Power Management for Wireless Data Transmission Using Complex Event Processing

Yu Xiao, Wei Li, Matti Siekkinen, Petri Savolainen, Antti Ylä-Jääski, *Member*, *IEEE*, and Pan Hui, *Member*, *IEEE*

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 both 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. 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 percent more energy compared to normal operation.

Index Terms—Power management, complex event processing, mobile device, context-aware

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 wireless data transmission are therefore very much needed for extending the battery life of mobile devices. In this paper, we focus on power management software that runs on a mobile device and helps to reduce energy cost of wireless data transmission on the device by controlling the behavior of device components and applications.

Energy consumed in wireless data transmission is heavily dependent on the situation in which the transmission happens. The situation includes many factors, such as the state of the mobile device, the environment of the wireless network which the mobile device is connected to, and the patterns of the traffic generated by mobile applications. In order to manage the transmission cost, power management software must be able to detect the situational variation which the behavior of the device components and applications is adapted to.

As defined in [1], "*context* is any information that can be used to characterize the situation of an entity." The entity can be a person, a device, a network, or an application. For example, signal-to-noise ratio (SNR) is context that indicates the link quality in a wireless network. Changes in context can be represented as *events*. For example, an event can be used for indicating the increase of SNR in a wireless network, or for indicating an occurrence such as the arrival of a person in a room. In order to represent concepts on higher abstraction levels, events can also be derived from other events, and used to represent patterns in event occurrence.

According to definitions of context and event, each factor included in the situation can be described with a set of contexts and/or a sequence of events. For example, the state of the mobile device can be described with context such as CPU frequency level and the operating mode of the wireless network interface, whereas the pattern of the traffic generated by mobile applications can be represented by a sequence of events that indicate arrivals of data packets at the wireless network interface. Thus, the detection of situational variation becomes an event processing task with a collection of different events as its input.

In this paper, we present an event-driven framework that can be used for implementing power management for wireless data transmission on mobile devices. Even though the event-driven approach has been adopted in a few power management systems, such as wake-on-wireless [2] and process cruise control [3], these systems were mainly designed with specific scenarios in mind and only supported simple event processing [4]. Differently from these systems, our framework supports complex event processing [5].

Developers that use our framework can use 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

- Y. Xiao, W. Li, and M. Siekkinen are with the Department of Computer Science and Engineering, Aalto University, PO Box 15400, Aalto 00076, Finland. E-mail: {yu.xiao, wei.2.li, matti.siekkinen}@aalto.fi.
- P. Savolainen is with Helsinki Institute for Information Technology (HIIT), University of Helsinki, PO Box 68, FI-00014, Finland. E-mail: petri.savolainen@hiit.fi.
- A. Ylä-Jääski is with Helsinki Institute for Information Technology, Aalto University, PO Box 15400, Aalto 00076, Finland. E-mail: antti.yla-jaaski@aalto.fi.
- P. Hui is with the Deutsche Telekom Laboratories, Ernst-Reuter-Platz 7, Berlin 10587, Germany. E-mail: pan.hui@telekom.de.

Manuscript received 4 Nov. 2011; revised 29 Feb. 2012; accepted 22 Apr. 2012; published online 29 May 2012.
For information on obtaining reprints of this article, please send e-mail to: tcsi@computer.org, and reference IEEECS Log Number TCSI-2011-11-0845.
Digital Object Identifier no. 10.1109/TC.2012.113.

declared in the ECA rules can also be ones derived from other events and ones generated by correlating other events. The rules of the event processing are defined by the developer in event processing specifications and are parsed by the framework into combinations of event processing functions such as filtering, pattern matching, and derivation. Developers only need to define their event processing specifications and ECA rules using structural XML, and the framework will handle the rule-based adaptations, including event generation, event processing, and adaptation scheduling.

We have implemented the framework in C++ on Maemo, a Linux-based mobile platform. We have also demonstrated our framework by using it to implement two power management applications. 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 [6], [7] 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 11.8 percent of energy in Internet radio streaming and the second one saves 12.9 percent 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 deploying multiple power management solutions simultaneously on the same device. It can be therefore used for integrating existing power management applications and for enabling coordination between them for additional energy savings. For example, coordinating the power management of wireless network interface [6] with application-level traffic adaptations [8] has the potential of reducing the transmission cost while at the same time maintaining the application performance.

The rest of the paper is organized as follows: Section 2 briefly introduces techniques of power management on mobile devices. Section 3 gives an overview of the event-driven framework. Two example applications that are used for evaluation are introduced in Section 4. Section 5 describes the implementation of the framework itself and the example applications using the framework. The results of our experimental study are presented in Section 6 followed by the conclusion of the work.

2 Background and Related Work

2.1 Power Management of Wireless Network Interfaces

Energy consumption of wireless data transmission is mainly caused by the operations of wireless network interfaces on mobile devices. Many power management mechanisms for the wireless network interfaces have been proposed and some of them have already been successfully commercialized. The commercial ones are usually implemented as part of hardware resource management in mobile OSs. They adapt the operating modes to system workload and try to gain energy savings from the difference in power consumption between the operating modes of hardware components. In other words, they try to keep the hardware components in lower-power states as long as possible. For example, the PSM [9] for Wi-Fi forces the Wi-Fi network interface (WNI) to go to sleep if there is no data to transmit or to receive. As listed in Table 1, the power consumed in SLEEP mode is only 5 percent of that consumed in IDLE mode.

TABLE 1
Power Consumption of N900 when WNI is in Different Operating Modes

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

During measurement, only the basic components of the device were in use. The screen backlight, Bluetooth, and WCDMA were also turned off.

These power management mechanisms have shown their potential in saving energy. However, there are also downsides to using them. First, they might cause performance degradation, such as increased delay when using PSM [10], because the incoming packets that arrive when the WNI is sleeping will be buffered at the access point or dropped. Second, the transition between operating modes takes time and costs energy. Sometimes, the overhead can even overtake the energy savings gained by the transition.

One way to reduce the negative side effect is to improve the design of the power management itself. For example, WNIs on commercial devices usually use an adaptive version of PSM, also known as *PSM Adaptive*. 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 100 or 200 ms. Compared with the PSM defined in the IEEE standard, PSM Adaptive achieves better performance in terms of delay, whereas the energy savings decreases.

To solve the new problems generated by PSM Adaptive, some revisions have focused on reducing the energy wasted in IDLE mode through intelligent control over the transitions between operating modes. Intelligent control is based on both the adaptation to the traffic characteristics and on the performance requirements of the mobile applications. For example, STPM [6] proposed to switch between CAM and PSM based on two factors: the potential energy savings and the possible performance degradation. It enabled PSM only when the energy savings could be achieved while the latency was tolerable. The results showed that STPM was better suited for delay-tolerant applications than the delay-sensitive applications such as streaming.

As oppposed to STPM, which focused on coarse-grained adaptation, Liu and Zhong [11] proposed micro power management (μPM). μPM tries to put the WNI into power saving mode during idle intervals, which can be as short as several microseconds. To control the frame delay and data loss, μPM determined when to wake up the WNI and how long the WNI should stay awake to receive any possible retransmitted data. The decisions were made based on the history-based prediction of the next incoming and outgoing traffic frames and the load of the access network. The evaluation of μPM through simulation showed that more than 30 percent of energy could be reduced without perceptible quality degradation for certain applications such as audio streaming.

A key issue in the above solutions is the prediction of the future incoming and outgoing traffic. There are different ways to implement the prediction. One is to use the hints disclosed by mobile applications [6]. The hints reveal when the applications will transfer data, how much data to transfer, and the maximum delay the applications could tolerate. Another method is to predict statistically the next arrival time based on the history of the previous packet arrival information [11]. The latter one does not require revision to applications. However, a challenge to the accuracy of prediction comes from uncertainties such as the wireless link quality during data transmission and the workload of the network devices carrying the data through the network.

In this paper, we present an event-driven framework that supports complex event processing and propose to implement power management solutions using this framework. We evaluate the feasibility of this method with two example solutions in which the control of wireless network interface is adapted to the prediction of network traffic and the prediction of wireless link quality, respectively. The effectiveness of these solutions themselves was evaluated in terms of energy savings and network transmission performance. Compared with previous work on prediction-based wireless interface control, our work provides a novel traffic prediction algorithm based on the self-similar burstiness of Internet traffic. In addition, we provide impact analysis of prediction accuracy on energy savings and performance.

2.2 System-Level Power Management

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 [12] 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 adapt 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 [13] is a platform that allows policy-based control over power-performance tradeoffs, but it only focuses on Dynamic Voltage and Frequency Scaling (DVFS) for the CPU. Dynamo [8] comes closer to our solution by considering a cross-layer framework. It optimizes the energy consumption through dynamic adaptations for the CPU (through Dynamic Voltage Scaling), the display, and the network interfaces. However, this solution is designed only for video streaming. STPM [6] provides a solution closely resembling our first example application (see Section 4.1), whereas STPM requires revisions to mobile applications and does not provide mechanisms for integrating it with complementary solutions.

A middleware for power management presented in [14] described the system using several system states, each corresponding to a unique power state. However, the design of the middleware was based on the assumption that the real-time information about the power state of each hardware component is available, which limits the implementability of the system. The solution described in [15] 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 power management policies on a single device, a challenge comes from the complexity in policy management. In this paper, we present an event-driven framework for power management. In our framework, power management policies are described as a set of event-driven adaptations. Our framework provides methods of conflict detection and resolution in order to avoid conflicts in adaptations. In addition, thanks to the use of complex event processing, our framework simplifies the rule evaluation and promotes separation of concerns by placing the event processing functionality in the event processing agent.

Some systems take a different approach to the power management problems. For example, Turducken [16] combines different capacity devices (laptops, PDAs, and sensors) into a single mobile system, and prolongs the battery life of the entire system by intelligent workload scheduling among devices. SleepWell [17] 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 to support collaborative power management between devices in the future, while in this paper we focus solely on a single-device system.

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 Fig. 1. The power management mechanisms compose of event-driven adaptations that are described with 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.

The event generator only produces atomic events based on the context information it obtains from monitoring

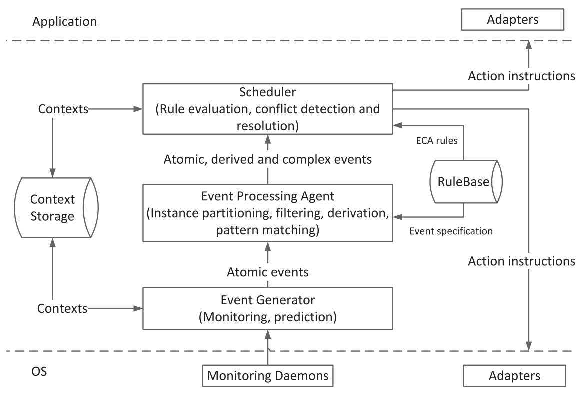

Application
Adapters
Action instructions
Contexts
Scheduler (Rule evaluation, conflict detection and resolution)
ECA rules
Atomic, derived and complex events
RuleBase
Context Storage
Event Processing Agent (Instance partitioning, filtering, derivation, pattern matching)
Event specification
Atomic events
Action instructions
Contexts
Event Generator (Monitoring, prediction)
OS
Monitoring Daemons
Adapters

Fig. 1. Architecture of our event-driven framework.

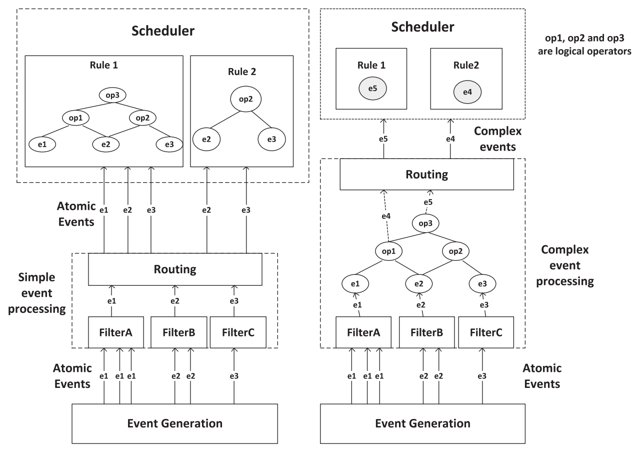


Fig. 2. Comparison between simple event processing and complex event processing.

daemons and from the context storage. Atomic events are filtered, partitioned into groups, and/or processed into more meaningful events according to event processing specifications by the event processing agent. 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 defined in the rule are satisfied, corresponding actions are scheduled.

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

3.2 Complex Event Processing

A key feature of our framework is the support for complex event processing in the event processing agent. The difference between simple and complex event processing systems lies in the functionality that they provide for processing the events. Whereas most simple event processing systems only support event filtering, complex event processing systems can also offer instance partitioning, event derivation, and pattern matching. Unlike simple event processing systems, complex event processing systems can also take into account the history of event occurrences and generate events of higher abstraction levels based on changes in the patterns of event occurrences. An example of this in the power management field is taking network packet arrival events as input and generating new events that indicate the beginning and the end of each new burst of traffic.

In a complex event processing framework, event processing is decoupled from event consumption. The action rules that are executed can be relatively simple as they are triggered by events of a higher abstraction level. If the same nontrivial application is implemented on a simple event processing framework, the action rules need to incorporate the event processing that is performed by the event processing agent in the complex event processing system. This means that a complex event processing framework allows for better separation concerns as the event-driven adaptations do not need to know how to extract meaningful information out of events of low abstraction levels, and the event processing specifications can be agnostic of what the events they generate will be used for. Decoupling of event processing from event consumption can also lead to reduced computational overhead as the events of high abstraction level that the event processing agent generates in a complex event processing system can be useful to more than just one adaptation rule. For example, the high-abstraction-level event that indicates the end of a burst of incoming traffic can trigger both a rule that pauses an ongoing upload and a rule that puts the wireless network interface to sleep.

The difference between simple and complex event processing frameworks is illustrated in Fig. 2. When executing the same actions, in the simple event processing case the event processing operations op2(e2, e3) get executed twice, whereas in the complex event processing case, only once.

3.3 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 the 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, while each event generator can generate more than one type of event. All the events generated by any event generator are imported into the same event processing agent for further processing.

TABLE 2
Contexts and Actions Included in Example Scenarios

	Traffic-aware WNI control	**SNR-based transmission adaptation**
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 PSM timeout	pause/resume transmission

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.

3.4 Event Processing Agent

Event processing specification written in structural XML is loaded into the event processing agent and is parsed into combinations of logical functions. Our event processing agent provides four logical functions of complex event processing: filtering, instance partitioning, derivation and pattern matching.

In *event filtering*, a filter is applied to event instances of the same event type in order to divide them into two distinct groups. Those instances whose attributes and/or metadata match the filter are submitted to one kind of processing, while the rest of the instances are submitted to another kind of processing.

Instance partitioning can be seen as a generalized version of event filtering. In instance partitioning, the event instances with the same event type can be divided into N groups instead of just two. Moreover, the partitioning can be done based on attribute values and metadata of these event instances or other factors such as the state of the device.

Event derivation takes one or more event instances as input and outputs a new event instance. The event instances used as input can be of different event types, while the event type of the output is different from that of the inputs. The attribute values of the new event instance are calculated from those of the inputs. We call an event that is the result of computational derivation from other events a *derived event*.

Pattern matching is a three-stage process for detecting patterns in an event flow. Event instances are first combined into intermediate event instance sets that are then fed into the actual pattern matching. Unlike event filtering and instance partitioning, pattern matching can involve events of different types. Algorithms used for grouping event instances are described in Section 5.3. In the second stage, the event processing agent applies *trend-based pattern matching* and/or *threshold-based pattern matching* to the intermediate event instance sets. Trend-based pattern matching is used for detecting whether an event attribute is showing an increasing, a decreasing or a stable trend over time. In threshold-based pattern matching, the pattern is defined by a binary operator and a threshold value. The binary operator is used for comparing the threshold to the input, and if the comparison returns TRUE, the input is considered to match the pattern. The input can consist of event attributes, metadata, or statistical data from a set of event instances. Finally, if the event instances match the pattern, a new *complex event* is created in the third stage.

3.5 Scheduler

Event-driven adaptations are defined in ECA rules by developers. These rules are loaded into the scheduler that is later responsible for rule evaluation during runtime. When loading a new rule, the scheduler checks if the newly loaded rule has any potential conflicts with the previously loaded rules. If the scheduler detects a potential conflict, it attaches a conflict resolution policy to the newly loaded rule or refuses to install the new rule whenever no conflict resolution policy can be applied. The details of conflict detection and resolution strategies will be described in Section 5.4.

If the installation of the new rule is successful, the scheduler will subscribe to the events that may trigger the new rule from the event processing agent. During runtime, whenever the scheduler receives a notification of an event occurrence it has subscribed to, it evaluates the conditions defined in the rules, and invokes corresponding actions if the conditions are satisfied.

4 Example 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 propose two power management applications aiming at energy savings without performance degradation through workload-aware hardware control and workload scheduling, respectively. We use these two applications for evaluating our framework and also as examples in Section 5 for explaining the implementation of our framework. The contexts and actions involved in the two example scenarios are listed in Table 2.

4.1 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 timestamps and sizes. 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 patterns. Our proposal improves PSM Adaptive, a variant of PSM that is widely used on commercial devices, by taking traffic patterns into account. The motivation 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 interpacket intervals in Internet traffic. For example, according to our measurement, 95.3 percent of packet intervals included in an Internet radio stream are smaller than 100 ms. During 75.5 percent of the aggregate interpacket 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 [18], without revisions to mobile applications or access points. According to the definition of "train burstiness" in [19], "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 no-data interval separating two bursts is by definition bigger than the packet interval threshold, predicting the beginning of a no-data interval bigger than the packet interval threshold is equivalent to predicting the ending of a burst.

We use a threshold of burst size, called *burst size threshold*, to predict the end 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 threshold, the packet that has just arrived is considered to be 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 5.

4.2 SNR-Based Transmission Adaptation

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 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 [20]. Energy efficiency of network transmission in WLAN increases when network throughput gets higher [21], [22], 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) [23] and Markov chains [24]. 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 10 Hz, 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 1 Hz in those scenarios. Let the monitored SNR at time x be $m(x)$ and the predicted SNR at time $(x+1)$ be $p(x+1)$. So our prediction algorithm can be simply defined as $p(x+1) = m(x)$.

5 Implementation

We implemented the framework and the two applications 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. The event processing agent and the scheduler are platform-independent. To better explain the implementation of these components, we use the two scenarios introduced in Section 4 as examples. The events used in these scenarios are summarized in Table 3.

5.1 Event Generators

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 and changes in the status of UDP flows and TCP 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 generator in question.

5.1.1 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

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

TABLE 3
Description of Events Used in Our Example Scenarios

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 a WRONG_END event 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 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 4
Description of XML Elements in Event Processing and ECA Rules

XML Element	Event Specification	ECA Rule
<on>	A type of event used as input.	The only type of event to be handled by this rule.
<window>	A window used for event instance selection.	
<if>	Conditions	Conditions
<do>	Actions to be invoked if the conditions are satisfied	Actions to be invoked if the conditions are satisfied
<elsedo>		Actions to be invoked if the conditions are not satisfied

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 flows and 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 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.

5.1.2 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.

5.2 Event Specification and ECA Rules

Event processing specifications define rules for event processing in the event processing agent, while ECA rules define event-driven adaptations to be scheduled by the scheduler. We use structural XML to represent these two types of rules. Each rule type includes four types of XML elements, as listed in Table 4.

If the input of event processing includes different types of events, the corresponding event processing rule can have more than one <on> elements each of which corresponds to one type of event. If the input has to be partitioned or combined into groups of event instances for further processing, <on> elements can be replaced with <window> elements. A <window> element can define a time window or a moving window used for selecting event instances that satisfy some criterias. Examples of <window> elements will be given in Fig. 4.

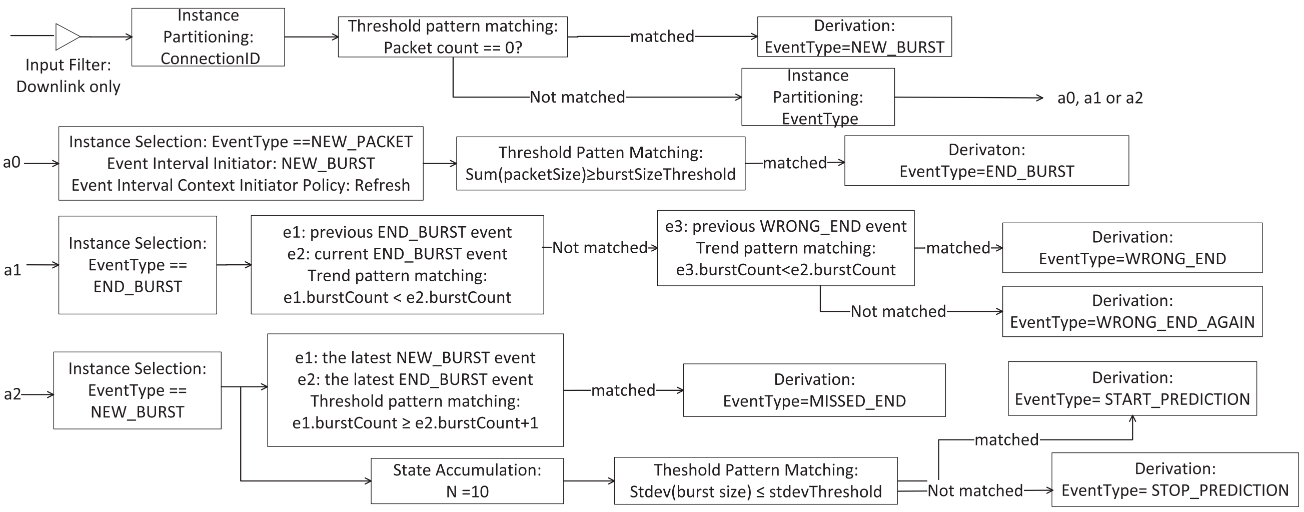

Input Filter: Downlink only
Instance Partitioning: ConnectionID
Threshold pattern matching: Packet count == 0?
matched
Derivation: EventType=NEW_BURST
Not matched
Instance Partitioning: EventType
a0, a1 or a2
a0
Instance Selection: EventType ==NEW_PACKET
Event Interval Initiator: NEW_BURST
Event Interval Context Initiator Policy: Refresh
Threshold Patten Matching: Sum(packetSize)≥burstSizeThreshold
matched
Derivaton: EventType=END_BURST
a1
Instance Selection: EventType == END_BURST
e1: previous END_BURST event
e2: current END_BURST event
Trend pattern matching: e1.burstCount < e2.burstCount
Not matched
e3: previous WRONG_END event
Trend pattern matching: e3.burstCount<e2.burstCount
matched
Derivation: EventType=WRONG_END
Not matched
Derivation: EventType=WRONG_END_AGAIN
a2
Instance Selection: EventType == NEW_BURST
e1: the latest NEW_BURST event
e2: the latest END_BURST event
Threshold pattern matching: e1.burstCount ≥ e2.burstCount+1
matched
Derivation: EventType=MISSED_END
Derivation: EventType= START_PREDICTION
matched
State Accumulation: N =10
Theshold Pattern Matching: Stdev(burst size) ≤ stdevThreshold
Not matched
Derivation: EventType= STOP_PREDICTION

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

Both event processing rules and ECA rules include the <if> element. The <if> describes the conditions that must be satisfied for executing actions defined in the <do> elements. The conditions are defined as a complex state, which is basically a combination of atomic states and other complex states formed with logical operators AND, OR, and NOT. During runtime, the condition expression can be parsed into a tree structure. The leaf nodes are capability values and their reference values. The nonleaf nodes are binary operators if their children are leaf nodes, or logical operators if their children are nonleaf nodes. We use a tree-traversal algorithm to implement the evaluation of the conditions.

The <do> and <elsedo> elements define actions that may be invoked. We define an action as a tuple: *(type, target, name, paramlist)*, where *type* denotes the type of the action, *component* is the identifier of the target hardware or software component, *name* is the identifier of the target attribute within the component and *paramlist* is a set of parameters for the action.

There are three operation types, *set*, *subscribe*, and *unsubscribe*. "Set" operations set hardware and software parameters and generate new event instances, while "subscribe" and "unsubscribe" operations subscribe and unsubscribe events, respectively.

5.3 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 events that are generated during event processing for further processing.

In this section, we present the implementation of event processing agent using traffic-aware WNI control as an example. The logical functions used in the example are shown in Fig. 3. When loading event specification into the event processing agent, different combinations of XML elements are mapped to logical functions according to Table 5.

Instance partitioning is divided into two types, segmentation-oriented partitioning and temporal-oriented partitioning. Segmentation-oriented partitioning classifies the event instances based on a certain event attribute. As shown in Fig. 3, the events accepted by the input filter are first partitioned based on the connection identifier. It means that the events related to different connections will be processed independently. The difference between filtering and segmentation-oriented partitioning in XML elements is that the event attributes used for differing event instances in segmentation-oriented partitioning are defined in the attributes of the <on> element.

Temporal-oriented partitioning divides the input into groups based on the timestamps of the event occurrence. As shown in Fig. 3, in branch a0, temporal-oriented partitioning is applied to the events with a type called NEW_PACKET. We implement temporal-oriented partitioning with a time window. Only the events 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 coexist with the old one, depending on the usage scenarios. In our example, as shown in Fig. 4, a new event instance with type of 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.

TABLE 5
Mapping XML Elements to Logical Functions

Logical function	XML elements
Filtering	<on> and <if>
Segmentation-oriented partitioning	<on> and <if>
Temporal-oriented partitioning	<window>
Derivation	<on> and <do>, or <window> and <do>
Pattern matching	<window>, <if> and <do>

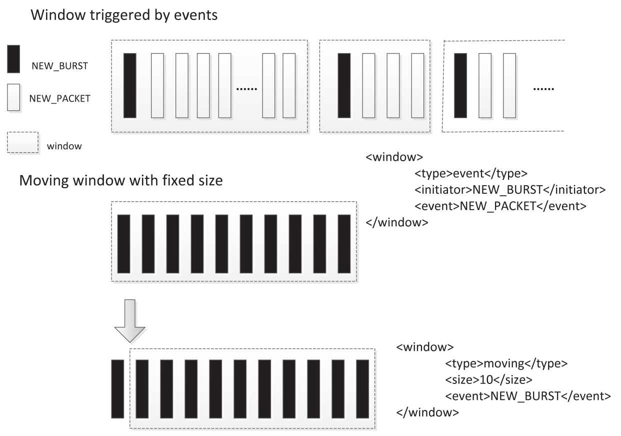


Fig. 4. Examples of windows used in event processing.

Besides time window, moving window is another kind of window defined in <window> elements. Moving window is used for selecting event instances for pattern matching in the example. In the sequence starting from a2 in Fig. 3, the standard deviation of the last 10 burst sizes is used when doing threshold pattern matching. As shown in Fig. 4, a moving window with fixed size of 10 is used for selecting the previous 10 NEW_BURST event instances as input.

5.4 Conflict Detection and Resolution

The scheduler checks for potential conflicts whenever loading a new rule. Given two rules, as described in Fig. 5, it first parses the actions defined in the new rule into operations to be applied to certain hardware/software components. After that, it searches for the previously loaded rules that may execute operations to the same component. If another rule that tries to set different values to the same parameter of the same component is found, there might be a conflict, and further investigation is needed. In this case, the scheduler parses the events defined in the two rules into tree structures according to their event specifications. If either of the resulting trees is a subtree of the other, the scheduler assigns a specificity-based conflict resolution policy to the rules as shown in Fig. 5. If neither of the trees is a subtree of the other, the scheduler checks whether the two events may occur at the same time. If simultaneous occurrence is possible, the scheduler will refuse to install the new rule and will ask the developer to set priorities to the conflicting rules. The information about which rules may cause conflict and which conflict resolution policy should be used is saved in the scheduler and is used for solving conflicts during runtime.

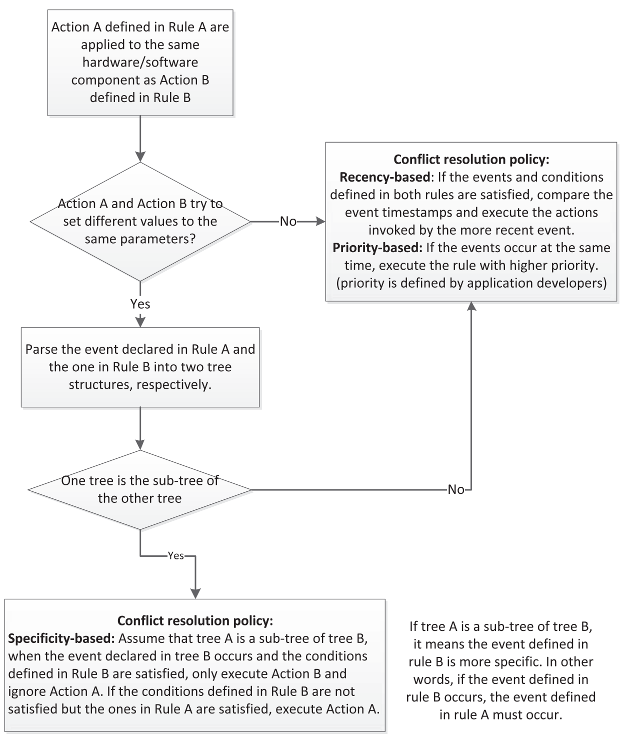

Action A defined in Rule A are applied to the same hardware/software component as Action B defined in Rule B
Action A and Action B try to set different values to the same parameters?
No
Conflict resolution policy:
Recency-based: If the events and conditions defined in both rules are satisfied, compare the event timestamps and execute the actions invoked by the more recent event.
Priority-based: If the events occur at the same time, execute the rule with higher priority. (priority is defined by application developers)
Yes
Parse the event declared in Rule A and the one in Rule B into two tree structures, respectively.
One tree is the sub-tree of the other tree
No
Yes
Conflict resolution policy:
Specificity-based: Assume that tree A is a sub-tree of tree B, when the event declared in tree B occurs and the conditions defined in Rule B are satisfied, only execute Action B and ignore Action A. If the conditions defined in Rule B are not satisfied but the ones in Rule A are satisfied, execute Action A.
If tree A is a sub-tree of tree B, it means the event defined in rule B is more specific. In other words, if the event defined in rule B occurs, the event defined in rule A must occur.

Fig. 5. Conflict detection and resolution.

6 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.

6.1 Experimental Setup

We ran the test on a Nokia N900. As shown in Fig. 6, the device was connected to a public 802.11 b/g access point, whose beacon interval was 100 ms. It means the mobile device woke up every 100 ms 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 1 MHz. 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.

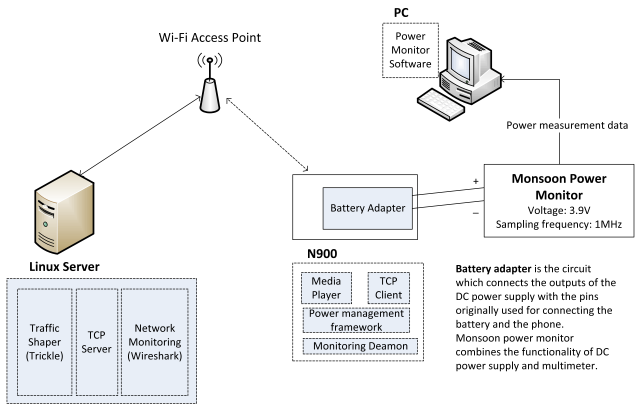
Traffic traces: We ran Wireshark[3] directly on the N900, and on the TCP server if the N900 was connected to it, to capture the packet information.

In the test cases where the network data rates needed to be specified, we used Trickle,[4] a bandwidth throttling software, to limit the data rate on the TCP servers.

6.2 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 1. 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 6, 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 percent 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 percent.

2. http://www.msoon.com/LabEquipment/PowerMonitor/.
3. http://www.wireshark.org.
4. http://monkey.org/~marius/pages/?page=trickle.



Battery adapter is the circuit which connects the outputs of the DC power supply with the pins originally used for connecting the battery and the phone.
Monsoon power monitor combines the functionality of DC power supply and multimeter.

Fig. 6. Experimental setup.

6.3 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 processed automatically, as shown in Fig. 3.

1. When START_PREDICTION occurs, start predicting the ending of each burst.
2. When STOP_PREDICTION occurs, stop predicting the burst endings.
3. When END_BURST occurs, set PSM timeout to 0 ms.
4. When NEW_BURST occurs and if the prediction of burst endings is enabled, set PSM timeout to 10 ms.
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 was turned either on or off. When the power management application was turned off, the PSM was enabled with the PSM timeout set to 100 ms. When the power management was running, we first initialized the burst size threshold to 4,000 Bytes. This threshold was updated with the unweighted mean of the previous five burst sizes. The exception was that when WRONG_END event 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 5,000 Bytes during initialization. It was updated with the simple moving average over the previous 10 burst sizes. We set the packet interval threshold to 10 ms. Choosing this particular setting was based on the observations we had made of the Internet traffic. In our measurements, 61.6 percent of the packet intervals were smaller than 10 ms and would thus be included inside bursts.

5. http://83.145.249.98:80/.

TABLE 6
Total Overhead Caused by Power Management

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

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.

TABLE 7
The Power Consumption of Listening to Internet Radio with and without Power Management

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

TABLE 8
Accuracy of Burst Prediction for Internet Radio Streaming and TCP File Download

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%

We repeated the experiments five times. Each run lasted for 4 minutes. The results, as listed in Table 7, show that the average power consumption of the device was 11.9 percent less with the power management system turned on.

Next we downloaded a 3,400 KB 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 shaped into 4 KB 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 Trickle traffic shaper. The average data rate during the file transmission was 15.97 KB/s. With the adaptations turned on, the energy consumed during the file download decreased by 13.6 percent from 87.37 to 76.94 J, at the cost of a 2 percent 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 8.

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 6 ms 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 type of WRONG_END. We then used the number of these event instances with type of WRONG_END for calculating the frequency of this first type of prediction errors.

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

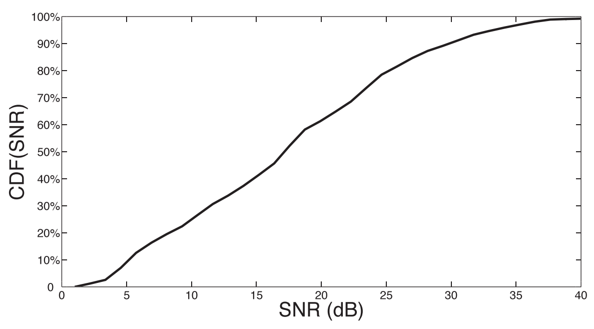


Fig. 7. Cumulative distribution of SNR.

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 50 ms if connected to an access point with a beacon interval of 100 ms. The average network throughput in our Internet radio experiment was 22.17 KB/s, 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 250 KB in size. From this information, we could calculate that a delay less than 10 second 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 adaptations 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.

6.4 SNR-Based Adaptive Network Transmission

We tested the file download from a remote server to the mobile 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 1 Hz. As shown in Fig. 7, 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 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 [23]. A file with size of 39.3 MB was downloaded from a TCP server to the mobile device. The data rate was limited to 512 KBps on server side. We adopted the following adaptation rules:

1. LOW_TO_HIGH_SNR and HIGH_TO_LOW_SNR events are subscribed when FIRST_CONNECTION occurs, and unsubscribed when NO_CONNECTION occurs.
2. Upon the occurrence of HIGH_TO_LOW_SNR, in case of TCP connection, it will be paused by setting the TCP receive window size to 0.

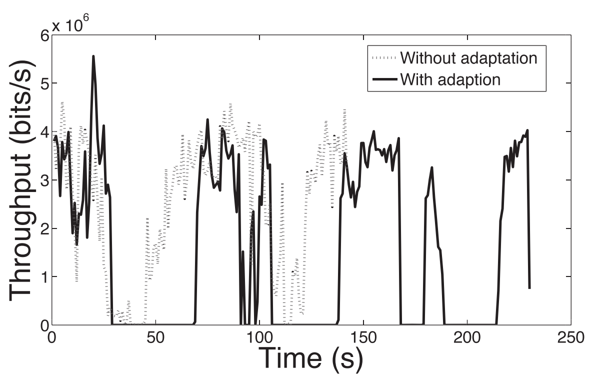


Fig. 8. Comparison of network throughput with/without adaptation.

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

As shown in Fig. 8, the download duration measured from the arrival of the first packet to the arrival of last packet was 161.97 seconds without adaptations, and 229.15 seconds with adaptations. Without adaptations, the TCP connection was 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 modes by 20 percent from 132.18 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 power models shown below.

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

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

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

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 [22] and [21]. 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.

There are two kinds of intervals, burst intervals, and the packet intervals inside bursts. The former are always bigger than the packet interval threshold, whereas the latter are smaller than it. In our measurement, we set the PSM timeout to 100 ms and packet interval threshold to 10 ms.

TABLE 9
Power Analysis of SNR-Based Transmission Adaptation

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

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 100 ms. 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 was 1.328 Mbps.

The values listed in Table 1 were used for our calculation. On average, it costs 93.68 J to download the file without adaptations, whereas it costs only 81.64 J with adaptations. With the adaptations, 12.85 percent less energy was consumed. To explain the energy savings achieved by our adaptation, we choose two samples. One is from the result without adaptation, and the other one from the result with adaptation. As shown in Table 9, the majority of savings come from the reduction in the energy used during burst intervals.

7 Conclusion

In this paper, we proposed an event-driven framework for rule-based power management for wireless data transmission on mobile devices. The framework supports complex event processing, which helps in decoupling event processing from event consumption and has the potential for reducing the event processing overhead if compared to simple event processing. We demonstrated the framework by using it to implement two power-saving applications, a traffic-aware WNI control application and a SNR-based transmission adaptation application. Measurements showed on average 12 percent of energy savings in both cases.

In the future, we see the potential of extending our framework toward collaborative power management between mobile devices. Contextual information collected from mobile devices can be shared to other mobile devices using central web services or peer-to-peer mechanisms. Our event-driven framework presented in this paper can be extended to support the collection and sharing of context data between mobile devices and also to utilize the context data that the other users produce for saving energy. The sharing of context information, such as traces of SNR measured in different locations, can help reduce the energy consumption spent in context monitoring and processing.

Acknowledgments

This work was supported by the Academy of Finland, grants number 253860 and 135230, and by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT). The authors also would like to thank Thomas Plagemann, and Vera Goebel for their valuable comments.

References

[1] G.D. Abowd, A.K. Dey, P.J. Brownd, N. Davies, M. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," *Proc. First Int'l Symp. Handheld and Ubiquitous Computing,* pp. 304-307, http://dl.acm.org/citation.cfm?id=647985.743843, 1999.

[2] E. Shih, P. Bahl, and M.J. Sinclair, "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," *Proc. MobiCom '02,* pp. 160-171, http://dx.doi.org/10.1145/570645.570666, 2002.

[3] A. Weissel and F. Bellosa, "Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management," *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems (CASES '02),* pp. 238-246, http://doi.acm.org/10.1145/581630.581668, 2002.

[4] B. Michelson, "Event-Driven Architecture Overview," Patricia Seybold Group, Feb. 2006.

[5] O. Etzion and P. Niblett, *Event Processing in Action.* Manning Publications, 2011.

[6] M. Anand, E.B. Nightingale, and J. Flinn, "Self-Tuning Wireless Network Power Management," *Wireless Networks,* vol. 11, pp. 451-469, July 2005.

[7] D. Bertozzi, L. Benini, and B. Ricco, "Power Aware Network Interface Management for Streaming Multimedia," *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC),* vol. 2, pp. 926-930, Mar. 2002.

[8] 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 J. Selected Areas in Comm.,* vol. 25, no. 4, pp. 722-737, May 2007.

[9] *IEEE Standard for Information Technology - Telecomm. 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 Standard 802.11-2007, Revision of IEEE Std 802.11-1999, p. C1-1184, June 2007.

[10] R. Krashinsky and H. Balakrishnan, "Minimizing Energy for Wireless Web Access with Bounded Slowdown," *Wireless Networks,* vol. 11, pp. 135-148, http://dx.doi.org/10.1007/s11276-004-4751-z, Jan. 2005.

[11] J. Liu and L. Zhong, "Micro Power Management of Active 802.11 Interfaces," *Proc. Sixth Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '08),* pp. 146-159, http://doi.acm.org/10.1145/1378600.1378617, 2008.

[12] F.R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices," *Proc. Eighth Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '10),* pp. 107-122, http://doi.acm.org/10.1145/1814433.1814446, 2010.

[13] D.C. Snowdon, E. Le Sueur, S.M. Petters, and G. Heiser, "Koala: A Platform for OS-Level Power Management," *Proc. Fourth ACM European Conf. Computer Systems (EuroSys '09),* pp. 289-302, 2009.

[14] H.S. Ashwini, A. Thawani, and Y.N. Srikant, "Middleware for Efficient Power Management in Mobile Devices," *Proc. Third Int'l Conf. Mobile Technology, Applications and Systems (Mobility '06),* 2006.

[15] A.B. Lago and I. Larizgoitia, "An Application-Aware Approach to Efficient Power Management in Mobile Devices," *Proc. Fourth Int'l ICST Conf. Comm. System Software and Middleware (COMSWARE '09),* pp. 11:1-11:10, 2009.

[16] J. Sorber, N. Banerjee, M.D. Corner, and S. Rollins, "Turducken: Hierarchical Power Management for Mobile Devices," *Proc. Third Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '05),* pp. 261-274, 2005.

[17] J. Manweiler and R. Roy Choudhury, "Avoiding the Rush Hours: Wifi Energy Management via Traffic Isolation," *Proc. Ninth Int'l Conf. Mobile Systems, Applications, and Services (MobiSys),* pp. 253-266, http://doi.acm.org/10.1145/1999995.2000020, 2011.

[18] M. Grossglauser and J.-C. Bolot, "On the Relevance of Long-Range Dependence in Network Traffic," *IEEE/ACM Trans. Networks,* vol. 7, no. 5, pp. 629-640, http://dx.doi.org/10.1109/90.803379, Oct. 1999.

[19] K.-c. Lan and J. Heidemann, "A Measurement Study of Correlations of Internet Flow Characteristics," *Computer Networks,* vol. 50, pp. 46-62, http://portal.acm.org/citation.cfm?id=1119569.1648543, Jan. 2006.

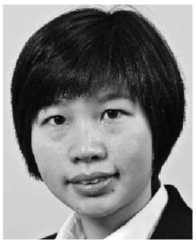
[20] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V.N. Padmanabhan, "Bartendr: A Practical Approach to Energy-Aware Cellular Data Scheduling," *Proc. MobiCom '10,* pp. 85-96, http://doi.acm.org/10.1145/1859995.1860006, 2010.

[21] R. Friedman, A. Kogan, and K. Yevgeny, "On Power and Throughput Tradeoffs of Wifi and Bluetooth in Smartphones," *Proc. INFOCOM '11,* Apr. 2011.

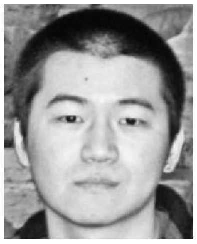
[22] Y. Xiao, P. Savolainen, A. Karpanen, M. Siekkinen, and A. Ylä-Jääski, "Practical Power Modeling of Data Transmission over 802.11g for Wireless Applications," *Proc. First Int'l Conf. Energy-Efficient Computing and Networking E-Energy,* pp. 75-84, 2010.

[23] R. Sri Kalyanaraman, Y. Xiao, and A. Ylä-Jääski, "Network Prediction for Energy-Aware Transmission in Mobile Applicatioins," *J. Advances in Telecomm.,* vol. 3, pp. 72-82, Nov. 2010.

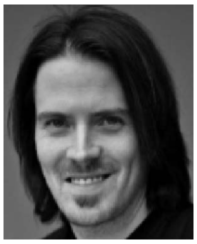
[24] G. Osman and S. Hasan, and C.M. Rahman, "Prediction of State of Wireless Network Using Markov and Hidden Markov Model," *Networks,* vol. 4, no. 10, pp. 976-984, 2009.



Yu Xiao received the MSc degree in computer science from Beijing University of Posts and Telecommunications in 2007, and the PhD degree in computer science from Aalto University in 2012. She is currently a postdoctoral researcher at the Department of Computer Science and Engineering, Aalto University. Her current research interests include energy-efficient wireless networking, crowd-sensing, and mobile cloud computing.



Wei Li received the BS degree in communications engineering from Nankai University, China in 2007 and the MS degree in digital signal processing from Aalto university, Finland in 2011. His current research interest include event processing on complex systems, mathematical modeling on computer as well as mobile platform programming.



Matti Siekkinen received the MSc degree in computer science from Helsinki University of Technology and in Networks and Distributed Systems from University of Nice Sophia-Antipolis in 2003, and PhD degree from Eurecom/University of Nice Sophia-Antipolis in 2006. He is currently a senior research scientist at the Department of Computer Science and Engineering, Aalto University. His research interests include energy efficiency in ICT, network measurements, and Internet protocols.

Petri Savolainen received the MSc degree in computer science from the University of Helsinki in 2004. He is currently a doctoral student at the Department of Computer Science, University of Helsinki and also a researcher at Helsinki Institute for Information Technology. His current research interests include peer-to-peer networking, energy-efficient computing, and mesh networking.

Antti Ylä-Jääski received the PhD degree in ETH Zuerich 1993. He has worked with Nokia 1994-2009 in several research and research management positions with focus on future Internet, mobile networks, applications, services and service architectures. He has been a professor for Telecommunications Software, Department of Computer Science and Engineering, Aalto University since 2004. His current research interests include green ICT, mobile computing, service, and service architecture. He is a member of the IEEE and the IEEE Computer Society.

Pan Hui received the PhD degree in computer science from the University of Cambridge, and the BEng and MPhil degrees both from the University of Hong Kong. He is a senior research scientist and principal investigator at Deutsche Telekom Laboratories (T-labs) Germany and an adjunct professor at Aalto University Finland. He was also an affiliated researcher with Intel Research Cambridge. His current research interests include social networking and computing, cloud computing, mobile and pervasive systems, mining of large-scale mobility data, and the application of complex network science in communication systems design. He is a member of the IEEE.