Application Characterization for Wireless Network Power Management

Andreas Weissel, Matthias Faerber and Frank Bellosa

University of Erlangen, Department of Computer Science 4
{weissel,faerber,bellosa}@cs.fau.de

Abstract. The popular IEEE 802.11 standard defines a power saving mode that keeps the network interface in a low power sleep state and periodically powers it up to synchronize with the base station. The length of the sleep interval, the so called beacon period, affects two dimensions, namely application performance and energy consumption. The main disadvantage of this power saving policy lies in its static nature: a short beacon period wastes energy due to frequent activations of the interface while a long beacon period can cause diminished application responsiveness and performance. While the first aspect, reduction of power consumption, has been studied extensively, the implications on application performance have received only little attention. We argue that the tolerable reduction of performance or quality depends on the application and the user. As an example, a beacon period of only 100 ms slows down RPC-based operations like NFS dramatically, while the user will probably not recognize the additional delay when using a web browser. If at all, known power management algorithms guarantee a system wide limit on performance degradation without differentiating between different application profiles.
This work presents an approach to identify on-line the currently running application class by a mapping of network traffic characteristics to a predefined set of application profiles. We propose a power saving policy which dynamically adapts to the detected application profile, thus identifying the application- and user-specific power/performance trade-off. An implementation of the characterization algorithm is presented and evaluated running several typical applications for mobile devices.

1 Introduction

We present an approach to on-line characterization of applications based solely on information obtained from the network link layer. This information is used to realize dynamic power management for mobile communication which aims at maximizing power savings without degrading application dependent, user-perceived performance and quality.

1.1 Motivation

For wireless networking, the IEEE 802.11 standard defines two operating modes: the continuously-aware mode *CAM*, which leaves the interface fully powered, and the power saving mode *PSP*, which keeps the network interface in a low power sleep mode and periodically activates it to synchronize with the base station. These periodic synchronizations are called *beacons*, the length of the sleep interval is the so called *beacon period* with a default value of 100 ms. Some network interfaces support additional power saving modes, e.g. *PSPCAM* or *PSP-adaptive* which automatically switch between the two modes depending on the network traffic. While this mechanism is defined for managed as well as ad-hoc networks, for reasons of simplicity, we concentrate our discussion on a configuration with one base station, the *access point*, communicating with at least one client.

The beacon period affects two dimensions, namely performance by generating network delays and energy savings. While this policy dramatically reduces the time the network interface has to be fully powered, receiving data is only possible after synchronizing with the base station. Incoming traffic is buffered at the access point and signalled in the "traffic indication map" (TIM) sent at each beacon. If data is waiting, the client activates the network interface and polls the data from the access point. After the transmission, the sleep cycle is established again. If data is buffered at the access point, the client is not aware of the incoming data for at most one whole beacon period. Thus, additional network delays are introduced and user-perceived application performance or quality can be affected.

The amount of performance or quality degradation depends not only on the beacon period, but also on the type of application, the send/receive characteristics and the user sensitiveness and tolerance. This aspect is often neglected by power management algorithms. Typically, only a system wide performance level is guaranteed without differentiating between different application profiles. As performance or latency is a subjective measure, the leverage of these factors differs from application to application and from user to user. For interactive processes, small delays are usually tolerated depending on the application and the sensitivity of the user. Performance degradation should be completely avoided for jobs that the user wants to finish as fast as possible, e.g. a download operation. Streaming applications like a radio or video player have to provide a certain quality of service which can not be achieved if the additional network delays caused by power management exceed the playtime of the buffered data. We argue that the tolerable reduction of performance or quality depends on the application and the user, and therefore has to be considered by operating system power management policies.

1.2 Contributions of this Work

An approach to on-line characterization of networked applications is presented. Several parameters derived from send/receive statistics are mapped to a predefined set of application profiles. The necessary data is already available in the kernel network stack and the computational overhead to determine the parameters is low. If a profile is identified with enough certainty, an application-specific power management setting is triggered.

We examine the characteristics of typical networked applications for mobile devices and identify their power/performance trade-offs. Alternative approaches to characterize applications are outlined and discussed.

The application profiles are characterized by several different parameters regarding network traffic, e.g. the proportion of data received to data sent, the average length of idle or active periods, the deviation of these values etc. A secure shell (SSH) session, e.g., can be identified by rather small packets together with short active and long passive periods while an audio stream can be recognized by periodic transmissions, i.e. a small deviation from the average length of periods of inactivity. Information on packet timing gets coarser with longer beacon intervals due to the increasing delays in receiving packets. For the range of beacon intervals considered in this work (up to 500 ms), the chosen parameters are quite robust and show only little variation over different delays.

The target parameters for the set of profiles are identified using numerous traces from application runs with different power management settings. The characterization is evaluated with another set of recorded traces and with extensive on-line tests including power management. We show that applications running in isolation are identified correctly. If the user works with two programs that generate network traffic simultaneously, the algorithm does either detect no profile at all or identifies both correctly, but switches frequently between the two.

The application dependent power management decision is configurable from user space and can be extended with more sophisticated power management algorithms.

We present related work in the following section. Our approach to application characterization is outlined in section 3. Section 4 describes the implementation in detail; followed by an evaluation of the characterization algorithm in section 5.

2 Related Work

2.1 Application Characterization

The operating system can be identified by analyzing the network traffic originating from it [11]. The IP implementation slightly differs from operating system to operating system, depending on RFC interpretation. The TCP SYN packets provide enough information to accurately determine the system.

To gather more information about the system and available services on the network, the packet payload can be analyzed. With tools like `ngrep`, a regular expression based analysis of network traffic is possible.

A straightforward and simple approach to identify applications is to use the port number and the protocol (TCP, UDP) from the headers of network packets. Unfortunately, ports can easily be mapped to or tunnelled through other ports. Firewalls often restrict connections to only a few open ports, e.g. port 80 for HTTP and 22 for SSH. To enable networked applications based on other ports to run, tunnelling of connections has become a common technique. A proxy (caching) server outside of the firewall serves not only HTTP requests, but also Multimedia streams. Identification using this method is also problematic with applications that use dynamically assigned ports, such as FTP and RPC. For all these cases, the proposed technique can complement the simple method of mapping port numbers to applications.

A more sophisticated method is to look at the contents of the packets. By reassembling the packets and by recognizing certain patterns the application can be identified from the contents of the data stream. This classification can be made without relying on the port numbers. Projects like “l7-filter” [10] classify packets based on patterns in layer 7 (application layer) at the cost of high processor utilization. The overhead of packet introspection can negatively affect power consumption.

2.2 Power Management for Wireless Networking

Stemm et al. [12] and Feeney at al. [6] investigate the energy consumption of wireless network interfaces and different network protocols in detail. Power management policies can be classified into three categories.

2.2.1 Static Protocols

Static protocols use one fixed, system wide beacon period, time-out value or inactivity threshold to trigger transitions from active to low power modes with reduced performance. The IEEE 802.11 power management algorithm is a typical representative of this class. Static protocols are often implemented in hardware because they are simple and do not require much storage space or computational effort.

2.2.2 Adaptive Protocols

Dynamic link-layer protocols adapt the beacon period or time-out threshold to the current usage characteristics of the device. These algorithms are often called *history based* as they draw upon the observed device utilization of the past.

Krashinsky and Balakrishnan present the Bounded Slowdown (BSD) protocol [8]. This power management protocol minimizes energy consumption while guaranteeing that the round trip time (RTT) does not increase by more than a predefined factor p over the RTT without power management. The factor controls the maximum percentage slowdown, defining the trade-off between energy savings and latency. If at time t_1 the network interface has not received a response to a request sent at time t_0, the interface can switch to sleep mode for a duration of up to $p(t_1 - t_0)$; i.e., the RTT, which is at least $t_1 - t_0$, will not be increased by more than the factor p. Thus, the beacon period is dynamically adapted to the length of the inactivity period. When data is transmitted, the wireless interface is set to continuously-aware mode. As this approach requires only information available at the link layer, it can be implemented in hardware.

Chandra examined the energy characteristics of streams of different multimedia formats, namely Microsoft Media, Real and Quicktime, received by a wireless network interface and under varying network conditions [3]. A simple history based policy is presented which predicts the length of the next idle phase according to the average of the last idle phases. As Microsoft Media exhibits regular transmission intervals, high energy savings can be achieved using this policy.

Chandra and Vahdat [4] propose energy aware traffic shaping for multimedia streams in order to create predictable transmission intervals. Varying the transmission periods reveals a trade-off between frequent mode transitions and added delays in the multimedia stream reception. Traffic shaping can be performed in the origin server, in the network infrastructure or in the access point itself. Regular packet arrival times enable client side mechanisms to effectively utilize the low power sleep state of the wireless interface.

The approach of Chandra and Vahdat addresses the trade-off between energy savings and performance, but is limited to streaming applications. As traffic shaping is performed at the server, user-specific preferences can not be taken into account. Application-specific server side traffic shaping and client side power management should add to each other nicely.

Several proposals for energy efficient transport layer protocols can be found in the literature. Bertozzi et al. show that the TCP buffering mechanism can be exploited to increase energy efficiency of the transport layer with minimum performances overheads [2].

2.2.3 Application-specific Protocols

These protocols require the support of applications to enable operating system power management. To achieve this, applications typically have to use a certain API to inform the operating system about their intended use of the network interface.

Anand et al. [1] propose STPM: self-tuning wireless network power management. STPM considers the time and energy costs of changing power modes. Applications can provide hints to the operating system about the time and volume of data transmission over the network interface. STPM dynamically adapts its power management policy.

Kravets and Krishnan propose a power management algorithm which shuts down the wireless interface after a certain period of inactivity and reactivates it periodically [9]. Variations of the algorithm with fixed and variable sleep periods are evaluated. Predictive algorithms are proposed to determine the length of the sleep periods. An application level interface to the power management protocol allows applications to control the policies used for determining sleep durations. Predictive algorithms can be used to adjust the power management parameters (inactivity time-out and sleep duration) using application-specific strategies. An implementation of a simple adaptive algorithm is presented which responds to communication activity by reducing the sleep duration to 250 ms and to idle periods by doubling the sleep duration up to 5 minutes.

Energy aware adaptation as presented by Flinn [7] is another approach to application dependent power management. By dynamically modifying their behavior, applications can reduce system power consumption, e.g. to achieve a specific battery lifetime. The operating system monitors the battery status, selects the correct power/quality-of-service trade-off and informs the currently running applications, guiding their adaptation.

While application-specific power management strategies address the power/performance trade-off, applications are required to be rewritten to support the operating system in its efforts to save energy. Our approach circumvents this drawback by providing the necessary information separate from the applications and, at the same time, allowing not only application-, but also user-specific power management policies.

3 Application Characterization

We examined the network characteristics of several typical applications for mobile systems (laptops and hand-helds):

- Mozilla (web browser)
- Secure Shell (SSH) session
- Network File System (NFS) operations

- FTP download
- Netradio: low bandwidth Real audio stream
- MP3 audio stream
- high bandwidth Real video stream

These applications can roughly be divided into three groups: interactive, foreground applications (web browser and SSH), non-interactive applications where execution time is key (network file system, download) and streaming applications.

With interactive applications, the user is sensitive to reduced application responsiveness. Small additional network delays may be tolerated or probably not even recognized. Thus, power management with small beacon intervals would be an option for these applications. Figure 1 shows the energy consumption of a five minute run of Mozilla under different power management settings. Power management increases the round trip time of network packets by an average of (beacon period / 2). The sensitivity to application responsiveness can not be measured directly; it depends on the individual user.

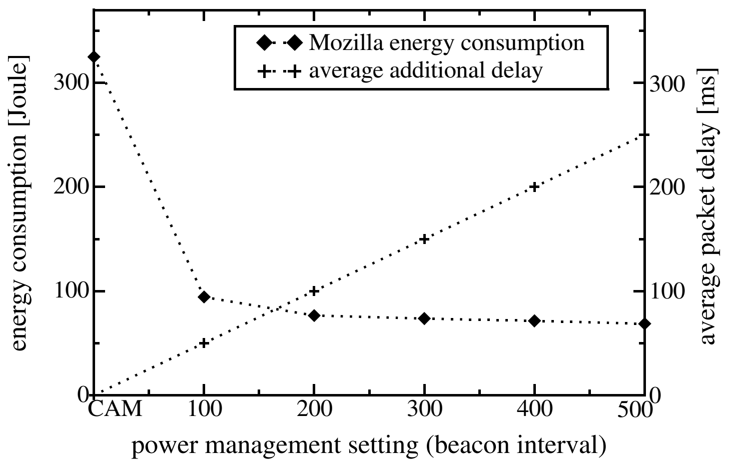

Figure 1: energy consumption of Mozilla and average packet delays under different power management settings

As Anand et al. [1] demonstrate, power management can dramatically increase the execution time and, as a consequence, the energy consumption of NFS. The same applies to a download or copy operation over the network. It is important to identify these applications and switch off power management to avoid wasting energy. Figure 2 shows the runtime of a simple find operation on a directory mounted via NFS. Without power management, the job finished after a few seconds and consumed approximately 4J. For a beacon period of 100ms, the runtime increases to over 250 seconds and the energy consumption to 67J. For all network services based on RPC, as well as copy or download operations, power management should be disabled.

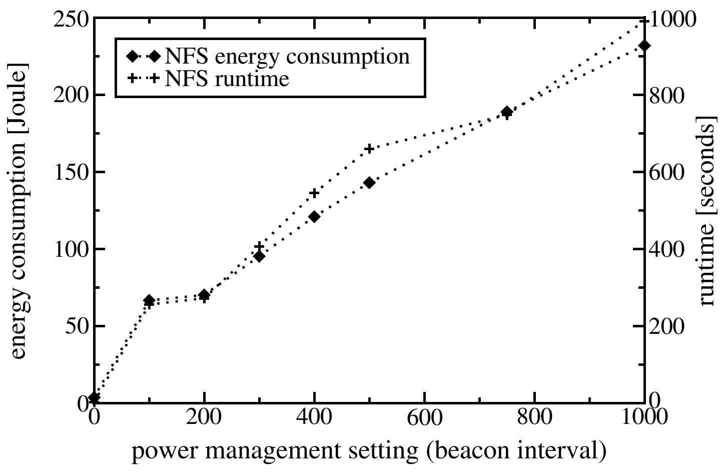

Figure 2: energy consumption and performance degradation of NFS

Streaming applications buffer a certain amount of data to reduce the impact of network delays or varying bandwidth. As a consequence, they are insensitive to small delays introduced by power management algorithms. In case of high delays, the quality of the presentation, e.g. the number of frames per second, is reduced, pauses are introduced or frames are skipped. In our experiments, a beacon interval of 500ms can be used without influencing a 200kbit video stream.

In order to achieve energy savings, not only the power consumption but also the runtime of the task has to be considered. The beacon mechanism reduces the average power consumption but introduces additional delays when receiving packets. This leads to an increased runtime for certain types of networked applications, e.g. RPC-based and copy or download operations. The resulting higher energy consumption can outweigh the savings due to the reduced average power. In contrast to this, for many interactive and streaming applications, the runtime is mainly determined by the user think time or other factors. In these cases, reduced power consumption will lead to reduced energy consumption because the runtime is not or only marginally influenced by the power management algorithm.

To sum up, the performance or, more generally, the quality-of-service degradation a user is willing to tolerate depends on the currently running application and the user expectations. Therefore, we propose a power management policy which incorporates user preferences into its decisions. Different application profiles are identified on-line and the user-defined, application-specific power management setting is chosen.

3.1 Application Profiles

In order to be able to distinguish different application profiles during runtime, we examined several different parameters identifying the network characteristics in a single user system. At the link layer, information about the number of packets and the size of

CAM	continuously-aware mode (no energy savings)
100 ... 500	IEEE 802.11 power management mode The beacon interval was set to the following values: 100ms, 200ms, 300ms, 400ms and 500ms

Table 1: power management settings for parameter training

parameter	**very small**	**small**	**medium**	**large**	**very large**
size of received packets [byte]	(< 250) SSH		(250–800) Browser, Stream, NFS	(800–1300) Stream	(> 1300) Download
size of sent packets [byte]	(0–50)		(50–100)	(100–200) NFS	(> 200)
ratio of the last two values	(< 12) SSH		(12–20) Stream	(20–25)	(> 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) Stream		(2–30)	(> 30) Browser	

Table 2: characterization of different profiles

the packets sent or received is available. Using these values, we constructed the following parameters:

1. average size of packets received (= number of packets received / bytes received)
2. average size of packets sent
3. average length of inactive periods (time intervals with no transmissions)
4. average length of active periods (time intervals with transmissions)
5. traffic volume received
6. traffic volume sent

We collected traces for every application under six different power management settings (see table 1) and computed the parameters presented above, together with the averages and standard deviations. In addition to that, we experimented with several ratios and combinations of the values. We decided to drop several parameters that showed high deviations or low correlation to the corresponding applications. The final, reduced set consists of the following parameters:

1. average size of packets received
2. average size of packets sent
3. ratio of average length of inactive to length of active periods

4. ratio of average size of packets received to size of packets sent
5. ratio of traffic volume received to traffic volume sent
6. standard deviation of the average size of packets received
7. standard deviation of the average length of inactive periods

Table 2 shows the highest correlations of parameters to application profiles. The numbers in brackets depict the range of parameter values. With some parameters, all profiles can be distinguished while others, e.g. the size of sent packets, can only be used to confirm classifications.

4 Implementation

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.

4.1 Collector Module

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

4.2 Characterization Module

The user space part of the characterization algorithm performs a mapping of the network statistics to application profiles. To perform the identification the module maintains tables containing the target values for all parameters and all applications. Periodically, these parameters are extracted from the network traffic statistics and compared to the parameters of the training runs. For every parameter, the application with the minimum difference between the current and the target values is chosen as a candidate. From the set of candidates, an application is only selected if it has the majority, i.e. if at least four of the seven candidates indicate the same. If no majority is reached, the decision is considered uncertain and the algorithm stops, i.e. the last identification is retained.

We measured the overhead of mode transitions of a Cisco Aironet wireless adapter (table 3). It can be seen that changing the beacon interval and the operating mode is equally expensive. In order to avoid frequent mode transitions which would prevent the interface from achieving any energy savings, we introduce a minimum time span t_{wait} between two mode transitions. In our experiments, this time span was set to 10 seconds.

4.3 Profile Management

The corresponding power management settings for the different application profiles are read from a configuration file or can be specified as command line parameters when

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

Table 3: overhead of mode transitions

running the characterization module. This way, varying preferences or preferences of different users can be taken into account by, e.g., switching to another set of power management settings when a new user logs into the system. A user who wants to change the currently active settings just needs to invoke the characterization module with the new set of parameters as command line attributes. It is possible to activate and deactivate the characterization module and use another power management algorithm instead, e.g. PSPCAM.

4.4 Future Work

We plan to integrate additional information available on the network and transport layer into our decision rule. Examples are the port number of a network connection, the protocol used (TCP or UDP) and the user id of the receiving or sending process. This information can support a decision based purely on link layer statistics. With the port number, some applications can be identified immediately. However, if the original port is mapped to port 80 due to firewall restrictions, this information is not available. The user id enables a distinction of user-specific application profiles or different power management policies for different users.

Statistical data stored in the task structure of the receiving or sending process, e.g. the runtime or priority, could be used to distinguish between interactive and non-interactive, background processes, similar to the approach taken in typical schedulers.

We would also like to investigate a combination of application-specific server side traffic shaping, like the approach presented by Chandra and Vahdat [4], with application-specific power management on the client side.

5 Evaluation

5.1 Data Acquisition

For power measurements we used the Cisco Aironet wireless adapter [5], connected to a notebook via a PC Card extender card from Sycard Technology. The extender card allows to isolate the power buses, so we attached a 4-terminal precision resistor of 50 mOhm to the 5 V supply line. The voltage drop at the sense resistor was measured with an A/D-converter with up to 40000 samples per second and a resolution of 256 steps. The maximum voltage drop that is correctly converted is 50 mV.

Figure 3 shows the power consumption of different power management settings of the Cisco Aironet card.

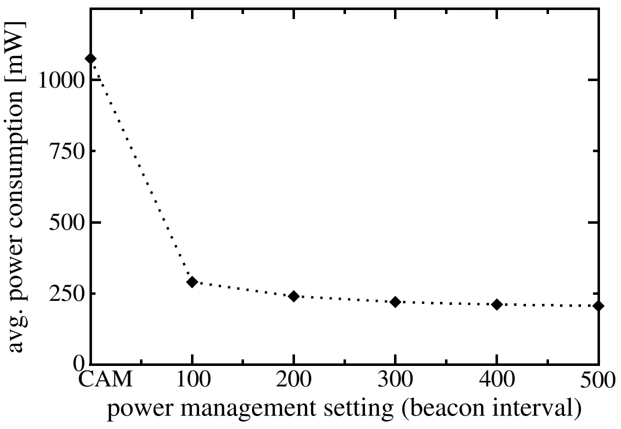

Figure 3: average power consumption of different power management settings

5.2 Off-line Application Characterization

We determined parameters for the following application profiles:

- Web browser/text
- Web browser/images
- NFS operations (kernel compile run, find)
- SSH (terminal session)
- FTP download
- Low-bandwidth stream (< 64 kbit, audio, Realplayer and MP3)
- High-bandwidth stream (video)

The two web browser profiles are treated as one application, as well as the different types of streams.

In order to evaluate the characterization algorithm, we recorded several traces of application runs with two different users. Figure 4 shows the percentage of time the algorithm identifies the correct application, reaches no decision (keeping the correct decision) and performs a wrong classification. The last value is determined counting identifications of other profiles and the time that follows until the correct application is identified again.

"Mozilla 1" is a browser session viewing web pages with many pictures, in "Mozilla 2" mainly text is viewed. SSH with X-forwarding shows a high error rate; in this case, an additional application profile should be created. During the "Mozilla 2" run, SSH was detected several times. As both are interactive applications they could also be combined to one profile. When combining Mozilla and SSH to one profile (which could be called "Interactive"), the percentage of time this profile is correctly classified is almost 100%. The two NFS sessions represent different file system operations (`find` & `grep` and `chmod`). Different streaming applications and download operations are identified correctly.

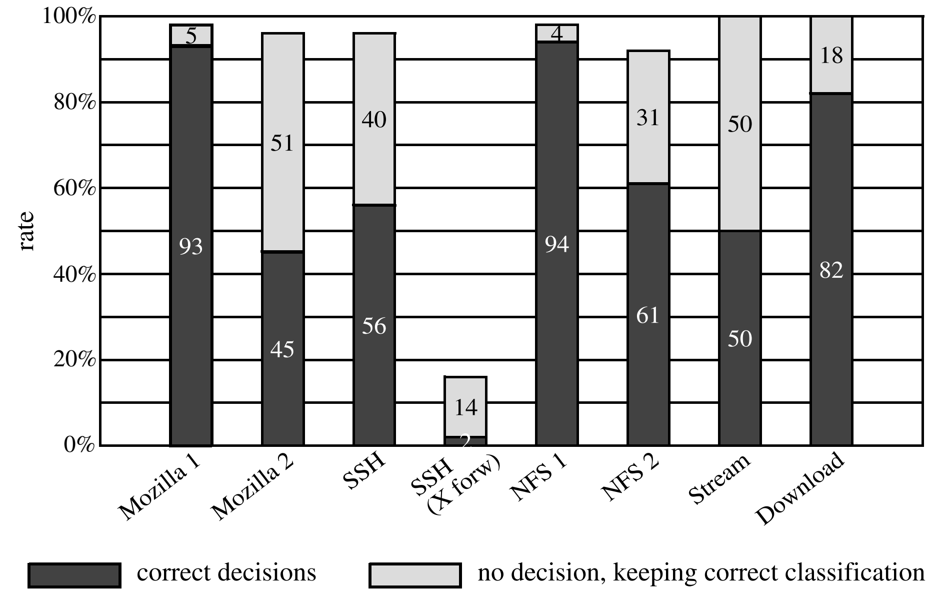

Figure 4: off-line application characterization

5.3 On-line Application Characterization

In order to be of practical use, our system has to characterize running applications correctly. We performed several on-line tests with a mix of networked applications and recorded the decisions of the characterization module and the power consumption of the system. Figure 5 shows an 8 minute run of four different programs: Mozilla, an SSH session, `find` on a NFS-mounted directory and Netradio (low-bandwidth Real audio stream). Approximately every 2 minutes the user performing the test switched to another application. The characterization module changes the power management setting according to the identified application. For Mozilla, we chose a beacon period of 200 ms, for SSH 100 ms and for Netradio 500 ms. When running NFS, we activated the continuously-aware mode (CAM). The figure shows the power consumption during the test run together with the detected applications. Mozilla (time stamp 0–136 s) is identified except for a short period between 74 and 88 s. The following two application profiles are recognized correctly (SSH: 136–255 s and NFS: 255–394 s). At timestamp 394 s the user activates the Real audio stream; however Mozilla is detected. The characterization module needs 11 seconds to correctly identify Netradio. This profile is kept until the end of the test.

In order to evaluate the system, we recorded the decisions of the characterization module during several test runs of over 70 minutes. For these tests, we used two different power management settings: CAM mode for NFS and download/copy operations and PSP mode with a beacon period of 100 ms for other application profiles.

If the user switches to a different application, the algorithm needs some time to recognize this change. We determined the length of these *recognition delays* and the time

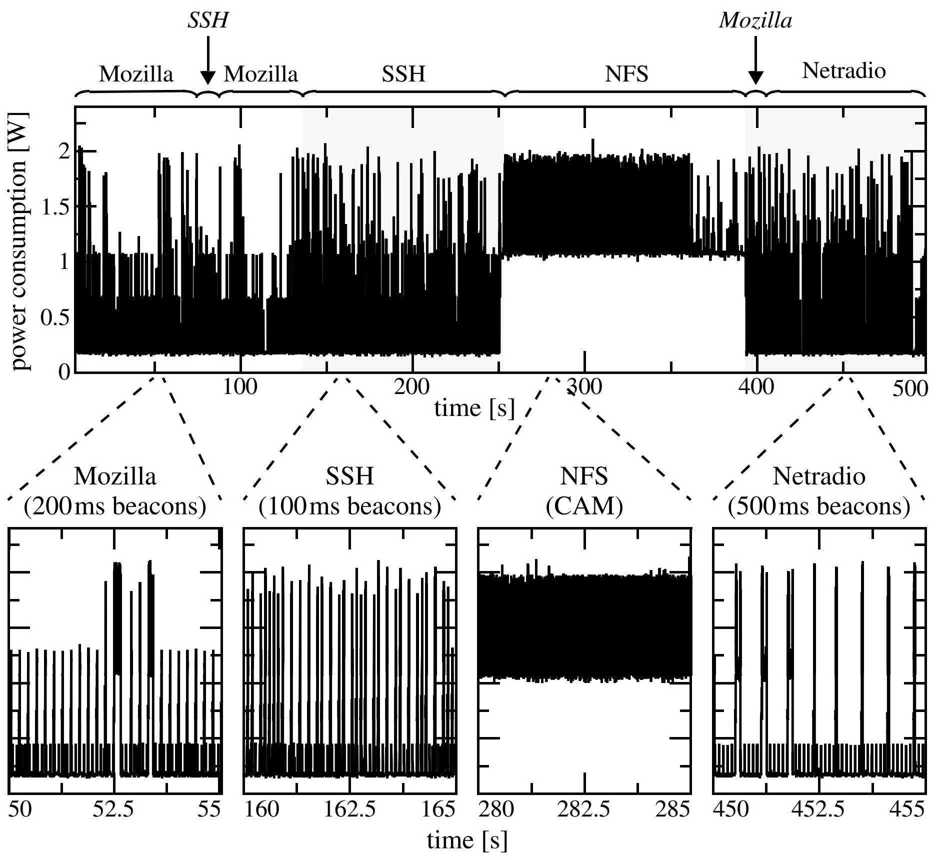

Figure 5: power consumption during a run of four different applications

intervals during which wrong decisions are taken, i.e. during which the running application is not correctly detected. In these cases, the user either experiences delays (if the power management setting is more aggressive than it should be) or energy is wasted (if the power management setting could be more aggressive). We determined an error rate of 6.5% over all test runs, composed of 4.5% wrong classifications and 2% recognition delays.

5.4 Running Applications in Parallel

In order to evaluate our characterization algorithm when a mixture of application profiles is observed, we performed tests where two applications run in parallel. The first scenario is a run of Netradio and Mozilla, the user browsing the web while listening to radio. In this case, both applications are recognized, but the algorithm switches frequently between the two (almost every 10 seconds). A solution would be to choose the application profile for which the user is more sensitive to network delays (the application with the smaller beacon interval or with power management disabled).

Our second test is a SSH session with Netradio running in the background. This time, the characterization algorithm does not reach a decision for the whole test run. If no profile is detected for a certain period of time, the algorithm could disable power management, switch to the card's own power management mechanism (PSPCAM) or activate a user-specified default setting.

5.5 Comparison with PSPCAM

In order to compare the energy savings achieved by our characterization approach to the card's own adaptive power management algorithm, PSPCAM, we recorded the power consumption of test runs with several applications under different power management settings. During periods of network activity, PSPCAM stays in CAM mode. After a short period of inactivity (approx. 2 seconds) the card transitions to PSP mode. This transition comes, in contrast to "manual" (de)activation, with almost no overhead (energy and time). Table 4 shows the energy consumption of 5 minute test runs. It can be seen that for all applications which are robust to the beacon mechanism, that are all except NFS, more energy can be saved in PSP than in PSPCAM mode. As a consequence, a power management algorithm which distinguishes different profiles can achieve higher energy savings than an energy aware link-layer protocol. When running e.g. Netradio, PSPCAM would unnecessarily deactivate power management during data transfers, although the audio stream works equally well when the beacon mechanism is active.

app. / mode	NFS	SSH	Netradio	Mozilla
CAM	250.0J	214.6J	215.4J	213.0J
PSPCAM	225.9J	97.6J	169.3J	72.4J
PSP (100ms)	62.9J	21.4J	75.9J	65.5J

Table 4: energy consumption over 5 minutes

6 Conclusion

We argue that power management for mobile communication has to take the application- and user-specific power/performance trade-off into account. This work presents an approach to on-line characterization of applications, enabling application-specific power management. A small set of parameters based on network traffic is mapped to predefined application profiles. We present an implementation with low computational overhead and evaluate it under several typical applications for mobile devices. Higher energy savings can be achieved when exploiting the tolerable performance degradation of different profiles than using the card's own power management mechanism.

This approach does not require source code modifications to programs. The user can specify individual power management settings for all application profiles. Thus, not only application- but also user-specific power management can be realized, addressing the power/performance trade-off of the individual user.

References

[1] ANAND, M., NIGHTINGALE, E. B., AND FLINN, J. Self-tuning wireless network power management. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MOBICOM '03)* (September 2003).

[2] BERTOZZI, D., RAGHUNATHAN, A., BENINI, L., AND SRIVATHS, R. Transport protocol optimization for energy efficient wireless embedded systems. In *Proceedings of the Design, Automation and Test Conference in Europe (DATE '03)* (March 2003).

[3] CHANDRA, S. Wireless network interface energy consumption implications of popular streaming formats. In *Proc. Multimedia Computing and Networking (MMCN'02)* (January 2002).

[4] CHANDRA, S., AND VAHDAT, A. 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 350 Series Client Adapters, June 2003.

[6] FEENEY, L., AND NILSSON, M. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of IEEE Infocom* (April 2001).

[7] FLINN, J., AND SATYANARAYANAN, M. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th Symposium on Operating System Principles SOSP'99* (December 1999).

[8] KRASHINSKY, R., AND BALAKRISHNAN, H. 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).

[9] KRAVETS, R., AND KRISHNAN, P. Application-driven power management for mobile communication. *ACM/URSI/Baltzer Wireless Networks (WINET) special issue of Best Papers from MobiCom'98* (1998).

[10] LEVANDOSKI, J., SOMMER, E., AND STRAIT, M. Application layer packet classifier for Linux.

[11] SPITZNER, L. Passive fingerprinting. In *SecurityFocus* (2000).

[12] STEMM, M., AND KATZ, R. H. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEEE Transactions on Communications E80-B*, 8 (1997), 1125–31.