Powertrace: Network-level Power Profiling for Low-power Wireless Networks

Adam Dunkels, Joakim Eriksson, Niclas Finne, Nicolas Tsiftes
{adam,joakime,nfi,nvt}@sics.se
Swedish Institute of Computer Science

March 2011
SICS Technical Report T2011:05
ISSN 1100-3154

Abstract

Low-power wireless networks are quickly becoming a critical part of our everyday infrastructure. Power consumption is a critical concern, but power measurement and estimation is a challenge. We present Powertrace, which to the best of our knowledge is the first system for network-level power profiling of low-power wireless systems. Powertrace uses power state tracking to estimate system power consumption and a structure called energy capsules to attribute energy consumption to activities such as packet transmissions and receptions. With Powertrace, the power consumption of a system can be broken down into individual activities which allows us to answer questions such as "*How much energy is spent forwarding packets for node X?*", "*How much energy is spent on control traffic and how much on critical data?*", and "*How much energy does application X account for*?". Experiments show that Powertrace is accurate to 94% of the energy consumption of a device. To demonstrate the usefulness of Powertrace, we use it to experimentally analyze the power behavior of the proposed IETF standard IPv6 RPL routing protocol and a sensor network data collection protocol. Through using Powertrace, we find the highest power consumers and are able to reduce the power consumption of data collection with 24%. It is our hope that Powertrace will help the community to make empirical energy evaluation a widely used tool in the low-power wireless research community toolbox.

1 Introduction

Low-power wireless networking is rapidly becoming a critical part of our everyday infrastructure through the deployment of electrical metering, building energy management, future smart cities, and the smart grid [30].

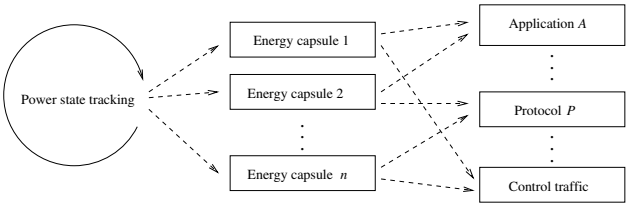


Figure 1: Powertrace uses power state tracking to estimate the power consumption of the system. The energy for individual activities, such as packet transmissions and receptions, are captured in energy capsules. The energy capsules can be attributed to applications, protocols, or other activities such as control traffic. Combining power profiles from nodes in the network gives a network-level power profile.

Low-power wireless devices are used in many situations, for example to monitor the power consumption of appliances in smart grid networks, to sense temperature and other environmental data in city automation systems, and to control heating systems for office building automation. Low-power wireless devices are often battery-powered, making power consumption a central concern.

Power-efficient mechanisms is an active topic in many fields [25], such as server systems [1, 32], data centers [19], mobile and handheld applications [9, 20], and low-power wireless networking, which has been the focus of research for over a decade in the sensor network community. The community has developed protocols, mechanisms, and algorithms for low-power wireless networking, but surprisingly few of those have been empirically evaluated in terms of power consumption. In part, this is because energy is difficult to measure [13], but also because systems and tools for measuring the power consumption of individual sensor network nodes [5, 8, 10, 13, 24] have required custom hardware

designs or have been difficult to use, and have not provided any means to view energy consumption at the network level.

We present Powertrace, a system for network-level power profiling for low-power wireless networks. Powertrace attributes network-level power consumption to the activities that cause the power to be spent. Powertrace uses power state tracking to estimate the power consumption of the local node, records the energy consumption in a energy capsules that represent node-level activities such as packet reception or packet transmission, and attributes the energy capsules to network-level activities such as individual applications or individual protocol activities including routing, forwarding, or control traffic. The process is shown in Figure 1. By attributing energy capsules to network-level activities, Powertrace can answer questions such as *"What is the average power spent on control traffic?"*, *"What is the power cost for forwarding packets on behalf of other nodes?"*, and *"Are there nodes that spend more energy than others, and why is this the case?"*. With Powertrace, researchers can measure, empirically evaluate, and compare the energy consumption of their low-power wireless systems; protocol designers can profile the power consumption of their protocols; system developers can debug and verify the operation of their system; and network operators can inspect the energy consumption of their networks.

We make two primary contributions with this paper. To the best of our knowledge, we are the first to demonstrate network-level power profiling for low-power wireless networks. We demonstrate that the method is useful by applying it to existing network protocols from the literature. Our results gives many insights into the power behavior of these protocols that have previously not been seen, such as that the largest part of power consumption is spent on idle wake-ups. Moreover, we quantify and evaluate the accuracy of software-based power state tracking as a way to measure energy consumption in low-power wireless systems. Additionally, we are the first to provide an empirical power evaluation of the proposed standard IETF RPL routing protocol for low-power wireless IPv6 networks. Taken together, we hope that the simplicity and accuracy of Powertrace combined with its broad usefulness will allow the community to move towards empirical energy evaluation as a preferred tool in the low-power wireless research toolbox.

We have implemented Powertrace for the Contiki operating system, but the methods are general enough to be applied to any operating system. A prototype version of Powertrace has been available for Contiki for some time, and has already been used by others to gain insight into their low-power wireless systems, resulting in a number of published papers. Powertrace has been used on at least four different hardware platforms (Tmote Sky, ESB, MicaZ, Zolertia Z1), with no code modifications needed when crossing platforms. Powertrace needs instrumentation, but the instrumentation is simple, requiring only a handful of lines of code in device drivers and radio duty cycling protocols.

The rest of this paper is structured as follows. Section 2 motivates the need for energy profiling for low-power wireless networks. In Section 3 we introduce Powertrace and describe how its power state tracking works and how energy capsules are used to attribute energy consumption to activities. Section 4 describes our implementation of Powertrace for Contiki. In Section 5 we evaluate the accuracy and overhead of Powertrace and demonstrate its usefulness by applying it to two important protocols from the literature: sensor network data collection and the proposed standard IETF low-power IPv6 RPL routing protocol. We discuss the implications of our findings and future directions in Section 6, review related work in Section 7, and conclude the paper in Section 8.

2 Communication and Power in Low-power Wireless

In low-power wireless networks, communication and power consumption are intertwined. The communication device is typically the most power-consuming component, but merely refraining from transmissions is not enough to attain a low power consumption: the radio consumes as much power in listen mode as when actively transmitting. To reduce power consumption, the radio must be switched completely off—duty cycled—as often as possible.

Low power consumption is important in different types of wireless systems for a number of reasons. For battery-powered systems, the energy consumption of the system determines its lifetime. Since batteries often cannot be easily replaced, low energy consumption translates into lower operating costs. For systems that are powered by an energy scavenging source, such as solar cells or vibration energy, the power supplied by the power source is small. Even systems with an always-on power source often need to maintain a low power consumption. For example, smart grid electrical meters, whose one purpose is to reduce overall power consumption of the electrical grid, cannot themselves have a too high power consumption, lest the power consumption of the smart grid infrastructure outweighs the potential savings. Moreover, the power consumption of a device often determines the cost and the physical size of the power converter of the device, so a low power consumption results in a smaller system, with a lower cost.

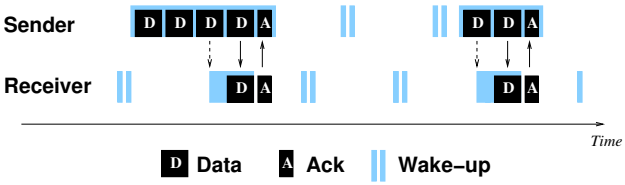


Figure 2: The basic operation of the ContikiMAC low-power radio duty cycling mechanism.

2.1 Attaining Low Power Consumption: Radio Duty Cycling

The purpose of radio duty cycling is to turn off the radio as much as possible but still retain the ability to communicate. When the radio transceiver is off, a node cannot receive transmissions from neighbors. To be able to communicate while keeping the radio switched off as much as possible, the radio must wake up periodically to be able to receive packets from neighbors. Numerous radio duty cycling mechanisms have been developed [2, 4, 21, 22, 28, 31].

As a specific example, for the popular Tmote Sky sensor network mote platform with its CC2420 radio transceiver, the CC2420 consumes 63 mW of power in listen mode. When transmitting data, the transceiver draws slightly less, 60 mW. In comparison, the CPU draws 5 mW in active mode and 0.1 mW in sleep mode. If the radio is always turned on, regardless of the activity on the node, the power consumption will remain at approximately 60 mW.

In a duty-cycled network, nodes perform three distinct actions: transmit packets, receive packets, and periodically wake up to be able to receive packets from neighbors. Wake-ups can be either scheduled or non-scheduled. With scheduled wake-ups, nodes agree on specific times for wake-ups so that senders always know when potential receivers will be awake. Scheduled schemes has an overhead in setting up and maintaining their schedules and are suitable for static networks, such as industrial monitoring systems [21]. Examples of duty cycling mechanisms with scheduled wake-ups are S-MAC [31] and TSMP [21]. With non-scheduled wake-ups, nodes wake up independently, either with an exact periodicity or randomly. A sender, who may not know when its receiver is awake, must send a preamble of wake-up transmissions to the receiver, before sending the actual data packet. Alternatively, the receiver can transmit a probe packet every time it wakes up to let potential senders know that it is awake. Examples of protocols with opportunistic wake-ups are B-MAC [22], ContikiMAC [4], X-MAC [2], and RI-MAC [28].

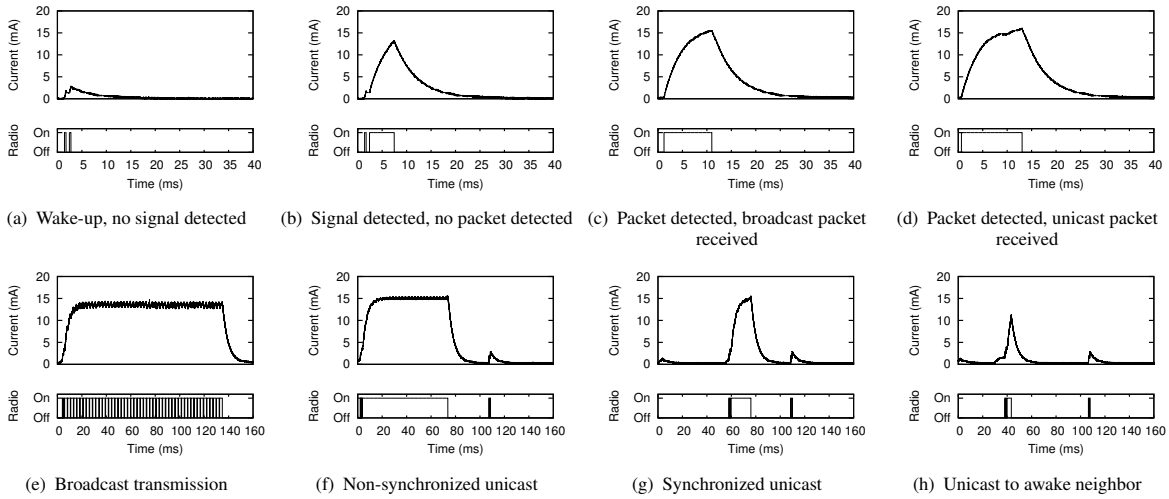
Throughout this paper, we use the ContikiMAC duty-cycling mechanism [4], the default radio duty cycling mechanism in the Contiki operating system. As shown in Figure 2, ContikiMAC uses a periodic wake-up mechanism configured with constant, uniform intervals. A ContikiMAC wake-up consists of two consecutive radio channel samples, whose timing is selected to allow either of the two samples to catch a transmission from neighbors. To send a packet to a ContikiMAC node, the sender repeatedly transmits the data packet until it hits the receiver's wake-up slot. After discovering that a packet is in the air, the receiver keeps its radio on to receive the transmission. Before going to sleep again, the receiver replies with an acknowledgment packet if the packet reception was successful. The sender stops sending packets and records the time at which the acknowledgment was received. Since wake-ups are periodic, the sender can synchronize with the receiver's wake-up phase to make subsequent transmissions shorter.

2.2 Counting Transmissions is Not Enough

In the absence of easy-to-use energy measurement techniques, the sensor network community has often used packet counting as a proxy for energy consumption. This energy estimation method is based on the assumptions that radio wake-ups and packet receptions consume no or an insignificant amount of energy, and that packet transmission energy is constant. To challenge these assumptions, we measure the energy consumption of the actions of the ContikiMAC duty cycling protocol by attaching an oscilloscope to a 100 Ω resistor connected in series with a power source to a Tmote Sky sensor network mote.

Figure 3 (a)-(d) shows the current draw for wake-ups. The energy consumption depends on the activity in the radio medium. If no radio signal is detected, the radio can quickly go back to sleep (a). If a radio signal is detected, the radio is kept on in anticipation of an incoming packet, but if no valid packet is detected, the radio can go back to sleep (b). We call this a false positive wake-up. If a packet is detected, the radio is kept on to receive the full packet. If a broadcast transmission is received (c), no link-layer acknowledgment is sent. If a unicast transmission is received (d), a link-layer acknowledgment is sent, which causes more energy to be spent.

Figure 3 (e)-(h) shows the current draw for packet transmissions in four different situations: one broadcast transmission and three unicast transmission. The broadcast transmission (e) is the longest of them because the transmission must reach all neighbors. Since the wake-up schedule of the neighbors is unknown, the broadcast transmission must extend over the full wake-up period, which in this case is 125 ms. For unicast transmissions, it is enough to reach one receiver, and the transmission can stop when the sender has heard a link-layer acknowledg-



(a) Wake-up, no signal detected (b) Signal detected, no packet detected (c) Packet detected, broadcast packet received (d) Packet detected, unicast packet received

(e) Broadcast transmission (f) Non-synchronized unicast (g) Synchronized unicast (h) Unicast to awake neighbor

Figure 3: Current draw of radio transmissions with ContikiMAC on the Tmote Sky. The shark fin-like patterns are due to capacitor buffers in the Tmote Sky designed to smooth out current spikes. Notice that timescales are different in figures (a)-(d) and in (e)-(h).

Activity	**Energy (uJ)**
Wake-up, no signal detected	12
False positive wake-up	100
Broadcast reception	178
Unicast reception	222
Broadcast transmission	1790
Non-synchronized unicast transmission	1090
Synchronized unicast transmission	120
Unicast transmission to awake receiver	96

Table 1: Energy consumption for the ContikiMAC activities in Figure 3.

ment from the receiver (f). With fixed wake-up schedules, the sender can then phase-lock onto the neighbor so that the next transmission can start just before the neighbor wakes up, thus reducing the transmission cost (g). Neighbors may sometimes be fully awake, for example, because they are designated routers that have a continuous power supply. If the neighbor was fully awake, the transmission does not need to wake the neighbor up, and is therefore much faster (h).

Table 1 summarizes the energy consumption of the ContikiMAC actions from Figure 3. We see that the energy cost of wake-ups and receptions may differ by an order of magnitude, depending on the situation. Likewise, we see that the cost of a packet transmission may differ by an order of magnitude depending on the type of transmission and on the receiver, even when the size of the data is constant. Also, the cost of wake-ups is on the same order of magnitude as unicast transmissions. Taken together, these data suggest that the number of transmissions is not a good predictor of total system energy, but that more detailed energy models are needed.

3 Power Profiling with Powertrace

Powertrace is a run-time power profiling mechanism that uses power state tracking to estimate the power consumption of each node, breaks the power consumption into energy capsules, and attributes them to higher-level activities. Examples of such activities are individual applications, individual protocols, or protocol mechanisms such as control traffic, packet forwarding, or retransmissions. Powertrace allows both inspection of node-level energy behavior and of network-level protocol power profiles.

An example of how Powertrace operates is shown in Figure 4. The figure shows a timeline of activities in a Powertrace system. The system first performs a periodic wake-up, then attempts to transmit a packet but senses a simultaneous radio transmission from a neighbor, then successfully retransmits the same packet, and finally performs a second periodic wake-up. Powertrace tracks the power states of the system as it goes through its activities. For each activity, Powertrace records the estimated energy in a corresponding energy capsule. As seen in the figure, multiple power states contribute to an energy capsule and energy capsules. For example, a transmission capsule contains energy contributions from both the radio transmission and the radio listen state.

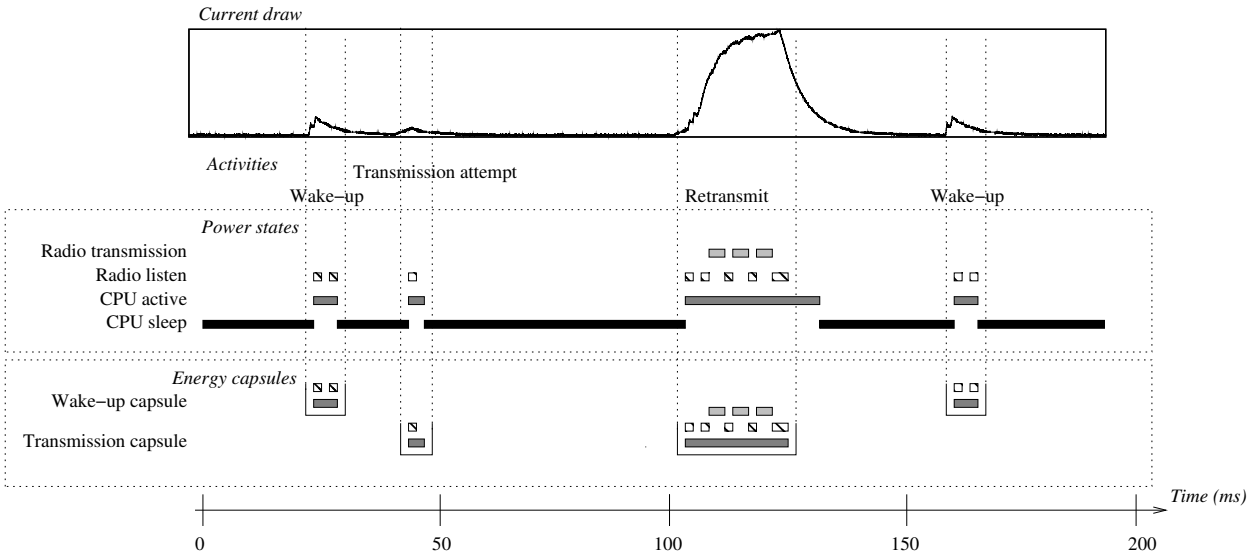


Figure 4: Measuring communication energy expenditure with Powertrace: the radio duty cycling layer maintains energy capsules for wake-ups, transmissions, and receptions. In the figure, capsules for wake-up and transmissions are shown. The transmission capsule is split across two activities: the first transmission attempt at 40 ms, which sensed another transmission in the ether and backed off, and the retransmission at 100 ms.

3.1 Power Model

Powertrace uses a linear power model in which the instantaneous power is estimated as the sum of all active power states. System energy is derived from the time that the system spends in each power state. The instantaneous system power consumption $P_{\text{system}}(t)$ at time t be described as

$$P_{\text{system}}(t) = \sum_{m,n} P_{m,n} s_{m,n}(t), \qquad (1)$$

where $P_{m,n}$ is the power consumption of component m in power state n and $s_{m,n}(t)$ is either 0 or 1, depending on whether the state is entered at time t or not. Examples of components are the CPU, the radio transceiver, onboard flash memories, and sensors. Examples of power states are the CPU in active mode or in sleep mode and the radio in listening mode or in transmission mode. Likewise, the energy model E_{system} is

$$E_{\text{system}} = \sum_{n,m} P_{m,n} T_{m,n}, \qquad (2)$$

where $T_{m,n}$ is the time during which component m has been in state n.

The constant factors $P_{m,n}$ can be either pre-calibrated using off-line measurement or calibrated at run-time, e.g., by using techniques such as those developed by as Fonseca et al. [10]. If the power model is used for comparing two protocols or mechanisms the actual constants cancel out and only the time factors $T_{m,n}$ remain. For example, in many cases it is advantageous to report power consumption of low-power wireless protocols as the measured radio duty cycle: the percentage of time in which the radio was turned on. Reporting power consumption as its radio duty cycle makes it possible to compare protocols across hardware platforms, which may have different constant factors for Equation 2 but have similar timing.

3.2 Power State Tracking

Powertrace tracks system power states by measuring the time during which components are in each power state. Device drivers are instrumented to record a time stamp when a component enters a new state. When the component leaves its state, the time difference is computed, and added to the corresponding $T_{m,n}$.

Power state tracking is widely used to assess the energy consumption of general-purpose computers [32] and handheld systems such as Android and Windows Mobile [20]. It has also previously been used in networked embedded systems [5, 12].

The power state tracking in Powertrace is done entirely in software and no additional hardware is needed. This has both advantages and disadvantages compared to hardware-based energy measurement mechanisms, such as the methods developed by Ritter et al. [24] and Fon-

seca et al. [10]. A software-based mechanism is not affected by environmental factors that affect the energy consumption of the system, such as temperature and humidity. Moreover, software-based techniques yield the same result on different batches of the same hardware, which is not necessarily true for a hardware-based method.

3.3 Energy Capsules

In Powertrace, an energy capsule contains a representation of the energy of an activity, such as the transmission or reception of a packet, or reading a block of data from a file on external flash memory. Each energy capsule is associated with a set of power states. Powertrace records the energy consumption of the activity by *opening* an energy capsule when the activity starts and *closing* when the activity ends. When the capsule is open, it records the energy from the power states with which it is associated. An individual activity can span across multiple lower level operations that are separated in time. Capsules can therefore be opened and closed multiple times. Likewise, multiple energy capsules can be open at any given time, tracking energy for multiple operations over different power states.

Energy capsules are controlled by the module in charge of its energy consumer. For example, a flash file system maintains energy capsules for creating, reading, and writing files. A radio duty cycling mechanism maintains energy capsules for radio communication.

When sending a packet, the radio duty cycling layer opens an energy capsule to hold the energy consumption for the transmission. Many low-level activities contribute to the energy consumption of a transmission. In Figure 4, the packet transmission is first initiated after approximately 40 ms. The radio duty cycling layer samples the radio channel, but finds a transmission from a neighbor and consequently sets a retransmission timer to expire approximately 60 ms in the future. The energy capsule is closed and opened again for the next transmission. The radio is again set to listen mode to check for transmissions from neighbors, but this time the packet can be sent. The radio duty cycling layer now repeatedly transmits its packet until a link layer acknowledgment is received from the receiver. Between the transmissions it sets the radio to listen mode to be able to receive the acknowledgment. When the acknowledgment is received, the radio is turned off and energy capsule is closed. Control is then returned up the network stack, which keeps the CPU in active mode a while longer, but this energy is not attributed to the transmission capsule since it is not part of the radio transmission.

3.4 Capsule Aggregates

Multiple energy capsules can be combined into energy capsule aggregates. A capsule aggregate contains the sum of the energy from all its capsules. Powertrace uses capsule aggregates to attribute communication power consumption to network protocols, which are identified by port number or protocol identifier. Applications or protocols that use Powertrace can also use them to aggregate energy information.

Powertrace allows applications or protocols to subscribe to its capsule feed. When a capsule's activity has completed, the controlling module informs Powertrace, which distributes the capsule to its subscribers. Subscribers can process the energy capsules immediately, or store them on secondary storage, such as a flash memory, for later off-line analysis.

Applications or protocols may use capsule aggregates to maintain their own information about the energy consumption of the system, which allows them to make energy-aware decisions. For example, a routing protocol can subscribe to the capsule feed to get information about the energy costs of transmitting data to particular neighbors. This information can be aggregated into the routing table, which allows the protocol to base its routing decisions on the average transmission energy for its potential parents, allowing routing protocols to be more energy-efficient.

4 Implementation

We have implemented Powertrace in the Contiki operating system for low-power wireless networks[1]. Contiki includes the uIPv6 certified low-power IPv6 stack [7] and the Rime sensornet communication stack [6], allowing us to evaluate Powertrace with two different network stacks. Contiki also provides a set of duty cycling mechanisms, including ContikiMAC [4] and X-MAC [2], that we use when evaluating Powertrace. The principles behind Powertrace are generic enough for it to be implemented for any low-power wireless operating system, however.

4.1 Instrumentation

Powertrace requires instrumentation of the system to track power states and to generate energy capsules, but the amount of instrumentation needed is small: instrumentation typically requires only a handful lines of code. Figure 5 shows the Powertrace instrumentation of the Contiki CC2420 low-power radio device driver, in pseudo-code. The listing shows the functions for turning

[1] http://www.sics.se/contiki

on listen mode, turning off listen mode, and for transmitting a packet. The instrumentation adds one line of code every time a power state is changed. In our Powertrace implementation in Contiki, we needed to add 7 lines of instrumentation code to the existing 900 lines of code in the CC2420 driver.

Figure 6 shows the energy capsule instrumentation, in pseudo-code, for the radio duty cycling layer. This instrumentation is needed to detect when the radio is transmitting packets and when it is waking up to sample the radio channel for activity. For transmissions, a transmission energy capsule is opened when initiating a transmission. The transmission code first samples the channel to see if someone else is currently transmitting. If so, it turns of the radio, closes the transmission capsule, and sets up a retransmission timer. By opening the capsule before sampling the channel, the cost of the channel sample will be included in the transmission capsule.

If no transmission was detected, the packet is sent repeatedly until an acknowledgment is received. The packet is sent repeatedly because the receiver may be asleep. In this simplified example, there is no synchronization, but in a real protocol such as ContikiMAC, the sender records the wake-up schedule of its neighbors to optimize subsequent transmissions. When the transmission has been completed, the capsule is closed. By calling capsule_done(), Powertrace is informed that the energy capsule is ready to be sent to subscribers.

The wake-up code is straightforward, but uses two energy capsules to distinguish between an idle wake-up and a packet reception. Before turning the radio on, both the idle wake-up capsule and the reception capsule are opened. If no packet was detected, the wake-up capsule is closed and the reception capsule is rewound: the capsule is reverted to its last state. If a packet was detected, it is received by the radio, and the reception capsule is closed and the wake-up capsule is rewound.

The amount of instrumentation needed for energy capsules is small. In our ContikiMAC implementation, we added 15 lines of instrumentation to the existing 1200 lines of code. In Contiki's X-MAC implementation, we added 13 lines of code to its 1000 lines of code, and the ContikiTDMA time-synchronized duty cycling protocol got its 700 lines of code extended by 7 lines of instrumentation code.

4.2 Two-timer Calibration

Powertrace's power state tracking uses hardware timers to measure the time for each power state. For long-lived power states, the timer must be able to count the entire lifetime of the power state, without wrapping. For short-lived power states, the timer must have a tick rate that is high enough to measure the power state.

```
cc2420_listen() {
  powerstate_on(RADIO_LISTEN);
  strobe(CC2420_SRXON);
  is_listening = 1;
}
cc2420_off() {
  strobe(CC2420_SRFOFF);
  powerstate_off(RADIO_LISTEN);
  is_listening = 0;
}
cc2420_send() {
  if(is_listening) powerstate_off(RADIO_LISTEN);
  powerstate_on(RADIO_TRANSMIT);
  strobe(CC2420_STXON);
  powerstate_off(RADIO_TRANSMIT);
  if(is_listening) powerstate_on(RADIO_LISTEN);
}
```

Figure 5: Excerpts of the CC2420 low-power radio driver, instrumented to track Powertrace power state changes. The strobe() function is CC2420-specific and used to send commands to the radio hardware.

```
send_packet() {
  capsule_open(transmission_capsule);
  cc2420_listen();
  if(cc2420_packet_detected()) {
    /* Someone else is sending. */
    cc2420_off();
    capsule_close(transmission_capsule);
    set_retransmission_timer();
    return;
  }
  do {
    /* Repeatedly transmit until
    receiver wakes up */
    cc2420_send();
    /* Listen for ACK */
    if(cc2420_packet_received()) {
      /* Read ACK */
      cc2420_read();
      break;
    }
  } until(timeout);
  cc2420_off();
  capsule_close(transmission_capsule);
  capsule_done(transmission_capsule);
}
wakeup_radio() {
  capsule_open(wakeup_capsule);
  capsule_open(receive_capsule);
  /* Sample channel for activity. */
  cc2420_listen();
  if(cc2420_packet_detected()) {
    /* A packet was detected */
    capsule_rewind(wakeup_capsule);
    cc2420_receive_packet();
    capsule_close(receive_capsule);
    capsule_done(receive_capsule);
  } else {
    /* No packet detected */
    capsule_rewind(receive_capsule);
    capsule_close(wakeup_capsule);
    capsule_done(wakeup_capsule);
  }
}
```

Figure 6: Powertrace instrumentation of the radio transmission and wakeup code.

Timer resolution is particularly important for actions

that are timer-driven. If the action is driven by the same time source as the power state tracking, the power state timing may be consistently under-measured. For example, the ContikiMAC wake-up mechanism is driven by a timer interrupt that uses Tmote Sky 32768 Hz hardware timer. By using the same timer source to measure its power state, the measurement is consistently skewed: the radio listen power state during a ContikiMAC wake-up is exactly 12.3 ticks long.

To combine measurement of long-lived and short-lived phenomena, our Tmote Sky Powertrace implementation uses a combination of two hardware clocks: the 32768 Hz timer source and the internal CPU cycle clock. Power states are always measured with the 32768 Hz timer, but for states that have a lifetime below 32 ticks (1 millisecond), the CPU cycle clock is used. A conversion table is used to convert between the two clocks. Since the CPU cycle clock is unstable and affected by external factors such as temperature, Powertrace must make sure to calibrate the clocks and regenerate the conversion table if the clocks drift. In our current implementation, we periodically (every 30 seconds) recalibrate the clocks, but it is also possible to trigger calibration by events that are known to skew the clocks, such as rapid temperature changes.

5 Evaluation

We evaluate Powertrace in three ways. First, we evaluate the accuracy of the energy consumption as reported by Powertrace through comparing it with oscilloscope energy measurements. Our results show that Powertrace is accurate within 94% compared with hardware-based power measurement. Second, we evaluate the overhead of Powertrace and the energy capsule mechanism. We find that Powertrace has a negligible overhead, but if energy capsules are stored to flash, performance is affected. Third, we demonstrate the usefulness of Powertrace through a set of case studies where we use Powertrace to study the energy consumption of two communication protocols for low-power wireless: the Contiki Collect data collection protocol and the IETF proposed standard RPL protocol for routing in low-power IPv6 networks. Based on information from the Powertrace power profile, we are able to reduce the system energy consumption of the data collection network by 24%.

For our experiments, we use the Tmote Sky mote [23], a widely used sensor network platform. We use a small-scale experimental setup for the accuracy and overhead experiments, and a 17-node testbed for the network experiments. To ensure repeatability, we also use the Contiki simulation environment for a subset of the case studies. The Contiki simulation environment combines a cycle-accurate simulation of the Tmote Sky platform with a bit-level accurate simulation of its CC2420 radio transceiver.

5.1 Accuracy

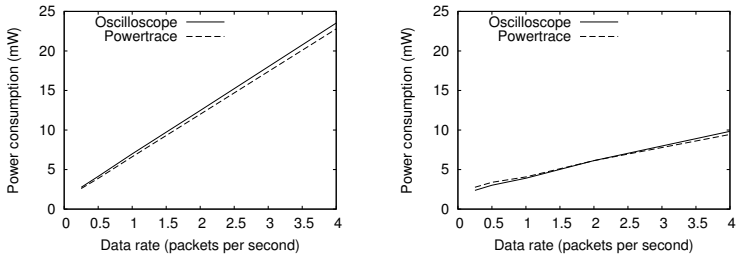
The accuracy of Powertrace depends on two factors: how closely the energy estimation through power state tracking matches the actual energy consumed by the hardware, and if the energy capsule mechanism is able to correctly attribute energy to activities.

To evaluate the accuracy of the power state tracking mechanism, we run an experiment where one Tmote Sky is instrumented with energy measurement hardware and where we compare the output of Powertrace with the hardware energy measurement. To measure the energy of the hardware, we use an oscilloscope to measure the voltage across a $100\,\Omega$ resistor connected in series with an external power source, which is set to deliver 4.5 V. The oscilloscope samples the voltage over the resistor at 2 MHz. We use three auxiliary motes to act as communication partners and to generate background traffic noise.

On the mote under test, we run an application that transmits broadcast and unicast packets to one of the auxiliary motes. For all incoming packets, the auxiliary mote responds with either a broadcast or a unicast transmission. We vary the data rate between one packet every eight seconds to four packets per second. We vary the background noise conditions from fully silent to WiFi background noise with cross-traffic from the auxiliary motes. This experiment thus exposes the mote under test both to direct traffic, non-intelligible background noise, and background noise in the form of packets that are processed by the mote. Each experiment is run for 10 minutes.

The result of the experiment is shown in Figure 7, which shows the energy converted into power consumption. To convert the Powertrace timing data into power, we insert numbers from the Tmote Sky data sheet into Equation 2. We empirically found the data from the data sheet to match the power consumption of the Tmote Sky mote used in the measurements. As shown in the figure, the power consumption reported by Powertrace is accurate to 94% of the measured power consumption. We expect that run-time calibration [10] can improve the accuracy further.

To validate the correctness of the energy capsule attribution mechanism, we run an experiment with two Tmote Sky motes. One mote acts as a sender and the other as receiver. The sender sends unicast traffic to the receiver over two different application-layer connections. Both the sender and the receiver record their per-application energy consumption. Figure 8 shows the result. The energy attributed to the two applications match



(a) No radio interference.

(b) With radio interference.

Figure 7: The power consumption as a function of the application-level data rate, measured with Powertrace and with an oscilloscope, with and without external interference. With interference, the power consumption is lower because fewer transmissions are made when collisions are detected.

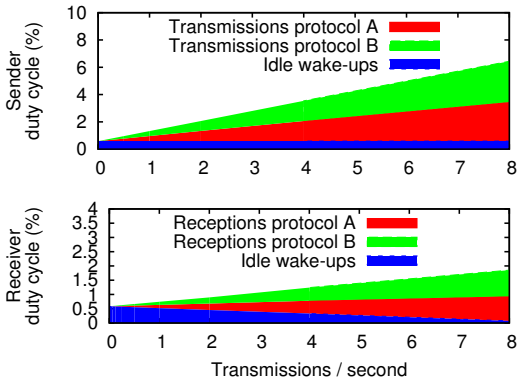


Figure 8: Two applications running on a sender node sends data to a receiver. Powertrace attributes their transmission and reception energy to the two applications. The energy spent on idle wake-ups drops as the number of receptions increase.

their transmission rate. We also see that the energy spent on idle wake-ups decrease as the traffic rate increases. This is expected, as the number of wake-ups that result in a packet reception increases with the traffic rate.

5.2 Overhead

We evaluate the run-time overhead of Powertrace by measuring its impact at the system level. We run an experiment where two nodes send exchange packets as fast as they can, without any radio duty cycling, and measure the number of packets they are able to send. We vary the type of processing that is done on the energy capsules: on-line aggregation of energy values and storing the energy capsules to flash for off-line analysis. For storing capsules to flash, we use the Contiki Coffee file system [29]. We would expect that storing the capsules

Setup	Throughput
Without Powertrace	69.8 ± 0.017
Powertrace with on-line analysis	69.4 ± 0.017
Powertrace with capsules to flash	66.6 ± 0.019
On-line analysis, w/ ContikiMAC	7.51 ± 1.3
Capsules to flash, w/ ContikiMAC	6.22 ± 2.0

Table 2: Throughput in packets per second with and without Powertrace, and with and without writing energy capsules to flash.

to flash to have a measurable effect on performance. On purpose, this experiment have high data rates that typically falls outside of the usage pattern of low-power wireless networks, but exposes the overhead of Powertrace.

The results of the experiments are shown in Table 2. We see the effect of Powertrace with on-line analysis is negligible on the packet rate. Powertrace reduces the data rate with only 0.6%. Storing every energy capsule to flash reduces the data rate by 4.6%, however. But when radio duty cycling is introduced, the throughput drops dramatically, and the spread increases. The high variance in this measurement makes it impossible to statistically show any difference when writing capsules to flash. The reason for this is that ContikiMAC receptions can occur no more often than the wake-up rate, which means that any additional processing following a packet transmission or reception has a negligible effect on the system.

5.3 Case Studies

To demonstrate the usefulness of Powertrace, we perform a set of case studies where we use Powertrace to study low-power wireless protocols from the literature. This is the first network-scale empirical evaluation of low-power wireless protocols where we are able to break

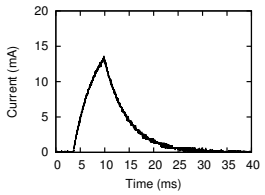


Figure 9: The X-MAC wake-up, which is approximately ten times as expansive as the ContikiMAC wake-up (Figure 3).

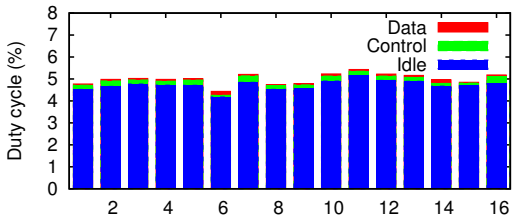


Figure 10: The power profile for the data collection network with X-MAC. Note that the y axis scale is different from Figure 11 and Figure 12.

down the network power consumption into components.

5.3.1 Sensornet Data Collection

We first look at the power profile of a sensor network data collection protocol. Data collection protocols occur frequently in sensor networking research and have been used in many sensor network deployments. Many data collection protocols exist. In this case study, we use Contiki Collect, the data collection protocol provided with the Contiki operating system. Nodes in a Contiki Collect network periodically broadcast beacons that announce their distance from a sink node. To send a packet towards a sink, nodes pick a parent that is closer to the sink than itself. Contiki Collect uses expected transmissions (ETX) as its path cost metric [3]. Contiki Collect uses ideas from the TinyOS CTP protocol [11], including adaptive beaconing and datapath route validation, both of which reduce the control traffic overhead.

We set up a 17 node Tmote Sky network in an office environment and let one node be the sink of the network. The other nodes send data towards the sink at a rate of one packet every other minute, a typical data rate used in sensor networks [11]. We use Powertrace to collect power profiling data from the system and transmit the energy readings as part of the data collection traffic. The sink, which is connected to a PC, sends the collected data over its USB port to the PC, which logs the data. We use two system configurations, one that uses the X-MAC duty cycling protocol [2] and one that uses ContikiMAC—both with a wake-up rate of 8 Hz. For both configurations, the resulting collection networks had an average depth of 1.5-1.8 hops and were able to deliver 93%-98% of their packets to the sink. In their power profiles, we distinguish between idle wake-ups, control traffic, and data traffic. Idle wake-ups are wake-ups that do not result in any packet reception.

The power profile of the X-MAC network is shown in Figure 10, which reports the power consumption as radio duty cycle. It is evident that the idle power consumption is the dominating factor of the system power consumption. This is due to the X-MAC wake-up mechanism (Figure 9) being comparatively expensive in comparison to transmissions and receptions. These results are consistent with power measurements obtained from a TinyOS CTP network [17], which uses an X-MAC-like duty cycling scheme by default.

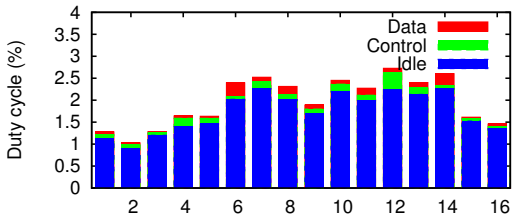


Figure 11: The power profile for data collection with ContikiMAC and the default clear-channel assessment wake-up mechanism. Note that the y axis scale is different from Figure 10.

The results for ContikiMAC are shown in Figure 11. Although ContikiMAC has a lower power consumption than X-MAC, we see that the idle power consumption dominates also in this case. The per-node breakdown shows that nodes in vicinity of the sink, which was placed in the middle of the network in the vicinity of

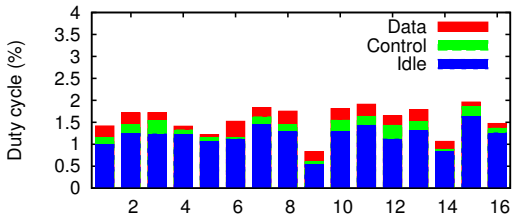


Figure 12: The power profile for data collection with ContikiMAC and a more conservative clear-channel assessment. Note that the y axis scale is different from Figure 10.

nodes 7 and 8, have a higher idle power consumption. This is because nodes in the center of the network are affected by transmissions from fringe nodes that are too far away for successful receptions, but close enough to trigger a false positive wake-up.

Seeing that false positive wake-ups had such an effect on power consumption, we configured ContikiMAC to have a more conservative wake-up trigger. By default, ContikiMAC uses the clear-channel assessment mechanism provided by the radio chip to trigger a wake-up, but this mechanism is intended for avoiding collisions when transmitting packets. It is therefore intended to be sensitive to transmissions that are out of the reception range. By setting a more conservative clear-channel assessment threshold for ContikiMAC's wake-ups, we were able to reduce the amount of false positives, as shown in Figure 12. With this configuration, we reduced the total power consumption by 24%, but also increased the average path length from 1.5 hops to 1.8 hops. This is also evident in the power profile, where the cost for data traffic increases as more packets must be forwarded.

5.3.2 Low-power IPv6 Routing with RPL

We next turn to applying Powertrace on RPL, a low-power IPv6 routing protocol that is on the verge of becoming an IETF standard [30]. RPL is designed to operate efficiently across a wide range of network types, but with a particular focus on low-power networks with potentially high loss links. RPL is a distance-vector protocol that builds a directed acyclic graph rooted at the network border router. RPL is optimized for the many-to-one communication pattern, where network nodes primarily send data towards the border router, but has provisions for any-to-any routing as well. As RPL is about to become an IETF standard, it is important to profile its power behavior.

We use the same testbed setup as above and use the sink node as the IPv6 network border router. We set up an IPv6/RPL network between the other nodes and use Powertrace to measure the power consumption of the network. We break the power consumption into Internet Control Message Protocol (ICMP) packets that constitute control traffic and User Datagram Protocol (UDP) packets that constitute application data. Since we established the role of idle wake-ups in the previous section, we disregard them here. We run a simple data collection application on top of UDP. The result is shown in Figure 13. We see that the control traffic power consumption goes down over time but that the cost for data traffic stays relatively constant.

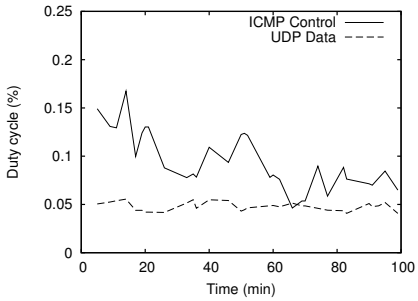


Figure 13: The power profile of the RPL routing protocol, broken down into control and data traffic.

6 Future Directions

Although power consumption has been a primary concern for low-power wireless research in the sensor networking community, surprisingly few protocols and mechanisms have been empirically studied in terms of their energy consumption. We hope that Powertrace and its mechanisms, which are intended to be simple yet powerful, will help to move the community towards more empirical and experimental energy evaluations.

6.1 Usefulness

We evaluate the usefulness of Powertrace in this paper, but the ultimate test of usefulness is if the system is useful for others. An earlier version of Powertrace has been included in the Contiki OS open source distribution and has been used for empirical evaluation of low-power wireless networking mechanisms. Lachenmann et al. [14] use it to demonstrate that their uDSSP implementation has a power overhead of 0.07 mW. Meier et al. [18] use Powertrace to perform run-time optimization of radio duty cycling wake-up schedules, which increased network lifetime by 50%. Dunkels et al. [4] measured the control traffic overhead with Powertrace and devised a method to reduce it by to reduce it by piggybacking control traffic transmission on each other. Similarly, Österlind et al. [33] used Powertrace to find that redundant wake-ups accounted for a large portion of the power consumption and designed a mechanism that reduces the number of wake-ups. Powertrace also makes it possible to distinguish power consumption of traffic classes—a feature whose usefulness was demonstrated in the development of the politecast communication primitive for low-power wireless [16], which used Powertrace to identify the cost of redundant control traffic broadcasts. Powertrace was also used extensively during the development and tuning of the ContikiMAC protocol [4]. With the results provided by Powertrace, it was evident

that the wake-up cost dominated energy consumption, and this observation prompted the development of ContikiMAC's energy-efficient wake-up mechanism.

6.2 Research Directions

Being able to track energy at the network scale opens up new research opportunities. First, it makes is possible to experimentally evaluate existing protocols and mechanisms to expose the system-level trade-offs that are difficult to reach without the ability to do system-level measurements. Second, with run-time power profiling, new energy-aware protocols and mechanisms that make energy-aware decisions can be developed. We have already seen examples of those, such as the ZeroCal mechanism by Meier et al. [18], which uses Powertrace to perform duty cycling run-time optimization. Moreover, the results obtained with Powertrace can lend empirical support to research debates. For example, Schmid et al. [26] argue that that low-power wireless networks need to move away from the mesh networking model in order to reduce the power consumption because idle wake-ups consume a disproportional amount of energy. The empirical results in this paper demonstrate that idle wake-ups constitute the largest part of the power consumption in low-power wireless systems.

6.3 Generalization

We have thus far only used Powertrace in the context of low-power wireless systems, but we believe that its concepts generalizes into other areas. In its current implementation, Powertrace relies on the accuracy of its power state tracking mechanism. In more complex devices, such as handheld devices or servers, it normally is not possible to directly track power states, however. But Powertrace's energy capsules do not directly depend on the power state tracking, but can work with any underlying power estimation model, of which there are many, both for servers [25] and handheld devices [20]. It would be interesting to generalize Powertrace to such devices as well, providing them with the network-level power profiling ability.

6.4 Technology Trends

The low-power wireless area still is in its infancy. With today's technology, radio duty cycling must be done in software, but next-generation low-power radios such as IEEE 802.15.4e are likely to have built-in hardware support for duty cycling. This makes the current Powertrace power state tracking impossible. But the design of the energy capsule mechanism extends to hardware-implementations of radio duty cycling. For example, the radio chip could hold an energy capsule for its wake-up, which could be periodically read by the radio device driver. Likewise, packet transmissions and packet receptions could also return an energy capsule to the device driver. With this functionality, network-level power profiling would be possible even with hardware implementations of radio duty cycling.

7 Related Work

Power management, power measurement, and power profiling are active research areas in many fields [1, 9, 19, 25]. In particular, power state tracking has a long history. Zeng et al. [32] used the power state tracking mechanism to estimate the energy consumption but for general-purpose computers. Their problem is, however, more complex than ours, due to the higher complexity of general-purpose peripherals such as hard drivers. Similarly, Pathak et al. [20] use system call tracing to estimate the power profile of systems using Android or Windows Mobile. Their mechanism is similar to the power state tracking used by Powertrace, but works at the system call level rather than the device driver layer because the device driver layer cannot be accessed in modern smartphones.

To the best of our knowledge, Powertrace is the first system to profile the power behavior of low-power wireless protocols and mechanisms at the network-level. The work closest to Powertrace is Quanto by Fonseca et al. [10]. Quanto uses a hardware-based energy measurement device [8] to track the energy consumption of the system and stores this for later off-line analysis. The authors show that the energy consumption of individual activities can be extracted through the off-line analysis. Powertrace is different from Quanto in many aspects. First, unlike Quanto, Powertrace is a run-time mechanism that continuously provides power profiling data, which allows applications and protocols to make energy-aware decisions. Moreover, unlike Quanto, Powertrace breaks down energy into individual network activities, which enabled network-level profiling.

A number of energy measurement or energy estimation mechanisms have been developed in the sensor network community. Ritter et al [24] estimate the energy consumption of a sensor mote by using a 1 F capacitors, so-called GoldCaps, to power the motes. Since the energy storage of the capacitor is constant, it is possible to estimate the energy consumption by measuring the(relatively short lifetime of the mote. Jiang et al [13] developed hardware add-on board that measured the current consumption for a mote. This required a significant attention to detail as the board needs to measure both short-lived current bursts and long-lived trends. Dutta

et al [8] present a hardware-based technique where a switching capacitors were used to estimate the power consumption of the mote. When one capacitor ran out of energy, it triggered a processor interrupt. By counting the interrupts, it was possible to estimate the energy consumed by the device. Unlike these approaches, Powertrace does not need any additional hardware or custom designs but only use software which makes adoption and deployment easier.

Software-based estimation techniques have also been developed, both for use in simulation and in deployed systems. Shnayder et al. [27] provide energy models to the TOSSIM sensor network simulator that allows energy to be estimated through simulation. Network-level simulators such as TOSSIM have the drawback of not capturing low-level behavior and therefore cannot accurately simulate radio duty cycling mechanisms. Unlike simulation-based approaches, Powertrace allows power profiling in experimental setups and in deployed systems, which allows the system developer to see the exact conditions in which the network is operating.

Hurni et al. [12] and Dunkels et al. [5] use a power state tracking technique that estimate the total energy consumption of a sensor mote by measuring the time during which components are switched on. Lorincz et al. [15] use a similar technique for energy and resource management in the Pixie operating system. Powertrace incorporates these techniques but adds network-level profiling that makes it possible to break down power consumption into network-level activities.

8 Conclusions

Energy consumption is of primary importance in low-power wireless networks, but to optimize energy we have to be able to measure it. We present Powertrace, a system for network-level power profiling of low-power wireless networks. Powertrace enables not only run-time energy estimation of low-power wireless systems but also power profiling that show the per-activity power cost. This provides system developers with the means to optimize their systems for power consumption, as they are able both to identify the worst power consumer and to evaluate any savings their optimizations do. Powertrace uses power state tracking to estimate the node-level power consumption and a structure called energy capsules to attribute energy consumption to activities.

Using Powertrace, we demonstrate that the number of radio transmissions is a poor estimator for the energy consumption of a system and that low-power wireless systems spend a large portion of their power on idle listening. Based on these results, we tune the ContikiMAC duty cycling mechanism and reduce system energy consumption by 24%, but idle listening still dominates.

Powertrace is already being used in the community and we hope that it will further help towards more empirical studies of energy and power consumption in low-power wireless systems.

References

[1] G. Banga, P. Druschel, and J. Mogul. Resource containers: a new facility for resource management in server systems. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 45–58, New Orleans, Louisiana, United States, 1999.

[2] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, pages 307–320, Boulder, Colorado, USA, 2006.

[3] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the International Conference on Mobile Computing and Networking (ACM MobiCom)*, pages 134–146, San Diego, CA, USA, 2003. ACM.

[4] A. Dunkels, L. Mottola, N. Tsiftes, F. Österlind, J. Eriksson, and N. Finne. The announcement layer: Beacon coordination for the sensornet stack. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2011.

[5] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the IEEE Workshop on Embedded Networked Sensor Systems (IEEE Emnets)*, Cork, Ireland, June 2007.

[6] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Sydney, Australia, November 2007.

[7] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making Sensor Networks IPv6 Ready. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Raleigh, North Carolina, USA, November 2008.

[8] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, pages 283–294, 2008.

[9] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, pages 48–63, Charleston, South Carolina, United States, 1999.

[10] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 323–338, 2008.

[11] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Berkeley, CA, USA, 2009.

[12] P. Hurni, T. Braun, B. Nyffenegger, and A. Hergenroeder. On the accuracy of software-based energy estimation techniques. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, Bonn, Germany, February 2011.

[13] X. Jiang, P. Dutta, D. Culler, and I. Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, Cambridge, Massachusetts, USA, 2007.

[14] A. Lachenmann, U. Müller, R. Sugar, L. Latour, M. Neugebauer, and A. Gefflaut. Programming sensor networks with state-centric services. In *Proceedings of Distributed Computing in Sensor Systems (DCOSS)*, 2010.

[15] K. Lorincz, B. Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource aware programming in the pixie os. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, pages 211–224, Raleigh, NC, USA, 2008.

[16] M. Lunden and A. Dunkels. The politecast communcation primtive for low-power wireless. *The ACM Computer Communications Review*, April 2011.

[17] M. Martins, R. Fonseca, T. Schmid, and P. Dutta. Poster abstract: Network-wide energy profiling of ctp. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, pages 439–440, Zurich, Switzerland, 2010.

[18] A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele. Zerocal: Automatic mac protocol calibration. In *Proceedings of Distributed Computing in Sensor Systems (DCOSS)*, pages 31–44, 2010.

[19] R. Nathuji and K. Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, pages 265–278, 2007.

[20] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y. Wang. In *Proceedings of the ACM European Conference on Computer Systems (ACM EuroSys)*, 2011.

[21] K. Pister and L. Doherty. TSMP: Time Synchronized Mesh Protocol. In *Proceedings of the IASTED International Symposium on Distributed Sensor Networks (DSN08)*, Orlando, Florida, USA, November 2008.

[22] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, pages 95–107, Baltimore, MD, USA, 2004. ACM Press.

[23] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, Los Angeles, CA, USA, April 2005.

[24] H. Ritter, J. Schiller, T. Voigt, A. Dunkels, and J. Alonso. Experimental Evaluation of Lifetime Bounds for Wireless Sensor Networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, Istanbul, Turkey, January 2005.

[25] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower)*, San Diego, CA, December 2008.

[26] T. Schmid, R. Shea, M. B. Srivastava, and P. Dutta. Disentangling wireless sensing from mesh networking. In *Proceedings of the Workshop on Hot Topics in Embedded Networked Sensor Systems (HotEmnets)*, Killarney, Ireland, 2010.

[27] V. Shnayder, M. Hempstead, B. Chen, and M. Welsh. Powertossim: Efficient power simulation for tinyos applications. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2004.

[28] Y. Sun, O. Gurewitz, and D. Johnson. RI-MAC: A Receiver-Initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Raleigh, NC, USA, 2008.

[29] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling Large-Scale Storage in Sensor Networks with the Coffee File System. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, San Francisco, USA, April 2009.

[30] J.P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[31] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, New York, NY, USA, June 2002.

[32] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. Ecosystem: managing energy as a first class operating system resource. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, California, 2002.

[33] F. Österlind, N. Wirström, N. Tsiftes, N. Finne, T. Voigt, and A. Dunkels. StrawMAN: Making Sudden Traffic Surges Graceful in Low-Power Wireless Networks. In *Proceedings of the 2010 ACM HotEMNETS Workshop on Hot Topics in Embedded Networked Sensosr*, Killarney, Ireland, June 2010.