kapitel werden die Ergebnisse der Arbeit rekapituliert und ein Ausblick auf zukünftige Arbeiten gegeben.

6.1 Zusammenfassung

Im Rahmen dieser Studienarbeit wurden verschiedene theoretische Batteriemodelle untersucht. Für die Batteriemodelle wurden Bewertungskriterien aufgestellt und die Modelle auf ihre Funktionsweise und Tauglichkeit für eine Implementierung in den *NET*-Cluster beurteilt. Weiterhin hat die Untersuchung des Energieverbrauchs von 802.11 Netzwerkkarten einen großen Teil dieser Arbeit eingenommen. Die Implementierung eines ausgewählten Batteriemodells mit Verbraucher in das System bildete den Abschluss dieser Arbeit.
Die emulierte Batterie ist unabhängig von ihren Verbrauchern und kann auf mehrere Verbraucher in Zukunft erweitert werden. Da zum jetzigen Zeitpunkt noch keine 802.11 Geräte emuliert werde können, gestaltete sich die Modellierung des Energieverbrauchs eines 802.11 Gerätes in *NET* als schwierig.

6.2 Zukünftige Arbeiten

Das momentan implementierte Batteriemodell modelliert das Entladeverhalten einer vollständig geladenen Batterie unter variablen Belastungen. Für eine Erweiterung des Batteriemodells wäre es denkbar die Möglichkeit einer Wiederaufladung der Batteriekapazität mit zu modellieren. Wiederaufladbare Batterien verlieren mit der Anzahl an Lade- und Entladezyklen einen Teil ihrer Kapazität. Diese Zyklenfestigkeit wird in der momentan emulierten Batterie nicht berücksichtigt. Sarkar und Adamou entwickeln in [SA03] ein Framework, wie die unterschiedlichen Batteriezellen einer Batterie zu belasten sind. Ziel dieser Arbeit ist es, die maximal verfügbare Energie einer Batterie auszuschöpfen. Das im Rahmen dieser Studienarbeit implementierte Batteriemodell belastet die Batteriezellen der Reihe nach. Eine Erweiterung des Batteriemodells durch das Framework von Sarkar und Adamou wäre denkbar.
Die Berechnung des Energieverbrauchs eines 802.11 Geräts wird realistischer, wenn 802.11 Geräte in *NET* emuliert werde können, da mehrere, im Moment noch unbekannte, Parameter für die Verbrauchsberechnung verfügbar wären (Sendestärke, Übertragungsrate oder Betriebsmodus der Karte). Auch die Modellierung des Energieverbrauchs im Falle einer fehlerhaften Übertragung wäre viel leichter als in den Messexperimenten aus Kapitel4 modellierbar, da *NET* eine kontrollierte Umgebung zur Verfügung stellt.

7 Literatur

[Acp] http://www.acpi.info

[Apm01] Advanced Power Management V1.2 Specification
http://www.microsoft.com/whdc/hwdev/archive/BUSBIOS/amp_12.mspx

[BF01] A. Bard und L. Faulkner. 2001. Electrochemical Methods, Second Edition. Wiley, New York. }[0.5cm] [Buc01] I. Buchmann. Batteries in a Portable World. Cadex Electronics. 2001. http://www.cadex.com

[Buc03] I. Buchmann. Battery University. http://www.batteryuniversity.com/

[CSA+98] Jyh-Cheng Chen, Krishna M. Sivalingam, Prathima Agrawal und Shalinee Kishore. A Comparison of MAC Protocols for Wireless Local Networks Based on Battery Power Consumption. Proceedings of IEEE Einfocom 1998. San Francisco, CA, März, 1998.

[Dur03] Duracell. Technisches Bulletin NiMh 2003. http://www.duracell.de

[DFN93] M. Doyle, T.F. Fuller und J.S. Newman. Modeling of galvanostic charge and discharge of the lithium/polymer/insertion cell. J. Electrochem. Soc., vol.140, 1993.

[EBW02] Jean-Pierre Ebert, Brian Burns und Adam Wolisz. A trace-based approach for determining the energy consumption of a WLAN network interface. In Proc. of European Wireless 2002. pp. 230-236.

[Fee01] Laura Marie Feeney. An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Newtorks. Mobile Networks and Applications, 2001.

[FN01] Laura Marie Feeney und Martin Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. IEEE Infocom 2001.

[FS99] J. Flinn und M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. Second IEEE Workshop on Mobile Computing Systems and Applications, 1999.

[GHS96] Paul Gauthies, Daishi Harada und Mark Stemm. Reducing Power Consumption for the Next Generation of PDAs: It's in the Network Interface. Proceddings of MoMuC'96.

[GW00] W.B. Gu und C.Y.Wang. Thermal-Electrochemical Modeling of Battery Systems. In J. Electrochem. Soc. vol. 147, 2000.

[Häb03] M. Häberlein: Vorlesungsskript Anorganische und Allgemeine Chemie, FH Frankfurt a.M., 2003.
http://www.fbv.fh-frankfurt.de/mhwww/ach-vorlesung/inhalt.HTM

[Hag93] S. Hageman: Simple PSPICE models let you simulate common battery types. Electronic Design News, vol. 38, 1993.

[HBZ+03] Z. Hu, D. Brooks, V. Zyuban und P. Bose. Microarchitecture-Level Power-Performance Simulators: Modeling, Validation, and Impact on Design. IBM Research, Harvard University. 2003
http://www.eecs.harvard.edu/ dbrooks/micro2003-tutorial-final.pdf

[HM03] Daniel Herrscher und Steffen Maier. NET: The Network Emulation Testbed Manual. University of Stuttgart 2003.

[Hog03] Emma Jane Hogbin, ACPI: Advanced Configuration and Power interface. In The Linux Documentation Project, Juli 2003.

[HT03] M. Handy und D. Timmermann. Simulation of Mobile Wireless Networks with accurate modelling of non-linear Battery Effects. 4th International Conference on Applied Simulation and Modelling 2003, Marbella.

[Joh03] W.E.Johns: Notes on Batteries, 2003.
http://www.gizmology.net/batteries.htm (Stand 14.12.03)

[LRD+ 02] Kanishka Lahiri, Anand Raghunathan, Sujit Dey und Debashis Panigrahi. Battery-Driven System Design: A New Frontier in Low Power Design. Proceedings of the 2002 conference on Asia South Pacific design automation/VLSI Design.

[Mar99] Thomas L. Martin: Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing. Carnegie Mellon University 1999.

[MS96] Thomas L. Martin und Daniel P. Siewiorek: A Power Metric for Mobile Systems. Engineering Design Research Center, Pittsburgh, 1996.

[New03] J.S.Newman, FORTRAN Programs for Simulation of Electrochemical Systems. Dualfoil program for lithium battery simulation 2003.
http://www.cchem.berkeley.edu/ jsngrp/ (Stand 15.01.2004)

[PCD+01] D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan und K. Lahiri.: Battery life Estimation for mobile embedded Systems. In Proc. Int. Conf. VLSI Design, Jan 2001.

[Red01] Red Hat Linux 7.2: The official Red Hat Linux Reference Guide, 2001. http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual /ref-guide/index.html

[Rot01] S. Rothwell. APM BIOS driver for Linux. 2001. http://lxr.linux.no/source/arch/i386/kernel/apm.c

[RV02] D.Rakhmatov und S. Vrunhula. Battery Lifetime Prediction for Energy-Aware Computing. In Proceedings of Int. Symposiums on Low Power Electronics and Design, 2002.

[RV03] D. Rakhmatov und S. Vrunhula. Energy Management for Battery - Powered Embedded Systems. ACM Transactions on Embedded Computing Systems, Vol. 2, No. 3, August 2003.

[SA03] S. Sarkar und M. Adamou. A Framework for Optimal Battery Management for Wireless Nodes. IEEE Journal on selected Areas in Communications, Vol. 21, No.2, February 2003.

[TMW+96] V. Tiwari, S. Malik, A. Wolfe und M.T.C. Lee. Instruction Level-power analysis and optimization of software. J. VLSI Signal Processing, vol. 13, no.2, 1996.

[Zin01] Rolf Zinniker: Batterien und Akkus: Elektrochemische Beschreibung. ETH Zürich, Institut für Elektronik, Januar 2001.

[Zin96] Rolf Zinniker: Regenerierung von Alkalibatterien. Bulletin SEV/VSE 15/96.

A Bedienung der Batterieemulationssoftware

Die Bedienung der Batterieemulationssoftware funktioniert folgendermaßen:

- Den Proc File Simulator starten, damit die Batterieschnittstelle von der Batterieemulationssoftware gesteuert werden kann.
- Die Batterieemulationssoftware starten.
- Den Proc File Simulator nach erfolgter Batterieemulation beenden, damit das ursprüngliche Power-Management die Arbeit wieder übernehmen kann.

A.1 Proc File Simulator

Der Proc File Simulator ist ein Kernelmodul. Die Aufgabe des Kernelmoduls ist es, die Ausgabe des APM-Moduls umzuleiten und die Ausgabe der Batterieemulationssoftware stattdessen in die Datei `/proc/apm` zu schreiben. Das Kernelmodul wird mit `insmod procfsim.o` in den Kernel geladen. Per Default wird die Datei `/proc/apm` durch die Datei `/proc/apm.sim` ersetzt. Über Parameter können diese Defaultwerte geändert werden. Parameter sind:

- orgName= Dateiname der Datei, welche ersetzt werden soll (Default: apm).
- simName= Dateiname der Datei, welche die Datei aus orgName ersetzen soll (Default: apm.sim).
- bufSize= Größe des Datenpuffers (Default 1024).
- perm = Die Permissions der neuen Datei (Default 0600).
- uid = Die UserID der neuen Datei.
- gid = Die GruppenID der neuen Datei.

Nach erfolgreicher Emulation der Batterie kann das Kernelmodul wieder mit `rmmod procfsim` entfernt werden.

A.2 Die Batterieemulationssoftware

Die Batterieemulationssoftware benötigt zum Start mehrere Parameter vom Benutzer:

- Die Programmlaufzeit in Sekunden (als natürliche Zahl)
- Der Dateiname der Batteriekonfigurationsdatei.

- Der Dateiname der Verbraucherkonfigurationsdatei.
- Der Dateiname der Ausgabedatei, welche dem Proc File Simulator übergeben werden kann.
- Der Dateiname der Statistikdatei, welche vom Programm erzeugt werden soll.

Der Progammaufruf kann beispielsweise folgendermaßen erfolgen:
`java battemu 100 battery consumer apm.sim log.txt`
Der Aufbau der Konfigurationsdateien für die Batterie und Verbraucher ist in Kapitel 5.3.4 beschrieben. Beispielskonfigurationsdateien sind im Anhang B abgebildet. Wichtig ist, dass in jeder Zeile entweder ein Schlüsselwort oder ein Konfigurationswert steht. Der Benutzer muss alle Schlüsselwörter in den Konfigurationsdateien verwenden! Soll beispielsweise kein Recoveryeffekt in der Batterie stattfinden, dass muss der Recoverywert auf 0 gesetzt werden. Die Recovery-Lookup-Tabelle muss allerdings mit mindesten einem Wert belegt werden (analog für die Rate-Lookup-Tabelle). Der Benutzer hat sicherzustellen, dass in den Konfigurationsdateien nur erlaubte Werte stehen, siehe Kapitel 5.3.4.

B Beispielskonfigurationsdateien

Eine Batteriekonfiguration kann folgendermaßen aufgebaut sein:

```
t-capacity
1000000
n-capacity
500000
cells
3
rec-value
100
rec-limit
0
rec-start
10.0
30.0
50.0
end
rate-start
10000 1.5
20000 2
end
```

Eine Verbraucherkonfiguration kann folgendermaßen aufgebaut sein:

```
device
vlan0
m-receive
2.2
m-send
2.4
b-receive
300
b-send
400
constant
250
```