FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems

Roman Lim Federico Ferrari Marco Zimmerling Christoph Walser
Philipp Sommer[*] Jan Beutel
Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
[*]Autonomous Systems Lab, CSIRO ICT Centre, Australia
lim@tik.ee.ethz.ch http://www.flocklab.ethz.ch/

ABSTRACT

Testbeds are indispensable for debugging and evaluating wireless embedded systems. While existing testbeds provide ample opportunities for realistic, large-scale experiments, they are limited in their ability to closely observe and control the distributed operation of resource-constrained nodes—access to the nodes is restricted to the serial port. This paper presents FlockLab, a testbed that overcomes this limitation by allowing multiple services to run *simultaneously* and *synchronously* against all nodes under test in addition to the traditional serial port service: tracing of GPIO pins to record logical events occurring on a node, actuation of GPIO pins to trigger actions on a node, and high-resolution power profiling. FlockLab's accurate timing information in the low microsecond range enables logical events to be correlated with power samples, thus providing a previously unattained level of visibility into the distributed behavior of wireless embedded systems. In this paper, we describe FlockLab's design, benchmark its performance, and demonstrate its capabilities through several real-world test cases.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*measurement techniques*; D.2.5 [**Software Engineering**]: Testing and Debugging—*distributed debugging, tracing*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based System—*real-time and embedded systems*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Testbed, GPIO tracing, GPIO actuation, power profiling, adjustable power supply, wireless sensor network

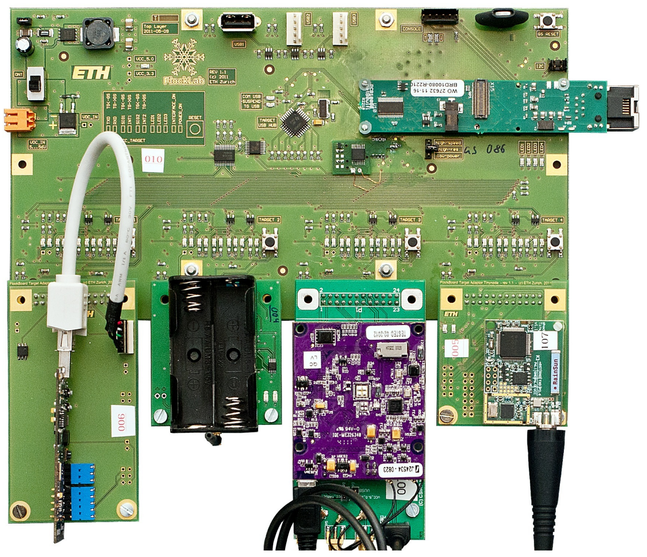


Figure 1: FlockLab observer with Tmote Sky, IRIS, Opal, and Tinynode connected via interface boards.

1. INTRODUCTION

Testbeds play a key role in developing real-world wireless embedded systems by providing the facilities to debug and evaluate protocols and applications in a controlled, yet realistic distributed environment. A review of the spectrum of existing testbeds yields a long list: relocatable testbeds to study applications in the intended target environment [29], testbeds with robots for controlled mobility experiments [18], testbeds performing distributed power measurements [15], homogeneous testbeds with hundreds of devices [9], and emulation platforms [12] and heterogeneous testbed federations [4] to assess large-scale services on thousands of nodes.

Despite this broad spectrum, the current practice of testbed-assisted development revolves around LED and `printf` debugging: developers use the nodes' on-board LEDs to observe conditions in the running program and `printf` statements to log diagnostic messages, performance counters, or program state over the serial port. However, it is well known that `printf`s alter the timing behavior and are therefore unsuitable for analyzing timing sensitive code such as radio drivers and MAC protocols. Perhaps one reason for the unchallenged popularity of these techniques is their ease of use [31]. Another reason is that current testbeds allow access to the devices under test only through the serial port. As a result, developers are left with no other option than to use `printf`s, a means suitable for a number of long-term profiling tasks but cumbersome, highly intrusive, and unsuitable for detailed investigation of interactions among multiple devices, especially real-time issues.

The current solution for debugging low-level software and hardware interactions is a logic analyzer and a mixed-signal oscilloscope allowing to capture and trigger events of interest (*e.g.*, changes in program state or packet transmissions) at high timing resolution. Different from `printf`s, setting digital GPIO pins on a node introduces a known delay of just a few clock cycles, which makes GPIO tracing a powerful tool for debugging timing sensitive code. The required equipment, however, limits the setup to a few nodes on a table, bearing little resemblance to a real multi-hop setting.

The main contribution of this paper is FlockLab, a testbed with services providing a previously unattained level of visibility into wireless embedded systems. FlockLab's novelty stems from the combined capability of tracing and actuating logical state changes at high level of detail, accurate timing information in the low microsecond range, and the possibility to profile and control power over the whole testbed. By coupling a powerful, stateful *observer* platform directly with every device under test, the *target*, FlockLab overcomes the bottleneck at the single sink of centralized data collection systems. FlockLab leverages distributed target-observer pairs with deep local storage that are capable of capturing event and power traces of all targets locally, simultaneously, synchronously, and at high rates without sacrificing on timing accuracy or incurring data rate limitations of traditional backchannel-based testbeds [14, 37].

As such, FlockLab combines the capability of a logic analyzer, power analyzer, serial data logger, and programmable power supply with network synchronization and deep local storage adjacent to each target—distributed across the entire testbed. FlockLab also supports multiple target platforms, allowing for comparative analysis of applications and protocols on the same physical topology. It performs distributed power measurements at higher rate, resolution, and synchronization accuracy than prior testbeds. Users may apply power profiling and GPIO tracing against all targets to correlate power samples and logical events, or dynamically adjust the target supply voltage to emulate battery depletion effects. Sec. 2 details the services available in FlockLab.

Sec. 3 presents the design of FlockLab to meet the challenges that arise when providing these services. Based on our current FlockLab deployment at ETH Zurich, which consists of 30 observers in a mixed indoor/outdoor setting that host Opal, IRIS, Tinynode 184, and Tmote Sky targets as shown in Fig. 1, we benchmark FlockLab's performance in Sec. 4. We find, for instance, that FlockLab can capture GPIO events reliably up to a rate of 10 kHz; it can timestamp distributed events and power samples with an average pairwise error below 40 μs; and it measures power draw with an average error smaller than 0.4 % over six orders of magnitude, while providing a highly stable and programmable supply voltage. We further demonstrate in Sec. 5 the utility of FlockLab through various real-world test cases, including an experiment in which we take a detailed look into packet propagation and power draw during a Glossy network flood [10]. Gaining similar multi-modal insights at this level of detail would hardly be feasible with any prior testbed. We review related work in Sec. 6 and conclude in Sec. 7.

Platform	Microcontroller	Speed	Cycles	Time
Tmote Sky	MSP430 F1611	4 MHz	5	1,250 ns
Tinynode 184	MSP430 F2417	12 MHz	5	417 ns
Opal	ARM Cortex-M3	96 MHz	5	52 ns
IRIS	ATMega1281	8 MHz	2	250 ns

Table 1: Clock cycles and time needed to set a GPIO pin on the current FlockLab targets. *The known, minimal delay of GPIO tracing allows for virtually non-intrusive debugging of timing sensitive code.*

2. FLOCKLAB SERVICES

FlockLab delivers new insights into wireless embedded systems by providing the following key services.

GPIO tracing. An observer can trace level changes of five target GPIO pins at a rate of up to 10 kHz. Setting a GPIO pin takes only 2–5 clock cycles on current target platforms, as listed in Table 1. Thus, using simple code instrumentation, this service allows for virtually non-intrusive tracing of events of interest; for example, a trace of packet exchanges may help debug a MAC or routing protocol. Like a mixed-signal oscilloscope that can trigger on digital signals and capture on analog signals, it is also possible to couple GPIO tracing with GPIO actuation and power profiling using a callback mechanism: upon detecting a defined pin edge, an observer can set another GPIO pin or start measuring power.

GPIO actuation. An observer can set, clear, and toggle up to three target GPIO pins, one of which is the target's reset pin, either periodically or at predefined times. This is useful, for example, to create controlled experiments by triggering some action on all targets at the same time, such as starting or stopping the nodes, turning on the radio, transmitting a packet, or freezing and logging a state variable.

Power profiling. An observer can sample the current draw of the target at a maximum frequency of 28 kHz when operating the ADC in high-resolution mode and up to 56 kHz when operating it in high-speed mode. FlockLab defaults to the high-resolution mode since it provides a higher SNR than the high-speed mode, as further described in Sec. 3.4. Users specify time windows during which this service should be running. The resulting power traces can aid in developing energy-efficient applications and have also been used for conformance testing [38] and failure diagnosis [20].

Adjustable supply voltage. An observer can dynamically adjust the target supply voltage between 1.8 V and 3.3 V in steps of 100 mV. To introduce repeatable voltage changes, users can select from a range of predefined charge/discharge curves or define their own voltage-time profiles. This can be used, for example, to study discharge-dependent behavior.

Serial I/O. Finally, an observer can read or inject data over the target's serial port, which is a standard service available on almost any testbed. FlockLab currently supports ASCII data, TOS messages, and SLIP datagrams, making it compliant with the serial communication available in state-of-the-art operating systems like TinyOS and Contiki.

FlockLab allows a user to run any combination of the above services *simultaneously* and *synchronously* on any subset of observers. FlockLab *accurately timestamps* data acquired during a test *across all services and observers*, thus providing previously unattained insights into local and distributed system behavior both in detail and at scale. To the best of our knowledge, this makes FlockLab unique in the spectrum of testbeds for wireless embedded systems.

3. FLOCKLAB ARCHITECTURE

Providing the above services presents several challenges to the design of FlockLab. This section highlights these challenges and describes FlockLab's hardware and software architecture designed to solve them.

3.1 Challenges

- **Minimum disruption:** FlockLab must not perturb the behavior of the system under test beyond the minimum necessary to obtain the desired measurements.
- **High accuracy and resolution:** FlockLab needs to provide highly accurate power samples over a dynamic range that spans six orders of magnitude in current draw, from sleep currents of just 2 μA on a Tinynode up to active currents on the order of 100 mA. The resolution of power measurements and event traces must approach or exceed 10 kHz to capture ephemeral radio events, such as clear channel assessments, which last only 100–200 μs.
- **Time synchronization:** FlockLab must tightly time-synchronize the observers against a stable global clock, so as to precisely correlate events and power samples of one observer as well as across multiple observers. With sampling rates of at least 10 kHz, events and power samples must thus be timestamped with 50 μs accuracy or better.
- **Large data volume:** FlockLab needs to cope with large data volumes that arise particularly during high-resolution power profiling. Samples should not be lost and be quickly processed to achieve high testbed utilization.
- **Platform support:** FlockLab's hardware and software architecture must be designed in such a way that new platforms can be supported with little effort and cost.

3.2 Overview

FlockLab consists of several distributed target-observer pairs and a set of servers. *Observers* are powerful platforms that can host up to four devices under test, the *targets*, connected through relatively simple *interface boards*. Observers implement all services available in FlockLab in hardware or software. They connect to several backend servers responsible for coordinating their distributed and synchronized operation, for processing and storing collected results, and interacting with FlockLab users.

3.3 Observer Hardware

Observers are based on a custom-designed PCB assembly as depicted in Fig. 2. Its heart forms a Gumstix XL6P COM embedded computer, which is driven by a 624 MHz Marvell XScale PXA270 microprocessor and equipped with 128 MB SDRAM and 32 MB flash memory. We add an 8 GB SD card to cache test configurations, program images, and test results. Observers connect to FlockLab servers preferably through the Ethernet expansion of the Gumstix; USB Wi-Fi adapters can be used if Ethernet proximity is lacking.

A switching regulator converts a 5–56 V DC input voltage to the 5 V on-board voltage required by the Gumstix. A linear regulator with low output noise further down-converts to the 3.3 V on-board voltage required by other components. The ADS1271, a 24-bit delta-sigma ADC, is used for power profiling as detailed in Sec. 3.4. Additionally, there are three USB connectors and a humidity/temperature sensor. In our current deployment, described in Sec. 3.8, we use the readings of the latter to control a USB-powered fan on four outdoor observers to prevent humidity and overheating issues.

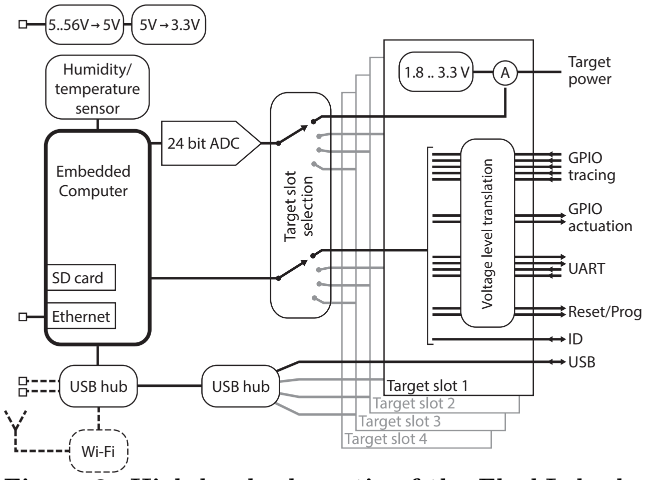


Figure 2: High-level schematic of the FlockLab observer hardware.

An observer provides four pin header connectors to attach targets through interface boards, and replicates the following main components for each connector: an LM3370 switching regulator to adjust the target supply voltage in the range of 1.8–3.3 V with 100 mV resolution; a MAX9923H current-sense amplifier for power measurements; five incoming and two outgoing GPIO lines to trace and actuate GPIO pins of the target; UART lines to read and inject data over the target's serial port; lines to reset and program the target; an ID line to identify the interface board as further discussed in Sec. 3.6; and a USB port for USB-enabled targets and interface boards. Two 8-bit signal translators match the variable voltage of the target with the 3.3 V on-board voltage of the observer. Finally, an observer provides nine LEDs controlled by the GPIO and UART data lines for visual inspection.

Because of the limited number of GPIO pins on the Gumstix, we need to multiplex the available signal lines between the four targets. We achieve this by letting the Gumstix enable the two voltage level translators and the current-sense amplifier of the desired target and disable them for all other slots. The Gumstix can thus control one target at a time.

The cost of the complete observer PCB assembly amounts to a rough total of 1000 USD including manufacturing costs.

3.4 Measuring Power

To measure power, we put a small shunt resistor between the switching regulator and the target. The voltage across the resistor is proportional to the current draw of the target. We use a MAX9923H high-side current-sense amplifier to amplify the sense voltage by a gain of 100. The MAX9923H has low offset voltage and high gain accuracy, providing precise measurements also at low sense voltages. The output of the amplifier is then fed into an ADS1271 ADC, whose samples are fetched by the Gumstix over an SPI bus. Conversion into current is done in the FlockLab backend based on the shunt resistance, the gain of the amplifier, the reference voltage of the ADC, and the supply voltage of the target. We assess the stability of the latter in Sec. 4.2, showing that the voltage drops only by a few mV at typical current draws.

The choice of the shunt resistor presents a tradeoff. Using a small resistor reduces the influence on the measurements, whereas using a large resistor gives a better SNR. Another important factor is the wide dynamic range of current draw. For instance, a Tinynode draws only 2 μA in sleep mode,

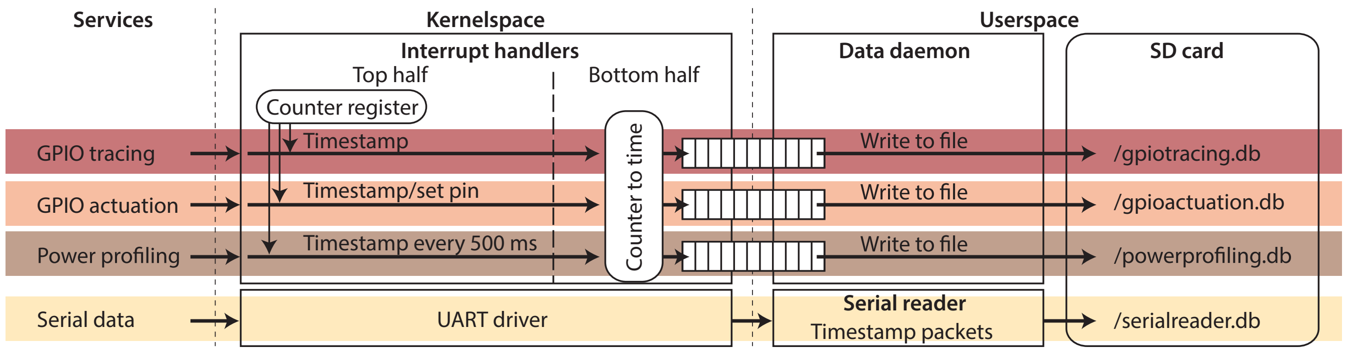


Figure 3: Processing of GPIO events, power samples, and serial data on an observer. *Timestamping occurs in the bottom half of an interrupt handler using a tick count taken in the top half, which increases precision and throughput.*

whereas an Opal draws as much as 49 mA when both radios are turned on. To prepare FlockLab for even higher current draws of future platforms, we want to support up to 160 mA. Based on these considerations, we decided to use a relatively small 150 mΩ shunt resistor, which still enables the high-gain amplifier to accurately measure low signal levels.

The ADC has a resolution of 24 bits, which gives a theoretical resolution of 10 nA in current draw based on the specifications of shunt resistor, amplifier, and ADC. The ADC features two modes of operation that are interesting for FlockLab, selectable by a jumper: high-speed and high-resolution. Using a 14.3 MHz clock source, the ADC samples at 56 kHz in high-speed mode and at 28 kHz in high-resolution mode. FlockLab defaults to the latter as it has a higher SNR of 109 dB, while still providing a sufficiently high sampling rate to capture short-lived radio events.

3.5 Observer Software

Observers run OpenEmbedded Linux and use Chrony as an NTP [25] client to synchronize every 1–2 min with the FlockLab NTP server (see Sec. 3.7). This provides the basis for accurately timestamping GPIO events, power samples, and serial messages. Observers cache the timestamped data locally before uploading them to the FlockLab database server, and have a collection of Python scripts that are used by the FlockLab test management server to trigger scheduled actions such as starting and stopping a test, reprogramming a target, and setting the target supply voltage.

Data acquisition and timestamping. To gain access to hardware connected to the Gumstix—in our case the GPIO lines and the SPI bus which interfaces with the ADC—we implement data acquisition and timestamping as kernel modules. Kernel processes run with highest priority, which helps reduce processing delays and thus increase throughput.

As shown in Fig. 3, data acquisition for GPIO tracing, GPIO actuation, and power profiling starts in interrupt handlers. Triggered by a hardware or timer interrupt, the top half of a handler serves the interrupt, reads a register to obtain the current tick count, and requests that the bottom half of the handler be executed at some future time. The bottom half uses then the tick count to compute a precise timestamp. This approach increases throughput and timestamp precision, because it minimizes the execution time of the top halves, enabling interrupt requests to be served at high rate and low jitter.

As for GPIO tracing and GPIO actuation, an observer timestamps single events. This is however different for power profiling. To reduce system load and memory consumption, we generate a timestamp only every 500 ms. Using the constant sampling rate of the ADC, the FlockLab backend later interpolates the timestamps of single power samples.

Timestamping of serial messages is less critical since these are already affected by non-deterministic UART transfer delays [5] and therefore should not be used to log data that require highly accurate timestamps. For this reason, we process and timestamp serial messages in userspace.

Data caching. When using FlockLab's power profiling service, the observers have to deal with enormous amounts of data, so efficient data handling is key. Motivated by this, we use a custom-built binary log file mechanism rather than a full-blown database system. As shown in Fig. 3, kernel FIFO queues are used for transferring acquired data from kernel to userspace, where a daemon receives the data and writes them into separate files on the SD card. Upon request from the FlockLab test management server, an observer uploads accumulated data to the database server.

3.6 Supporting Diverse Target Platforms

FlockLab possesses the flexibility to support diverse target platforms with little effort in terms of hardware and software. Every observer can host four targets of possibly different form factors, connectors, features, and tools required for installing program images. Key to this flexibility is the use of interface boards: simple PCB assemblies that interconnect the components on an observer (see Fig. 2 and Sec. 3.3) with the corresponding components on the target.

Every platform requires its own custom-designed interface board, since there is no standardized connector or pin layout for wireless embedded devices. An interface board may also need to make provisions for different logic levels.

Additionally, FlockLab imposes a few constraints on the design of an interface board. First, it needs to fit certain maximum dimensions and have an appropriate header connector. Second, the components on an interface board must work with one of the available power supplies: 3.3 V, 5.0 V, or the 1.8–3.3 V DC adjustable voltage. Third, an interface board must feature a serial ID chip that is compliant with the widely used DS2401, which is needed to automatically identify the mapping of target slots to interface boards.

Besides interface boards for Tmote Sky, Tinynode, and Opal designed by us, external collaborators from IBM designed an interface board for IRIS, which also supports Mica2 and MicaZ due to pin-compatibility. We leverage these interface boards in our current FlockLab deployment at ETH Zurich to attach four different platforms to each observer, as shown in Fig. 1.

On the software side, it is sufficient to port the reprogramming tool to the Gumstix to support a new platform. As for serial I/O, FlockLab observers already support ASCII data, TOS messages, and SLIP datagrams. The target software requires no special measures, since embedded operating systems already provide functions for serial I/O and accessing GPIO pins, and power is measured by the observer.

3.7 Backend Infrastructure

Observers connect via Ethernet or Wi-Fi to a set of servers that provide all what it takes to make FlockLab a testbed.

Time synchronization server. FlockLab operates its own NTP server that synchronizes against another server on campus and a high-accuracy pulse per second (PPS) signal output by a GPS receiver, which provides a precise time reference. All observers synchronize against this NTP server.

Web server. Users interact with this server to schedule and configure their tests. Every user is allowed to reserve FlockLab for a certain maximum duration and number of tests at a time. A test configuration consists of a single XML file to setup the services and one or more compiled binaries. A user can run a test as soon as possible or during some specified time slot, abort a running test, and fetch the results of successfully completed tests. If requested, a user receives email notifications about started and completed tests.

Test management server. This server is responsible for operations related to starting, running, and finalizing scheduled tests. If a test is about to start, it parses the configuration, prepares flashable images from the supplied binaries, and dispatches these data to the observers. While a test is running, it periodically queries the observers for results and stores them in a database. When a test has finished, it processes the raw data (*e.g.*, interpolate timestamps, convert to current) and stores them in a compressed archive.

Database server. This server hosts a MySQL database, which stores test configurations, test results, and user-specific data such as quotas and login information.

Monitoring server. Finally, we use Zabbix and Cacti to constantly monitor all server instances, networking components, and observers. In case of an abnormal situation, FlockLab admins are automatically informed via e-mail and/or SMS to ensure maximum uptime of the testbed.

3.8 Deployment

The current FlockLab deployment at ETH Zurich consists of 30 observers, each hosting a Tmote Sky, IRIS, Opal, and Tinynode 184. As illustrated in Fig. 4, 26 observers are deployed indoors across one floor in an office building, distributed in offices, hallways, and storerooms. Four observers are deployed outside, sitting on the roof of an adjacent building a few meters beneath the floor with the indoor observers.

All indoor observers connect via Ethernet, and have a light acrylic glass cover to protect against dust. To help the accuracy of NTP by reducing communication latency and jitter, they are all in the same LAN segment. The outdoor observers use Wi-Fi due to lack of Ethernet on the roof, and are housed in robust polycarbonate boxes with controlled ventilation to avert humidity and overheating problems.

During testbed idle times, the test management server runs an RSSI scanner on all target platforms, determining the noise level on all channels and frequency bands, and a test where targets broadcast 500 30-byte packets each and then report the number of packets they received from any

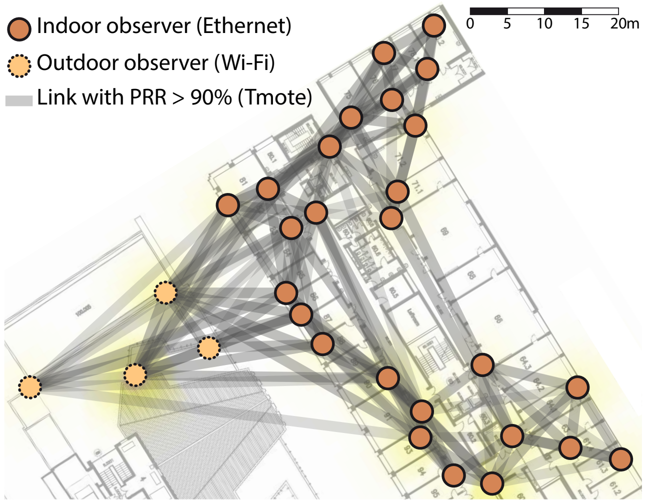


Figure 4: Layout of FlockLab deployment including information about link qualities and noise.

other target, which gives an estimate of the link qualities in the testbed. This information is stored in the database and displayed on the FlockLab website as an overlay on the deployment map as shown in Fig. 4, giving users an idea as to what extent their tests may be affected by external interference (*e.g.*, from co-located Wi-Fi) or limited connectivity.

4. BENCHMARKING FLOCKLAB

Using our current deployment, we benchmark in this section the accuracy and the limits of key FlockLab services. We start by evaluating FlockLab's timing accuracy, which is fundamental to exploit the full potential of the GPIO and power profiling services, check the stability of the power supply and the accuracy of the power measurements, and finally determine the maximum rate for capturing GPIO events.

4.1 Timing Accuracy

4.1.1 GPIO Tracing and Actuation

Setup. We randomly select 7 Ethernet-connected observers, and put one Wi-Fi-connected observer indoors on a table. We evaluate GPIO tracing and actuation in two separate 1 h tests. In the first test, we use a signal cable to connect a GPS clock to one GPIO pin of each observer. The GPS clock generates a PPS signal, and the observers timestamp the corresponding GPIO events. In the second test, we connect one GPIO pin of each observer to a Tektronix MSO4054B mixed-signal oscilloscope. All observers simultaneously toggle the pins every second, and the oscilloscope measures the actual timing of these events.

Pairwise timing error. We first measure the pairwise timing error between simultaneous GPIO events at different observers. This evaluates the alignment of GPIO traces collected by different observers and, for GPIO actuation, the precision with which simultaneous actions can be triggered.

Table 2 shows that the average pairwise error is smaller than 40 μs when using the 7 Ethernet-connected observers. If we add the Wi-Fi-connected observer, the error increases significantly due to higher and more variable delays in the exchange of NTP packets over Wi-Fi. The error is similar for GPIO tracing and actuation, as an observer executes similar operations when timestamping an event or setting a pin.

GPIO service	7 Ethernet			7 Ethernet, 1 Wi-Fi		
	avg	85th	max	avg	85th	max
Tracing	36 μs	69 μs	255 μs	166 μs	527 μs	1,161 μs
Actuation	30 μs	54 μs	394 μs	138 μs	334 μs	1,170 μs

Table 2: Pairwise timing error of GPIO services. *The average error is smaller than 40 μs with Ethernet observers.*

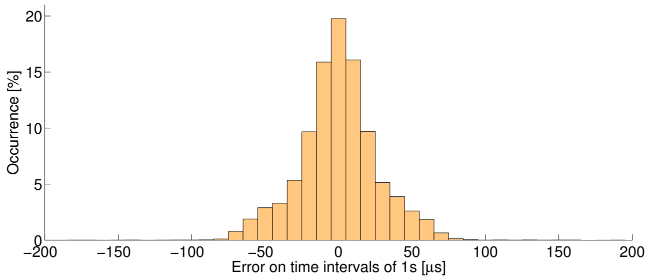


Figure 5: Distribution of the error on time intervals between GPIO events. *The average error is -0.011 μs.*

These results show that FlockLab allows users to align GPIO traces and to set GPIO pins with an error as small as a few tens of microseconds when using the indoor observers. This high accuracy is more than sufficient to trace packet transmissions among targets, as we demonstrate in Sec. 5.5. The results also show that because of the higher NTP synchronization error over Wi-Fi, the outdoor observers are less suited for tests that require sub-millisecond timing accuracy.

Error on time intervals. Using data from the previous experiment, we also assess the error on time intervals. We compute for each observer the difference between timestamps of consecutive GPIO events and compare it to the PPS signal. In this way, we evaluate the precision with which an observer measures the interval between GPIO events.

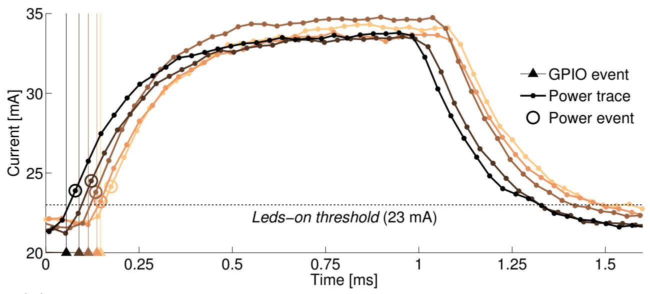
Fig. 5 shows the distribution of the error on time intervals, as measured by all 8 observers used in the experiment. We see that it approaches a normal distribution with a sample mean of -0.011 μs and a sample standard deviation of 27 μs. The average error is small because each timestamp is similarly affected by variable interrupt delays on an observer. We show in Sec. 5 that this precision allows to profile the radio activity or to measure the clock drift of a target.

4.1.2 Power Profiling

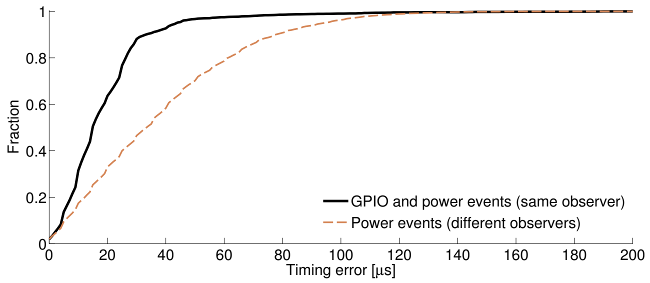
Setup. To evaluate the timing accuracy of the power profiling service, we run a 2 min test on 6 Tmote Sky targets attached to Ethernet-connected observers. One *transmitter* generates a 30-byte packet every 62.5 ms. The other 5 *receivers*, located in the transmission range of the transmitter, have their radios turned on and receive the packets. The corresponding observers enable GPIO tracing and power profiling, measuring current[1] draws at 28 kHz.

When a start frame delimiter (SFD) interrupt signals the start of a packet reception, a receiver toggles a GPIO pin and turns on its three on-board LEDs. As shown in Fig. 6(a), these operations generate a GPIO event and an increase in current draw from 22 mA to 34 mA. When the next SFD interrupt signals the end of a reception, each receiver turns off its LEDs and the current decreases accordingly. We consider these events as occurring at the same time, as we measure with an oscilloscope that the lag due to different time of flight and interrupt delays is smaller than 1 μs. To compare power timestamps, we define that a *power event* occurs when the current rises above a *leds-on threshold* of 23 mA.

[1]We use power and current interchangeably in Secs. 4 and 5, because FlockLab supplies a known, stable voltage (see Sec. 4.2) and thus power is directly proportional to current.



(a) Simultaneous GPIO and power events on 5 observers.



(b) Cumulative distribution of timing errors.

Figure 6: Timing errors of power profiling. *Observers timestamp simultaneous power events with an average pairwise timing error of 39 μs.*

Timing error between GPIO and power events. We measure the timing error between GPIO and power events on the *same* observer by computing the interval between the GPIO and the respective power timestamp (*i.e.*, between a vertical line and the corresponding circle in Fig. 6(a)).

The solid line in Fig. 6(b) shows the cumulative distribution of the timing error, which is 20 μs on average and smaller than 29 μs in 85 % of the cases. We see that the average error is close to half the power sampling period (17 μs): power profiling has a lower resolution than GPIO tracing and most of the timing error comes from the random delay between a GPIO event and the following power sample.

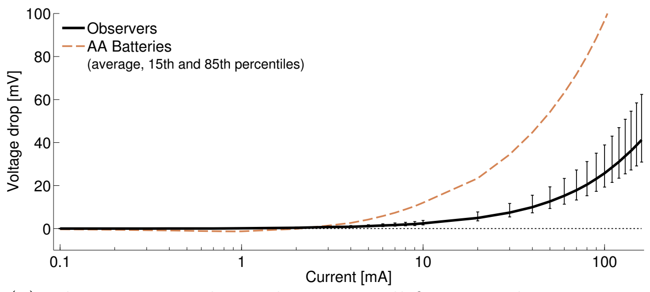
Pairwise timing error. We now look at the pairwise timing error between simultaneous power events on *different* observers (*i.e.*, between two circles in Fig. 6(a)).

The dashed line in Fig. 6(b) shows the cumulative distribution of this pairwise timing error, averaging around 39 μs with an 85th percentile below 68 μs. The error is comparable to that of simultaneous GPIO events in Sec. 4.1.1, since the sources of time inaccuracies are similar. Fig. 6(a) and the test case in Sec. 5.5 confirm that the precise alignment of power traces in FlockLab allows to match the power draw of a target to packets exchanged with other targets.

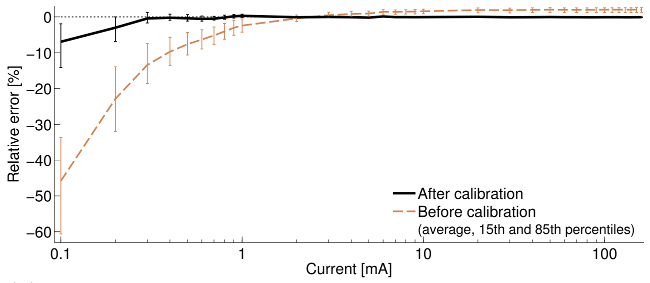
4.2 Power Accuracy

We use ad-hoc experiments to check whether an observer accurately measures the current draw of the target with only minimal impact on the stability of the target supply voltage.

Setup. We connect the target slot of an observer to a high-precision Agilent N6705A power analyzer, which acts as a target that draws predefined currents. The current draws



(a) The average voltage drop is small for typical target currents and less than 42 mV even when a target draws 160 mA.



(b) Calibration using linear regression reduces the average relative error on current draw to -0.39 %.

Figure 7: Stability of the power supply and accuracy of the power measurements in FlockLab.

cover the full dynamic range and proceed in a step-wise fashion as follows: from 0 mA to 1 mA in steps of 0.1 mA, from 1 mA to 10 mA in steps of 1 mA, and from 10 mA to 160 mA in steps of 10 mA; each of the 35 steps lasts 3 s. During the experiment, the observer supplies a nominal voltage of 3.3 V and records the current drawn by the power analyzer with a resolution of 14 kHz, while the power analyzer records the voltage supplied by the observer with a resolution of 24 kHz. We repeat the experiment 32 times, using the four target slots of eight randomly chosen observers.

Stability of power supply. We first look at the stability of the supply voltage. To this end, we measure the voltage drop as the difference between the zero-load voltage (*i.e.*, when no current is drawn by the power analyzer) and the voltage supplied at a certain current draw. The solid line in Fig. 7(a) shows that the average voltage drop is at most a few mV for typical current draws of our targets; for example, an Opal draws 49 mA when fully active, yielding an average voltage drop of 13 mV. The voltage drop of the other target platforms is even smaller, because they draw less current.

To put these numbers into perspective, we measure the voltage drop of two AA alkaline batteries, a typical power supply in real deployments. The dashed line in Fig. 7(a) shows that their average voltage drop is higher than that of an observer, and is above 23 mV already at a current draw of 20 mA. We compute a linear fit between voltages and currents and find that a target sees an average resistance of 259 mΩ when connected to an observer, which is almost four times smaller than what a target would see with AA batteries (947 mΩ). The results show that power profiling with FlockLab minimally affects the target supply voltage.

Accuracy of power measurements. Next, we evaluate the accuracy of power measurements by computing the relative error between the current draw measured by the observers and the current drawn by the power analyzer.

Number of GPIO pins	Power profiling	Captured GPIO events 99 %	100 %
1	no	80 μs	290 μs
	yes	90 μs	280 μs
5 interleaved	no	20 μs	80 μs
	yes	30 μs	90 μs

Table 3: Minimum required interval between consecutive GPIO events to capture 99% and 100% of generated events. *An observer captures 99 % of events on one GPIO pin if they are at least 90 μs apart.*

The dashed line in Fig. 7(b) shows that FlockLab underestimates at currents below 2 mA and slightly overestimates at higher currents. The relative error is particularly significant for low currents: static offset errors of the current-sense amplifier, manufacturing errors of the shunt resistor, and inaccuracies of the amplifier gain introduce a constant offset and a constant multiplication factor into the measurements. Motivated by this observation, we use linear regression to estimate these constants by comparing the measurements from the observers with those from the power analyzer, effectively calibrating FlockLab's power profiling service.

For each observer and target slot, we repeat the experiment and correct the measured current draw by applying our calibration based on the constants computed from the previous experiment. The solid line in Fig. 7(b) shows that the calibration reduces the relative error on current draw significantly, especially for currents below 1 mA. For currents between 0.1 mA and 160 mA, the accuracy of the power measurements increases by a factor of 6 after calibration.

Based on calibration parameters we computed for all target slots on all 30 observers, the FlockLab test management server corrects the power measurements before delivering them to the user. We show in Sec. 5.3 that this results in accurate power measurements allowing to precisely measure the energy consumed by a target throughout a test.

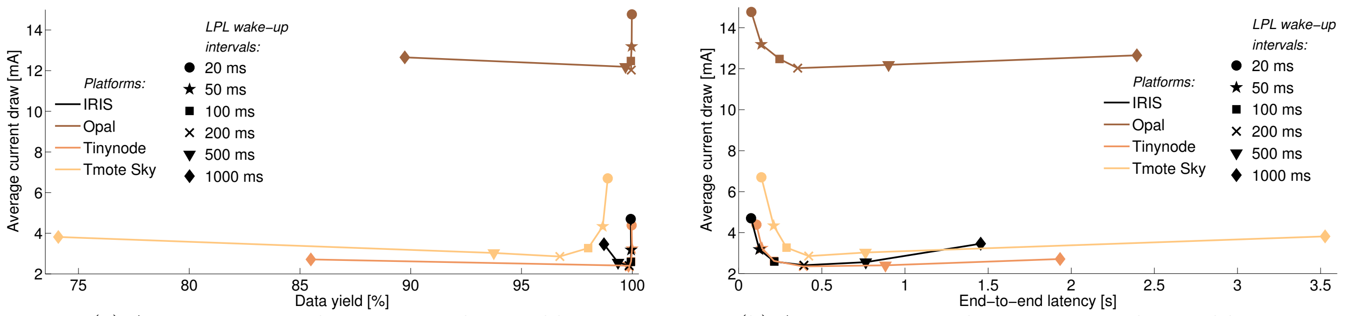
4.3 Limits in Capturing GPIO Events

The sampling rate of the ADC defines the interval between power samples. This is different for tracing GPIO events on an observer: the minimum required interval to reliably capture consecutive events depends on the interrupt delay and the execution time of the top half of the interrupt handler. We run experiments to determine this minimum interval.

Setup. We use all 30 Tmote Skys and let them toggle GPIO pins with an increasing interval. Starting from 10 μs, targets increase the interval in steps of 10 μs up to 1 ms, and generate at each setting 100 GPIO events. We run four tests, each repeated ten times: two where they toggle a single pin and two where they toggle five pins interleaved. In both cases, we run one test with and one test without power profiling.

Minimum interval between GPIO events. For each interval, we compare the number of captured events with the number of generated events. Table 3 lists the minimum required interval to capture 99 % and 100 % of events. First, we see that FlockLab captures more frequent events when they are interleaved on 5 GPIO pins. This is because every GPIO pin is mapped to a specific interrupt flag, and no new events can be captured until the respective flag is cleared.

We further observe that the minimum required interval to capture GPIO events increases by 10 μs when the power profiling service is enabled. This service increases the load on an observer, leading to higher interrupt delays and thus



(a) Average current draw against data yield.
(b) Average current draw against end-to-end latency.

Figure 8: FlockLab enables comparative performance analyses of the same application on multiple platforms. *The plots show performance results from CTP running atop LPL for different LPL wake-up intervals and platforms, including IRIS (2.4 GHz, 3 dBm), Opal (868 MHz, 6 dBm), Tinynode (868 MHz, 12.5 dBm), and Tmote Sky (2.4 GHz, 0 dBm).*

to a lower probability that events are successfully captured. Finally, we note the significant difference between the minimum required intervals for capturing 99 % or 100 % of events, since sporadic activity on the observers (*e.g.*, exchanging and processing NTP packets or storing measurement data into a file) may sometimes increase the interrupt delay, too.

The following section shows that FlockLab's GPIO tracing service allows to accurately record MCU and radio activity, measure end-to-end packet delays, monitor the exchange of packets, and measure the clock drift of a target.

5. FLOCKLAB IN ACTION

After presenting the architecture of FlockLab and evaluating its performance, we now demonstrate the utility of FlockLab for testing, debugging, and evaluating wireless embedded systems through several real-world test cases.

5.1 Comparative Multi-Platform Analysis

One feature that sets FlockLab apart from other testbeds is the possibility to test multiple platforms on the same physical topology. Comparative analyses of this type can provide valuable feedback, for example, to developers of communication protocols, because the characteristics of the underlying platform may affect the performance of these protocols considerably and in non-trivial ways.

In this test case we perform a comparative multi-platform analysis of the standard TinyOS data collection application. The application uses CTP [13] on top of the LPL [27] link layer to collect data from a set of nodes at a single sink. We use all 30 observers and all four targets available in FlockLab: Opal, Tinynode, Tmote Sky, and IRIS. The radios of these platforms differ in terms of frequency band, maximum transmit power, modulation scheme, and data rate. For each individual platform we let the nodes transmit at the highest power setting, and all nodes but the sink generate a packet every 5 s for a duration of 35 min.

We are interested in how each platform affects the trade-offs between energy consumption, data yield, and end-to-end latency. Since these key performance metrics are known to be influenced by the operational parameters of the link-layer protocol [40], we further test for each platform six different LPL wake-up intervals: 20, 50, 100, 200, 500, and 1,000 ms. We thus expect to gain insights into the platform-dependent sensitivity of the system performance to changes in the LPL wake-up interval, too.

Without FlockLab. Despite the lack of multiple platforms on other testbeds, it is not trivial to obtain reliable and unobtrusive measurements of the performance metrics we are interested in. Data yield can be measured quite straightforwardly based on the sequence numbers of received packets, but measuring energy and latency is more difficult.

On testbeds that do not support power profiling, energy consumption must be estimated in software. For example, Energest [6] provides accurate energy estimations in Contiki but is also intrusive (see Sec. 5.3). Other operating systems like TinyOS lack a standard energy estimator. This increases the overhead to obtain energy estimates in the first place, may lead to incomparable results from different custom-built estimators, and generally encourages the use of radio duty cycle as a proxy for energy consumption, which may not be meaningful toward the total node energy budget.

One approach to measure the end-to-end latency is to log a message over the serial port when generating a packet at the source and another message when receiving a packet at the sink. However, serial logging alters the timing behavior of the application, and the resulting timestamps are inaccurate due to non-deterministic UART delays [5]. Another approach is to run a dedicated time synchronization protocol such as FTSP [23] concurrently to the protocol under test and to timestamp packets at the source. But, as shown in [3], running multiple network protocols concurrently entails the risk of unanticipated interactions between protocols that can lead to performence losses or even failures. Furthermore, for some combinations of platforms and operating systems there may not be a synchronization protocol readily available.

With FlockLab. The power profiling service in FlockLab provides non-intrusive current measurements for computing the energy consumption. The GPIO tracing service can be used to measure the end-to-end latency: the application toggles a GPIO pin when a source generates a packet and another GPIO pin when the sink receives a packet. Taking the interval between both events of the same packet, we obtain non-intrusive measurements of the end-to-end latency of received packets. The effort is limited to inserting two GPIO tracing statements in the application code and configuring the FlockLab services in an XML file.

Fig. 8(a) shows data yield and Fig. 8(b) shows end-to-end latency against average current draw[2], for all platforms and

[2]The high current draw with Opal is due to a software issue that prevents the nodes from entering a low-power mode.

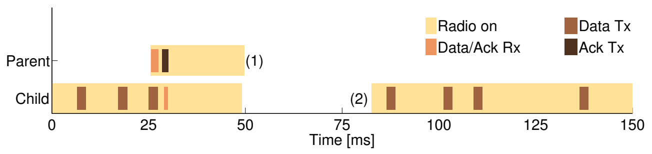


Figure 9: GPIO trace showing a misconfiguration of CTP and LPL on Tinynodes. *After receiving a packet, the parent turns off the radio (1) before the child sends the next packet (2), causing packet loss due to queue overflows.*

LPL wake-up intervals. As expected, higher data yield and lower end-to-end latency can generally be achieved at the expense of higher average current draw. While this holds for all platforms, data yield and end-to-end latency are better with IRIS, Opal, and Tinynode than with Tmote Sky, since the higher transmit power of the former platforms leads to shorter routing paths with CTP. Interestingly, IRIS is least sensitive to changes in the LPL wake-up interval, and all four platforms draw minimum current at 200 ms LPL wake-up interval, which is thus the most energy-efficient parameter setting for this particular topology and traffic load.

5.2 Finding and Fixing Bugs

GPIO tracing is also a very powerful debugging tool. We already found and fixed several bugs this way, and as a concrete example we describe next how we found and fixed a protocol misconfiguration that caused a poor performance during initial experiments of the previous test case.

Finding the bug. With LPL wake-up intervals of 500 ms and 1 s, the initial results from Tinynode and Opal nodes were significantly worse than expected in terms of data yield and end-to-end latency. All sources were seemingly affected, and we could not pinpoint specific nodes to debug with a logic analyzer. We thus instrumented the radio stacks to set different GPIO pins according to the current radio state (*i.e.*, sleeping, active, receiving, or transmitting) and repeated the experiments with GPIO tracing enabled. Using `printf`s instead of GPIO, we would have run the risk of breaking the timing-sensitive operation of the radio driver and LPL.

After aligning the GPIO traces of all nodes, we noticed that nodes located farther away from the sink could communicate properly. The bug indeed affected mostly nodes close to the sink, which delivered only a small fraction of the many packets they had to forward. We decided to focus on these nodes and, by looking deeper into the transfers between a child and its parent, we found that children were transmitting at most one packet during an LPL wake-up interval, although they had multiple packets ready to be sent.

Fixing the bug. With the help of GPIO traces, we were also able to find and fix the cause of this bug. Fig. 9 shows an example of the problem, based on GPIO traces collected from two Tinynodes. After a successful packet reception, the parent kept the radio on for a short time but went to sleep (1) before the child could transmit the next packet (2). As a result, children had to wait until the next regular wake-up of their parents before they could send the next packet, which caused severe data loss at long wake-up intervals.

This prompted us to check the configurations of CTP and LPL. Based on our settings, a Tinynode or Opal node kept the radio on for 20 ms after a reception, but its children transmitted additional packets only after 32 ms (the default CTP setting for generic platforms). We fixed this misconfiguration by changing the value of these parameters based on the radios' data rate and experimental results. For example, on Tinynodes a parent keeps the radio on for 36 ms after a reception, and a child transmits the next packet after 10 ms.

5.3 Controlling and Profiling Applications

When evaluating applications like data collection it is often desirable not only to precisely measure performance figures but also to control nodes during an experiment, for example, to specify which nodes generate packets and when [40], or to emulate failures by turning some nodes off during a certain interval [13]. We now show that FlockLab greatly helps control and profile typical data collection applications.

In this test case, we run the default data collection application of Contiki on Tmote Sky targets, that is, Collect on top of ContikiMAC. The wake-up interval of the latter is 128 ms. We want one node to generate a packet every 2 s for 260 s, from $t = 30$ s to $t = 290$ s. We also want to measure the energy consumed by that node during these 260 s.

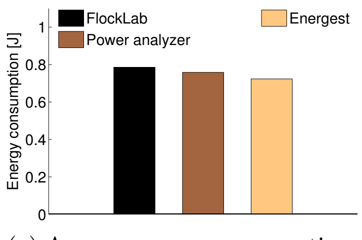
Controlling without FlockLab. A common approach to control an experiment is to add some logic that, for example, starts and stops the generation of packets depending on the current time and the identifier of the node. This approach requires to recompile the application program for tests that need different parameterization, which is time-consuming. Most importantly, some form of in-band time synchronization is also needed if several nodes are to simultaneously start and stop generating packets, which can, however, degrade the performance of the application under test [3].

Controlling with FlockLab. With GPIO actuation we can control the targets without employing an additional time synchronization within the application. In our test case, the observer connected to the node of interest sets a GPIO pin at $t = 30$ s and clears it at $t = 290$ s: the target starts and stops generating packets accordingly. Because the observers are time-synchronized, it is also possible to let multiple targets start and stop generating packets simultaneously. Moreover, we can test different generation patterns by simply modifying the GPIO actuation timings in the test configuration.

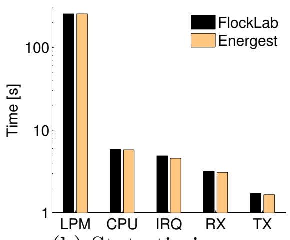
Profiling without FlockLab. On testbeds without power profiling, the energy consumption of a node can be estimated in software. For example, Energest measures the time spent by the node in different states [6], which can be combined with the current draws in each state to estimate energy. This method is however intrusive, since it requires nodes to start and stop counters whenever they change state, and requires changes to existing code to be used on a different platform.

Profiling with FlockLab. Power profiling allows to measure the energy consumption of any supported target in a completely non-intrusive fashion. GPIO tracing allows also to profile a target's operation and measure state timings by setting GPIO pins according to the MCU and radio states.

In our test case, we enable power profiling between $t = 30$ s and $t = 290$ s. Fig. 10(a) shows the energy consumption measured by FlockLab, averaged over eight test repetitions, and compares it with measurements from a power analyzer attached to the target and with software-based estimations using Energest. The non-intrusive FlockLab measurements are slightly more accurate than the estimations provided by Energest: the former measures an average energy consumption that is 3.6 % higher than the one measured with the power analyzer, while the latter underestimates it by 4.6 %. We also see from Fig. 10(b) that the state timings



(a) Average energy consumption.



(b) State timings.

Figure 10: FlockLab can be leveraged to obtain non-intrusive and highly accurate energy measurements.

measured with GPIO tracing correspond on average within 2.7 % to those reported by Energest. FlockLab is however less intrusive than Energest; for example, we measure on a Tmote Sky that Energest requires 11 and 21 MCU cycles to start and stop a counter, whereas only 5 cycles are required to set the level of a GPIO pin on a Tmote Sky.

5.4 Measuring Clock Drift

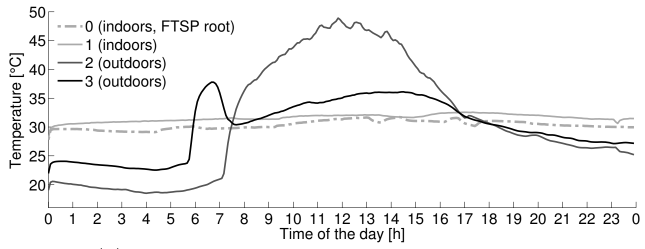
When evaluating communication and time synchronization protocols, it is often desirable to measure how much the clock of a target drifts from the nominal frequency during a test. We now demonstrate that FlockLab allows to run tests where targets experience different clock drifts (*e.g.*, by using targets located outdoors) and to measure the actual drift during a test accurately and minimally intrusive.

In this test case, we want to measure the clock drift of 30 Tmote Sky targets during a 24 h experiment. In particular, we are interested in comparing the drift of indoor and outdoor targets, and in relating the drift to the temperature measured by the targets' on-board sensors during the test.

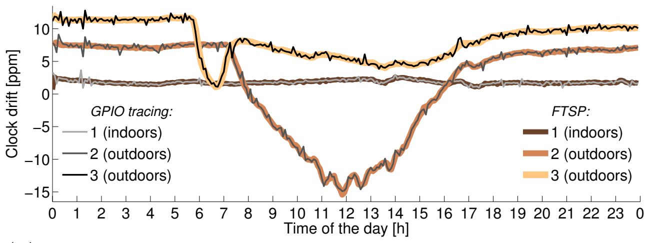
Without FlockLab. A possible method to measure the clock drift is to employ FTSP, the default time synchronization protocol in TinyOS, as it periodically estimates how much the clock of a node drifts compared to the clock of a root [23]. As previously discussed, time synchronization protocols are however intrusive and may affect the behavior and the performance of the application under test.

With FlockLab. With GPIO tracing we can measure the clock drift of a target in a simpler and less intrusive way, without the need of running a synchronization protocol on the target. In our test case, we instrument the application to toggle a GPIO pin every 0.5 s, and measure the clock drift by comparing the difference between consecutive GPIO timestamps with the nominal value of 0.5 s. We then average these drift values over intervals of 5 min to limit the GPIO timing errors discussed in Sec. 4.1.1. To evaluate the accuracy of our measurements, we enable FTSP with a resynchronization interval of 3 s and use an indoor target as the root.

Fig. 11(a) shows the temperature measured during the 24 h by the FTSP root and three other targets, one located indoors and two outdoors. We notice that during daytime the outdoor targets experience significant (but different) temperature variations, while the indoor targets measure fairly constant temperatures. Fig. 11(b) shows how the clocks of the three targets drift compared to the clock of the FTSP root, measured with GPIO tracing and by FTSP. As expected, we see that variations in temperature translate into variations in the targets' clock speed and thus into varying drift. We also notice that the drift measured with GPIO tracing corresponds to that estimated by FTSP: their



(a) Temperature measured by four targets.



(b) Clock drift of three targets compared to the FTSP root, measured with GPIO tracing and by FTSP.

Figure 11: With FlockLab it is possible to accurately measure clock drift on multiple targets during an experiment with minimal intrusiveness.

difference is hardly noticeable in Fig. 11(b) and averages 0.003 ppm. An observer performs such accurate drift measurements despite temperature variations affect also its clock speed, because it resynchronizes at least every 2 min with the FlockLab NTP server.

5.5 Multi-Modal Monitoring at Network Scale

The possibility of monitoring the activity of multiple targets while simultaneously measuring their current draws is invaluable for developers of low-power wireless applications. This allows, for example, to trace the exchange of packets among targets, to analyze in which states targets consume most energy, or to detect possible misbehaviors that may cause targets to reach undesired states or to draw more current than expected. Unlike any existing testbed, FlockLab offers this possibility, and with a minimal effort from a user.

As a test case we use the Glossy flooding protocol, which lets an initiator flood a packet to all receivers within a few milliseconds [10]. We set the transmit power of 26 Tmote Sky targets to -10 dBm and let Glossy flood a 30-byte packet every 24 ms, using different initiators in consecutive floods.

Without FlockLab. With current testbeds, the only possibility to monitor state transitions or packet exchanges is to instrument an application to store timestamps (*e.g.*, into external flash memory) whenever an event of interest occurs. Nodes print these timestamps at the end of a test, and the testbed collects them from the serial ports. As mentioned before, this approach is very intrusive and provides meaningful results only if the nodes employ some form of in-band time synchronization. Alternatively, network simulators like Cooja [26] can be used to visualize the exchange of packets, but their channel and hardware models may not accurately reproduce what happens during an experiment on real devices. With either approach, however, no information about the instantaneous current draws of the nodes is available.

With FlockLab. GPIO tracing allows to monitor the radio states of the targets and the exchange of packets among them with minimal intrusiveness. In our test case, we simply

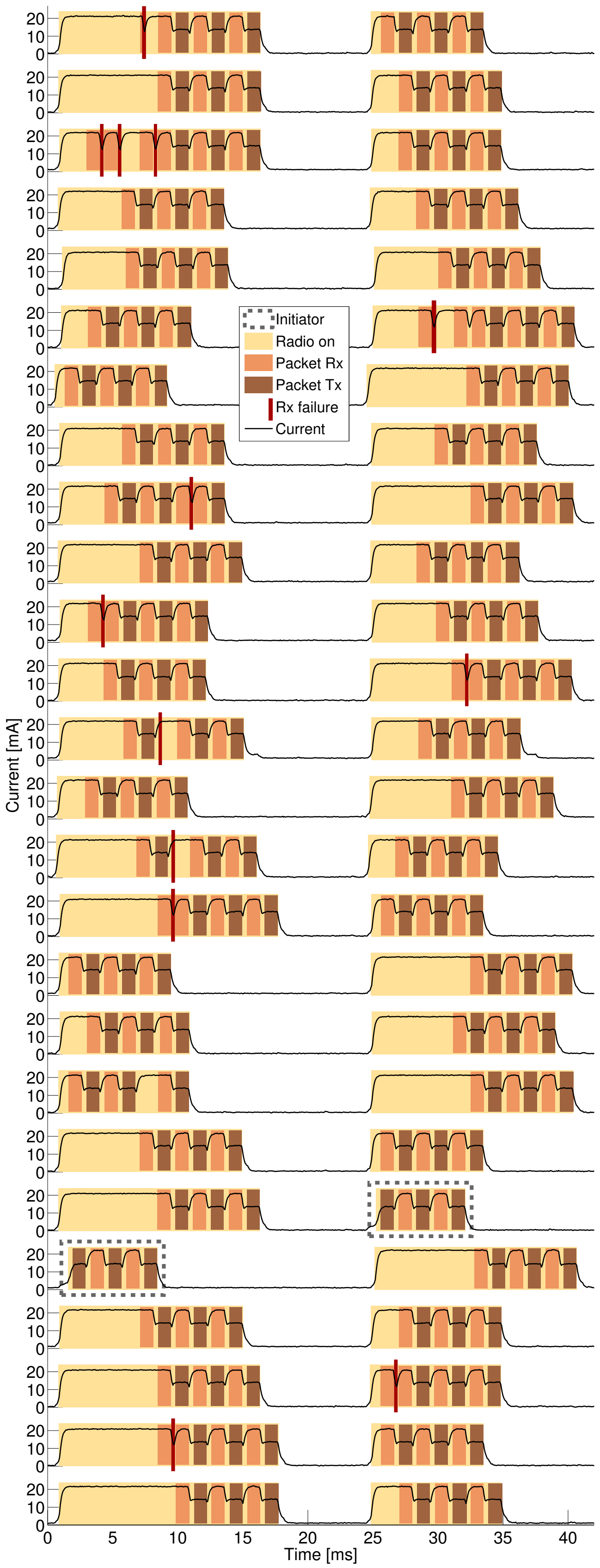


Figure 12: GPIO tracing and power profiling allow for monitoring the activity of multiple targets while simultaneously measuring their current draw.

instrument Glossy to toggle four GPIO pins whenever the radio state changes: when it is turned on or off, when it starts or stops receiving or transmitting a packet, and when a packet reception fails (*e.g.*, because of packet corruption). Together with GPIO tracing, we enable also power profiling to measure the current draw of the 26 targets.

Fig. 12 shows a graphical representation of the radio states and the current draws of the 26 targets during two consecutive floods with different initiators, based on a short excerpt of the GPIO and power traces collected with FlockLab. The timing accuracy of FlockLab allows to precisely monitor how packets propagate in the network based on the reported radio states. For example, it is clearly visible that multiple targets transmit packets simultaneously, which is a peculiarity of Glossy. It is also possible to analyze the sets of targets transmitting or receiving at a certain time instant, and study how they are related to the network topology.

This type of visualization resembles that of the Cooja simulator, but with FlockLab it is based on information collected during experiments on real devices and over real wireless links. FlockLab provides also the current draws of the targets during the experiment. It is thus possible to correlate logical states and power samples and, for example, to measure the energy cost of different states. As expected, Fig. 12 shows that targets draw most current when the radio is turned on, and in particular when they are receiving or waiting for a packet; transmissions are indeed cheaper due to the low transmit power used in the experiment.

To the best of our knowledge, FlockLab is the first testbed that, among other features, provides the functionality of multiple logic analyzers and power analyzers—distributed and synchronized across the entire testbed. We maintain that developers of distributed applications and low-power wireless protocols can significantly benefit from such augmented debugging and testing capabilities.

6. RELATED WORK

Sensor network testbeds. Departing from most of the early installations [9, 37], emerging testbeds are increasingly diverse and specialized: relocatable testbeds to evaluate applications in the intended target environment [29], testbeds with robots for controlled mobility experiments [18], and testbed federations to assess large-scale services [4]. FlockLab, instead, aims to provide visibility into the distributed behavior of protocols and applications, to detect bugs and inefficiencies early in the development cycle. As such, to the best of our knowledge, FlockLab is the first testbed with verified support for distributed, synchronized GPIO tracing and actuation coupled with high-resolution power profiling.

Closest to FlockLab are PowerBench [15], SANDbed [16], and w-iLab.t [2]. As shown in Table 4, these testbeds also provide distributed power measurements at comparable or lower rates and resolutions. FlockLab also achieves a better synchronization, allowing for a better alignment of power traces recorded at different nodes. Furthermore, FlockLab supports four different platforms and future platforms can be added with little effort, whereas the other testbeds support only one platform. We note that w-iLab.t also seems to support GPIO-based services, but there exists no public information on the performance of these services in w-iLab.t.

DSN provides coarse network-wide power sensing by sampling the nodes' current draw every few minutes [8]. In addition, like MoteLab [37], DSN instruments one node with

Testbed	#Nodes	Supported Platforms	Sampling Rate	Resolution	Synchronization Error
PowerBench	28	TNOde	5 kHz	12 bit	∼1,000 μs
SANDbed	28	MicaZ	40 kHz	16 bit	∼10,000 μs
w-iLab.t	200	Tmote Sky	10 kHz	12 bit	unknown
FlockLab	4×30	Opal, Tinynode, IRIS, Tmote Sky	high-resolution: 28 kHz high-speed: 56 kHz	24 bit	avg/85th%: 39 μs/68 μs

Table 4: Existing testbeds supporting distributed power measurements. FlockLab *is the only testbed with multiple platforms; it provides the highest resolution, and timestamps power samples with 20–200× better synchronization accuracy.*

a high-precision multimeter. FlockLab clearly exceeds the capabilities of these testbeds. However, the approach of coupling targets with powerful observers is inspired by these and other systems [12]. Like FlockLab, many support multiple platforms [4, 14], but, unlike FlockLab, only serial I/O.

Power and energy estimation. Several sensor network simulators [1, 32] and emulators [21] provide power or energy estimation capabilities. They mainly differ in the level of detail in which they model hardware components and program execution, and hence in the accuracy of their estimates. The basic approach consists of recording the time each hardware component spends in each power state, and combining these data with a calibrated power model of the target node.

Software-based online energy estimation follows the same approach, but performs time measurements on real nodes [6]. Different from simulation or emulation, intricate effects of interrupts and timers are automatically taken into account. Changes to existing code, overhead in terms of processing, memory, and code footprint, and lack of visibility into the instantaneous power draw are the downside of this approach.

By contrast, FlockLab measures power non-intrusively on several platforms, enabling detailed profiling prior to deployment. Thus, FlockLab has advantages especially in the early stages of development, whereas software-based estimation allows for energy profiling on larger testbeds.

Power and energy measurement. A number of methods exist for measuring rather than estimating power or energy. Some target external profiling [24, 35], while others enable a node to measure its own consumption [7, 17]. Different from FlockLab, none of them addresses the challenge of synchronizing measurements across multiple nodes.

SPOT [17] uses a voltage-to-frequency converter to feed an energy counter that is read by the node. It achieves high accuracy across a large dynamic range, assuming a constant supply voltage. Aveksha adopts a similar approach to obtain power traces [35]. The design in [36] measures also the supply voltage to accurately calculate energy as the voltage varies. iCount provides energy metering at nearly zero cost by counting the cycles of a node's switching regulator [7].

Quanto [11] builds on iCount to obtain the energy breakdown per programmer-defined activity, using regression models and causal activity tracking. Targeting high-performance sensing platforms, [33] resolves energy usage at the level of processes and hardware components using a dedicated integrated circuit. By combining GPIO tracing with power profiling, also FlockLab can be used to track network-wide activities and subsequently attribute costs to each activity.

Sensor network debugging. A wealth of research has been devoted to diagnosing and debugging wireless embedded systems. Existing approaches target failures caused by interactions among multiple nodes [19, 30], network faults such as routing and node failures [22, 28], or node-local bugs including data races and stack overflows [34, 39].

Most of these systems feature a frontend that collects data about the running system and a backend that analyzes these data for possible failures. FlockLab does not solve the latter, but it provides correlated power and event traces in a way that is nearly unobtrusive for the debugged application. This is in sharp contrast with many debugging techniques that perturb the timing behavior by adding debug statements, logging events into non-volatile memory, or transmitting debug messages in-band with application traffic. Because of this, FlockLab can be highly effective in detecting failures due to time-critical interactions among multiple nodes, possibly by applying distributed assertions [30] or data mining techniques [19] on event traces. Moreover, power traces can be exploited for conformance testing [38] and failure diagnosis [20]. For cycle-accurate debugging of a single node, however, other solutions may be more suitable.

For instance, Aveksha uses a custom-built debug board to interface with the on-chip debug module through JTAG [35]. It provides breakpoints, watchpoints, and program counter polling for very detailed event tracing, and power measurements that can be correlated with events of interest. Aveksha is truly non-intrusive, except for breakpoints. However, the design is tied to MSP430-based platforms, and setting triggers correctly may require detailed knowledge of machine code and memory addresses. Instead, FlockLab makes distributed event tracing as simple as LED and `printf` debugging, supports several platforms and MCUs, and facilitates the integration of new ones with little effort.

7. CONCLUSIONS AND FUTURE WORK

FlockLab is the result of a multi-year effort to push beyond the capabilities of contemporary testbeds, providing the research community with a shared tool to study wireless embedded system in unprecedented detail. We presented the design of FlockLab, benchmarked its performance, and demonstrated its utility through real-world test cases.

Looking ahead, we see the need for systematic approaches that guide developers in (or partially relieve them from) instrumenting their code with tracing statements, and for tools that visualize and aid in analyzing the possibly huge amount of test data. As a first step, we have published the sources of the test cases presented in this paper as tutorials on the FlockLab website `http://www.flocklab.ethz.ch/`.

Acknowledgements

We thank all the people that helped us over the past years in building FlockLab, most notably Matthias Woehrle for starting off the project, Mustafa Yuecel for his contribution to the design of the observer hardware, and Alexandru Caracas for adding support for IRIS. We also thank our shepherd, Prabal Dutta, and the anonymous reviewers for their valuable comments. This work was supported by nano-tera.ch and NCCR-MICS under SNSF grant number 5005-67322.

8. REFERENCES

[1] A. Barberis, L. Barboni, and M. Valle. Evaluating energy consumption in wireless sensor networks applications. In *10th Euromicro Conf. on Digital System Design Architectures, Methods and Tools (DSD)*, 2007.

[2] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester. The w-iLab.t testbed. In *Proc. of the 6th ICST Intl. Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, 2010.

[3] J. Choi, M. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In *Proc. of the 7th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2009.

[4] G. Coulson, B. Porter, I. Chatzigiannakis, C. Koninis, S. Fischer, D. Pfisterer, D. Bimschas, T. Braun, P. Hurni, M. Anwander, G. Wagenknecht, S. P. Fekete, A. Kröller, and T. Baumgartner. Flexible experimentation in wireless sensor networks. *Communications of the ACM*, 55(1), 2012.

[5] M. Delvai, U. Eisenmann, and W. Elmenreich. Intelligent UART module for real-time applications. In *Proc. of the 1st Workshop on Intelligent Solutions in Embedded Systems (WISES)*, 2003.

[6] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proc. of the 4th Workshop on Embedded networked sensors (EmNets)*, 2007.

[7] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proc. of the 7th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2008.

[8] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum. Deployment support network: A toolkit for the development of WSNs. In *Proc. of the 4th European Conf. on Wireless sensor networks (EWSN)*, 2007.

[9] E. Ertin, A. Arora, R. Ramnath, M. Nesterenko, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, and H. Cao. Kansei: A testbed for sensing at scale. In *Proc. of the 5th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2006.

[10] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proc. of the 10th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2011.

[11] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proc. of the 8th USENIX Conf. on Operating systems design and implementation (OSDI)*, 2008.

[12] L. Girod, J. Elson, T. Stathopoulos, M. Lukac, and D. Estrin. EmStar: A software environment for developing and deploying wireless sensor networks. In *Proc. of the USENIX Annual Technical Conf.*, 2004.

[13] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proc. of the 7th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2009.

[14] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. TWIST: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proc. of the 2nd Intl. Workshop on Multi-hop ad hoc networks: from theory to reality (REALMAN)*, 2006.

[15] I. Haratcherev, G. Halkes, T. Parker, O. Visser, and K. Langendoen. PowerBench: A scalable testbed infrastructure for benchmarking power consumption. In *Proc. of the Intl. Workshop on Sensor Network Engineering (IWSNE)*, 2008.

[16] A. Hergenröder, J. Wilke, and D. Meier. Distributed energy measurements in WSN testbeds with a sensor node management device (SNMD). In *Workshop Proceedings of the 23rd Intl. Conf. on Architecture of Computing Systems (ARCS)*, 2010.

[17] X. Jiang, P. Dutta, D. Culler, and I. Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *Proc. of the 6th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2007.

[18] A. Jiménez-González, J. Martínez-de Dios, and A. Ollero. An integrated testbed for heterogeneous mobile robots and other cooperating objects. In *Proc. of the 23rd IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[19] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han. Dustminer: Troubleshooting interactive complexity bugs in sensor networks. In *Proc. of the 6th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2008.

[20] M. M. H. Khan, H. K. Le, M. LeMay, P. Moinzadeh, L. Wang, Y. Yang, D. K. Noh, T. Abdelzaher, C. A. Gunter, J. Han, and X. Jin. Diagnostic powertracing for sensor node failure analysis. In *Proc. of the 9th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2010.

[21] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proc. of the 2nd Workshop on Embedded networked sensors (EmNets)*, 2005.

[22] K. Liu, M. Li, Y. Liu, M. Li, Z. Guo, and F. Hong. Passive diagnosis for wireless sensor networks. In *Proc. of the 6th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2008.

[23] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *Proc. of the 2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2004.

[24] A. Milenkovic, M. Milenkovic, E. Jovanov, D. Hite, and D. Raskovic. An environment for runtime power monitoring of wireless sensor network platforms. In *Proc. of the 37th Southeastern Symp. on System Theory (SSST)*, 2005.

[25] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network time protocol version 4: Protocol and algorithms specification. RFC 5905, 2010.

[26] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with COOJA. In *Proc. of the 7th IEEE Intl. Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*, 2006.

[27] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proc. of the 2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2004.

[28] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proc. of the 3rd ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2005.

[29] O. Rensfelt, F. Hermans, L.-Å. Larzon, and P. Gunningberg. Sensei-UU: A relocatable sensor network testbed. In *Proc. of the 5th ACM Intl. Workshop on Wireless network testbeds, experimental evaluation and characterization (WiNTECH)*, 2010.

[30] K. Römer and J. Ma. Passive distributed assertions for sensor networks. In *Proc. of the 8th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2009.

[31] R. Shea, M. Srivastava, and Y. Cho. Scoped identifiers for efficient bit aligned logging. In *Proc. of the Conf. on Design, Automation and Test in Europe (DATE)*, 2010.

[32] V. Shnayder, M. Hempstead, B. Chen, G. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. of the 2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2004.

[33] T. Stathopoulos, D. McIntire, and W. J. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In *Proc. of the 7th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2008.

[34] V. Sundaram, P. Eugster, and X. Zhang. Efficient diagnostic tracing for wireless sensor networks. In *Proc. of the 8th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2010.

[35] M. Tancreti, M. S. Hossain, S. Bagchi, and V. Raghunathan. Aveksha: A hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems. In *Proc. of the 9th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2011.

[36] T. Trathnigg and R. Weiss. A runtime energy monitoring system for wireless sensor networks. In *Proc. of the 3rd Intl. Symp. on Wireless Pervasive Computing (ISWPC)*, 2008.

[37] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A wireless sensor network testbed. In *Proc. of the 4th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2005.

[38] M. Woehrle, K. Lampka, and L. Thiele. Exploiting timed automata for conformance testing of power measurements. In *Proc. of the 7th Intl. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2009.

[39] J. Yang, M. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: A comprehensive source-level debugger for wireless sensor networks. In *Proc. of the 5th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2007.

[40] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. pTunes: Runtime parameter adaptation for low-power MAC protocols. In *Proc. of the 11th Intl. Conf. on Information Processing in Sensor Networks (IPSN)*, 2012.