Anwendungsspezifische Energiesparverfahren für WLAN

Studienarbeit im Fach Informatik

vorgelegt von

Matthias Faerber

geboren am 21. Februar 1979 in Fürth (Bay.)

Institut für Informatik,
Lehrstuhl für verteilte Systeme und Betriebssysteme,
Friedrich Alexander Universität Erlangen-Nürnberg

Betreuer:	Dr. Ing. Frank Bellosa
	Dipl.-Inf. Andreas Weißel
	Prof. Dr. Wolfgang Schröder-Preikschat

Beginn der Arbeit: 3. Juni 2003
Abgabedatum: 19. Januar 2004

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 16. Januar 2004

Application-Specific Power Management for Wireless Networks

Student Thesis

by

Matthias Faerber

born February 21th, 1979, in Fürth (Bay.)

Department of Computer Science,
Distributed Systems and Operating Systems,
University of Erlangen-Nürnberg

Advisors:	Dr. Ing. Frank Bellosa
	Dipl.-Inf. Andreas Weißel
	Prof. Dr. Wolfgang Schröder-Preikschat

Begin:	June 3rd, 2003
Submission:	January 19th, 2004

Abstract

In recent years computer networks have gained importance in many fields. After the success of wired networks and the internet, radio based wireless networks are spreading. In many cases wireless networks are used when it is impossible to connect the computer to a network using a cable or it is just uncomfortable. This not only applies to network cables but also to power cables and so most devices are battery powered.

Wireless network adapters that are using the IEEE 802.11b standard are already equipped with a power management mode. In this power management mode the card continously changes between a sleep and an awake state. The sleeping interval can be altered by setting the so called beacon interval.

This work examines how network applications can be identified only with information available at the network layer. It is shown that statistical information derived from the linux kernel can be used to characterize and identify applications. For a set of applications such profiles are developed and explained.

Based on these profiles an algorithm is provided that is capable of detecting applications during their runtime and adjust the beacon interval of the IEEE Standard. Thus the static behavior of the power management mode it changed into a dynamic one. The algorithm considers both application performance and energy conumption and sets the power management to an user-defined trade-off between both.

The results of the characterization algorithm are evaluated in two tests. First offline tests are used to analyze the detection rates for one application alone. Second, online tests are used to analyze the delay between the start of an application and its detection and the impact of false characterizations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Portable computers, notebooks as well as handhelds are becoming more and more popular. As they often rely on batteries, their power consumption has become a major issue. In recent years hardware components have been equipped with capabilities to save energy. They range from processors supporting multiple speed steppings up to hard drives that shut down when they are not accessed for a longer period.

The IEEE 802.11 standard for wireless LAN cards was already designed with power management in mind. But this algortithm has a flaw that is common for all these standard algorithms, either for CPUs or network cards. They mostly are static and do not adapt to the application or user.

The power management mode included in the IEEE 802.11 defines two modes of operation for a wireless node. It can either be in Constantly Awake Mode (CAM) or in Power Saving Mode (PSP). Whereas in CAM the card is always ready to receive and transmit, the card is asleep most of the time while it is in PSP mode. Before going to sleep, it has to inform the base station that it is using power saving mode. While the client card is in power saving mode, the base station may not transmit any data destined for the client and has to buffer it. Clients that currently have data buffered at the access point are regularly informed by the base station when data is available for them and can poll the base station for that data.

In the power saving algorithm of the standard the card is always put to sleep for a fixed period of time. This time interval cannot be altered dynamically but has to be set by a user if he wants to adjust it. The sleeping interval has two effects. While it reduces the amount of energy the network card is consuming, it also can have negative impact on application performance. The performance penalty differs from application to application. Some applications like multimedia players already have mechanisms to handle packet losses. Others, for example those that are RPC-based, do not and suffer greatly from power management.

Several algorithms have been proposed to reduce or completely get rid of this static behavior and adapt power management to the applications. But most either alter the transmission protocol between two stations or extend the kernel with an API so that applications can reveal information about their likely future behavior.

This work shows how the power management algorithm of the IEEE standard can be extended to a dynamic behavior without having to change the transmission protocol or alter applications. Instead a heuristic is used to detect running applications and change the power management settings according to the currently running application.

This work is organized as followed. The IEEE 802.11b standard is explained in chapter two with its most import features. This includes possible network topologies, important services and frame types that are relevant to power management. At the end of this chapter the power management algorithm of the standard is explained.

The third chapter takes a look at the measurement environment. The setup to record energy traces, the Linux driver and tools that are necessary to operate a wireless adapter are part of this.

In the fourth chapter the applications that were used to test the performance of the algorithm in this work are described.

The focus of the fifth chapter lies on the characterization part of the algorithm. It explains the evolution of the algorithm and how parameters are mapped to applications.

The work ends with an outlook over what can be done to enhance the usability and performance of the algorithm further. Also other works in the field of power managent are referenced and their relation to this work is given.

Chapter 2

IEEE 802.11 Standard

The IEEE 802 protocol family was designed to enable the exchange of information between hosts on a network. The most established protocol of that family is the 802.3. It is used to connect computers in wired local area networks. The 802.11 protocols are used to connect computers wireless using radio hardware.

Before the IEEE has adopted the 802.11 standard in 1997 nearly all vendors of network equipment had their own proprietary protocol. Devices of two vendors often were not able to comunicate with each other. The 802.11 was an effort to solve that problem and create a standard for wireless communication in local area networks.

In addition to that first 802.11 standard (802.11-1997) other standards for wireless networks have been developed. But although these later standards are based on the first one they are not always compatible. A summary of all protocols of the 802.11 family is given in table 2.1.

Standard	Range	Frequency	Bit-Rate
802.11-1997		2.4 GHz	1-2 mbps
802.11a	20-50m	5 GHz	54mbps
802.11b	30-100m	2.4 GHz	11mbps
802.11g	30-100m	2.4 GHz	2-54mbps

Table 2.1: IEEE 802.11 Standards

The 802.11-1997, which was the first to be adopted, had only a limited bitrate. So two enhancements were developed: 802.11a and 802.11b. 802.11a offers higher bit-rate than the former standard, but operates at the 5GHz frequenzy band and is not available in Europe due to radio frequency regulations. 802.11a compatible networks are used mainly in the United States.

802.11b alters the way data is transmitted in regard to the 802.11-1997 standard and is able to provide bit-rates of up to 11mbps. As it uses also the 2.4Ghz band, it is mostly compatible with the older 802.11-1997 standard.

2.1 Network Topology

2.1.1 Ad-Hoc Networks

The most simple configuration of a wireless network is an ad-hoc cell. It is displayed in figure 2.1.

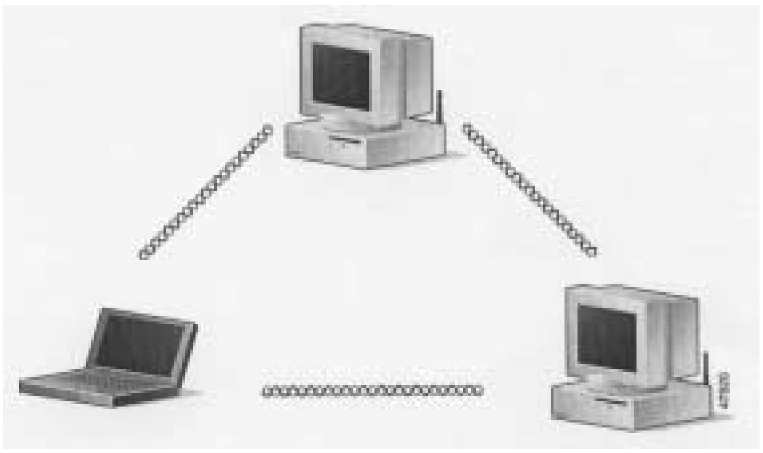

Figure 2.1: Wireless cell in ad-hoc mode

In this configuration no additional hardware except the network cards is necessary to connect the stations. Every station is directly connected with every other stationin the cell. This can be compared to an ethernet in the way that there is no central controller. Two stations can communicate only as long as they are able to see each other. One station can act as a proxy or bridge and connect the wireless cell with other networks.

2.1.2 Infrastructure Network

An infrastructure network is a more complex configuration of a wireless network. It is displayed in figure 2.2.

In an infrastructure network every cell is controlled by an access point. The access point is the center of every cell and can act as a bridge to connect the wireless cell with other networks. Each station that wants to join the network have to be able to reach the access point. A station always connects with the access point that has the strongest signal.

A network system with only one access point is called a *basic service set* (BSS). Several BSS can be connected via a *distribution system* and are then called *extended service set* (ESS). Every BSS or ESS can be identified by a unique number called SSID. The *distribution system* enables a station in one BSS to communicate with a station in an other BSS of the same extended service set. The standard specifies no special distribution system and in practice ethernet is used most often. In order to communicate with stations in other parts of the network a station must be trackable. This requires a special service that is provided by the distribution system and is called *assoziation*

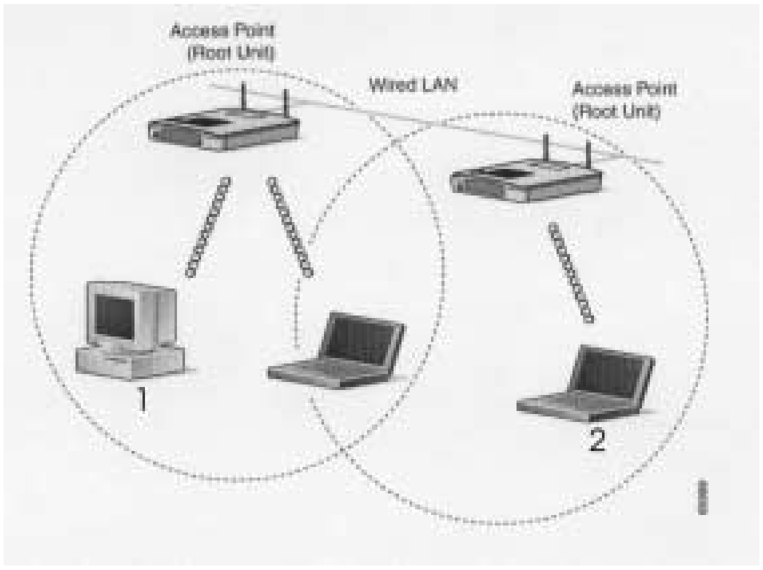

Figure 2.2: Wireless cell in infrastructure mode

(see section 2.2.1). With the help of this service the distribution system is able to route the packets through the net from the sender to the receiver.

In an ESS a station that leaves one cell (BSS) of the network can join another cell. This is called *roaming* and is transparent to the user. This way wireless networks can be extended beyond the range of one access point.

2.2 Important Services

2.2.1 Distribution

The distribution service is a key service of an extended service set. It is invoked every time a station wants to send packets to a station that is localized in a different cell of the ESS. For example: Station 1 in figure 2.2 is not able to directly communicate with station 2 as they are not located in the same cell. Instead the distribution system that connects the two access points has to be used.

In order to indentify the access point that is responsible for the cell in which a station is located, a station first has to *associate* itself with the access point. One station can at any time be assiciated with only one access point. But an access point can be assiciated with any number of stations.

To be able to send messages in a wireless network it is necessary for a station to be associated with an access point.

2.2.2 Authentication

In wired networks it is relatively simple to control the access to the network. Every station that is connected to the wire has access, all others don't. In wireless networks such restrictions do not apply. As the transport medium is air in principle every station within the range of the access point can join the network.

IEEE 802.11 provides control to LAN access via the *authentication* service. Although no particular authentication scheme is required in the specifications in most cases shared key authentication is used. In shared key authentication schemes both adapters have the same key. The access point sends the station a request (challenge) and the station as to respond with the correctly encrypted request message. The same key is also used to encrypt transmitted data. This encryption is not considered to be safe and was intended to provide about the same security as a wire in ethernets does. Therefore it is called *Wired Equivalent Privacy* (WEP).

2.2.3 Interrelationship of Association and Authentication

Directly after activating a network adapter, a station is not connected to a network. First it has to associate with an access point and authenticate itself. In this process only the following three states are possible:

1. Unauthenticated & Unassociated
2. Authenticated & Unassociated
3. Authenticated & Associated

States 1 and 2 are only used during the registration procedure. In state 3 the station is in logged on to the network and can exchange data with all other stations.

The interrelationship of the services and states is shown in figure 2.3.

A detailed description of all services can be found in [6] and an overview of wireless LANs in [9] and [10].

2.3 Frame Types

In a network during transmission data is split into packets and is send in TCP frames over the net. In wireless networks three frame types exist:

- Data frames
- Control frames
- Management frames

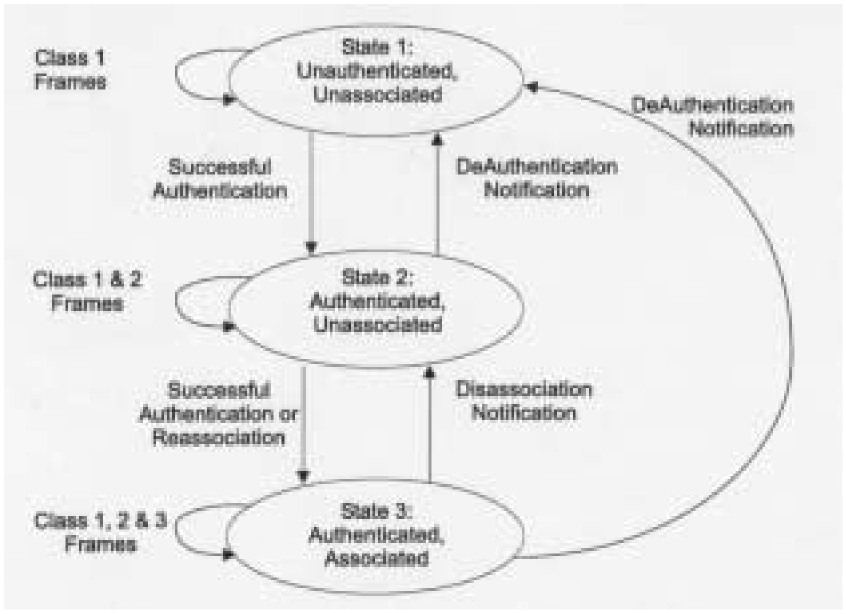
Class 1 Frames
State 1: Unauthenticated, Unassociated
DeAuthentication Notification
Successful Authentication
DeAuthentication Notification
Class 1 & 2 Frames
State 2: Authenticated, Unassociated
Successful Authentication or Reassociation
Disassociation Notification
Class 1, 2 & 3 Frames
State 3: Authenticated, Associated

Figure 2.3: Relationship between states and servcies

All stations that want to participate in a wireless network have to be able to understand and send all these frame formats.

Data frames are the most common type of frames in a wireless network. They are used to exchange application data between stations in a network. *Control frames* are used to regulate the access to the medium (air). They include among others the request to send, clear to send and power save poll frames. *Management frames* are used to control infrastructure aspects of a wireless cell. Most of the services offered by a network (like those in section 2.2) depend on management frames. Possible management frames are, for example, association and disassociation frames, their responses and authentication frames. A special management frame is the beacon frame, its contents are displayed in table 2.2. It is transmitted continuously by the access point to inform all stations about changes in the network. Beacon frames play an important role in power management as all stations have to listen to them in predefined intervals. Each station may set its own interval. In the following chapters it will be shown how this interval can be changed to save energy.

2.4 Power Management

As most stations that use wireless adapters are battery powered, an additional power management mode was included in the IEEE standard. Two modes of operations are defined:

- Active Mode
- Power Saving Mode

Information	Length	Meaning
Timestamp	8 Bytes	Time of the sender to synchronize the clocks of all stations in a cell
Beacon-Interval	2 Bytes	Time between to beacon frames. With the clock time, every station can evaluate the time for the next beacon and thus detect the missing of a beacon frame
Capability	8 Bytes	Information about the cell: Ad-hoc or infrastructure mode, encryption, ...
Address	6 Bytes	Address of the cell, in infrastructure mode it is the address of the access point
⋮	18-23 Bytes	⋮
TIM	4 Bytes	Transmition Indication Map to controll power management modes

Table 2.2: Beacon frame format

A station that is in active mode, is always *awake* and can send and receive data at any moment. A station in power saving mode, on the other hand can assume an additional *doze* state. In this state the station is not able to receive or transmit any data and consumes very few power.

$$\begin{tikzcd}
\text{Active Mode} \arrow[r] & \text{Awake State} \\
 & \text{Awake State} \\
\text{Power Saving Mode} \arrow[ru] \arrow[r] & \text{Doze State}
\end{tikzcd}$$

Before a station may go to power saving mode it has to inform the access point. This is necessary as the station is virtually offline while it is in doze state. The access point has to buffer all data that is destined for a station in power saving mode. A station on the other hand has to periodically listen to beacon frames. As shown in table 2.2, the access point informs all stations whether data is available or not. Upon determining that data is available at the access point, a station may poll the access point for it and receive the data in response.

The transitions between these two states are described in the power management modes, which are summarized in table 2.3.

As an access point has to know whether a station is in power saving or active mode, a station may not change modes without successfully informing the access point about the change.

The traffic indication map (TIM) of a beacon frane is used to notify stations that data is available for them at the access point. There are two types of TIM messages: TIM and DTIM. DTIMs (delivery traffic indication map) are special TIM messages as the access point can send broadcast and multicast messages for staions in the cell.

Figure 2.4 illustrates the data exchange between an access point and two stations. Both stations have different polling interval settings. A ramp-up indicates that the station is powering up and

Active mode or AM	Station may receive frames at any time. In Active mode, a station shall be in the Awake state. [...]
Power Save or PS	Station listens to selected beacons (based upon the Listen-Interval parameter [...]) and sends PS-Poll frames to the access point if the TIM element in the most recent beacon indicated a directed MAC service data unit (MSDU) buffered for the station. The access point shall transmit buffered directed MSDU to a station in powersave mode only in reponse to a PS-Poll from that station [..] In PS mode, a station shall be in Doze state and shall enter the Awake state to receive selected beacons, to receive broadcast and multicast transmissions following certain received beacons, to transmit, and to await responses to transmitted PS-Poll frames [...].

Table 2.3: Power Management Modes (as defined in [6])

switching from doze to awake state. At the first arrow in the figure station one receives a beacon frame with a TIM indicating that data is available. The station sends a poll frame and data is transmitted from the access point to the station. If the medium is already used by another station, beacon frames may be delayed. Every station can decide itself how long it wants to stay in doze state. But the longer a station is in doze state, the higher is the latency to receive a packet.

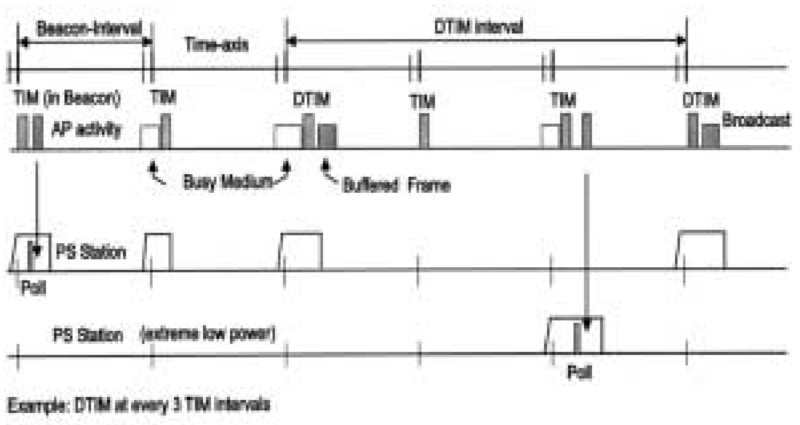

Figure 2.4: Infrastructure Power Management Operation [6]

Chapter 3

Measurement Environment

3.1 Energy Measurement/ Platform

The network card for which the Linux kernel driver was adapted, was a Cisco Aironet 350 wireless LAN adapter [5]. The card was used with an IBM Thinkpad 600 notebook that ran a Debian GNU/ Linux. The operating system was a Linux standard kernel version 2.4.18. In order to measure the energy consumption of the network adaptor it was connected to the computer via an PCMCIA extender. The extender has several contacts to monitor the card. Figure 3.1 shows a diagram of the measurement setup.

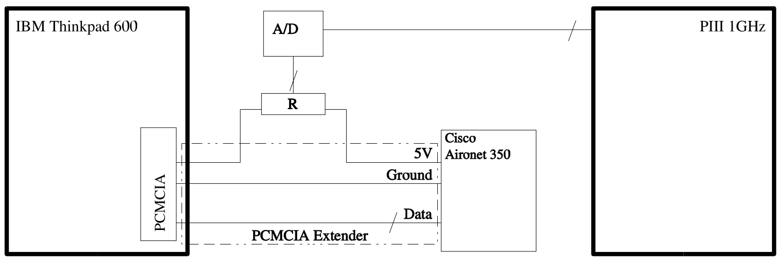

Figure 3.1: Measurement Environment

The network card is supplied with $5V$. A small resistor was attached to the power supply line. The resistor is connected via a A/D converter to the parallel port of another computer. The A/D converter is able to sample up to 20000 values per second. With the fall of voltage at the resistor U_{R_1} and the size of the resistor R_1 it is possible to compute the power consumption P_{LAN} of the network card using the formula:

$$P_{LAN} \approx \frac{5V \cdot U_{R_1}}{R_1}$$

3.2 Linux Airo Kernel Drivers

Drivers for the Cisco Aironet 350 are already included in the Linux kernel. During the development of this algorithm several versions of the Linux kernel up to 2.4.21-pre5 have been tested. Finally kernel version 2.4.18 was chosen as it was the only one that has proved to be reliable.

The kernel driver is composed of two modules. While the `airo_cs.o` module is responsible for interfacing with the pcmcia bus, the `airo.o` module is the actual driver. The `airo.c` source file contains all the code that is necessary to operate the hardware.

3.3 Wireless Tools

The *Wireless Extensions* are an extension to the Linux kernel and provide a generic API to manipulate and query various settings of network adapters. These include the SSID of the network cell, transmission power, encryption keys and the power management configuration. *Wireless Extensions* are included in the kernel and every driver that wants to make use of it needs to implement it.

While the *Wirless Extensions* are an API in the kernel, the *wireless tools* are a reference implementation and provide user space tools. The central tool is called `iwconfig`. It is a command line interface to alter basic settings. The others are `iwlist`, `iwspy` and `iwpriv` and allow access to more private settings.

3.4 Cisco Aironet 350

For the experiments in this thesis, a Cisco Aironet 350 Wireless LAN adapter has been used. This card offers three different power saving modes:

- CAM (equal to AM mode)
- PSP (equal to PS mode)
- PSPCAM

In addition to switching from CAM to PSP (or PSPCAM) the adapter also has the ability to change the beacon interval. Switching between two modes or changing the listen interval costs both time and energy. Table 3.1 summarizes the costs with Aironet 500.

To overcome these costs, Cisco has introduced a new power saving mode that is called PSPCAM. PSPCAM differs from PSP in the manner that after being in awake state, it does not immediately fall back to doze state but stays in awake state for a short period (approx. 200ms).

mode transitions	time	energy
PSP to CAM	320ms	280mJ
CAM to PSP	317ms	283mJ
change beacon interval	333ms	300mJ

Table 3.1: Overhead of Mode Transitions

This additional mode reduces some of the shortcomings of the power saving mode of the standard. But as it is a proprietary mode, and not likely to be implemented in cards of other vendors, it is not regarded any further in this work.

Chapter 4

Application Parameters

4.1 Parameters

The data source in the Linux kernel was chosen so that it causes as few overhead as possible. This includes both programming overhead during the implementation and runtime overhead. The Linux kernel already contains data structures with information about traffic on a network interface. This information is maintained within the driver in `struct net_device_stats`. In an unmodified kernel the contents of this structure can be viewed for example with the `ifconfig` command. Among other information it contains the following:

- Received packets
- Transmitted packets
- Received bytes
- Transmitted bytes

Within kernel space `struct net_device_stats` is accessible via the `get_stats` operation. Periodically this operation is called and the information containing the amount of data transmitted in the last period is written to a log file.

The data from the log file is read by a program that computes a series of derived key figures. These key figures should satisfy two rules.

1. The key figures should be stable for one application. Stable in this context means that the key figure should differ as little as possible for the whole runtime of the application. But stable also means that it should be independent of the polling interval if the card is operated in power management mode.

2. The key figures should vary for two distinct applications. Vary means that the distance for two values for two different applications of the same key figure is as large as possible. The further one key figure is distributed, the better is its success rate during the characterization.

The program that performs the characterization will be called from now on characterization module. In this characterization modules seven key figures were used to identify applications.

1. Average size of received packets
2. Average size of transmitted packets
3. Ratio of inactive to active periods
4. Ratio of average size of received to average size of transmitted packets
5. Amount of transmitted bytes
6. Standard deviation of average sizes of received packets
7. Standard deviation of average length of inactive periods

The *average size of received packets* and those for transmitted packets can be used to distinguish whether an application causes a lot of network traffic or rather few one. Targeted at another aspect of network applications is the *ratio of inactive to active periods*. It is a measure whether an application transmits data continously or has periods with delays. A similar key figure is the *standard deviation of average length of inactive periods*. It can be used to distinguish interactive applications from non-interactive ones. For interactive applications it is more likely to have periods of inactivity that vary in their length then for batch applications due to user think time.

In addition to the previous seven key figures that were finally included into the characterization module a number of other figures were evaluated. This included:

- Standard deviation of average sizes of transmitted packets
- Ratio of standard deviations for the sizes of received and transmitted packets
- Standard deviation of average lengths of active periods

Although these key figures could be easily computed as results from the previous key figures, they were not included in the characterization module. They were either not stable for one application or grouped around one single value and thus were omitted.

4.2 Applications

To test the characterization algorithm it was necessary to select a set of applications that is common to run on a wireless node. One proposed scenario included research activity in the Internet, finding and editing of some files that are mounted on a remote share using NFS like file systems and a remote session on a host using SSH.

In another scenario it a user listens to a radio stream on the internet or watches live videos on the internet. Also the download of larger files should be identified.

To cover the proposed scenarios, the following applications were chosen to test the characterization module and profiles for them were created:

- Web browser (Mozilla)
- Download (FTP-Client)
- Low-bandwidth stream (Real Audio Stream/ mp3 Stream)
- High-bandwidth stream (Real Video Stream)
- RPC based service (NFS)
- Terminal session (SSH)

In addition to these six application a number of others would also be possible. Not included in the list above are for example an email client, a chat program and a news ticker. As an application like an email client is called only very rarely (once about every 10min) power management for them alone is not necessary. Chat programs and news tickers on the other hand are often based on the http protocol and should fit into the web browser profile. So again no seperate profile is necessary.

4.3 Application Classification

Application can be classified into four categories regarding power management. They can be *interactive* or *non-interactive* and power management *friendly* or *unfriendly*. Friendly applications show little or no performance loss if power management is turned on, unfriendly applications do. Figure 4.1 classifies the six test applications within this scheme. Audio and video streams are combined into one single profile.

Mozilla and SSH belong to the second group of applications. Although performance is a critical issue, both applications tolerate power mangement. The intervals for which a user is able to recognize delay vary. Whereas for the secure shell beacon intervals above 100ms are disturbing, a web browser can tolerate beacon intervals up to 300ms and even more.

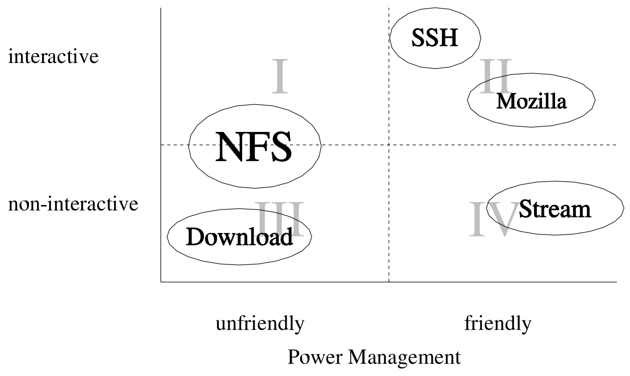

Figure 4.1: Application classification

NFS belongs to the first as well as to the third group of applications. NFS falls into to first group if the interactive part dominates. Examples would be sessions in which a lot of shell command like `ls`, `find` or `grep` are used and the user has great influence on the workflow. The compilation of a Linux kernel would be an example where NFS falls into the third group. Workflows in this group are triggered by the user but then run on their own for a longer period.

The network file system uses RPC services of the operating system and is very unfriendly in regard to power management. This has been examanined in detail by [2]. Our own tests show that performance for NFS with power management enabled decreases performance extremely. Figure 4.2 shows a copy operation on an NFS mounted filesystem. The operation finished after a few seconds when power management was turned off (0*s* beacon interval), but took more than 1000*s* for a beacon interval of 1000*ms*. Therefore the profile for NFS disables power management completely.

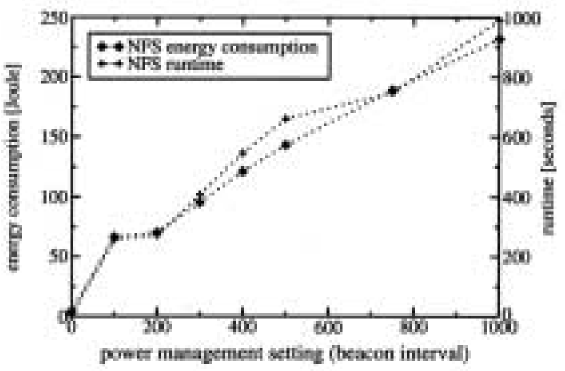

Figure 4.2: NFS performance degrading

FTP-Client falls into the third group. It is a batch application and mostly independent of user input. It is different in so far as it utilizes the complete bandwidth of the wireless network. In the profile for FTP power management was switched off.

Audio and video streams are the only ones from the set of test applications that fall within the fourth category. The transfer protocols for web stream were designed so that they can cope with interrupts in the transfer for a short time. This is done by prefetching data at the client and playing the stream with a short delay. Activating power management is nothing else than introducing an artificial delay. For all test applications the audio and video play could tolerate the biggest polling interval. In the profile it was set to 500ms.

Chapter 5

Implementation

5.1 Algorithms

The implementation of the algorithm for the Linux operating system consists of two parts, the collector module located inside the kernel and the characterization module in user space.

5.1.1 Collector Module

The kernel part of the system is responsible for retrieving data from the network interface level. The Linux kernel already maintains a data structure that contains statistical data about the traffic for each network interface. Thus no additional overhead is implied on the system to obtain the necessary information. The collector module periodically (100ms) retrieves the number of bytes and packets that have been received and transmitted during the last time slot. It takes the data from the kernel structure and passes the statistics to user space via the `/proc` interface.

5.1.2 Characterization Module

The characterization module is responsible for mapping the network statistics to an application. The final version has evolved in several steps.

5.1.2.1 Base Version

To perform the identification, the characterization module maintains tables containing the target values for all parameters (dimensions) and applications. Periodically the collector module reads the data and calculates the parameters newly. Then the calculated values are compared with those in the tables. For every parameter the application with the minimum difference between the calculated and the stored value is chosen as a candidate. The application that is most often represented in that set of candidates is selected.

Although detection rates are quite good as shown in table 5.1, too many other wrong applications are still detected.

5.1.2.2 Enhanced Version

To reduce this problem another selection algorithm was introduced. The set of candidate applications is choosen as described in 5.1.2.1. But instead of simple majority, it is now necessary for an application to be selected as winner to have absolute majority. That is, more than half of all candidates indicate the same application. If there is no application that has absolute majority, the characterization module retains the decision of the last period and return *unknown* for that time slot. Detection rates of the algorithm are summarized in table 5.2.

Despite the improved detection rates, another factor was introduced. *Unknown* indicates how many times no decision could be made and the former decision was retained. This way a large part of the *unknown* decisions can be added to the *correct* ones.

Application	detected
Mozilla	79%
Video	81%
Radio	86%
SSH	88%
NFS	78%
FTP	100%

Table 5.1: Detection Rates Base Version

Application	detected	unknown
Mozilla	97%	61%
Video	94%	63%
Radio	99%	36%
SSH	92%	52%
NFS	99%	88%
FTP	100%	0%

Table 5.2: Detection Rates Enhanced Version

5.1.2.3 Final Version

To address the problem of the high *unknown* rate, another modification was introduced into the detection algorithm.

In table 5.4 it can be seen that for two applications, it is possible that key figures are very close to each other. In algorithms 5.1.2.1 and 5.1.2.2 only one candidate application may be selected for one parameter although other applications may be located very close.

To reduce this problem a level of uncertainity was introduced. All applications whose calculated value lie near the stored sample are also selected as candidates. Experiments have shown that a circle with a diameter of 1.2 times the distance between the closest candidate and the sample yields the best results.

A second modification was made to the selection algorithm. Whereas in algorithm 5.1.2.2 it was necessary to have absolute majority, it now suffices for one application to be selected to have two times more candidates in the set than the application with second most candidates. Detection rates are shown in table 5.3.

Application	detected	unknown
Mozilla	96%	36%
Video	97%	58%
Radio	99%	37%
SSH	98%	32%
NFS	99%	79%
FTP	100%	18%

Table 5.3: Detection Rates Final Version

5.1.3 Characterization Module Implementation

The characterization module was implemented in the Perl programming language. Perl was chosen, because it supports the rapid development of small modules and parsing of text files. But the characterization module is not language dependend and could be written in any programming language.

The characterization is composed of mainly four steps. First data is extracted from the log file, then the parameters are calculated. Based on these parameters applications are detected and the polling intervall of the network adapter is adjusted.

The first step, extracting the values from the log file is easily done in Perl. The logfile is broken up using the `split` command and is stored into an array.

Based on this array, active and inactive periods are determined. This task is executed in its own method. Slots of inactivity, where no data is transmitted can be identified as all their values are zero. The length of each period of inactivity or activity are counted and their history is stored in an array with 100 entries. From this array, the average length of inactive and active periods and their standard deviations are calculated. After the lengths of inactive and active periods, the parameters that are connected to the package sizes are calculated. Again this is done in an method of its own. Average packet sizes are calculated using the data of active slots only, so that packets with a size of zero do not alter the result.

Then applications need to be identified. For each parameter that was computed in the previous step all applications are examined and chosen as candidates if the stored value in their profile lies within the range around the calculated value. In the range algorithm (5.1.2.3) for every parameter several candidate applications are possible. All candidates are stored into an array. Then based on this array of possible candidate applications the running application is identified. The result of the identification can also be that no application has been detected and the value *unknown* is returned.

Finally the polling interval of the network adapter has to be adjusted to the policy for that application. If the application that has been detected in this slot is already running or the result of the identification was that no application could be identified the polling interval does not need to be altered. If in contrast a new application was detected, the interval needs to be adjusted. This is done using the `iwconfig` (section 3.3) operation. It is also checked whether the last change in the polling interval has happend more than 10s ago to prevent the card from frequently switching the polling interval if many applications are identified in a short period of time.

5.2 Parameter Mapping

Chapter 4.1 introduced the parameters that are used to identify applications. To obtain values for them, the applications (Mozilla, NFS operations, FTP client, Real Player, SSH session) were run on the test computer and trace files were recorded from the kernel. This was repeated for several beacon interval settings. The beacon intervals were: No power management and sleep intervals of 100, 200, 300, 400, 500, 750 and 1000ms. The trace files were analyzed afterwards and the key figures for all parameters were calculated.

The following values were used in the evaluation of the characterization module. They were computed using an analyzing tool that is similar to the characterization module but lacks the last stage and performs no identification. Instead it lists the values of each parameter. Like the characterization module the analysis tools computes values for every timeslot. To obtain one value for all timeslots the average is computed and used as key figure.

- The *average size of received packets* ranges from 58 bytes per packet for the Mozilla profile up to 1380 bytes for FTP transfers. The distribution of the key figures is relatively even and the distance between two key figures is at least 98.
- The *average size of transmitted packets* starts at 48 bytes for text based web pages and ends with 161 bytes for NFS operations. The Real Player audio stream, the video stream and the FTP client lie close together ranging from 54 to 55 bytes.
- The *ratio of inactive to active periods* ranges from 1.7 up to 31. The non interactive applications Realplayer (for both radio and video stream), FTP and NFS are grouped between 1.7 and 2.9. SSH has a ratio of 3.4 and the web browser 6.4 for web pages with many figures and 31 for pages that contain mainly text.
- The *ratio of average size of received to average size of transmitted packets* ranges from 1.91 up to 25.56. The values are distributed evenly across the whole range, only Mozilla (when viewing text pages) and NFS operations have almost the same ratio of 1.92 resp. 1.91.
- The *amount of transmitted bytes* ranges from 1.23 up to 60.76. SSH, Mozilla and NFS lie close to each other at 1.23, 1.8 and 1.84.
- The *standard deviation of average sizes of received packets* ranges from 67.26 to 246.60. Again, the values are distributed well, but the Real Player video and radio streams are hard to distinguish.

- The *standard deviation of average length of inactive periods* spans from 0.36 to 36.64. But it has to be noted that only Mozilla has values above 3. The rest of the applications is spread from 0.36 to 2.56 only.

The most significant parameters and their mapping to applications together with key figures is given in table 5.4.

parameter	**very small**	**small**	**medium**	**large**	**very large**
size of received packets [byte]	(< 250) SSH		(250-800) Browser, Radio, NFS	(800-1300) Video	(> 1300) Download
size of sent packets [byte]			(100-200) NFS		
ratio of the last two values	(< 12) SSH		(12-20) Radio, Video		(> 25) Download
ratio of inactive to active periods	(< 0.2) Download	(0.2-2) NFS	(2-6) SSH	(> 6) Browser	
standard deviation of the length of inactive periods	(0-2) Radio, Video			(> 30) Browser	

Table 5.4: Overview of application key figures

While one parameter on its own is unable to distinguish between two applications, a combination of them provides good detection rates.

Chapter 6

Evaluation

In the first part the characterization algorithm is analyzed offline. That means, that the traces were recorded first and afterwards in a second step the characterization module was run. In this test, no change in the polling interval takes place.

In the second part, the module is analyzed online. This means that while the application is running, the characterization module is analyzing the traces and adjusting the polling interval according to a predefined policy.

6.1 Offline Detection Rates

In the offline test, detection rates for the various applications are evaluated after having recorded the traces. The traces were analyzed afterwards by the characterization module. For the offline tests the characterization module was equipped with a mechanism to know in advance which application the trace belonged to. The module differentiates between three results:

1. Direct (positive identification based on key figures)
2. Unknown positive (positive identification, but only because no other is identified)
3. False (identification of a wrong application, either direct or indirect)

Figure 6.1 displays the results of the offline tests. The dark grey part of the columns means that an application has been identified directly while the light grey that the result of the characterization was unknown but still positive. The parenthesis denotes the polling interval for that test run. For all test runs two factors will be evaluated. They are *detection rate* and *certainty*. *Certainty* in this context does not mean that applications were characterized wrong, but it indicates the proportion of direct and unknown positive identifications.

The two Mozilla runs were of equal length, both of about 5 minutes. In testcase ”Mozilla 1” a web page that consisted mainly of pictures was viewed. Detection rates and certainty for that testcase are very high with 98% resp. 95%. The second Mozilla test case, with text web pages, has a lower certainty (47%) but equally high detection rate (96%).

Testcases ”NFS 1” and ”NFS 2” have a length of 7 and 9 minutes. In these testcases, a `find` command was executed on a directory that was mounted using NFS. The difference in certainty between 59% for the first testcase and 96% for the second test case cannot be explained with the change in the polling interval. Rather differences in the network environment should be responible. Actually certainty at *CAM* polling interval should be higher as the network card is not put to sleep. In testcases 3 and 4 the `chmod` was used to changed the access rights of a directory recursively. Detection rates and certainty of both test cases are at about the same level.

For the radio testcase, an audio stream was played for 12 minutes. Whereas detection rate is at 99%, certainty is only at 50%.

A terminal session on a remote computer was started for each of the two SSH testcases ”SSH 1” and ”SSH 2”. This session was used to perform administrative tasks on that computer like listing a directory, finding files and editing text files. While for both testcases roughly the same operations were executed, detection rate for the second testcase is at 96% and only at 62% for the first testcase. Certainty is at 39% for the ”SSH 1” and at 58% for ”SSH 2”. In a third SSH testcase, a X-Window program was started on the remote computer and forwarded to the local. The detection rate for this testcase (16%), can be explained with the change in the application profile (transfer of text information to transfer of X-Window information). A new profile describing X-Forwarding via SSH should lead to higher detection rates.

The video in the last testcase is only recognized at a rate of 53% and a very low certainty of 3%. In another testcase not displayed in the diagram, a video that was transmitted using a low bitrate was never characterized correctly. Instead the web radio profile was detected at a rate of 64% and a certainty of 15%. This might be an indicator that an application profile for low and high bitrate multimedia stream might produce better results than profiles for audio and video streams.

6.2 Online Detection Rates

In the test, the applications were executed sequentially. Approximately every 2 minutes the application was changed. The test started with browsing internet pages for two minutes. Following a terminal session on a remote computer was started using the SSH. After two minutes more, a `find` command was exectued on a directory that is located on a NFS share. The last two minutes realplayer was used to play a netradio transmission.

Figure 6.2 displays the energy trace, that was taken while executing the test. The test started with the Mozilla web browser. It was correctly identified execept for 14s starting at timestamp 74s. During this period the characterization module identified SSH. The following two applications SSH and NFS were identified correctly and no other applictations were found in this period. The

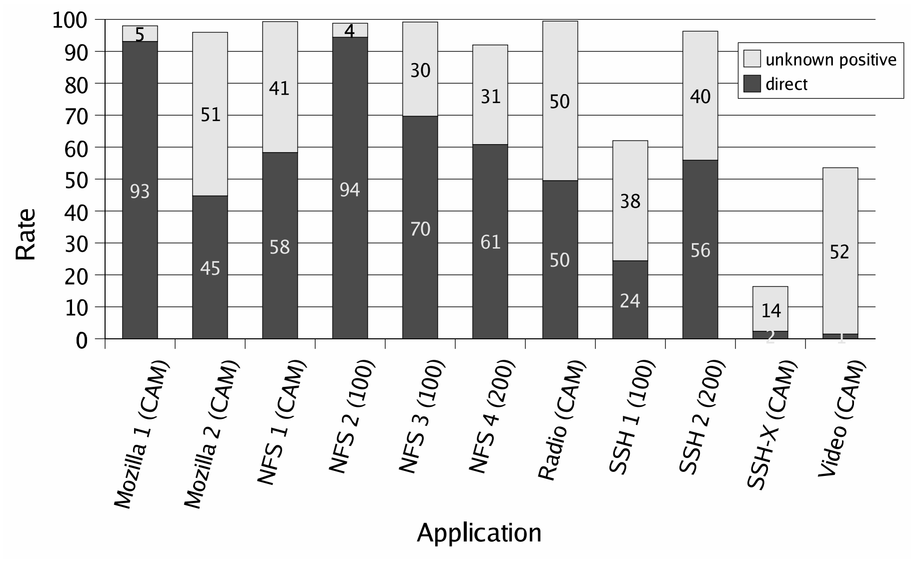

Figure 6.1: Offline Application Detection Rates

characterization module needed about 10s to realize that the application had changed and switch the profile. Starting at timestamp 394s the realplayer was activated. For the first 11s of its runtime Mozilla was identified falsely. But after this period the profile was switched to radio and was kept for the rest of the test.

6.3 Running two Applications in Parallel

Detection rates when two applications are running and particularly transmitting data over the network device concurrently are low. The characterization module was tested by running two applications in parallel.

In the first test, netradio and Mozilla were run in parallel. In this testrun both applications were detected, but the characterization mode switched frequently between them. In the second test, SSH and Real Player were run in parallel. This time, the characterization module could reach no decision.

As the trace data was extracted at network level, packets could not be associated with single applications, but the two or more applications running in parallel were treated as one, adding the network traffic.

If it is necessary that two applications which are running in parallel were recognized, a new profile that consists of both should be added. Another option would be to disable power management if the characterization module can reach no decision or to switch to a default profile.

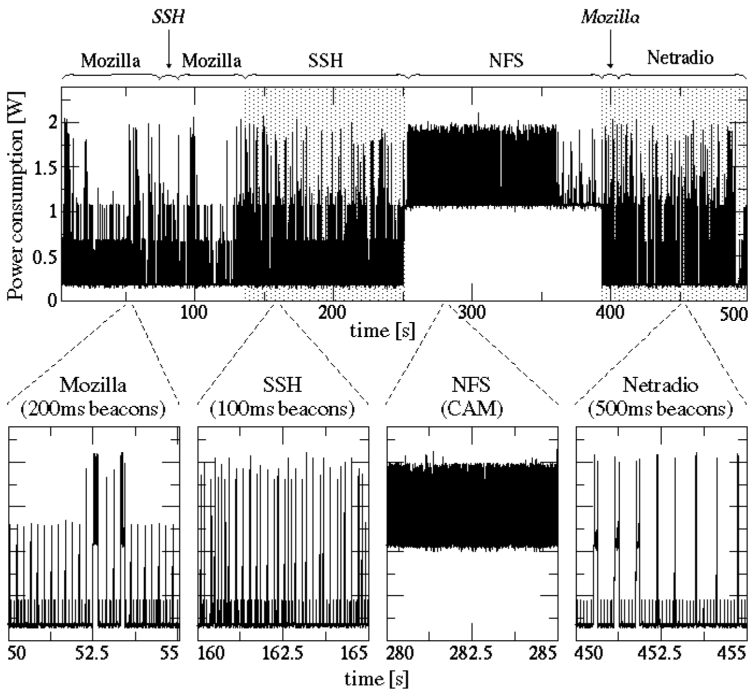

Figure 6.2: Online Application Detection

Chapter 7

Related Work

Several techniques have been proposed to reduce energy consumption of wireless network adapters. Generally works in the field of wireless power management follow two different approaches. One approach relies on applications that are cooperative in respect of power management and implement a special API to give hints about their intended behavior. The other tries to modify the transmission protocol so that less energy is needed.

7.1 Algorithms with Hints

The works of Flinn et al. [2] propose a self-tuning wireless power management. They introduce an API so that applications can give hints about their future behavior to the power management layer. Their API is composed of the following commands: `TransferHintBegin`, `ListenHintBegin`, `HintEnd`, `SetKnob` and `SetBasePower`. All hints are passed to an algorithm that decides whether power managemant can be used or not. Programs that want to use this application specific power management have to implement the API.

Kravets et. al [8] not only propose a API to make predictions about application behavior but also suggest a new transport layer protocol. While in 802.11 the control over a network cell and the transmission of data lies at the access point, Kravets argues the a client has more knowledge about application and therefore should decide about transmission windows.

7.2 Modifications of the Transport Protocol

The Bounded Slowdown Protocol [7] modifies the time the network adapter is awake after a packet is transmitted. The work is oriented at the *Round Trip Time* (RTT) of packets. Krashinsky and Balakrishnan introduce a factor p that represents an upper limit of delay. In the Bounded Slowdown Protocol two timeouts T_{awake} and T_{sleep} exist. After a packet is send, the adapter stays awake for T_{awake}. Then the adapter is sent to sleep and awakes after T_{sleep} to check for a response for his perviously sent packet. If no packets have arrived the card is again put to sleep, but this time T_{sleep} is extended by the factor p.

Bertozzi et al. [3] also propose improvements to the transport protocol. But instead of looking at round trip times they use TCP buffer occupancy to identify periods in which power management can be used. They identify two situations when the card can be put to sleep. The first situation occurs when the receive buffer is full and a *zero window* message is sent to the server, the card can be powered down until the *non-zero* message is sent as the TCP protocol takes care that no data is transmitted. The second situation is when the receive buffer is not utilized due to inactivity on the network device.

Chandra and Vahdat [4] examine in their work the influence of power management on several multimedia data streams. To enhance performance of multimedia applications they propose two new proxies, one at the server side of the wireless network and one at the client's side. The function of these proxies is to buffer larger amounts of data at the server side proxy and transmit them to the client side proxy. The client side proxy stores the data and delivers it to the client application. By sending the data in one large packet instead of several smaller ones, windows of inactivity are created and can be used to power down the network adapter.

Chapter 8

Future Work

8.1 Improvements in Usability

So far it is complicated to add new application profiles to the characterization module. Generally it is done in two steps. First, the application has to be run and traces for it have to be recorded. Then these traces have to be analyzed, parameters have to be computed and finally the characterization module has to be altered. This is not just time-consuming, but also prone to errors. As the source code of the characterization module has to be altered directly, only experienced users could do it. But more important, a trace that is recorded in a test run is likely to differ from one that is recorded during work.

A more user friendly method would be to maintain these tables with a dedicated user tool. This tool could have a start and an end of recording button and output the recorded and analyzed session in a description file containing the parameters. This description file could be the input of another tool that makes the new parameters available to the characterization module. In a first implementation a command line interface would be enough, but a more elaborated tool could use a GUI.

A key feature of any user configuration tool is that a user can customize the behaviour of the characterization module. A user must be able to set the speed (polling interval) of the network adapter. If the user feels that the default policy does not fit his needs, he should be able to alter it and adjust it to his needs. The configuration tool should also be able to activate and deactivate the characterization module and use the standard power management algorithm instead, either in CAM or PSP mode.

8.2 Optimization of Key Figures

In this work key figures were computed in a very simple manner. After calculating the key figures for every slot, the average was computed for all slots and this value is taken for the characterization module. Another method would be to take the median (element in the middle of the sorted array) of all values. Both have about the same detection performance. For the tests in this work manually tuned key values were used. As this involves a lot of work, an optimization algorithm to compute key figures from traces would be valuable. Such an optimization algorithm would have to consider all other applications and compute key figures so that the distance for the same figure but two applications is maximal.

8.3 Additional Sources of Information

For the characterization algorithm other sources of information would be possible. Network connections always include a port number. This port number is standardized for most applications and could be used as source for applications. Port 80 for example indicates the the network connections belong to a web browser. On UNIX systems the a mapping of port numbers to applications can be found `/etc/services`. Another source of information that is available and can be used is the TOS-flag in the IP header. This flag indicates the type of service the network packet is belonging to. For example `0x4` stands for ftp data and `0x8` for telnet (SSH) data [1]. This flag is set by most applications and is another hint that can be used and although not many variations are possible it is another hint.

A possible place for an implementation would be the netfilter layer. Netfilter is the firewall implementation for Linux and is already equipped with mechanisms to filter for ports or TOS-flags. The LOG-target that is currently used to write log messages to the command line or the log files could be altered to report to the characterization module. Such an implementation would have minimal overhead as only a new module for the kernel part of netfilter and a library module for the user part would be necessary.

Chapter 9

Conclusions

This work presents a new approach to power saving for wireless network adapters. Using a power/ performance tradeoff it offers more flexibility to the user than previous algorithms.

It was shown how data available to the kernel can be instrumented to identify network applications and adjust the existing IEEE power management so that it will be adequate for both application performance and power management.

Previous algorithms were targeted at the redesign of the transmission protocol or the network stack. This involves big overhead, as either the operating system or network applications had to be rewritten. The algorithm in this work is nonintrusive in the way that only minimal changes to the kernel have been made and none to the network infrastructure or applications. Instead it uses existing infrastructure and data whenever possible.

Based on data from the kernel driver of the network adapter several key figures are calculated and mapped to applications using profiles. It was shown that such profiles can be added easily. In addition a user may adjust the energy saving profiles with a speed knob to his own needs.

The algorihm was implemented and tested with several network applications. Their performance was tested in offline as well as online tests. Offline tests show that applications were identified successfully for more that 85% of the running time. In online tests applications were also characterized correctly for most of the time. The online tests show that wrong applications can be identified. But these false identifications were all corrected after a short period of time.

As the algorithm is oriented at the applications it outperforms all static algorithms that are implemented in network adapters.

Bibliography

[1] Philip Almquist. Rfc 1349 - type of service in the internet protocol suite. `http://www.faqs.org/rfcs/rfc1349.html`.

[2] Manish Anand, Edmumd B. Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MOBICOM '03)*, September 2003.

[3] Davide Bertozzi, Anand Raghunathan, Luca Benini, and Srivaths Ravi. Transport protocol optimization for energy efficient wireless embedded systems. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE 2003)*, March 2003.

[4] Surendar Chandra and Amin Vahdat. Application-specific network management for energy-aware streaming of popular multimedia format. In *Proceedings of the 2002 USENIX Annual Technical Conference*, June 2002.

[5] Cisco Systems, Inc. *Cisco Aironet Wireless LAN Client Adapters Installation and Configuration Guide for Linux*.

[6] IEEE Computer Society LAN MAN Standards Committee. *IEEE 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications*, August 1999.

[7] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the Eighth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM 2002)*, September 2002.

[8] Robin Kravets and P. Krishnan. Application-driven power management for mobile communication. *ACM/URSI/Baltzer Wireless Networks (WINET) special issue of Best Papers from MobiCom'98*, 1998.

[9] Axel Sikora. *Wireless LAN*. Addisson-Wesley, 2001.

[10] Jean Tourrilhes. *Linux Wireless LAN Howto*, August 2000.

Anwendungsspezifische Energiesparverfahren für WLAN

In den letzten Jahren hat die Verbreitung von Computernetzwerken immer mehr zugenommen. Mit der Entwicklung von Funknetzwerkkarten gewann auch der Aspekt eines Energiesparmodus immer mehr an Bedeutung, da viele der Geräte die Funknetzwerkkarten benutzen batteriebetrieben sind.

Funknetzwerkkarten, die den IEEE 802.11b Standard benutzen, besitzen bereits einen solchen Energiesparmodus. Dabei wird die Karte, wenn sie den Sparmodus benutzt, in regelmässigen Abständen aus einem Schlafzustand aufgeweckt um mit der Basisstation in Kontakt treten zu können. Diese Abstände können mit dem sogenannten ”Beacon Interval” verändert werden.

Der Energiesparmodus des IEEE Standards hat, wenn er aktiv ist, zwei Auswirkungen. Zum einen wird der Energieverbrauch der Funknetzwerkkarte deutlich gesenkt, zum anderen wird aber auch die Anwendungsausführung beeinträchtigt. Diese Beeinträchtigung ist abhängig von der Art der Anwendung. So sind Multimedia Anwendungen oft mit Mechanismen ausgestattet um kurzzeitige Paketverluste auszugleichen und der Energiesparmodus hat keine Auswirkungen auf die Anwendung. Andere Anwendungen hingegen, zum Beispiel das Netzwerkdateisystem NFS, werden sehr stark beeinträchtigt.

Diese Arbeit untersucht, wie verschiedene Netzwerkanwendungen von einander unterschieden werden können und wie der IEEE Energiesparmodus auf sie abgestimmt werden kann. Dazu werden mit Hilfe von Informationen über gesendete Datenpakete aus dem Linux Kernel verschiedene statistische Kennzahlen entwickelt. Es wird gezeigt, dass diese Kennzahlen hinreichend eindeutig sind um verschiedene Programme von einander unterscheiden zu können. Für eine Menge von Anwendungen werden solche Kennzahlenprofile erstellt und erklärt. Die Profile enthalten neben den Kennzahlen auch noch eine für die Anwendung geeignete Einstellung des Energiesparmodus.

Basierend auf den Anwendungsprofilen wird ein Algorithmus vorgestellt, mit dessen Hilfe Anwendungen unterschieden werden können. Dazu werden zur Laufzeit errechnete Kennzahlen mit den Anwendungsprofilen verglichen. Diejenige Applikation, deren Laufzeitkennzahlen am nähesten bei den vorher berechneten liegen, wird von dem Algorithmus erkannt. Anschliessend wird der Energiesparmodus der Funknetzwerkkarte noch wie im Profil gespeichert eingestellt.

Auf diese Weise verliert der Energiesparmodus des IEEE Standards seinen statischen Charakter und kann dynamisch an verschiedene Applikationen angepasst werden. Dabei betrachtet der Algorithmus sowohl den Energieverbrauch als auch die Anwendungsqualität (Performance) und setzt

den Energiesparmodus so, dass ein Kompromiss zwischen den beiden Grössen gefunden wird.

Abschliessend werden die Ergebnisse der Arbeit in zwei Tests ausgwertet. Zuerst werden in Tests Log-Dateien ausgewertet um die Erkennungsraten für eine Anwendung zu ermitteln. Danach wird in Laufzeit Tests die Zeitdauer zwischen dem Starten der Anwendung und ihrer Erkennung, als auch der Einfluss von falschen Erkennungen untersucht.

In Zukunft könnte der Erkennungsalgorithmus um weitere Datenquellen erweitert werden. Mit Hilfe einer Schnittstelle zur Paketfilter Implementierung des Linux Kernels liesen sich zum Beispiel die IP-Port Adresse oder die Type-of-Service Markierung der Datenpakete gewinnen. Diese Informationen unterscheiden sich ebenfalls bei verschiedenen Anwendungen und sie können als zusätzliche Hinweise auf eine Anwendung genutzt werden.