Publication V

Yu Xiao, Wei Li, Matti Siekkinen, Petri Savolainen, Antti Ylä-Jääski, Pan Hui. Power Management for Mobile Devices Using Complex Event Processing. ***Aalto University publication series SCIENCE+TECHNOLOGY Aalto-ST 26/2011*****, 1-27, 2011.**

Abstract

Energy consumption of wireless data transmission, a significant part of the overall energy consumption on a mobile device, is context-dependent - it depends on several internal and external contexts, such as application workload and wireless signal strength. In this paper, we propose an event-driven framework that can be used for efficient power management on mobile devices. The framework adapts the behavior of a device component or an application to the changes in contexts, defined as events, according to developer-specified event-condition-action (ECA) rules that describe the power management mechanism. In contrast to previous work, our framework supports complex event processing in addition to simple event processing. By correlating events, complex event processing helps to discover complex events that are relevant to power consumption. Using our framework developers can implement and configure power management applications by editing event specifications and ECA rules through XML-based interfaces. We evaluate this framework with two applications in which the data transmission is adapted to traffic patterns and wireless link quality. These applications can save roughly 12% more energy compared to normal operation.

1 Introduction

Energy consumption caused by wireless data transmission has become a significant component of overall energy consumption on mobile devices, due to the increasing popularity of mobile Internet services. Solutions that can improve the energy efficiency in data transmission are therefore very much needed for improving the battery life of mobile devices. In this paper, we focus on power management software for wireless data transmission on a single device. This software controls the behavior of device components and applications in order to reduce transmission cost.

The effectiveness of power management for wireless data transmission depends on how well it can adapt the behavior to the workload of transmission and the situation in which the transmission happens. A key challenge is the detection of situational variation since the situations include many factors such as the state of the mobile device, the environment of the wireless network to which the mobile is connected to, and the patterns of the traffic generated by mobile applications.

Context has been widely used for describing the state of an entity, such as a person, a device, an application and a network. For example, signal-to-noise ratio (SNR) is a measure of wireless link quality, and the distribution of packet intervals reflects the burstiness in traffic. If the changes in contexts are defined as *events*, complex event processing [13] techniques can be applied and the situational variation can be modeled as complex events. This enables designing power management software as a number of event-driven adaptations.

Even though the event-driven approach has been adopted in a few power management systems, such as wake-on-wireless [16] and process cruise control [20], these systems were mainly designed with specific scenarios in mind. In addition, these systems only supported simple event processing [11]. However, previous work shows, that the adaptations for wireless data transmission have to handle more than one event at a time and require more complex processing of the events. For example, the self-tuning power management system, STPM [2], switches the operating mode of the Wi-Fi network interface (WNI) between the Continuously Active Mode (CAM) and Power Saving Mode (PSM), while taking into account application hints about traffic patterns, tolerable delay and the trade-off between transition cost and energy savings at the same time. To implement such adaptations following an event-driven approach, complex event processing is needed because it can process a collection of events and generate more meaningful ones based on the patterns shown in the event occurrences.

In this paper we propose an event-driven framework that can be used for implementing power management on mobile devices. Our framework uses event-condition-action (ECA) rules to describe the power management mechanism that explains which actions to invoke upon the occurrence of an event under certain conditions. As our framework supports complex event processing, the events declared in the ECA rules can also be ones derived from other events and ones generated by correlating other events. The rules of the complex event processing are defined in event specifications and can be parsed into different combinations of logical functions like filtering, pattern matching and derivation. Developers only need to define their event specifications and the ECA rules that describe the event-driven power management mechanism using structural XML, and the framework will handle the rule-based processing, including event generation, event processing, and adaptation scheduling.

We have implemented this framework in C++ on Maemo, a Linux-based mobile platform. We have also demonstrated our framework with two power management applications, in which complex event processing and simple event processing are applied respectively. The first application focuses on the adaptations of PSM settings to traffic patterns. We predict the no-data intervals based on the self-similar burstiness of network traffic. Our proposal differs from previous work [2, 4] in not requiring revisions to applications, as it can learn the traffic pattern online based on traffic statistics. The second application focuses on the adaptations of network transmission to SNR in Wi-Fi. Our experimental results show that the first application saves 11.78% of energy in Internet radio streaming and the second one saves 12.86% in TCP file downloading in a mobile scenario.

In summary, our contributions include:

1) Proposing an event-driven framework for power management on mobile devices. To the best of our knowledge, it is the first one that uses complex event processing for power management.

2) Providing user-friendly interfaces for implementing and configuring power management applications and hiding the low-level implementation from application developers.

3) Demonstrating the usage and effectiveness of the framework with two power management applications, one of which is original in itself.

Our framework enables implementing and deploying many power management solutions simultaneously. Therefore, it can be used for integrating existing power management applications, which has potential of gaining more energy savings through the coordination between applications. For example, coordinating the power management of wireless network interface [2] with application-level traffic adaptations [12] has potential of reducing more transmission cost while maintaining the application performance.

The rest of the paper is organized as follows. Section 2 introduces two power management applications which are used as examples for describing and evaluating our framework. Section 3 gives an overview of our event-driven framework. Section 4 describes the implementation of the framework itself and the example applications that use the framework. The results of our experimental study are presented in Section 5. Related work is reviewed in Section 6, and our work is concluded in Section 7.

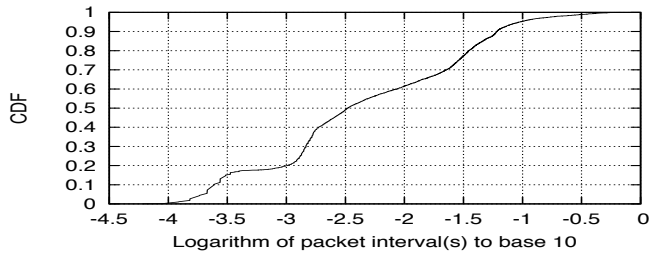

Figure 1. Cumulative distribution of packet interval in an Internet radio stream.

2 Motivating Scenarios

On a mobile device, hardware components, such as wireless network interfaces, are the actual power consumers. The rate of power consumption is determined by the physical characteristics of the hardware, whereas the total amount of energy consumption also depends on the workload generated by software. In this section, we present two motivating scenarios in which the power management applications aim at energy savings through workload-aware hardware control and workload scheduling, respectively.

2.1 Traffic-aware WNI Control

Alice is listening to an internet radio channel on her mobile phone through Wi-Fi. A traffic sniffer is running on the phone capturing packet information such as packet arrival time and packet size. At the same time, power management software is analyzing the statistics of packet information based on which it classifies the radio stream as self-similar bursty traffic, and starts to predict the occurrences of bursts. Whenever a burst is predicted to end, the power management software informs the WNI to go to sleep.

In this scenario, power management software learns the traffic patterns online and adapts the WNI operating mode to the discovered traffic patterns. Our proposal improves *PSM Adaptive*, a variant of PSM that is widely used on commercial devices, by taking traffic patterns into account. Differently from the PSM defined in the IEEE standard [1], PSM Adaptive adopts a timeout mechanism which forces the WNI to wait for a fixed period of time first, instead of going to sleep immediately when the WNI becomes idle. The length of this waiting time, called the *PSM timeout*, is usually fixed to 100ms or 200ms.

The motivation of our proposal comes from the fact that PSM Adaptive is inefficient for many applications, since the fixed PSM timeout used on commercial devices is longer than most packet intervals in Internet traffic. For example, as shown in Figure 1, 95.3% of packet intervals included in an Internet radio stream are smaller than 100ms. During 75.5% of the aggregate packet intervals the active WNI is in IDLE mode, which wastes energy. Hence, apart from shaping the traffic patterns, we argue that it is necessary to make changes to PSM Adaptive in order to adapt the operating mode of the WNI to the patterns of the traffic in a more energy-efficient manner.

We predict the traffic intervals during runtime and adjust the PSM timeout dynamically with the traffic intervals, taking the constraints of performance and energy overhead into account. We predict the traffic intervals based on the self-similar burstiness of Internet traffic [7], without revisions to mobile applications or access points. According to the definition of "train burstiness" in [9], "a burst can be defined as a train of packets with a packet interval less than a threshold". We call this threshold the *packet interval threshold*. As the intervals between two bursts must be bigger than the packet interval threshold, predicting the beginning of a big no-data interval can then be transformed into predicting the ending of a burst.

We use a threshold of burst size, called *burst size threshold*, to predict the ending of a burst. The burst size is equal to the total size of all the packets included in the burst. A variable that holds the size of the current burst is updated every time a new packet arrives. When the current burst size variable reaches the burst size threshold, the packet that has just arrived is considered as the last packet of the burst. In other words, a new burst interval is estimated to have begun.

We apply the moving average algorithm for calculating the burst size threshold. Given the sizes of the previous N bursts, the burst size threshold is set to the mean of these burst sizes. To gain high prediction accuracy, we use standard deviation of burst size to evaluate the self-similarity of burst size. Only when the standard deviation of the previous M burst sizes is smaller than a threshold, called *standard deviation threshold*, do we start to run the prediction. The implementation of this application using our framework will be detailed in Section 4.

2.2 SNR-based Transmission Adaptations

Alice is downloading a file from a TCP server to her mobile phone through Wi-Fi. As she is moving with the phone, the wireless link quality is not stable. A network monitor running on the phone is monitoring the wireless link quality in terms of SNR. Based on the history, the network monitor predicts the change in the wireless link quality that is going to happen in the next time slot. When the wireless link quality becomes unacceptable, the phone pauses the file transmission, until the wireless link quality becomes sufficiently good again.

In this scenario, the power management software adapts the network transmission to the wireless link quality for saving energy. We measure the wireless link quality using SNR, as it has been previously proved to be a good indicator of the wireless link quality [15]. Energy efficiency of network transmission in Wi-Fi increases when network throughput gets higher [6, 21], and the network throughput is at its best when the link quality is good. Hence, it is more energy-efficient to conduct data transmission when the link quality is good.

We adopt a threshold-based method for adaptation scheduling based on the prediction of SNR. Previous work has proposed several SNR prediction algorithms based on statistical models like Autoregressive integrated moving average (ARIMA) [19] and Markov chains [14]. Most of these models require complex offline model training. In this work, we show that with our simple online prediction algorithm it is still possible to gain energy savings that are comparable to those obtained with more complex methods.

Our simple online prediction algorithm works as follows. We monitor SNR at a fixed frequency, so that a time series can be divided into time slots with a fixed length. The length of the time slot is chosen so that the measured SNR value does not change more than once during one time slot. For example, according to our SNR measurement sampled at 10Hz, in the scenarios where the phones move with the mobile users at walking speed, the measured SNR does not change more than once in one second. Hence, it would be accurate enough to sample SNR at 1Hz in those scenarios. Let the monitored SNR at time x be $m(x)$ and the predicted SNR at time $(x+1)$ be $p(x+1)$. Our prediction algorithm can be simply defined as $p(x+1)=m(x)$.

	Traffic-aware WNI control	**SNR-based transmission adaptations**
Context	packet interval, packet interval threshold, packet size, burst size, burst size threshold, standard deviation of burst size, standard deviation threshold	monitored SNR, predicted SNR, SNR threshold
Action	adjust the PSM timeout	pause/resume transmission

Table 1. Contexts and actions included in motivating scenarios.

2.3 Why is Complex Event Processing Needed?

The contexts and actions involved in the two motivating scenarios are listed in Table 1. We propose to implement such context-aware adaptations following an event-driven approach, because the adaptations described in the scenarios are invoked only when a certain change in context occurs.

Although event-driven approach has been used in other power management systems [16, 20], our proposal differs from previous ones by supporting complex event processing. We argue that complex event processing is more suitable than simple event processing for power management, especially in wireless data transmission scenarios, due to the following reasons.

First, simple event processing generates events only based on the context that can be directly measured. It cannot generate events based on changes in predictions. In contrast, complex event processing can create events based on statistics and prediction of contexts. Some contexts such as predicted SNR can not be directly measured but rather need to be calculated based on statistics or prediction.

Second, simple event processing cannot generate events based on event patterns, whereas pattern matching is needed by many power management applications. For example, in the example application presented in Section 2.1, the ending of a burst is predicted based on the self-similarity of burst sizes in previous samples.

Third, complex event processing can bring extra benefits for power management at OS and middleware levels, compared with simple event processing which is usually accompanied by complex policy management. It

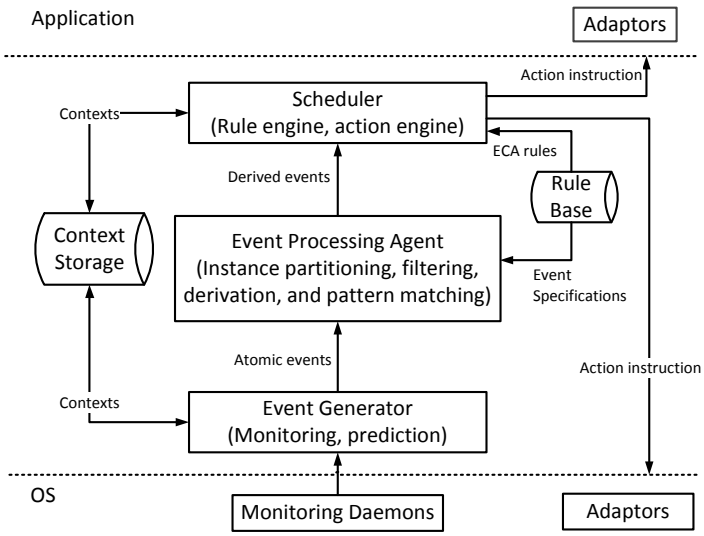

Figure 2. Architecture of our event-driven framework.

is because complex event processing can provide more meaningful information about the situations, which in turn makes it easier for application developers to define policies and to detect the potential conflicts between policies.

3 System Architecture

3.1 Overview

We propose an event-driven framework for power management of wireless data transmission on mobile devices. The architecture of our framework is shown in Figure 2. The power management mechanisms compose of event-driven adaptations that are defined using ECA rules. The lifecycle of events consists of three stages: event generation, event processing, and event consumption. Accordingly, there are three components in our framework, namely, event generator, event processing agent, and scheduler.

An event generator only produces atomic events based on the context information from monitoring daemon and context storage. Atomic events are processed into derived events by an event processing agent following event specifications. When the scheduler receives events from the event

processing agent, it matches the events to the ones defined in the ECA rules, and validates the conditions required for scheduling the actions using the available context information. Whenever an event matches a rule and all the conditions are satisfied, corresponding actions are scheduled.

The ECA rules and event specifications are stored in the rule base. Event processing states, historical events, external environment states and other global states are saved in a container called context storage. The content in the context storage gets updated when the relevant contexts change.

3.2 Event generator

The event generator is the software component that generates events based on the changes in contexts. The context information is collected from either the monitoring daemons or context storage. The relationship between context, state and event is described in the definitions of atomic state and atomic event below.

Definition 1: An atomic state is a tuple: S = *(c, op, val)* , where *c* is the capability value, *op* is one of the binary operators defined in a set: $\{<, >, \geq, \leq, =, \neq\}$, and *val* is the reference value of the capability.

Definition 2: An atomic event *e* indicates the change in a state from S_0 to S_1. It can be represented as *e:* $S_0 \rightarrow S_1$.

As shown in *Definition 1*, each context variable that can be monitored is modeled as *capability*, and the corresponding context providers are modeled as *sensors*. A sensor can be a hardware or software component with sensing capabilities. By comparing the capability values, an event can be created if there is a change in the values. In practice, events are only generated when there is at least one component subscribing to the event in question. In our system, there can be more than one event generator. An event generator can generate more than one type of event based on event specification. All the events generated by any event generator are imported into the same event processing agent for further processing.

Event instances are objects used for exchanging event information during runtime. An event instance includes the meta-data of the event such as event type identifier and occurrence timestamp, and a set of event-specific attributes. An attribute can be defined as a tuple including a unique attribute identifier and the indicator of attribute data type. Take a type of event which indicates the arrival of new packet as an example, its

identifier is NEW_PACKET. In addition to the meta-data of the event, an event instance of this type includes packet interval with previous packet, packet size, connection identifier, and transmission direction.

3.3 Event Processing Agent

Our event processing agent supports both simple event processing and complex event processing. In simple event processing, the event instances are directly forwarded to the scheduler. In complex event processing, the following functions will be applied, depending on event specifications.

Instance partitioning: The event instances with the same event type can be handled in different ways, depending on the values of certain event attributes, the time when the event occurs, and/or the state of the device and its environment. Accordingly, we partition the event instances and apply different processing functions to each partition afterwards. For example, the event instances with type of NEW_PACKET can be partitioned according to the connection identifier, one of its event attributes.

Filtering: We use filters to select the event instances that show certain features in their attributes or in meta-data properties associated to them. Different processing functions are performed on those events that match the filter than on those that do not. Similarly with instance partitioning, this function does not generate any new event or change the information included in the event instances.

Derivation: Derivation aims at generating new events by processing the input event instances based on different rules. These rules describe how the attributes of the new generated events are calculated from the attributes of the input event instances and/or the global state of the system.

Pattern matching: As defined in [13], "an event pattern is a template specifying one or more combinations of events." Pattern matching in the event processing agent is responsible for detecting whether the given set of event instances satisfies a particular pattern. In power management applications, the threshold pattern and trend pattern are often used.

3.4 Rule Specification

We use structural XML to represent the ECA rules of adaptations. There are four types of XML elements in a rule, <on>, <if>, <do> and <elsedo>. The <on> states the only type of the event to be handled by this rule. The <if> describes the conditions as a complex state, which is basically a

combination of atomic states formed with logical operators AND, OR, and NOT. The <do> elements in a rule define the actions to be invoked if the conditions are satisfied, while the <elsedo> elements define the ones to be invoked if the conditions are not satisfied.

Definition 3: We define an action as a tuple: *(id, type, name, paramlist)*, where *id* is the identifier of a hardware component or a software component that will handle the operation, *type* denotes the type of the operation, *name* is the identifier of the operation, and *paramlist* is a set of parameters used for the operation.

We define three types of operations, *set*, *subscribe*, and *unsubscribe*. 'Set' is used for setting hardware/software parameters, while the others are used for subscribing or unsubscribing events, with the event type stated in the identifier of the operation.

Event specifications can be written using the same XML schema. Event processing rules includes two types of 'set' operations, one for event generation and the other for context update. The processing of the event specifications in the event processing agent follows the same procedure as the processing of adaptation rules in the scheduler. This procedure will be explained in Section 4.3.

4 Implementation

We implemented the framework in C++ on Maemo 5, a Linux-based OS. In this section, we will describe the implementation of each component. Among them, event generators work closely with the context monitoring utilities which are platform-specific. Depending on the context information needed, event generation can be implemented as several event generators each of which handles a set of context information. In contrast, the event processing agent and the scheduler are context-independent. In addition to the framework itself, we also implemented the two power management applications introduced in Section 2 using the framework. The events used in these applications are summarized in Table 2. We will give examples of the related event generation and complex event processing in this section.

Event type	Event description
NEW_PACKET	A new packet arrives.
NEW_BURST	This packet is the first one in a new burst.
END_BURST	This packet is predicted to be the last packet of current burst.
WRONG_END	This packet does not belong to a new burst. However, the current burst has been predicted to end.
WRONG_END_AGAIN	This packet does not belong to a new burst, and an event instance with a type called WRONG_END for this burst has been sent earlier.
MISS_END	This packet belongs to a new burst. However, the end of previous burst was not detected beforehand.
START_PREDICTION	The standard deviation of burst sizes is small enough for prediction.
STOP_PREDICTION	The standard deviation of burst sizes becomes so big that it is difficult to predict the burst arrivals.
FIRST_CONNECTION	One connection is established and it is the only one at the moment.
NO_CONNECTION	The last connection on the mobile device disconnects.
LOW_TO_HIGH_SNR	The predicted SNR is becoming bigger than the SNR threshold in the next time slot.
HIGH_TO_LOW_SNR	The predicted SNR is becoming smaller than the SNR threshold in the next time slot.

Table 2. Description of events used in our example applications.

4.1 Event Generator

We implemented two event generators, the traffic monitor and the network monitor. The traffic monitor provides the atomic events, which indicate the arrivals of data packets or the changes in TCP/UDP connections. The network monitor generates events for the changes in the network environment such as the changes in SNR. As an event can be something that has happened in physical reality or something that we predict will happen in the near future, the event generators can provide events indicating the changes in the monitored context like SNR as well as the changes predicted to happen in near future, such as the predicted traffic intervals. The subscription of atomic events and the related context information is managed by the event generators in question.

Traffic monitor
The traffic monitor generates events based on real-time packet information. We implemented packet sniffing in a kernel module using Netfilter [1]. Netfilter is a set of hooks in the Linux kernel. For each hook, there is a callback function to be invoked whenever a packet traverses the hook in the network stack. We utilized two existing Netfilter hooks, *hook_local_in* and *hook_local_out*, which handle outgoing and incoming traffic, respectively. We customized their callback functions so that the packet information can be sent to the user space. In addition, we added a list to the kernel module for managing the identifiers of the ongoing connections. The identifier is the combination of the IP and port of the source and the destination, or simply just the local port in the case of a TCP connection. We utilized the handshake messages to detect the change in connection state. For example, when an ACK responding to SYN is detected and the connection identifier is not included in the list, a new connection is considered to be established.

The traffic monitor running in the user space opens a Netfilter socket for the incoming messages from the kernel module. For each message, the first byte states the message type, such as “connection opened”, “connection disconnected”, or “packet arrived”. The traffic monitor also maintains a list for managing the information on existing connections. An event instance with a type called FIRST_CONNECTION will be generated, if the connection count changes from 0 to 1. Conversely, one with a type called

[1] http://www.netfilter.org

NO_CONNECTION will be generated if the count changes from 1 to 0. For the message about a new packet, the traffic monitor generates an event instance with a type called NEW_PACKET and copies the packet information into the event attribute fields.

The traffic monitor can also generate atomic events using filtering on a single packet attribute. For example, it generates event instances with a type called NEW_BURST based on the packet interval with the previous packet that belongs to the same connection and has the same transmission direction. A packet that is the first packet of a new burst satisfies the conditions of two event types, NEW_PACKET and NEW_BURST. If both event types have been subscribed, one event instance will be generated for each type. Otherwise, only the one that has been subscribed will be generated.

Network monitor

The network monitor was implemented in a different way than the traffic monitor. The network monitor does not passively listen for messages coming from other components. Instead, it periodically pulls information directly through OS APIs. For example, it gets the signal strength and noise level every 1 second through *ioctl* functions.

The network monitor generates events that indicate changes in the predicted SNR. For example, LOW_TO_HIGH_SNR for the change from a value lower than the threshold to one higher than that. We use only the predicted SNR here, because the predicted SNR for the current time slot implies the current state of network transmission. For example, when the SNR in the current time slot was earlier predicted to be lower than the threshold, even if the actual measured SNR went over the threshold, the network transmission would have been paused in accordance with our adaptation rules.

4.2 Event Processing Agent

Event generators and the event processing agent share an event queue. Event generators only push event instances into the queue, while the event processing agent can get them out of the queue and also push back derived events for further processing. Our event processing agent performs both simple event processing and complex event processing. In this section, we will focus on the implementation of complex event processing,

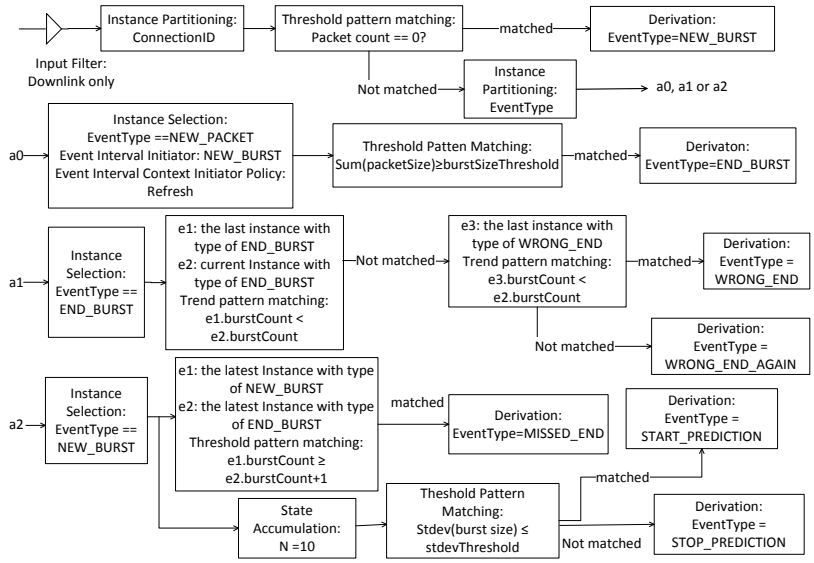

Figure 3. Complex event processing in traffic-aware WNI control.

using traffic-aware WNI control as an example. As shown in Figure 3, the complex event processing in the example uses filtering, instance partitioning, derivation and pattern matching. Filtering is implemented in the same way as done by the traffic monitor for generating event instances with a type called NEW_BURST. The implementation of the other three is explained below.

Instance partitioning: The example uses two types of instance partitioning. One is called segmentation-oriented partitioning, which classifies the event instances based on a certain event attribute. As shown in Figure 3, the event instances accepted by the input filter are first partitioned based on the connection identifier. It means that the event instances related to different connections will be processed independently. After the threshold pattern matching stage, the event instances that do not match will be further partitioned into three sets based on the event type. Accordingly, there are three branches, marked as a0, a1 and a2, for the following processing for each partition.

In branch a0, temporal-oriented partitioning is applied to the event instances with a type called NEW_PACKET. We implement it with a time window. Only the instances with their occurrence timestamps covered by the time window will be included. The time window is usually defined using the beginning and ending points in time. However, in this case, these points in time are uncertain, because the time window is initialized or

closed only when a certain event occurs. Hence, we define the time window using the events that initiate or terminate the time window, and call them initiator and terminator, respectively. If a time window already exists when an initiator occurs, there are different ways to handle the new initiator. The initiator can be either ignored or interpreted as a signal to create a new time window. The new time window can then replace the old one, or co-exist with the old one, depending on the usage scenarios. In our example, a new event instance with a type called NEW_BURST will refresh the time window. It means that the related burst statistics will be updated. For example, the size of current burst will be reset to 0.

Pattern matching: Our example uses threshold pattern matching and trend pattern matching. A threshold pattern can be described by a binary operator and a threshold value. The binary operator is used for comparing the threshold to the input. If the comparison returns TRUE, the input is considered to match the pattern. Trend pattern matching is used to detect whether an event attribute is showing an increasing, a decreasing or a stable trend over time.

Unlike in filtering, the input used in pattern matching is not limited to mere event attributes and event meta-data. For example, in the sequence starting from a2, the standard deviation of the last 10 burst sizes is used as input when doing threshold pattern matching. The pattern matching algorithms can also be replaced with other pattern matching algorithms during runtime, which makes the pattern matching mechanism even more flexible if compared to the filtering mechanism.

Derivation: In Figure 3, there are 7 types of derived events that may be generated. The attributes of derived events come from input events or context storage.

4.3 Scheduler

When the scheduler loads the ECA rules, it must first check for any potential conflict in the actions defined in different rules. Given two event instances, *e1* and *e2*, with different event types. If *e1* and *e2* can occur simultaneously and it is possible for the conditions defined in both rules to be satisfied, we must check whether their corresponding actions conflict with each other. For example, when the actions try to modify the same parameters of the same hardware component, conflict occurs. If no conflict is reported, the scheduler checks from event specifications which atomic

events are needed for generating the events defined in the rules. It then subscribes for the events from the event generators which can provide them.

The scheduler parses the ECA rules according to their XML schema, and executes the functions when the notification of an event occurrence arrives. This procedure includes two steps. First, the condition expression is parsed into a tree structure. The leaf nodes are capability values and their reference values. The non-leaf nodes are binary operators if their children are leaf nodes, or logical operators if their children are non-leaf nodes. Second, when the event defined in the rules occurs, we use tree-traversal algorithm to implement the evaluation of the conditions. The capability values defined in leaf nodes are obtained from the context storage.

5 Evaluation

We evaluated the gained energy savings, the energy consumption overhead caused by power management itself, and the impact of running power management on the system performance.

5.1 Experimental Setup

We ran the test on a Nokia N900. The device was connected to a public 802.11 b/g access point, whose beacon interval was 100ms. It means the mobile device woke up every 100ms to check for incoming data, whenever it was in SLEEP mode. Throughout the experiments we collected power consumption and traffic traces.

Power consumption traces: We used a Monsoon power monitor [2] to measure the power consumption during runtime. The sampling frequency of the power monitor was set to 1MHz. The power monitor itself combines the functionalities of a DC power supply and a power meter. We replaced the battery of the N900 with an electrical circuit, through which the N900 was powered by the DC power supply of the power monitor.

Traffic traces: We ran Wireshark [3] directly on the N900, and on the TCP server if the N900 was connected to it, to capture the packet information.

In the test cases where the network data rates needed to be specified,

[2] http://www.msoon.com/LabEquipment/PowerMonitor/
[3] http://www.wireshark.org

IDLE P_I	SLEEP P_S	RECEIVE P_R
668.88mW	32.25mW	980.68mW

Table 3. Power consumption of N900 when WNI is in different operating modes. During measurement, only the basic components of the device were in use. The screen backlight, Bluetooth and WCDMA were turned off.

SNR monitoring	Traffic sniffing	Event handling
11.27mW	8.11mW	0.75mW

Table 4. Total overhead caused by power management. The overhead of SNR monitoring was measured when the SNR was collected every 1 second. The overhead of traffic sniffing was measured when Internet radio streaming was running.

we used Trickle[4], a bandwidth throttling software, to limit the data rate on the TCP servers.

5.2 Baseline Power Consumption

We first measured the power consumption of the N900 when the embedded WNI was in different operating modes. The results are listed in Table 3. After that, we measured the energy consumption overhead caused by our power management system, which includes the overhead of SNR monitoring, traffic sniffing, and event handling. As shown in Table 4, the overhead caused by the event handling, which includes all the operations except the event generation and the actions invoked by the scheduler, was about 1% of P_S. SNR monitoring and traffic sniffing were only used when there was network transmission going on. Compared with P_R or P_I, the energy consumption overhead of SNR monitoring and traffic sniffing was less than 2%.

5.3 Internet Radio Streaming and File Download

We used the embedded media player on the N900 to connect to an Internet radio station, called The Voice [5]. The real-time radio stream was delivered to the mobile through HTTP/TCP. Our power management system was running on the mobile device, independently of the media player.

We loaded the rules below during the initialization of our power management software. In practice, these rules were written as structural XML following the rule specifications. The events defined in these rules were

[4]http://monkey.org/ marius/pages/?page=trickle
[5]http://83.145.249.98:80/

processed automatically, as shown in Figure 3.

Rule 1: When START_PREDICTION occurs, start predicting the ending of each burst.

Rule 2: When STOP_PREDICTION occurs, stop predicting the burst endings.

Rule 3: When END_BURST occurs, set the PSM timeout to 0ms.

Rule 4: When NEW_BURST occurs and if the prediction of burst endings is enabled, set the PSM timeout to 10ms.

Rule 5: When WRONG_END occurs, double the burst size threshold.

In addition, WRONG_END_AGAIN and MISS_NEW are only used for analyzing prediction accuracy. When they occur, the relevant statistics are updated.

We measured the power consumption of listening to Internet radio in two scenarios where our power management application turned either on or off. When the power management application was turned off, the PSM was enabled with the PSM timeout set to 100ms. When the power management was running, we first initialized the burst size threshold to 4000 Bytes. This threshold was updated with the unweighted mean of the previous 5 burst sizes. The exception was that when an event with a type called WRONG_END occurred, which means the actual burst size is bigger than the threshold, the burst size threshold would be doubled. The standard deviation threshold was set to 5000 Bytes during initialization. It was updated with the simple moving average over the previous 10 burst sizes. We set the packet interval threshold to 10ms. Choosing this particular setting was based on the observations we had made of the Internet traffic. Figure 1 shows that 61.6% of the packet intervals in our measurements were smaller than 10ms and would thus be included inside bursts.

We repeated the experiments five times. Each run lasted for 4 minutes. The results, as listed in Table 5, show that the average power consumption of the device was 11.9% less with the power management system turned on.

Next we downloaded a 3400KB file using wget[6] from a Linux server to the N900. The file was saved to /dev/null in order to avoid the writes to persistent memory affecting the measurement results. The traffic was

[6] http://www.gnu.org/s/wget/

Without adaptations	947.680mW
With adaptations	835.168mW
Difference	-11.9%

Table 5. The power consumption of listening to Internet radio with and without adaptations.

Prediction results	Internet radio	File download
Burst ending was correctly predicted	29.74%	82.33%
Burst was predicted to end but did not	22.17%	7.31%
Burst ended, but the ending was not predicted beforehand	48.09%	10.27%

Table 6. Accuracy of burst prediction for Internet radio streaming and TCP file download.

shaped into 4KB bursts on the server using Trickle. However, the shape and the duration of the bursts were not constant, because the traffic passed through both wired and wireless networks after leaving the traffic shaper called Trickle. The average data rate during the file transmission was 15.97 KBps. With the adaptations turned on, the energy consumed during the file download decreased by 13.6% from 87.37J to 76.94J, at the cost of a 2% increase in download time. The decrease in energy consumption during the file download was a little higher than in the case of Internet radio. One reason for the difference is that the traffic prediction accuracy was much higher in the file download case, as shown in Table 6.

We calculated the accuracy of our traffic predictions based on the event counts. Two types of prediction errors were identified in the experiments. In the first case an event instance with a type called END_BURST was generated before the burst had actually ended. However, even in this case the remaining packets in the burst might still arrive on time, if they arrived during the 6ms it took the WNI to switch to SLEEP mode. If a packet was received by the WNI during this transition period, our traffic monitor generated an event instance with a type called WRONG_END. We then used the number of this type of event instances for calculating the frequency of this first type of prediction errors.

If packets arrive after the WNI has already fallen asleep, the packets will be buffered in the access point until the WNI wakes up. The average additional delay for these packets would be 50ms if connected to an access point with a beacon interval of 100ms. The average network through-

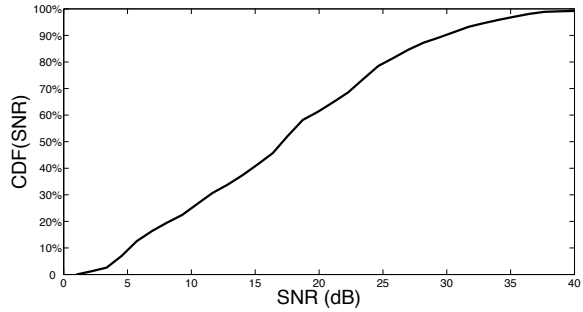

Figure 4. Cumulative distribution of SNR.

put in our Internet radio experiment was 22.17KBps, which is fairly close to the encoding rate of the Internet radio stream. The first big burst of data that arrived after the connection was established was measured to be about 250KB in size. From this information we could calculate that a delay less than 10 seconds was unlikely to have effect on playback quality, which was confirmed by the fact that no break during playback was empirically observed during our measurements.

The other prediction error happened, when the burst finished, but the power management system had failed to predict the ending of the burst and no adaptation could be invoked during the no-data interval. This kind of errors have a negative effect on energy savings, but they do not cause any additional delay.

5.4 SNR-based Transmission Adaptations

We tested the file download from a remote server to the mobile device on the move. We moved the mobile device along a straight line away from the access point and then took the device back at a stable walking speed. The sampling frequency of SNR was 1Hz. As shown in Figure 4, the SNR was uniformly distributed with a mean of 17 and a standard deviation of 9. We set the SNR threshold to 15 for generating event instances with type of LOW_TO_HIGH_SNR and HIGH_TO_LOW_SNR.

We evaluated the quality of our SNR prediction algorithm by comparing the predicted values with the measured ones. As evaluation metrics we used the mean squared error (MSE), which is the sum of the squares of prediction errors. The MSE of our predictions turned out to be 34.87, which is relatively good compared with [19]. A file with size of 39.3MB was downloaded from a TCP server to the mobile device. The data rate was

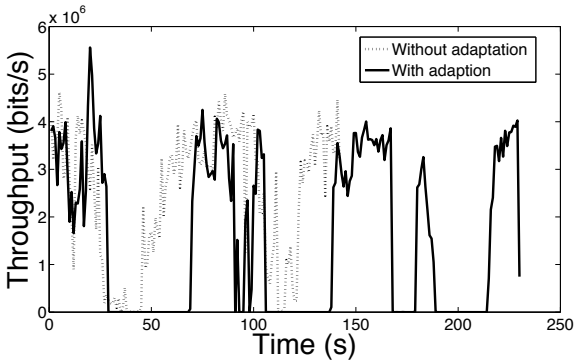

Figure 5. Comparison of network throughput with/without adaptations.

limited to 512KBps on server side. We adopted the following adaptation rules.

Rule 1: LOW_TO_HIGH_SNR and HIGH_TO_LOW_SNR events are subscribed when FIRST_CONNECTION occurs, and unsubscribed when NO_CONNECTION occurs.

Rule 2: Upon the occurrence of HIGH_TO_LOW_SNR, in case of TCP connection, the transmission will be paused by setting the TCP receive window size to 0.

Rule 3: Upon the occurrence of LOW_TO_HIGH_SNR, in case of TCP connection, the transmission will continue by resuming the TCP receive window size to its default value.

As shown in Figure 5, the download duration measured from the arrival of the first packet to the arrival of last packet is 161.97 seconds without adaptations, and 229.15 seconds with adaptations. Without adaptations, the TCP connection is disconnected due to the low SNR for 29.80 seconds between the 108th and 138th seconds. With adaptations, the download was paused by adaptations for 122.04 seconds. The WNI was put into SLEEP mode instead of letting it work inefficiently in RECEIVE mode. Hence, our adaptations reduced the time spent in RECEIVE or IDLE mode by 20 % from 132.18 seconds to 107.12 seconds.

As it proved unfeasible to conduct the physical power measurement of a moving device with the hardware we had at our disposal, we had to resort to calculating the energy consumption based on the traffic traces with the

	Without adaptations	With adaptations
Data processing on WNI (J)	29.95	30.11
Wasted in IDLE mode during burst intervals (J)	23.49	16.09
Wasted in IDLE mode inside bursts (J)	34.27	34.33
Total energy (J)	87.71	80.53

Table 7. Energy-efficiency analysis of SNR-based transmission adaptations.

power models shown in (1)-(3).

$$\mathbf{T_1} = \sum_{i \leq 0.1} (i) + \#(bursts \mid i > 0.1) \times T_{timeout}, \quad (1)$$

$$\mathbf{T_2} = T - \sum(i) - S/R, \quad (2)$$

$$\mathbf{Energy} = (P_R - P_S) \times S/R + (P_I - P_S) \times (T_1 + T_2), \quad (3)$$

where T is the total duration of file transmission, T_1 is the sum of burst intervals and T_2 is the sum of the packet intervals which are smaller than the packet interval threshold. In addition, i is the burst interval in seconds, $T_{timeout}$ is the PSM timeout, S is the total traffic size, and R is the maximum throughput of the WNI. P_R, P_S and P_I are the power consumption with WNI in RECEIVE, SLEEP and IDLE mode, respectively.

Our models are derived from the ones presented in [21] and [6]. We assume that the WNI is in RECEIVE mode during transmission and in IDLE mode during the interval if the interval is shorter than the PSM timeout. For the intervals longer than the PSM timeout, WNI stays in IDLE mode until the timer expires and then switches into SLEEP mode.

The intervals include two parts, burst intervals, and the packet intervals inside bursts. The first might become bigger than the PSM timeout, whereas the latter is always smaller than the packet interval threshold. In our measurement, we set the PSM timeout to 100ms and packet interval threshold to 10ms. Hence, WNI only stays in IDLE mode during the intervals inside bursts, whereas it might go into SLEEP mode if the burst intervals are bigger than 100ms. We calculated the total duration of WNI processing as the total traffic size divided by the maximum throughput of the WNI. According to our measurement, the maximum throughput is 1.328Mbps. The values listed in Table 3 are used for our calculation. On average, it costs 93.68 Joules to download the file without adaptations, whereas it costs only 81.64 Joules with adaptations. With the adapta-

tions, 12.85% less energy was consumed. To explain the energy savings achieved by our adaptation, we choose two samples. One is from the result without adaptations, and the other one from the result with adaptations. As shown in Table 7, the major savings come from the reduction in the energy wasted during burst intervals, because the random traffic like ARP (Address Resolution Protocol) packets is reduced when adaptations are applied.

6 Related Work

Many application-specific solutions have been proposed for improving the utilization of power management based on hardware resource management. For example, traffic shaping for streaming applications [5] and web prefetching have been proposed for reducing the transition overhead caused by the usage of PSM and increasing the duration spent in the low-power modes. However, since these solutions make adaptations to the changes in contexts individually, it is not clear how these application-specific solutions could work compatibly with each other, and collaboratively with the default power management software installed on the devices. Hence, in this paper, we look at the system from a *holistic* perspective and propose a framework that enables the management of these mechanisms through predefined rules. The crucial aspect is that we include the applications and the context in which the device operates in the overall picture because knowledge about them allows us to exploit specific power management mechanisms more efficiently.

Our work provides a platform for application developers to implement their power management applications on. There are several systems that also consider the system-level aspect.

Koala [17] is a platform that allows policy-based control over power-performance trade-offs, but it only focuses on Dynamic Voltage and Frequency Scaling (DVFS) for the CPU. Dynamo [12] comes closer to our solution by considering a cross-layer framework. It optimizes the energy consumption through dynamic adaptations for the CPU (through DVS), the display, and the wireless network interfaces. However, this solution is designed only for video streaming. STPM [2] provides a solution closely resembling our first example application (see Section 2.1), whereas STPM requires revisions to mobile applications and does not provide mecha-

nisms for integrating it with complementary solutions.

A middleware for power management presented in [3] was designed based on the assumption that the real-time power state of each hardware component is available. They tried to describe the system using several system states, each corresponding to a unique power state. However, it is unclear how such middleware can be implemented beyond a simple example and how applications make decisions on which state to choose to reach optimal power consumption. The solution described in [8] is also closely related to our approach but only focuses on well-defined enterprise application scenarios of accessing web services and synchronizing them with a database.

If we try to extend the above systems for the integration of different power management policies on a single device, a challenge comes from the increase in the complexity of policy management. In this paper we use complex event processing, which leads to simple policy management.

Some systems take a different approach to the power management problems. For example, Turducken[18] combines different capacity devices (laptop, PDA, and sensor) into a single mobile system, and prolongs the battery life of the entire system by intelligent workload scheduling among devices. SleepWell [10] focuses on reducing the energy consumption of mobile clients by reducing contention between different access points. The event-driven approach we propose can be extended for the collaborative power management between devices in future, while in this paper we focus solely on a single device system.

7 Conclusion

In this paper, we proposed an event-driven framework for rule-based power management for wireless data transmission on mobile devices. It supports both simple event processing and complex event processing, which eases the implementation of power management. We evaluated the framework with two power management applications, traffic-aware WNI control and SNR-based transmission adaptations. The results show on average 12% of energy savings in these cases.

Bibliography

[1] IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages C1 –1184, June 2007.

[2] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. *Wirel. Netw.*, 11:451–469, July 2005.

[3] H. S. Ashwini, A. Thawani, and Y. N. Srikant. Middleware for efficient power management in mobile devices. In *Proceedings of the 3rd International Conference on Mobile Technology, Applications & Systems*, Mobility'06. ACM, 2006.

[4] D. Bertozzi, L. Benini, and B. Ricco. Power aware network interface management for streaming multimedia. In *Proceedings of 2002 IEEE Wireless Communications and Networking Conference*, volume 2 of *WCNC'02*, pages 926–930, 2002.

[5] F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys'10, pages 107–122, New York, NY, USA, 2010. ACM.

[6] R. Friedman, A. Kogan, and K. Yevgeny. On power and throughput tradeoffs of wifi and bluetooth in smartphones. In *Proceedings of the 30th Conference on Computer Communications*, INFOCOM '11, Shanghai, China, 2011.

[7] M. Grossglauser and J.-C. Bolot. On the relevance of long-range dependence in network traffic. *IEEE/ACM Trans. Netw.*, 7:629–640, October 1999.

[8] A. B. Lago and I. Larizgoitia. An application-aware approach to efficient power management in mobile devices. In *Proceedings of the 4th International ICST Conference on Communication System Software and Middleware*, COMSWARE'09, pages 11:1–11:10. ACM, 2009.

[9] K.-c. Lan and J. Heidemann. A measurement study of correlations of internet flow characteristics. *Comput. Netw.*, 50:46–62, January 2006.

[10] J. Manweiler and R. Roy Choudhury. Avoiding the rush hours: Wifi energy management via traffic isolation. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys'11, pages 253–266, New York, NY, USA, 2011. ACM.

[11] B. Michelson. Event-driven architecture overview. *Patricia Seybold Group, Feb*, 2006.

[12] S. Mohapatra, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Dynamo: A cross-layer framework for end-to-end qos and energy optimization in mobile handheld devices. *IEEE Journal on Selected Areas in Communications*, 25(4):722 –737, may 2007.

[13] Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications Co., 2011.

[14] Osman, Gani and Hasan, Sarwar and Chowdhury Mofizur Rahman. Prediction of state of wireless network using markov and hidden markov model. *Networks*, 4(10):976–984, 2009.

[15] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking*, MobiCom'10, pages 85–96. ACM, 2010.

[16] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, MobiCom'02, pages 160–171. ACM, 2002.

[17] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for os-level power management. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys'09, pages 289–302. ACM, 2009.

[18] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical power management for mobile devices. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys'05, pages 261–274. ACM, 2005.

[19] R. Sri Kalyanaraman, Y. Xiao, and A. Ylä-Jääski. Network prediction for energy-aware transmission in mobile applicatioins. *Journal on Advances in Telecommunications*, 3:72–82, November 2010.

[20] A. Weissel and F. Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES'02, pages 238–246, New York, NY, USA, 2002. ACM.

[21] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski. Practical power modeling of data transmission over 802.11g for wireless applications. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 75–84, New York, NY, USA, 2010. ACM.