A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge

Markus Schuß, Carlo Alberto Boano, Manuel Weber, and Kay Römer
Institute for Technical Informatics
Graz University of Technology, Austria
{markus.schuss, cboano, manuel.weber, roemer}@tugraz.at

Abstract

A large number of low-power wireless communication protocols has been proposed in the last decade by both academia and industry in an attempt to deliver information in an increasingly reliable, timely, and energy-efficient manner. However, their level of dependability has rarely been benchmarked under the same settings and environmental conditions. In this paper we present the execution and results of a competition aimed to evaluate the dependability of state-of-the-art low-power wireless protocols under the same settings, and push their performance to the limit. We define a scenario emulating the operation of a wireless sensor network in industrial environments rich with radio interference and compare the end-to-end dependability of systems based on protocol strategies ranging from adaptive and time-slotted frequency-hopping to multi-modal routing and flooding. To increase fairness and realism, we allow the developers of the competing protocols to interact with the benchmarking infrastructure and optimize the protocol parameters for the scenario at hand. We achieve this by designing, implementing, and employing D-Cube, a low-cost tool that allows to accurately measure key dependability metrics such as end-to-end delay, reliability, and power consumption, as well as to graphically visualize their evolution in real-time. This interaction with the benchmarking infrastructure and the competitiveness of the event have incited the developers to push the performance of their protocols to the limit and reach impressive results.

Categories and Subject Descriptors

B8.2 [**Performance Analysis and Reliability**]

General Terms

Design, Measurement, Performance, Reliability.

Keywords

Competition, Dependability, Performance, Testbeds.

International Conference on Embedded Wireless Systems and Networks (EWSN) 2017
20–22 February, Uppsala, Sweden

ISBN: 978-0-9949886-1-4

1 Introduction

Low-power wireless sensor networks are becoming an integral part of the Internet of Things (IoT) and are envisioned to be soon employed in safety-critical application domains such as smart production, smart cities, and connected cars. This class of applications imposes strict dependability requirements on network performance and the employed communication protocols are envisioned to deliver information in a reliable, efficient, and timely manner, i.e., they are expected to minimize packet loss and energy expenditure, as well as to keep end-to-end delays below given bounds. To avoid major system failures, it is very important to thoroughly compare the end-to-end dependability of low-power wireless systems and get deeper insights about their strengths and limits.

The research community traditionally validates new communication protocols experimentally on publicly available large-scale testbeds such as Indriya [12] or TWIST [27], and compares their performance with the one of previously published solutions. Although very common (new protocols regularly appear and are evaluated following this methodology), this practice does *not* allow a fair and objective comparison of protocol performance, and does *not* enable a proper understanding of the end-to-end dependability of a system.

Same testbed $\not\Rightarrow$ comparable results. A first problem is that comparing communication protocols to each other by running experiments on the same testbed and quantitatively juxtaposing the numbers is not sufficient to provide a fair comparison. The wireless environment in the testbed can indeed change significantly even between consecutive experiments. Minor variations in the link quality, channel congestion, or temperature gradient across the testbed are sufficient to affect protocol performance and cause differences in the results that may lead to false conclusions [3]. This problem is exacerbated by the fact that testbeds are often deployed unattended in public buildings, and any disturbance (e.g., doors being opened or closed, and Wi-Fi networks operating in close proximity) may radically change the propagation characteristics and the connectivity between nodes. Also, testbeds are continuously evolving (i.e., subject to architectural changes or replacement of nodes) and shared with other researchers (i.e., other jobs may be scheduled in between the comparison of the same set of protocols), which further affects the comparability of results.

Same setup $\not\Rightarrow$ fair comparison. When experimenting on public testbeds, researchers typically embed the specifica-

tions of an experiment directly into the application firmware. They manually define themselves the traffic load and pattern, specify which nodes are (in)active, generating information, or collecting data, as well as select the transmission power of the nodes to control network density or to enforce a specific network diameter. The same setup is then used to compare the candidate protocol to the state-of-the-art. Although necessary, running the exact same setup is not sufficient to guarantee fair comparisons. On the one hand, an ill-suited setup may accidentally favor the performance of a specific protocol. On the other hand, each protocol has unique properties and its *parameters* need to be carefully tuned for the scenario at hand, as tiny differences in the parametrization may lead to quite different results. The common practice is to rely on default settings – a problem that is amplified by the lack of a central protocol repository. This often results in comparisons of protocols running on different (versions of) operating systems, which may further bias the results.

Detailed logs $\not\Rightarrow$ accurate metrics. Several public testbeds allow the collection of user data over a serial backchannel. Researchers typically use this feature to log diagnostic messages and extract evaluation metrics used as a basis to compare protocol performance (e.g., the packet loss ratio by counting the number of lost packets in the network). The ability of creating detailed logs, however, does not imply the extraction of metrics that can accurately characterize protocol performance. On the one hand, log messages of testbeds are typically not time-stamped at sub-millisecond scales and therefore cannot be used to accurately profile delays. On the other hand, log messages are generated using `printf` instructions that may alter the timing behavior of protocols and break their functionality. Recent feature-rich testbeds such as FlockLab [36] are equipped with expensive hardware and can help in unobtrusively and accurately profile power and delays without the need of logs, although on a limited number of nodes and hence not on a large scale.

Performing protocol $\not\Rightarrow$ dependable system. Evaluating the performance of a communication protocol and showing improvements on specific low-level metrics (e.g., number of parent switches, path length in hops, or communication overhead) does not allow to draw any conclusion on the *end-to-end dependability* of a system. When it comes, for example, to a safety-critical IoT application that monitors events and reports them to a central unit (e.g., a healthcare application monitoring the vital signs of a patient), it is fundamental to evaluate the end-to-end dependability of the system from real-world event to back-end notification.

All these problems call for a proper *benchmarking of the dependability of IoT protocols* under the same repeatable and controlled settings. This benchmarking should be carried out using objective and fair metrics, while allowing developers to optimize their protocols by tuning their parameters, and should include the repeatable generation of *environmental conditions* that impact network performance.

Furthermore, there is also a need to push the performance to the edge, as benchmarking protocols typically just consists in mere comparisons of existing implementations and does not encourage developers to optimize their solutions. In this context, official *competitions*, such as the RoboCup [22] and the Darpa Grand Challenge [11] are known to spur all contestants into producing new and better results.

In this paper we present the execution and the results of the *EWSN 2016 dependability competition*, a contest that aims to benchmark the dependability of state-of-the-art IoT protocols in environments rich with radio interference. We define a scenario that emulates the operation of a sensor network monitoring discrete events in an industrial setting where several co-existing wireless devices are crowding the RF spectrum, and evaluate how reliably, timely, and efficiently, the competing protocols can report the occurrence of these events to a sink node. The competing IoT protocols are based on techniques ranging from adaptive and time-slotted frequency-hopping to multi-modal routing and flooding, and are selected among the state-of-the-art.

To increase fairness and realism, we allow the developers of the competing protocols to interact with the benchmarking infrastructure and optimize the protocol parameters for the specific application scenario at hand. We achieve this by designing, implementing, and employing *D-Cube*, a low-cost tool that allows to accurately measure key dependability metrics and graphically depict their evolution in real-time. D-Cube is built using off-the-shelf components and allows to accurately profile the power consumption of a device, measure end-to-end latency at sub-μs accuracy, and detect the occurrence of specific events. We employ D-Cube to create our benchmark scenario with minimal costs (up to a 10-fold cost reduction w.r.t. the state-of-the-art [36]) and to allow the contestants to monitor the performance of their protocols live. As we will show in the paper, this interaction with the benchmarking infrastructure and the competitiveness of the event incite the developers to push the performance of their protocols to the limit, and obtain high levels of dependability.

This paper proceeds as follows. Sect. 2 discusses related works in the area. Sect. 3 gives an overview of the goals of the competition and describes the evaluation metrics, as well as the requirements of the necessary benchmarking infrastructure. We address these requirements by designing D-Cube and describe its architecture and implementation in Sect. 4 and evaluate its accuracy in Sect. 5. We describe the execution of the EWSN 2016 dependability competition and its results in Sect. 6, along with a discussion of the lessons learned. We finally conclude the paper and briefly outline the future work in Sect. 7.

2 Related Work

Although benchmarking the dependability of low-power wireless networks is a well-known problem in the research community, very few works related to benchmarking can be found in the literature. Their focus is mostly on the performance of operating systems [34, 50], processors [42, 33], hardware platforms [29, 43], data processing techniques [30, 38], and protocol stack design verification [23, 31].

A few works have experimentally benchmarked the performance of operating systems for low-power networked embedded systems [34, 50]. Watfa et al. [50] have profiled the performance of TinyOS, Contiki, and Mantis OS when running the same application, and compared the amount of time

spent in specific states. Lajara et al. [34] have carried out a similar study, but focused instead on current consumption.

TinyBench [29] is a benchmark suite created to test wireless sensor network devices running TinyOS firmware. The benchmark focuses solely on the internal metrics of a single sensor node such as code size, execution time, or power consumption, and the authors show an exemplary application by using it to compare the power consumption of the same TinyOS application on different mote platforms. Also tailored to TinyOS is the work by Van Gerwen et al. [23], who proposed a benchmark workflow to evaluate the interaction of communication protocols at different layers. This framework allows the binding of a MAC and a routing protocol at compile time, and the authors show an application of this framework by evaluating the performance of different combinations of MAC and routing protocols under the same settings [24]. Their performance comparison work is, however, specific for TinyOS, employs the default settings of the compared protocols, and does not focus on the impact of environmental influences on network performance. Another work aiming to benchmark the correctness of the protocol stack design was presented by Kim et al. [31], who designed a framework that verifies the correctness and interoperability of a given protocol stack automatically. The authors propose an XML schema to define test conditions and procedures in a formal way and to generate source code for exchanging messages and commands between test application and driver.

When it comes to benchmarking protocol performance, the research community still struggles to provide proper comparisons, as it lacks a reference suite [46]. Developers often compare the performance of a newly proposed protocol to the state-of-the-art employing large-scale public testbeds such as MoteLab [52], Kansei [19], Indriya [12], and TWIST [27]. They define an ad-hoc setup, extract the metrics of interest from the logs collected using the serial back-channel, and draw conclusions about the goodness of the proposed solution [16, 18, 25]. As delays and power consumption cannot be precisely extracted from logs, the community started to develop more advanced infrastructures that allow a fine-grained measurement of power consumption [8, 9, 28] and timing-sensitive information [36, 37]. A notable example of a public testbed with these advanced features is FlockLab [36], which is increasingly used by the community to evaluate protocol performance in a sparse network [10, 20, 41]. However, a disadvantage of FlockLab is the high cost per node (1000 USD), which makes it hard to replicate the infrastructure on a large-scale.

All works benchmarking a newly proposed protocol against the state-of-the-art, however, share the same limitations: the different setup, the hardly-repeatable settings, the use of different protocol parameters, and the lack of information about the end-to-end dependability that a system employing that solution can offer. Differently from these works, in this paper we run a *competition* and benchmark the dependability of IoT protocols (i) under the same controlled settings, (ii) by injecting environmental influences in the evaluation, (iii) allowing developers to tailor protocol parameters, and (iv) with the goal of finding which solution performs best in a specific application scenario. Another competition similar in spirit to the one presented in this paper is the Microsoft localization competition [39], which is co-located with the IPSN conference since 2014 and compares the performance of state-of-the-art localization algorithms.

A number of other works have studied the impact of environmental conditions on protocol performance. Boano et al. [5] have designed JamLab, a tool that allows the repeatable generation of interference patterns using off-the-shelf wireless sensor nodes. This tool has been used to generate repeatable interference patterns and evaluate the performance of protocols in the presence of interference [7, 17, 44, 45]. The main limitation when using JamLab in public testbeds, however, is the inability of controlling the background interference in the testbed environment, which limits reproducibility across experiments. Other researchers have focused on the impact of temperature variations on protocol performance and designed testbeds that allow to control the on-board temperature of nodes [6, 47]. These testbeds have been used to study specific protocol problems [4, 48] and not to benchmark protocols under the same settings.

3 Competition: Overview

In answer to the increasing need for dependable low-power wireless systems, we organized the *EWSN 2016 dependability competition* to benchmark the performance of different protocols under the same settings and environmental conditions. In this section, we summarize the competition goals (Sect. 3.1), the metrics used to evaluate the contending protocols (Sect. 3.2), and derive from these the requirements of the necessary benchmarking infrastructure (Sect. 3.3).

3.1 Goals

Differently from previous evaluations carried out in the literature that focused on the goodness of a specific protocol in comparison to other solutions, our focal point is the creation of an unbiased setup where any protocol can be run under the same reproducible conditions.

Realistic scenario. Such an unbiased setup should be derived from a representative real-world wireless sensing application that requires dependable performance. In our case, we focus on an industrial control application in which a multi-hop wireless sensor network observes and reports events to a central unit. We also aim to reproduce environmental effects that traditionally challenge network performance in industrial real-world settings, and focus on the presence of other wireless devices crowding the RF spectrum.

Reproducibility. All protocols need to run under the same conditions. We hence focus on a benchmarking setup where nodes are deployed in a static temperature-controlled environment that is off-limits to people, in order to minimize changes in wireless propagation and connectivity between nodes. To generate repeatable interference patterns we use JamLab [5], a tool that is well-known to the community, and make sure that the generated jamming sequence starts synchronously with every experiment. We further disable all Wi-Fi access points in the surroundings of the benchmarking setup, and monitor that no other interference source is present in the competition area.

Hardware-neutral comparisons. The focus of the competition is on protocol performance. As the latter is

strongly dependent on the capabilities of the underlying hardware platform (e.g., processor speed, available memory, transceiver efficiency), we select one specific platform and use it to benchmark all competing systems. This allows hardware-neutral comparisons that maximize fairness. The employed hardware platform can be arbitrary, but its choice is practically driven by the number of protocol implementations available for that specific platform, which leads to the selection of an off-the-shelf device such as the TelosB mote.

No limitation on competing solutions. Besides selecting a platform that maximizes the number of protocols for which an implementation exists, it is also our goal to have no constraint on the type of protocol that is benchmarked. We indeed intend to find which protocols perform best in a given scenario, regardless of the employed strategy. Another important goal is to attract solutions developed by both academia and industry. To allow this, we cannot require the contestants to disclose their source code, but only to provide the final application firmware (e.g., as a `.hex` file).

Unobtrusive measurements. We aim to create a setup free of probe effects, i.e, that does not alter in any way the performance of a protocol. To this end, we need to disable serial communication and avoid the reliance on `printf` instructions, as they may alter the timing behavior of a protocol and break its functionality [36].

On-site protocol parametrization. As we are interested in the best performance that a protocol can reach in the specified settings, we let contestants inspect the benchmarking scenario in a two-days preparation phase during which they are allowed to optimize the parameters of their protocol. We hence need to graphically show to the contestants the protocol performance in real-time. After this preparatory phase, the contestants need to provide a final version of the protocol that will be used for the actual competition.

3.2 Evaluation Metrics

We focus on three dependability attributes that are highly relevant for low-power wireless protocols employed in safety critical settings: reliability, timeliness, and availability.

The *reliability* of a protocol is traditionally measured in terms of packet reception rate. As we intentionally generate interference and focus on an industrial control scenario where nodes observe and report events to a central device, we do not aim to measure the fraction of packets that are received or lost, but rather the number of events that are missed or incorrectly reported.

The *timeliness* attribute is captured as the end-to-end delay with which every event is communicated to the back-end infrastructure. In order to let contestants push the performance of their protocol to the limit and prevent excessive optimization for a specific metric, we do not specify a maximum delay by which an event needs to be reported.

The *availability* attribute can be expressed in terms of power efficiency, as wireless devices are typically battery-powered and their power consumption affects the overall system lifetime and hence its availability. Minimizing power consumption while maximizing reliability and timeliness is, however, a *catch-22* dilemma, as methods to increase reliability and timeliness such as retransmissions and higher duty cycle cause a drastic increase in the power consumption.

3.3 Requirements for the Benchmarking Infrastructure

In order to benchmark protocols according to these specifications, we need to build a facility that allows us to create the desired scenario and profile the desired metrics accurately while showing their evolution in real-time. This facility is essentially an augmented testbed that, besides common features such as easy reprogramming of nodes and persistent logging of serial output, should satisfy a number of technical and non-technical requirements that we summarize below.

Accurate power profiling. A first requirement for the benchmarking infrastructure is the ability to accurately measure the voltage and especially the current draw of *each* sensor node over a large range (from sleep currents of a few μA to active current draws up to tens mA). As software-based power estimation requires changes in the contestants firmware and is not highly accurate, the measurement needs to be carried out using external hardware. To properly capture short events such as radio channel switching, clear channel assessments, and processor sleeping times, the infrastructure needs to sample both current and voltage at a high frequency (56 kHz as in [36] can be taken as a reference).

Latency profiling. The benchmarking infrastructure also needs to provide means to measure end-to-end delays between nodes at microsecond-scale, given that tiny differences in the payload size or processor usage may result in differences in the order of a few μs. Another requirement is a common time reference over a large scale, as the infrastructure can span across different floors of a building and over large distances (e.g., when used in conjunction with IoT platforms equipped with long-range radios). Furthermore, the common time reference should be obtained without generating traffic in a frequency band that is used by the contestants.

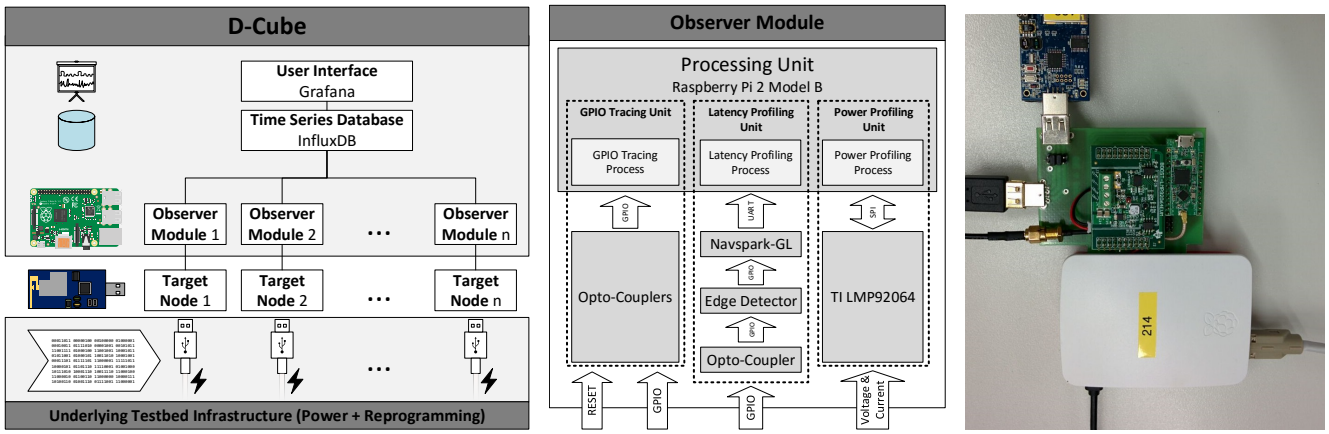
Event detection. The benchmarking infrastructure needs the ability to start and terminate an experiment automatically and read some of the GPIO pins of the target platform in order to associate logic-level changes to specific events.

Real-time visualization. In order to give rapid feedback about the performance of the different protocols to the contestants, the benchmarking infrastructure needs the ability to graphically summarize and depict the performance of a protocol w.r.t. each of the evaluation metrics in real-time.

Open-source design. Although the competition is a single event and focuses on a specific scenario, it is intended to be the first of a series, as well as an initial spark towards a comprehensive standardized benchmarking suite for low-power wireless protocols. Therefore, we aim to develop a generic reusable infrastructure and keep its design open-source, so that it can serve as a reference to the community.

Hardware agnostic. The developed infrastructure should be agnostic to the target hardware platform. Any IoT platform available on the market should ideally be pluggable in the benchmarking infrastructure with minimal effort.

Evolvability. As technology evolves continuously and components may need to be upgraded, it is important that the testbed is conceived as a modular design and that the software components are decoupled from the underlying hardware. This allows easier upgrades by only replacing obsolete components without affecting the rest of the system.



(a) D-Cube's general architecture (b) Structure of the observer module (c) Prototype of the observer module

Figure 1. D-Cube's general architecture (a) with sketch of the observer module structure (b) and prototype (c).

Affordability. When creating a feature-rich benchmarking infrastructure on a large-scale, costs may grow exponentially. In order to design a solution that can be reused by other research institutions and make it applicable on a large-scale, it is imperative to satisfy all the above requirements with minimal costs.

4 D-Cube: Design and Implementation

Existing testbeds infrastructures, unfortunately, do not satisfy the requirements outlined in Sect. 3.3. Common testbeds typically allow developers to record the serial output from each node and browse the logs (to extract the metrics of interest) only after an experiment has completed. To host a competition, we need instead an infrastructure that allows to log *timestamped events* at high speed, and that offers *real-time visualization* of custom performance metrics.

Furthermore, the required infrastructure also needs to accurately profile power and latency. Whilst a few feature-rich testbeds such as Flocklab embed these features, the high cost per node (FlockLab's observers cost approximately 1000 USD each [36]) makes it impossible to replicate these infrastructure on a large-scale. We hence need to find an affordable solution that offers comparable features to FlockLab, but shrinks down the costs by a factor of at least 10.

To address these problems, we design D-Cube[1], a tool that unobtrusively measures the three dependability metrics of interest with high accuracy, visualizes the results in real time, and shrinks the hardware costs down to 50 EUR per monitored node. In this section, we sketch D-Cube's architectural components (Sect. 4.1), the design and implementation choices allowing to extract the desired dependability metrics (Sect. 4.2 – 4.4), and outline how D-Cube's modular design allows to limit the hardware costs (Sect. 4.5).

4.1 Architecture

Fig. 1(a) summarizes D-Cube's architecture. D-Cube sits on top of an existing testbed infrastructure that provides power and easy reprogramming to the wireless sensor nodes used as target to test protocol performance.

Target nodes. The wireless sensor nodes running the code of the contestants are the *target nodes*. D-Cube is completely agnostic to the hardware platform chosen as a target node and to the underlying testbed infrastructure. Besides powering and reprogramming the nodes, the latter ideally allows to capture log messages for debugging purposes and to disable the UART interface of all target nodes to ensure fair comparisons[2]. For the competition, we use Advanticsys MTM-CM5000-MSP nodes (TelosB replicas) as target. These are connected to an existing testbed infrastructure powering and programming the nodes via USB active cables from a central control station.

Observer modules. Each target node is associated to one *observer module* that carries out latency profiling, power profiling, and GPIO tracing. The observer module consists of several components, as shown in Fig. 1(b). At the heart of an observer module lies the processing unit, which is used to schedule the measurements, aggregate data, and forward the collected information to a database for persistent storage. The observer module does not interact directly with its associated target node: it only monitors the GPIO and reset pins, and passively measures power consumption. In our prototype implementation, we employ a Raspberry Pi 2 (RPi2), model B, and use its on-board 100 Mb/s Ethernet interface to connect to the network for NTP synchronization and to access the time series database server (see Fig. 1(a)).

To accurately *profile latency*, the observer embeds a GPS unit to generate precise timestamps for externally triggered events. In our prototype implementation, this task is carried out by a Navspark-GL device [1]. The latter has an external trigger input that supports accurate timestamping of either rising or falling edges. To detect both of them, we use the Navspark-GL in conjunction with an edge detector.

To accurately *profile power*, the observer module needs to capture voltage and current at the same time. The capturing unit of the observer module consists of an ADC connected via SPI to the processing unit. In our prototype im-

[1] D-Cube (D^3) takes his name from the three dependability metrics that are observed: reliability, timeliness, and availability.

[2] These are typical features of almost any testbed infrastructure. In case these are not available, the observer module's processing unit of D-Cube can be used to accomplish these extra-tasks.

plementation, we employ the Texas Instruments LMP92064, a simultaneous-sampling 125 kSps 12-bit current and voltage monitor. A real-time process on the observer's processing unit reads the ADC values via SPI and writes them into a FIFO queue for further processing by a user space task.
The real-time task also takes care of *GPIO tracing* by recording the state of the GPIO pins from the target node. The GPIO tracing unit is used to also determine the start and termination of a measurement by monitoring the reset pin of the target node. The use of optocouplers as isolation layer between the GPIO pins of the target node and the ones of the observer module ensures unbiased energy measurements.
A key advantage of the observer design is that measurement units are kept independent from each other. This implies that modules can be disconnected (e.g., the GPS unit for latency profiling can be omitted if a target node does not generate or capture any event) and that the additional measurement units (e.g., to sample temperature) can be added without affecting the observer software. Each measurement unit, indeed, can be split into individual user space processes.

Time series database. The data collected by all observers is persistently stored on a time series database. Unlike traditional SQL databases, the latter does not have tables and keys, but uses series consisting of measurements. These are made up of fields containing the measured values (e.g., ADC voltage) and tags marking a given measurement as belonging to a specific category (hence enhancing and simplifying statistical analysis). As GPS modules return timestamps with nanoseconds precision, the database requires the ability of handling nanosecond time series. In our prototype implementation, we employ *InfluxDB*, an open-source time series database that is optimized for fast, high-availability storage and retrieval of time series data.

User interface. In order to support developers with real-time information about the performance of their protocols, a user interface extracts information from the database and displays it graphically. Detailed information can be displayed for each target node and statistics can be computed to summarize the performance of the running system. Users can autonomously combine and visualize the measurements of all sensors with a specific characteristic. In our prototype implementation, we employ *Grafana*, an open-source, general purpose dashboard and graph composer that is built on top of JavaScript and runs as a Web application. Grafana allows client-side rendering and embeds a full suite for user management including Lightweight Directory Access Protocol (LDAP) integration. Among others, it also acts as a proxy preventing unauthorized access to the underlying database.

4.2 Profiling Power Consumption

In order to profile the power consumption of a target node at a speed sufficiently high to detect short ephemeral radio events such as clear channel assessments and switches between low-power and active CPU mode, we use a high-precision dual channel ADC that allows simultaneous sampling of voltage and current. For the current channel an amplifier is required, as the voltage drop over a typical current shunt ($\leq 1\ \Omega$) is very small (≤ 100 mV). For this task, current sense amplifiers – specialized amplifiers with either a fixed or variable gain – are typically used. In our prototype, we select an off-the-shelf component, namely the Texas Instruments LMP92064EVM. This consists of a 12-bit ADC with one voltage and one current channel with integrated current sense amplifier at a fixed gain of 25. This component also embeds two level translators and isolators, as the integrated circuit operates at 5V, whilst the processing unit (RPi2) has 3.3V logic levels. We configured the voltage divider and the current shunt to values that are suitable for our target nodes. We limit the maximum current to 150mA and the maximum voltage to 10V, which allows us to measure tiny differences in voltage and current. We sample voltage and current at high frequency (62.5 kHz) using the RPi2 processing unit. In order to perform the measurement tasks at a constant period, we employ the real-time patch-set for the Linux Kernel, which allows to isolate a core solely for the data acquisition.

4.3 Profiling End-to-End Latency

In order to profile latency, a highly accurate and synchronized timestamp is required. While there are many options to achieve this, only a few are viable for large scale installations. A conventional synchronization pulse transmitted over a network of coax cables is impractical and expensive, as it requires calibration. Also the use of a radio module and a PLL implemented on an FPGA as in [37] is not a viable option. As D-Cube is agnostic to the employed target node by design, it must not employ a frequency band that may be used by the target nodes for their communications, as this would bias the benchmarking results. In D-Cube we hence make use of GPS to measure the timing of events, as little to no infrastructure is required to obtain sub-μs synchronization.

In our prototype, we select the Navspark-GL, an off-the-shelf component priced 25 USD that provides an external trigger input supporting timestamping. As the Navspark-GL can only create timestamps for rising *or* falling edges, we run it in conjunction with an edge detector (consisting of an XOR gate and an RC low-pass filter) that creates a short pulse whenever the input signal changes. An interrupt service routine on the Leon3 processor of the Navspark-GL checks for the level of the GPIO pin before the edge detector and writes the logical level and timestamp into a FIFO queue that is transmitted via UART to the RPi2.

A limitation of using GPS is that it requires a clear view of the sky to operate properly. This is typically possible also indoors, if the GPS module is placed in proximity of windows (this was the case in our competition setup, where the nodes generating and capturing events were close to windows and could receive proper GPS signal). Alternatively, one can deploy GPS re-radiators as done in the IoT-LAB testbed [9].

4.4 Capturing GPIO Events

In our implementation, for convenience, the observer module checks and resets the edge detection flags of the RPi2 at the same speed of the real-time process used to carry out the power profiling, i.e., it samples GPIO events at a frequency of 62.5 kHz (one sample every 16μs). This is more than sufficient to detect the events in our envisioned scenario at high speed. If necessary, the sampling rate can be further increased by two orders of magnitude by using one dedicated CPU core for this task.

4.5 Minimizing the Observer Costs

We built prototypes of the observer module using off-the-shelf components available at our institution or readily purchasable from standard retailers. Overall, the price for a single observer module amounts to 135 EUR. While this amount is already quite low (at least if compared to the 1000 USD of FlockLab's observers [36]), the price can be pushed down even further. Out of the 135 EUR, indeed, 65 are due to the LMP92064EVM evaluation module provided by Texas Instruments. The latter costs ten times more than its corresponding integrated circuit (IC) alone. When embedding the LMP92064 IC with level translator and edge detector circuit in a 2-layer PCB, the price of the observer module can be pushed down significantly, with the additional benefit of having a layer that shields the SPI traces (this would allow an even faster sampling rate of the ADC). The costs of D-Cube's observer module can further be reduced by using a cheaper processing unit such as an Allwinner H3 (Orange Pi One or NanoPi Neo) board. The latter costs approximately 10 EUR and has a comparable performance to the RPi2 that we employed. Altogether, these improvements would minimize the hardware costs down to roughly 50 EUR.

It is worth mentioning that all the individual components that we used to build D-Cube will be published open-source on-line[3] and will hence be available to the research community. We expect other contributors to make modifications and improve D-Cube towards a standardized benchmarking suite that increases the rigor of experimental validation.

5 D-Cube Validation

In this section, we experimentally evaluate how accurately D-Cube can measure the three evaluation metrics of interest and its suitability as benchmarking infrastructure for the competition. In particular, we evaluate the accuracy of D-Cube's observer modules when profiling power and latency in Sect. 5.1 and 5.2, respectively. We then evaluate the responsiveness of the real-time visualization component of D-Cube in Sect. 5.3.

5.1 Profiling End-to-End Latency

To evaluate how accurately D-Cube can measure end-to-end latency over large distances, we connect the input trigger of the Navspark-GL of different observer modules to the same trigger event and measure the timing deviation.

Experimental setup. We set up an experiment in which the input trigger pin of the Navspark-GL of three observer modules is connected in parallel to the same output of a STM32F103 microcontroller. We then configure the latter to produce a PWM signal with a frequency of 1 Hz and let each Navspark exploit the in-built timestamping function to print out the full timestamp in nanoseconds over the UART. We log the output from each observer module until several thousands of "rising-edge" timestamped events are collected, and repeat the experiment multiple times.

Accuracy of synchronization. Fig. 2 shows the pairwise error of the three observer modules. As expected, the time synchronization of the GPS modules is at a sub-μs scale. Approximately 95% of the timestamped events differ by only $\pm$250ns (shown as gray overlay in the figure).

[3] http://www.iti.tugraz.at/D-Cube

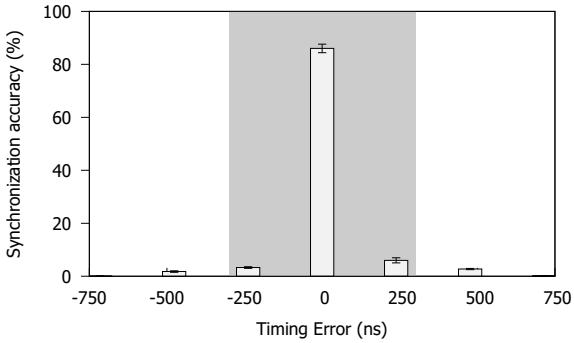


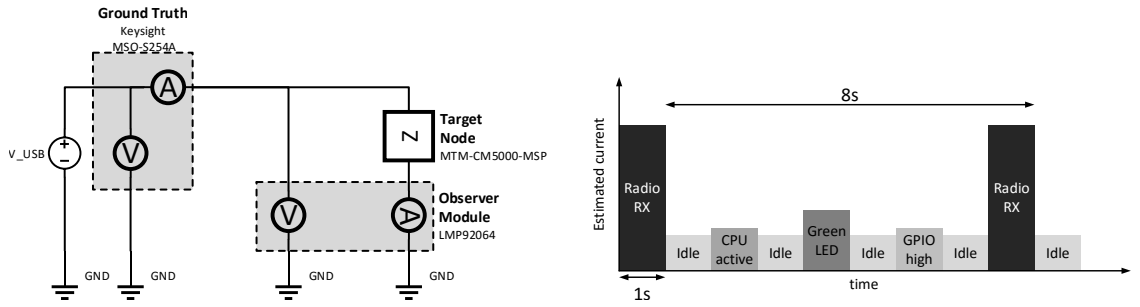
Figure 2. Accuracy of D-Cube's observer modules when profiling latency. GPS allows to keep time synchronization across different modules at a sub-μs scale.

5.2 Profiling Power Consumption

We measure how accurately D-Cube's observer modules can profile power consumption by comparing its measurements with the ones obtained using professional equipment, as well as evaluate the achievable sampling rate.

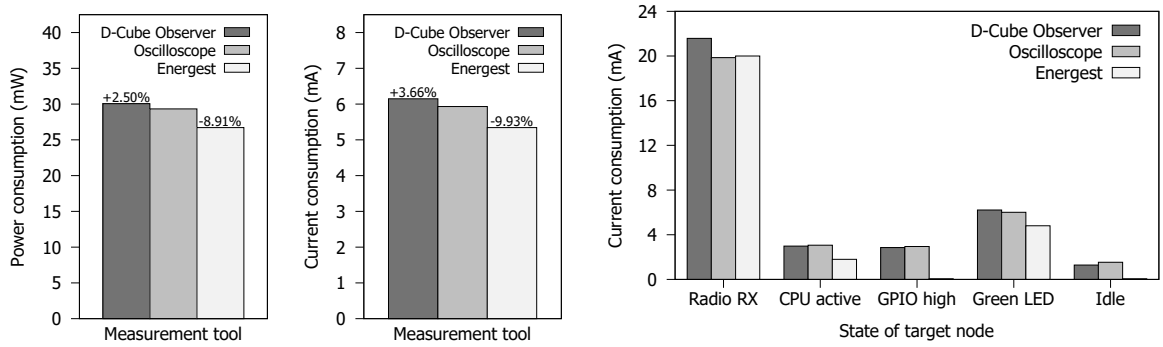
Experimental setup. We employ an Advanticsys MTM-CM5000-MSP node (TelosB replica) as target and measure its power consumption using D-Cube's observer modules and a Keysight MSO-S 254A mixed signal oscilloscope as ground truth. We create an experimental setup as sketched in Fig. 3(a), where using the oscilloscope we measure the current of the target node with a N2821A 3MHz/50uA high sensitivity AC/DC current probe on one channel, and the voltage using a regular probe on another channel. At the same time, we connect the target node to D-Cube's observer module and sample the current and voltage at a frequency of 62.5 kHz. We then create a test application using Contiki [14] that turns on and off individual components of the target node for a pre-defined period of time (e.g., LED, GPIO pin, radio in receiving mode, and CPU in active mode). Fig. 3(b) shows the employed pattern: each component is turned on for one second and an idle state (during which the CPU runs in low-power mode) interleaves each change. The pattern repeats periodically over time. To be able to precisely measure the power consumption of the node in the different states of the test pattern, we disable the FTDI, as its power draw was found not to be constant. In addition to this setup, we repeat the experiment enabling the FTDI using Energest (Contiki's software-based energy estimation [15]) to print the estimated power consumption.

Accuracy of power measurements. Fig. 4 shows the power and current consumption measured using the different tools. Compared to our reference oscilloscope, Energest underestimates the current consumption by about 9.9%, whilst D-Cube's observers overestimate it by 3.6%. Please notice that all currents measured by oscilloscope and observer module have been offset by 1.6 mA to compensate the constant consumption of the on-board optocouplers and provide a more fair comparison with Energest. The voltage measured by the observer module differs from the one obtained with the oscilloscope by only -0.84%. The voltage is also sufficiently stable to observe a drop of 50mV when the radio



(a) Experimental setup (b) Pattern of the test application

Figure 3. Experimental setup to measure the accuracy of D-Cube's power profiling (a) and test pattern used on target node when measuring current and voltage (b).



(a) Average power and current consumption (b) Breakdown of current consumption per state

Figure 4. Power and current consumption measured using different tools. (a) shows the average consumption of the test application, whilst (b) shows a breakdown of current consumption when only selected components are active.

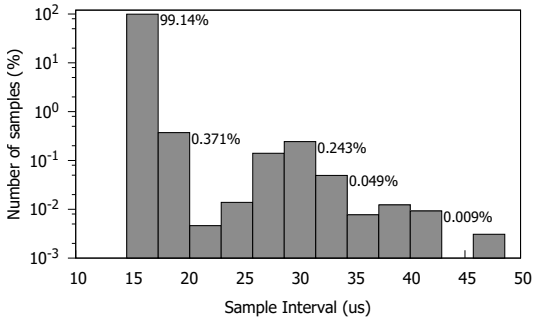


Figure 5. Stability of the sampling rate using D-Cube's observer modules. Please notice the logarithmic y-scale.

device is turned on and variations below 10mV in the remaining states. The resulting power consumption shown in Fig. 4 using Energest is computed by multiplying the estimated current consumption by 5V (constant USB voltage). Fig. 4(b) shows a breakdown of the power consumption in each state. The most significant difference can be observed when the GPIO pin is active, as the current implementation of Energest is unable to account for its consumption.

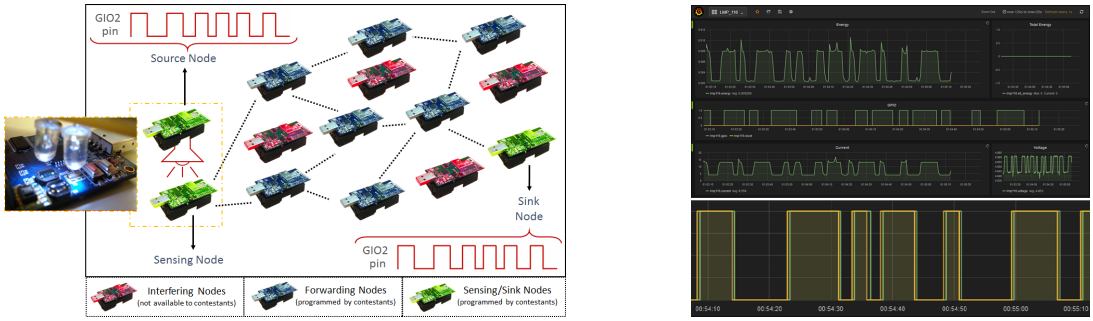
Sampling frequency. Using the same setup described above, we also evaluate the stability of the sampling rate of the observer modules. We use the Linux timestamps recorded by the RPi2 at nanosecond precision whenever a new sample is retrieved from the ADC, and compute the delay between consecutive samples. For this experiment we configure a sampling rate of 62.5 kHz and we hence expect two samples to be spaced by 16μs. Fig. 5 shows the stability of the sampling rate: in 99.14% of the cases, two consecutive ADC samples are indeed spaced by 16 μs. This means that our prototype with a single sided PCB and no shielding on the SPI lines is indeed capable of accurately sampling the power consumption at a frequency of up to 62.5 kHz (voltage and current are sampled simultaneously).

5.3 Timeliness of the GUI

The responsiveness of D-Cube's user interface displaying the performance of the system being tested is limited by two factors: (i) the speed of the selected time series database in providing queries, and (ii) the refresh rate of the GUI. In our case, InfluxDB answers queries in real-time with every data point being indexed as it comes in and immediately available in less than 100ms. The real-time visualization in Grafana, instead, is implemented by default using a periodic refresh interval of one second, i.e., the software automatically reloads the data from InfluxDB every second.

6 Competition: Results and Lessons Learned

We use D-Cube to setup the desired benchmarking infrastructure and run the EWSN 2016 dependability competition. After describing the competition setup (Sect. 6.1), we list the competing protocols (Sect. 6.2) study their performance (Sect. 6.3), and discuss the lessons learned in Sect. 6.4.



(a) Competition setup

(b) D-Cube's user interface

Figure 6. Competition setup (a) and user interface shown to the contestants (b).

6.1 Setup

We host the competition in a building of Graz University of Technology, where 45 wireless sensor nodes are deployed over an area of approximately 150 m^2. We select Advanticsys MTM-5000 sensor nodes (TelosB replicas) as target and select a portion of them to generate RF interference on multiple channels using JamLab [5]. All remaining nodes are monitored by D-Cube's observers that profile their power consumption over time.

One of the target nodes is placed in proximity of a light source and monitors its brightness using the embedded light sensors. Any sudden variation in the lighting condition needs to be promptly communicated to a sink node that will trigger one of its I/O pins accordingly. The sink node is not in the communication range of the sensing node, and additional forwarding nodes are available in the surroundings to reach the sink in a multi-hop fashion (at least three hops are necessary). The light source is a target node connected to a bright LED that is turned off and on according to a secret schedule (the same for all contestants). Node identities and positions are not disclosed to avoid engineered solutions, but the unique ID of the nodes can be read from flash. Fig. 6(a) sketches the competition setup at a glance.

Evaluation metrics. Employing D-Cube, the first dependability metric, *timeliness*, is measured as the delay with which each change in the lighting condition is reported to the sink. To this end, we use D-Cube's observer modules to timestamp the light changes on the node controlling the light source and the changes in the GIO2 pin of the sink node. The *availability* is inversely proportional to the power consumption measured across the network by all observer modules. Finally, the *reliability* is computed as the percentage of light changes that are correctly reported to the sink.

User interface. During a two-days preparation phase, the developers of the competing protocols have the possibility to monitor the performance of their protocols and optimize their parameters by connecting remotely to D-Cube's user interface. Contestants can see each other's performance live, which – as we will see in the next sections – helps significantly in pushing the dependability of their solutions to the limit. Fig. 6(b) shows a screenshot of the user interface displaying the performance of a protocol in real-time.

Final benchmarking. After the two preparation days, all contestants are asked to provide a final firmware to be used for the benchmarking on the competition day. The latter consists in a series of experiments of 35 minutes each, during which JamLab increasingly generates interference across the whole 2.4 GHz band, emulating the presence of several coexisting Wi-Fi networks. All Wi-Fi access points in the area are disabled and we make sure using an RF scanner that no other interfering device is present in the area.

6.2 Benchmarked Protocols

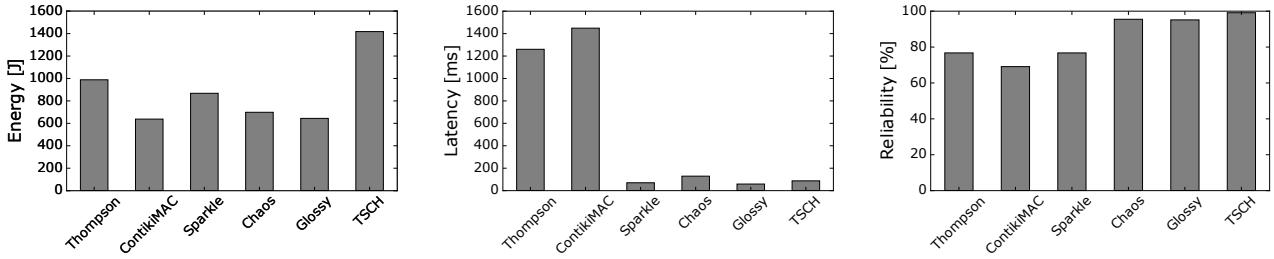
Using this setup, we benchmark the performance of the protocols developed by 11 teams that answered to an open call for competitors. Their solutions range from adaptive and time-slotted frequency-hopping to multi-modal routing and flooding. Due to space constraints we focus our study on six representative protocols, chosen among the ones having a detailed technical description available and performing best during the competition.

Enhanced ContikiMAC. ContikiMAC [13] is Contiki's default duty-cycling MAC protocol. We benchmark a version of ContikiMAC that uses a sophisticated CCA mechanism reducing the number of false wake-ups and maximizes energy-efficiency [32]. Differently from enhanced ContikiMAC versions such as MiCMAC [44], this version runs on a single channel.

Thompson-sampling based channel selection. This protocol implements in Contiki a solution that dynamically selects the best out of three channels. The channel to be used is chosen by solving a multi-arm bandit problem using Thompson sampling, which allows to minimize the number of samples necessary to obtain a good estimation [40].

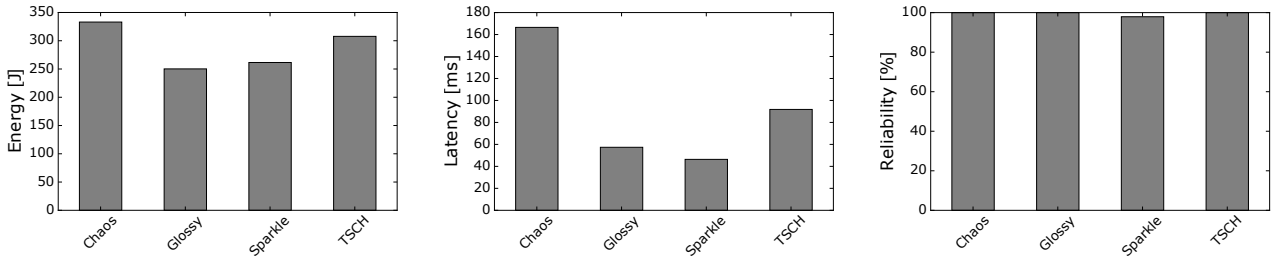
Glossy. Glossy [21] has been an influential flooding architecture for wireless sensor networks that exploits constructive interference of IEEE 802.15.4 symbols for fast network flooding and implicit time synchronization. The benchmarked protocol is an extended version of Glossy with channel-hopping, where the channel used for each communication slot is derived from the relay counter and the packet sequence number [49].

Chaos. Chaos [35] is a primitive for all-to-all data sharing of different packets. Inspired from Glossy's one-to-all flooding, Chaos parallelizes collection, processing, and dissemination inside a network by building on synchronous transmissions and user-defined merge operators. The benchmarked protocol is an extended version of Chaos that adds frequency-hopping and blacklisting of poor channels [2].



(a) Energy consumption (b) End-to-end delay (c) Reliability

Figure 7. Performance of the competing protocols during the competition day.



(a) Energy consumption (b) End-to-end delay (c) Reliability

Figure 8. Best performance obtained by the competing protocols during the preparation days.

Sparkle. Built upon Glossy, Sparkle [54] builds a multi-loop control network that controls each end-to-end flow based on run-time feedback. It employs capture effect to find a number of reliable paths between the source and the destination and activate nodes on one or more of these paths [53].

Time-slotted Channel Hopping. We also benchmark the OpenWSN open-source implementation of TSCH/6TiSCH [26], which employs time-slotted channel hopping at a medium access control level [51], and RPL with controlled flooding at the routing level.

6.3 Results

Fig. 7 shows the reliability, timeliness, and availability of the benchmarked protocols during the competition day.

Flooding pays off. A clear result is that solutions employing flooding are the ones that perform best across all three metrics. Whilst it is expected that flooding minimizes the end-to-end latency, it is usually naïvely assumed that this comes at a significantly higher energy-expenditure across the network, due to the higher number of transmissions and increased radio activity. The competition results show instead that flooding protocols have a comparable energy consumption to standard routing approaches.

Hopping is a necessity. While none of the competing protocols could achieve a perfect 100% end-to-end reliability, there is a clear cut between solutions relying on channel hopping across the whole frequency band and other solutions. Out of the six protocols that we benchmark, indeed, Sparkle, Chaos, Glossy, and TSCH achieve a significantly higher reliability and timeliness. ContikiMAC, which operates on a single-channel, exhibits the lowest reliability (69.1%) and highest average end-to-end delay (1.4 seconds), probably due to the high number of failed attempts to access the medium. The Thompson-sampling based channel selection, which uses a pool of three channels, improves reliability by 7% and reduces end-to-end delay by 200 ms. The other four protocols, which hop across the *whole* frequency band, instead, are able to push the reliability up to 99.2%.

New-generation protocols are reliable and timely. Another clear result is that new-generation protocols combining state-of-the-art techniques such as constructive interference, flooding, and (time-slotted) frequency-hopping, can sustain a reliability above 95% even in the presence of high interference and push down the end-to-end delay well below 150 ms for a network with at least 3 hops – a performance that approaches the requirements of safety-critical applications.

Over-optimizing may be fatal. Choosing how much energy to sacrifice for privileging timeliness and reliability is a well-known catch-22 dilemma. The contestants had to face this problem, which was exacerbated by the fact that they were aware of the performance of each other and strove for victory. This led to a situation in which contestants pushed the performance of their protocol to the edge, trying to achieve even the smallest gain that would allow them to beat another solution. The competition has clearly shown, however, that a minimal improvement in a metric can worsen the performance in another metric by almost a factor of two. An example is shown by comparing Fig. 7 and 8: TSCH exhibited a comparable performance to the other top three protocols during the preparation days, and tried to aggressively optimize its weakest point (timeliness). This resulted, however, during the competition day, in a drastic increase of energy consumption in comparison to all other protocols.

6.4 Lessons Learned

We summarize next the lessons learned during the preparation and execution of the competition.

Competitions trigger major advances. The best-performing solutions in the competition correctly captured more than 95% of the events with an end-to-end delay be-

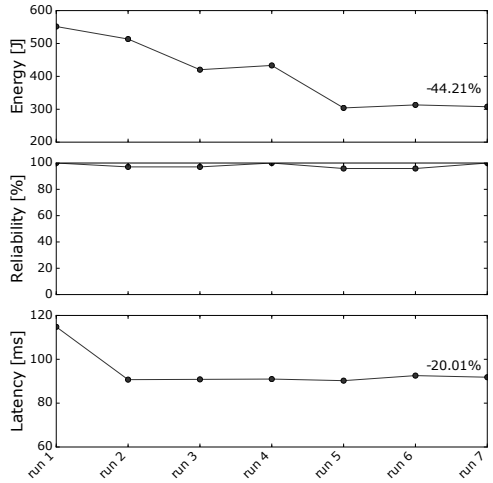


Figure 9. By only optimizing the parameters, contestants could significantly improve the dependability of their protocols. In this figure, the improvements of TSCH across multiple runs on the preparation day are shown.

low 75 ms within a network of at least three hops. Given the extreme environmental conditions that were created (severe radio interference across all frequency channels), this is a remarkable result that approaches the requirements of safety-critical applications. Undoubtedly, the essence of the competition contributed to this result, with people competing until late at night striving for victory.

Live feedback to participants. One of the aspects that contributed to the success of the competition was undoubtedly also the ability of D-Cube to graphically visualize the performance of protocols in real-time. We expect this feature to become a standard feature in public testbeds and future benchmarking suites.

The importance of proper parametrization. As mentioned in Sect. 1, a common practice when benchmarking different protocols is to rely on the default settings. The competition has shown that by only optimizing protocol parameters to the scenario at hand, the performance of some systems could significantly be improved without affecting any of the three dependability metrics. An example is shown in Fig. 9, which depicts the performance of TSCH during the competition day and the evolution of the three dependability metrics over consecutive experiments. In only seven test runs, the TSCH developers could decrease latency by 20% and energy-consumption by 45%, without affecting the end-to-end reliability of their system.

Impact of the operating system. Another important observation that stems from the competition is that the performance of a protocol competing in these settings is typically only minimally affected by the underlying operating system. Nevertheless, we have also learned that it is very important to benchmark the default settings of an operating system and look for any large discrepancy w.r.t. the expected values. While comparing the performance of the same application using the two most popular operating systems for IoT devices, Contiki and TinyOS, we indeed observed a large difference in the energy consumption. As shown in Fig. 10,

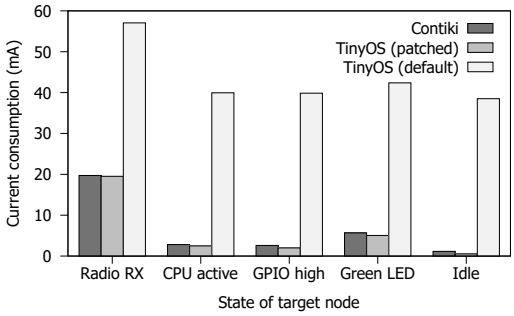


Figure 10. Current consumption on different operating systems. Due to misconfigured GPIO pins, large differences were observed between TinyOS and Contiki.

TinyOS exhibited a much higher energy consumption than Contiki when running the same application ($\approx$ 37mA more). The reason for such discrepancy turned out to be due to the default configuration of the GPIO pins. The latest version of TinyOS, indeed, configured by default the GPIO pins not as inputs (as the comment above the initialization section of the code suggested) but rather as outputs. Once the problem was fixed, the performance of the two operating systems was comparable, with Contiki consuming on average about only 0.5 mA more than TinyOS (see Fig. 10).

7 Conclusions and Future Work

In this paper we present the execution and results of a competition aimed at benchmarking the end-to-end dependability of state-of-the-art low-power wireless protocols under the same settings and environmental conditions. We focus on an exemplary industrial control scenario where several wireless devices crowd the RF spectrum and evaluate the reliability, timeliness, and energy-efficiency of the competing protocols. To create the benchmarking infrastructure, we have designed and implemented D-Cube, a tool that allows us to accurately measure end-to-end delays and power consumption with minimal hardware costs. Future work includes a complete automation of the benchmarking procedure with the ability to allow people to remotely contend without the need of physically hosting a competition.

Acknowledgments

The authors would like to thank Engelbert Meissl for his help while setting up the competition infrastructure. This work was performed within the LEAD-Project “Dependable Internet of Things in Adverse Environments”, funded by Graz University of Technology.

8 References

[1] *NavSpark User Guide, Rev. 0.9*, 2016.

[2] B. Al Nahas and O. Landsiedel. Towards low-latency, low-power wireless networking under interference. In *Proc. of the* 13^{th} *EWSN Conf., competition session*, 2016.

[3] N. Baccour, A. Koubâa, L. Mottola, H. Youssef, M. A. Zúñiga, C. A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: a survey. *ACM (TOSN)*, 8(4), 2012.

[4] C. A. Boano, K. Römer, and N. Tsiftes. Mitigating the adverse effects of temperature on low-power wireless protocols. In *Proc. of the* 11^{th} *IEEE MASS Conf.*, 2014.

[5] C. A. Boano, T. Voigt, C. Noda, K. Römer, and M. A. Zúñiga. JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation. In *Proc. of the* 10*th IPSN Conf.*, 2011.

[6] C. A. Boano, M. A. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer. TempLab: A testbed infrastructure to study the impact of temperature on wireless sensor networks. In *Proc. of IPSN*, 2014.

[7] C. A. Boano, M. A. Zúñiga, K. Römer, and T. Voigt. JAG: Reliable and predictable wireless agreement under external radio interference. In *Proc. of the* 33*rd RTSS Conf.*, 2012.

[8] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester. The w-ilab.t testbed. In *Proc. of the* 6*th TridentCom*, 2010.

[9] C. Adjih et al. FIT IoT-LAB: A large scale open experimental IoT testbed. In *Proc. of the* 2*nd WF-IoT*, 2015.

[10] M. Cattani, A. Loukas, M. Zimmerling, M. Zuniga, and K. Langendoen. Staffetta: Smart duty-cycling for opportunistic data collection. In *Proc. of the* 14*th SenSys Conf.*, 2016.

[11] Defense Advanced Research Projects Agency. DARPA Urban Challenge. `http://archive.darpa.mil/grandchallenge/`. Last visited: 27.06.2016.

[12] M. Doddavenkatappa, M. Chan, and A. Ananda. Indriya: A low-cost, 3D wireless sensor network testbed. In *Proc. of TridentCom*, 2011.

[13] A. Dunkels. The ContikiMAC radio duty cycling protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[14] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proc. of the* 1*st EmNetS Workshop*, 2004.

[15] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based online energy estimation for sensor nodes. In *Proc. of the* 4*th EmNetS Workshop*, 2007.

[16] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne. Orchestra: Robust mesh networks through autonomously scheduled TSCH. In *Proc. of the* 13*th SenSys Conf.*, 2015.

[17] S. Duquennoy, F. Österlind, and A. Dunkels. Lossy links, low power, high throughput. In *Proc. of the* 9*th SenSys Conf.*, 2011.

[18] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proc. of the* 8*th SenSys Conf.*, 2010.

[19] E. Ertin, A. Arora, R. Ramnath, M. Sridharan, and V. Kulathumani. Kansei: A testbed for sensing at scale. In *Proc. of the* 5*th IPSN*, 2006.

[20] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proc. of the* 10*th SenSys Conf.*, 2012.

[21] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proc. of the* 10*th IPSN Conf.*, 2011.

[22] K. Genter, T. Laue, and P. Stone. Benchmarking robot cooperation without pre-coordination in the robocup standard platform league drop-in player competition. In *Proc. of the IROS Conf.*, 2015.

[23] J. V.-V. Gerwen, S. Bouckaert, I. Moerman, and P. Demeester. Exploiting low-cost directional antennas in 2.4 GHz IEEE 802.15.4 wireless sensor networks. In *Proc. of the* 5*th SENSORCOMM*, 2011.

[24] J. V.-V. Gerwen, E. D. Poorter, B. Latré, I. Moerman, and P. Demeester. Real-life performance of protocol combinations for wireless sensor networks. In *Proc. of the SUTC Conf.*, 2010.

[25] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proc. of the* 7*th SenSys Conf.*, 2009.

[26] P. H. Gomes, T. Watteyne, P. Gosh, and B. Krishnamachari. Reliability through timeslotted channel hopping and flooding-based routing. In *Proc. of the* 13*th EWSN Conf., competition session*, 2016.

[27] 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* 2*nd REALMAN Workshop*, 2006.

[28] I. Haratcherev, G. Halkes, T. Parker, O. Visser, and K. Langendoen. PowerBench: A scalable testbed infrastructure for benchmarking power consumption. In *Proc. of the* 1*st IWSNE Workshop*, 2008.

[29] M. Hempstead, M. Welsh, and D. Brooks. Tinybench: The case for a standardized benchmark suite for TinyOS based wireless sensor network devices. In *Proc. of the* 29*th LCN Conf., poster session*, 2004.

[30] I. Galpin et al. SensorBench: Benchmarking approaches to processing wireless sensor network data. In *Proc. of the* 26*th SSDBM*, 2014.

[31] T. Kim, J. Kim, S. Lee, I. Ahn, M. Song, and K. Won. An automatic protocol verification framework for the development of wireless sensor networks. In *Proc. of the* 4*th TridentCom Conf.*, 2008.

[32] A. King, J. Hadley, and U. Roedig. Contikimac with differentiating clear channel assessment. In *Proc. of the* 13*th EWSN Conf., competition session*, 2016.

[33] L. Nazhandali et al. SenseBench: Toward an accurate evaluation of sensor network processors. In *Proc. of the IISWC Symposium*, 2005.

[34] R. Lajara, J. Pelegrí-Sebastiá, and J. J. P. Solano. Power consumption analysis of operating systems for wireless sensor networks. *Sensors*, 10(6), 2010.

[35] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *Proc. of the* 11*th SenSys Conf.*, 2013.

[36] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proc. of the IPSN*, 2013.

[37] R. Lim, B. Maag, B. Dissler, J. Beutel, and L. Thiele. TraceLab: A testbed for fine-grained tracing of time sensitive behavior in wireless sensor networks. In *Proc. of the* 10*th SenseApp Workshop*, 2015.

[38] Q. Luo, H. Wu, W. Xue, and B. He. Benchmarking in-network sensor query processing. Technical Report HKUST-CS05-09, The Hong Kong University of Science and Technology, 2005.

[39] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, S. Sen, and V. Handziski. Microsoft indoor localization competition: Experiences and lessons learned. *GetMobile*, 18(4), 2014.

[40] A. Maskooki, V. Toldov, L. Clavier, V. Loscrì, and N. Mitton. Channel exploration/exploitation based on a thompson sampling approach in a radio cognitive environment. In *Proc. of the* 13*th EWSN Conf., competition session*, 2016.

[41] M. Mohammad, X. Guo, and M. C. Chan. Oppcast: Exploiting spatial and channel diversity for robust data collection in urban environments. In *Proc. of the* 15*th IPSN Conf.*, 2016.

[42] S. Mysore, B. Agrawal, F. T. Chong, and T. Sherwood. Exploring the processor and ISA design for wireless sensor network applications. In *Proc. of the* 21*st International Conference on VLSI Design*, 2008.

[43] S. Nabar, A. Banerjee, S. K. Gupta, and R. Poovendran. Evaluation of body sensor network platforms: A design space and benchmarking analysis. In *Proc. of the Wireless Health Conference*, 2010.

[44] B. A. Nahas, S. Duquennoy, V. Iyer, and T. Voigt. Low-Power Listening Goes Multi-Channel. In *Proc. of the* 10*th IEEE DCOSS*, 2014.

[45] F. Österlind, L. Mottola, T. Voigt, N. Tsiftes, and A. Dunkels. Strawman: Resolving collisions in bursty low-power wireless networks. In *Proc. of the* 11*th IPSN Conf.*, 2012.

[46] S. Duquennoy et al. A benchmark for low-power wireless networking. In *Proc. of the* 14*th SenSys Conf., poster session*, 2016.

[47] F. Schmidt, M. Ceriotti, N. Hauser, and K. Wehrle. Hotbox: Testing temperature effects in sensor networks. Technical Report AIB-2014-14, RWTH Aachen, Germany, 2014.

[48] F. Schmidt, M. Ceriotti, N. Hauser, and K. Wehrle. If you can't take the heat: Temperature effects on low-power wireless networks and how to mitigate them. In *Proc. of the* 12*th EWSN Conf.*, 2015.

[49] P. Sommer and Y.-A. Pignolet. Dependable network flooding using glossy with channel-hopping. In *Proc. of the* 13*th EWSN Conf., competition session*, 2016.

[50] M. K. Watfa and M. Moubarak. A benchmarking tool for wireless sensor network embedded operating systems. *Journal of Networks*, 9(8), 2014.

[51] T. Watteyne, S. Lanzisera, A. Mehta, and K. S. Pister. Mitigating multipath fading through channel hopping in wireless sensor networks. In *Proc. of the IEEE ICC Conf.*, 2010.

[52] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: a wireless sensor network testbed. In *Proc. of the* 4*th IPSN*, 2005.

[53] D. Yuan and M. Hollick. Sparkle: Energy efficient, reliable, ultra-low latency communication in wireless control networks. In *Proc. of the* 13*th EWSN Conf., competition session*, 2016.

[54] D. Yuan, M. Riecker, and M. Hollick. Making 'glossy' networks sparkle: Exploiting concurrent transmissions for energy efficient, reliable, ultra-low latency communication in wireless control networks. In *Proc. of the* 11*st EWSN Conf.*, 2014.