Journal of the Indian Institute of Science
A Multidisciplinary Reviews Journal
ISSN: 0970-4140 Coden-JIISAD

Supporting Cyber-Physical Systems with Wireless Sensor Networks: An Outlook of Software and Services

Prasant Misra[1], Luca Mottola[1,2], Shahid Raza[1], Simon Duquennoy[1], Nicolas Tsiftes[1], Joel Höglund[1] and Thiemo Voigt[1,3]

Abstract | Sensing, communication, computation and control technologies are the essential building blocks of a cyber-physical system (CPS). Wireless sensor networks (WSNs) are a way to support CPS as they provide fine-grained spatial-temporal sensing, communication and computation at a low premium of cost and power. In this article, we explore the fundamental concepts guiding the design and implementation of WSNs. We report the latest developments in WSN software and services for meeting existing requirements and newer demands; particularly in the areas of: operating system, simulator and emulator, programming abstraction, virtualization, IP-based communication and security, time and location, and network monitoring and management. We also reflect on the ongoing efforts in providing dependable assurances for WSN-driven CPS. Finally, we report on its applicability with a case-study on smart buildings.

1 Introduction

A *cyber-physical system* (CPS) refers to a tightly integrated system that is engineered with a collection of technologies, and is designed to drive an application in a principled manner. For example, consider the case of adaptive lighting in road tunnels—a critical requirement for tunnel management and safety.[1] The goal, here, is to control the tunnel lighting levels in a manner that ensures continuity of light conditions from the outside to the inside (or vice-versa) such that drivers do not perceive the tunnel as too bright or dark. There are a number of possible solutions to this problem. It can, however, be simplified by designing a system that is able to account for the change in light intensity (i.e., *detect* physical conditions and *interpret*), and adjust the illumination levels of the tunnel lamps (i.e., *respond*) till a point along the length of the tunnel where this change is indiscernible to the drivers (i.e., *reason* and *control* in an optimal manner). Such a system embodies the principles of CPS.

CPS is built from the synergy of sensing, communication, computation, and control technologies.[2–5] While the functional description for different components of a CPS design are specific to the application requirements, its architectural abstraction remains the same (Figure 1). For example, consider the components of a proposed CPS architecture for adaptive tunnel lighting.[1] The system consists of: (i) a wireless *network* of *sensors* to record light measurements inside the tunnel, and an external sensor to measure luminance at the tunnel entrance, (ii) a *controller* to interpret the luminance levels, apply control logic, and make decisions, (iii) *actuators* to adjust the intensity of the tunnel lamp lights. The internal sensors wirelessly relay their measurements to a gateway, which then forwards it to the controller.

There are a range of application areas that can benefit from CPS. Zero-energy buildings, smart electric grid, smart transportation, extreme-yield agriculture and automation, patient and elderly care, smart medical technologies, safe evacuation from hazardous areas, search and rescue, firefighting, etc., are a few such examples.[3]

While the basic idea of CPS-like systems have been in existence before,[2] the information technology revolution of the last few decades has significantly advanced their frontiers. These latest

[1]*SICS Swedish ICT, Stockholm, Sweden.*

[2]*Politecnico di Milano, Italy.*

[3]*Uppsala University, Sweden.*

prasant@sics.se
luca.mottola@polimi.it
shahid@sics.se
simonduq@sics.se
nvt@sics.se
joel@sics.se
thiemo@sics.se

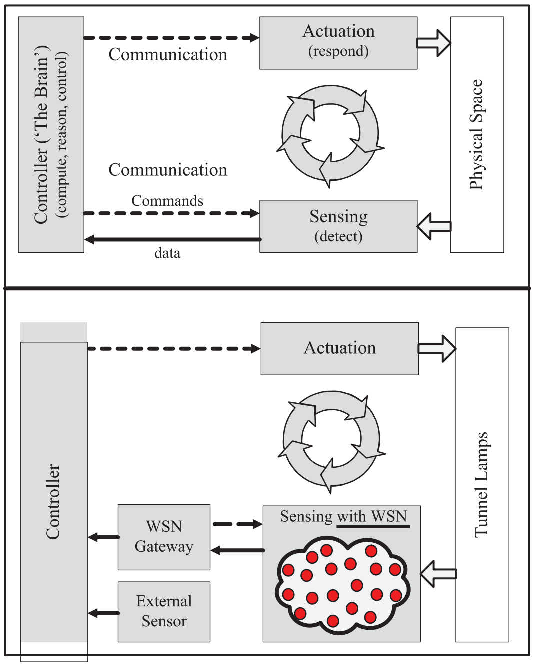


Figure 1. Top: Functional blocks of a cyber-physical system (CPS). Bottom: A wireless sensor network (WSN)-based CPS architecture for adaptive tunnel lighting.[1] While the functional description for different components of a CPS design is specific to the application requirement, its architectural abstraction remains the same.

developments can be used to drive CPS as demonstrated by Ceriotti et al.[1] for adaptive tunnel lighting. A wireless network of sensors, commonly referred as *wireless sensor network* (WSN), is one such technology that can be an essential component of CPS.

A WSN is composed of low-cost and low-power *nodes* of small form factor.[6] Each node is an autonomous battery-powered device with integrated sensing, on-board processing and wireless communication abilities. They are deployed in the region of interest to autonomously gather information from the environment, perform simple computations, and transfer only the required information to a remote device. Individually, these nodes appear to be of little value; however, deploying and networking them on a large scale is effective in fine-grained interaction with the physical space. *WSNs are, therefore, a way to support CPS applications where, besides sensing, communication and computation may be necessary.*

As with any emerging technology, the design of WSNs are not without technical challenges. These issues, predominately, arise due to the size, cost and power constraints of WSN platforms that translate to limited computing, communication, storage and energy resources.[a] Therefore, it is of utmost importance that each device makes very efficient use of its *constrained resources*, the impacts of which affect every aspect of design (from hardware to software) and architecture (communication, network and data management) of WSN.

As mentioned before, CPS is evolving with a rising influx of new technologies. There is a growing optimism that CPS may lead towards an Internet of physical objects, and this fascinating vision is being tagged as the *Internet of Things* (IoT). Therefore, CPS and IoT may not only transform the way people interact and control their physical surroundings, but also conduct and explore business opportunities. For example, consider the case of social media such as collaborative projects, blogs, content communities, social networking sites, virtual game worlds, and virtual social worlds. They act as soft sensors, and create new types of sensed data for CPS/IoT coupling that expand the potentials of business. *While the focus of past developments in WSN have been on fundamental design and energy issues, there are newer and increasing demands for supporting IoT by establishing interoperability with heterogeneous devices and technologies.*

To explore both these aspects of WSN with resource constrained devices, the rest of the article unfolds as follows.

- Section 2 and 3 explain the design guidelines of operating systems, simulators and emulators with a case-study on *Contiki* and *COOJA/MSPsim*.
- Section 4 explores the programming challenges and analyzes it within the context of *makeSense*, a recent programming framework.
- Section 5 argues for a software-based run-time control of application logic with *virtualization* on limited capability platforms.
- Section 6 explains the benefits of *IP-based* low-power wireless communication for achieving interoperability, and discusses the recently standardized IPv6 routing protocol *RPL*. It also explores the possible benefits of *RESTful* interactions, and platforms for *cloud computing*.

[a] For example, consider the case of a popular WSN platform such as TelosB.[7] It has a IEEE 802.15.4 wireless radio transceiver with a (low) bit-rate of 250 kbit/s, a 16-bit microcontroller with a (limited) memory capacity of less than 50 kB of code memory and 10 kB of RAM, a 12-bit ADC for sensing; and uses a 3.3 V AA-size battery to power its hardware peripherals for different operations.

- Section 7 presents the design of an *IP-based security* service for interconnecting low-power networks with the IoT.
- Section 8 details the concepts and challenges in provisioning *time* and *location* services, and outlines their performance limits.
- Section 9 outlines the existing tools for *network monitoring* and *management*.
- Section 10 crisply narrates the ongoing efforts in transcending WSN to be *dependable*.
- Section 11 demonstrates a WSN-driven CPS/IoT with a case-study on *smart building*.
- Section 12, finally, concludes with a summary of the article.

Here, we primarily focus on the software and services for WSN, and refer our readers to the article by Stankovic et al.[8] for a condensed discussion on energy management techniques. We believe that gaining insights into the aspects above will motivate and guide researchers and engineers from various disciplines to adopt WSN for applications of CPS.

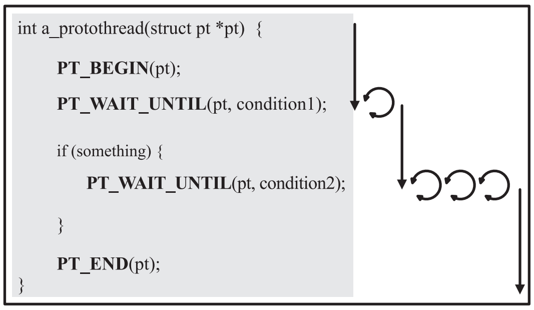
2 Operating Systems: A Case-Study on Contiki

The operating systems (OS) for WSN platforms differ significantly from those for general purpose computers, smart phones, and any other resource rich devices. They are designed to drive a single, dedicated application; and therefore, are more specialized, less complex, and consume limited system resources. Although these lean requirements translate to a smaller OS, provisioning it for memory constrained systems is a challenge. In spite of the difficulty, the concurrency models and abstractions provided by operating systems such as Contiki[9] and TinyOS[10] have successfully bridged this gap.

In this section, we briefly describe the major features of the *Contiki OS*. Contiki implements a lightweight process scheduler; and provides a set of libraries for memory management, sensor and communication abstractions, and low-power radio networking mechanisms. It is written in the *C programming language*, and can be compiled with any C99-compliant C compiler. It is designed to be easily portable, and runs on a variety of WSN platforms.

Contiki has an *event-driven kernel*, and uses a *single shared (memory) stack* for all process execution. Programming with Contiki requires defining event handlers that describe the response of the system to events (for example, sending and receiving radio packets, reading sensors, or invoking internal timers). The execution of a program is triggered either by events dispatched by the kernel or through the polling mechanism (for high priority events). Once an event is scheduled, the kernel does not preempt the event handler. Therefore, event handlers are allowed to run until their completion. Polling is, typically, used by processes that operate near the hardware; e.g., to respond to interrupts. Poll events are scheduled in between other events, wherein all processes that implement a poll handler are called in their priority order. Events are designed to be *asynchronous* in order to reduce the stack space requirements, as each invocation of event handlers rewinds the stack.

Contiki processes provide a sequential flow of control on top of its event-based kernel by using a programming abstraction called *Protothreads*.[11] They are extremely lightweight, stackless threads that provide conditional blocking through the PT_WAIT_UNTIL() wait statement. It is invoked within its body declared by the PT_BEGIN and PT_END statements (Figure 2). Protothreads combine the advantages of event driven (i.e., low memory usage) and multithreaded (i.e., good control flow with no explicit state machines by using the blocking wait semantics) concurrency models. They are a memory-efficient means of provisioning threads to run on a single, shared stack instead of allocating a different stack space to each thread, as required for ordinary threads. The *stack rewinding* mechanism helps in *context switching* between different threads. Moreover, they do not require any machine specific assembler code for executing. However, due to the *local continuation* method of implementation, protothreads do not save the stack context (i.e., call history and local variables) across a blocking call. Hence, it is advised not to use local variables inside a protothread, but rather to define their scope with the **static**



```
int a_protothread(struct pt *pt)  {

    PT_BEGIN(pt);

    PT_WAIT_UNTIL(pt, condition1);

    if (something) {

        PT_WAIT_UNTIL(pt, condition2);

    }

    PT_END(pt);
}
```

Figure 2. Protothreads in Contiki can be seen as a combination of event-driven and multi-threaded concurrency models. With protothreads, event-handlers can be made to block code execution while waiting for events to occur.

keyword in order to ensure that the variables keep their values between thread invocations.

Contiki supports two forms of memory management: (i) memory block allocation, and (ii) managed memory allocation. In the former way, all memory is allocated statically at compile time, and is useful when programmers know the memory requirements of the application. In the latter form, memory can be dynamically allocated at runtime, but from a fixed single memory block whose size is predefined and cannot be altered at runtime. Once memory is deallocated, a *compaction* mechanism is performed on the available memory blocks to reduce the risk of *fragmentation.*

Contiki provides a number of *hardware abstractions* for different components such as radio, LEDs, EEPROM, flash ROM, SPI, UART, RS232,[b] watchdog timers, etc., and sensors such as temperature, humidity, light, sound, etc., that are commonly found on WSN platforms. It also provides four types of *timers:* (i) passive *timers* that keep track of their expiration time without any event-posting ability, (ii) active *etimers* that have the ability to post events, (iii) active *ctimers* that have the ability to schedule function calls, and (iv) real-time *rtimer* that can schedule a function call at an exact time.

In realization of the IoT vision to connect constrained devices to the Internet (that is IP-based),[c] Contiki was the first WSN OS to provide *low-power Internet communication*[12] with μIP, and later incorporated μIPv6.[13] They are among the world's smallest IPv4 and IPv6 network protocol stacks with a memory footprint of approximately 5 kB and 11 kB respectively. Contiki's comprehensive IP stack includes standard IP protocols such as UDP, TCP and HTTP.[d] In addition, it supports new low-power IPv6 networking protocols such as 6LoWPAN (IPv6 over IEEE 802.15.4), RPL and CoAP.[e] It also provides *protocol-independent radio networking* with the Rime stack.[14]

Contiki supports *dynamic loading and linking* of modules at run-time,[15] a useful feature if there is a need to change the application behavior after deployment. It also provides a fast and lightweight *flash file system* called *Coffee*.[16] For devices with an external flash memory, Coffee provides easy-to-use file manipulation operations (e.g., open, read, write, append, close), and abstracts the low-level operation details (such as erasing before writing and flash wear-leveling) from the application program. Other useful features include an *interactive command-line shell* with a set of predefined commands that offer debugging convenience, software-based *power profiling*[17] of the system for enabling energy-aware mechanisms, *IPsec* stack[18] for secure networking, and *on device database* facility with *Antelope*.[19] We refer our readers to the Contiki wiki[20] for latest information and updates.

3 Simulators and Emulators: A Case-Study on COOJA and MSPsim

Developing and testing WSNs can be a difficult task because of the inherent traits of resource-constrained devices. Such typical traits include a lack of memory management units (MMUs) to catch system faults, highly constrained energy and power consumption, and low capacity to store software and run-time state. The latter trait sometimes require the software to be optimized for size rather than readability, which makes it more time-consuming to find and correct errors.

Simulators provide the means of capturing a large number of errors before deploying and using WSN devices. A plethora of simulation tools have been developed to this end, embodying different capabilities and trade-offs for developers and researchers. At the abstract protocol simulation level, we find simulators such as OMNeT++[21] and NS-3.[22] Whilst such simulators have the advantage of high performance and provide the means for rapid prototyping, the abstraction level puts the system far away from the real environment.

TinyOS Simulator (TOSSIM) is a discrete-event simulator that is able to execute unmodified TinyOS applications.[23] This simulator aims to strike a balance between simulation detail and speed by providing emulations of abstract hardware devices, and letting most of the code execute using native instructions. Multiple instances of a simulated node can form a network, for which TOSSIM manages the inter-node communication through the emulation of abstract radio devices. Different radio models give researchers the choice to achieve realism at different scales, with a trade-off against the simplicity of setup and the simulation speed. TOSSIM effectively attains high performance when simulating large networks of hundreds of nodes, but does not provide sufficient detail to develop and debug low-level protocols close to the hardware.

[b] LED: Light Emitting Diode—EEPROM: Electrically Erasable Programmable Read-Only Memory—ROM: Read Only Memory; SPI: Serial Peripheral Interface bus—UART: Universal Asynchronous Receiver/Transmitter.

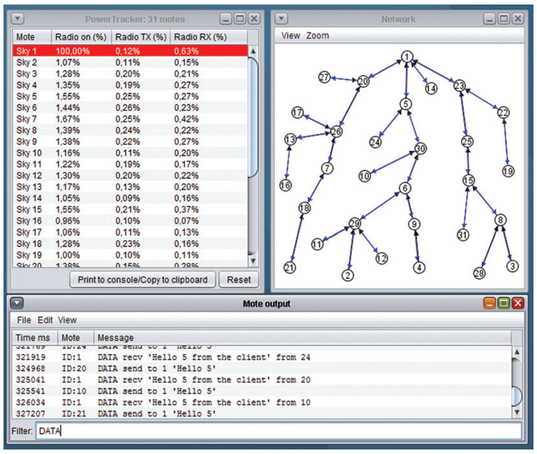
[c] IP: Internet Protocol, where IPv4 is the current protocol version whereas IPv6 is its successor.

[d] UDP: User Datagram Protocol—TCP: Transmission Control Protocol—HTTP: Hypertext Transfer Protocol.

[e] 6LoWPAN: IPv6 over Low power Wireless Personal Area Networks—RPL: IPv6 Routing Protocol—CoAP: Constrained Application Protocol.

COOJA/MSPsim (Figure 3) takes a different approach by emulating the nodes at a *cycle-accurate* level, while providing an abstract radio model in the same manner as TOSSIM.[24] In this approach, the same system firmware that executes on real nodes can be executed in COOJA/MSPsim. Since it has full visibility into the system state, it is able to control memory accesses and detect system faults. This feature is especially important when developing software for resource-constrained devices, which typically lack a MMU. Moreover, the emulator is able to track the transitions between system states and show a live power profile based on this information.[25]

With the current emergence of the IoT and standard-based communication, system interoperability is becoming increasingly important. COOJA/MSPsim, due of its ability to emulate a full WSN device, enables *interoperability*[26] testing of different operating systems and their communication stacks. The simulator is able to test interoperability not only on the network layer[27] but also on the link layer using different implementations of MAC[f] protocols.[28] COOJA/MSPsim also includes *TimeLine*, a tool that visualizes all radio-level events and allows the user to zoom in and inspect the behavior of a selected set of network nodes at any time. This tool leverages the combination of cycle-accurate hardware emulation and network simulation, and makes it considerably simpler to develop and debug interoperable, timing-sensitive network protocols.[24]



Mote	Radio on (%)	Radio TX (%)	Radio RX (%)
Sky 1	100,00%	0,12%	0,63%
Sky 2	1,07%	0,11%	0,15%
Sky 3	1,28%	0,20%	0,21%
Sky 4	1,35%	0,19%	0,27%
Sky 5	1,55%	0,25%	0,27%
Sky 6	1,44%	0,26%	0,23%
Sky 7	1,67%	0,25%	0,42%
Sky 8	1,39%	0,24%	0,22%
Sky 9	1,38%	0,22%	0,27%
Sky 10	1,16%	0,11%	0,20%
Sky 11	1,22%	0,19%	0,17%
Sky 12	1,30%	0,20%	0,22%
Sky 13	1,17%	0,13%	0,20%
Sky 14	1,05%	0,09%	0,16%
Sky 15	1,55%	0,21%	0,37%
Sky 16	0,96%	0,10%	0,07%
Sky 17	1,06%	0,11%	0,13%
Sky 18	1,28%	0,23%	0,16%
Sky 19	1,00%	0,10%	0,11%
Sky 20	1,38%	0,15%	0,28%

```
Time ms   Mote    Message
321919    ID:1    DATA recv 'Hello 5 from the client' from 24
324968    ID:20   DATA send to 1 'Hello 5'
325041    ID:1    DATA recv 'Hello 5 from the client' from 20
325541    ID:10   DATA send to 1 'Hello 5'
326034    ID:1    DATA recv 'Hello 5 from the client' from 10
327207    ID:21   DATA send to 1 'Hello 5'
```

Figure 3. A COOJA/MSPsim simulation of a network of nodes running Contiki with IPv6. This simulation environment combines cycle-accurate emulation with an abstract radio model, enabling developers and researchers to test the same system firmware as would be used on real CPS devices.

4 Wireless Sensor Network Programming: A Case-Study on makeSense

Application development is still one of the main hurdles to a wide adoption of WSN technology. In current real-world WSN deployments, programming is typically carried out very close to the operating system, therefore requiring the programmer to focus on low-level system issues. This not only distracts the programmer from the application logic, but also requires a technical background rarely found among application domain experts. The need for appropriate high-level programming abstractions, capable of simplifying the programming chore without sacrificing efficiency, has been long recognized and several solutions have been hitherto proposed, which differ along many dimensions.

Nonetheless, the problem is challenging: on one hand, existing approaches provide a wide and diverse set of functionality and, on the other hand, WSN applications have widely different characteristics and requirements. Choosing the best platform for a given application demands a clear understanding of the application needs and of the basic differences among programming approaches.

In the following, we start with a brief overview on the state of the art, and then proceed by analyzing a recent proposal whose goal is to blend existing solutions in a coherent and extensible programming framework.

4.1 *State of the art*

In WSN programming, the characterizing dimension that initially received some attention is the one of *node-centric programming* vs. *macro-programming*.[30] The former generally refers to programming abstractions used to express the application processing from the point of view of the individual nodes. The overall system behavior must therefore be described in terms of pairwise interactions between nodes within radio range. Macroprogramming solutions, instead, are usually characterized by higher-level abstractions that focus mainly on the behavior of the entire network, rather than on the individual nodes.

Nonetheless, under many respects the above distinction falls short of expectation in capturing the essence of currently available programming approaches. As a result, solutions offering radically different abstraction levels are considered under the same umbrella, ultimately rendering the distinction ineffective. For instance, both TinyDB[31] and Kairos[30] are commonly regarded as macroprogramming solutions. However, the former provides

[f] MAC: Medium Access Control.

an SQL-like interface where the entire network is abstracted as a relational table. Therefore, internode interactions are completely hidden from the programmer. The latter, on the other hand, is an imperative programming language where constructs are provided to iterate through the neighbors of a given node and communication occurs by reading or writing shared variables at specific nodes. Therefore, unlike TinyDB, in Kairos the application processing is still mostly expressed as pairwise interactions between neighboring nodes.

Mottola et al.[29] classify existing systems according to two macro dimensions: language and architecture. *Language* aspects analyze the primitives programmers can use to express communication and computation as well as the peculiarities of the programming model, while *architecture aspects* consider the programming solutions analyzing features such as their intended use, their reach into the low-level layers of the architecture and their execution environment. We briefly describe the relevant dimensions, and mention some representative systems along the way.

4.1.1 Language aspects

The taxonomy for language aspects is shown in Figure 4. *Communication* issues play a fundamental role, and therefore they can be divided further into sub-dimensions. The *scope* of communication is defined as the set of nodes that exchange data to accomplish a given application processing. The simplest form of scope is called *physical neighborhood* and refers to those cases where the constructs available to the programmer enable communication only among nodes within radio range. An example is Active Messages.[32] At the other extreme, *system-wide* scope refers to systems where constructs are such that all the nodes in the WSN are possibly involved in communication. This is for instance the case of TinyDB.[31] Between these extremes, several systems determine the communication scope based on a *multi-hop group*. The multi-hop group is *connected* if any two nodes in communication are connected via nodes belonging to the group; otherwise is *non-connected*. EnviroSuite[33] is an example of the former, while Logical Neighborhoods[34,35] is an example of the latter.

Further dimensions of communication are addressing and awareness. *Addressing* defines how the nodes involved in the communication are identified. Existing programming frameworks use either *physical* or *logical* addressing. The former relies on statically-assigned identifiers, while the latter uses application-level properties to identify the target nodes. Active Messages is an instance of the former class, while Logical Neighborhoods belongs to the latter. *Awareness*, on the other hand, is concerned with the extent to which the

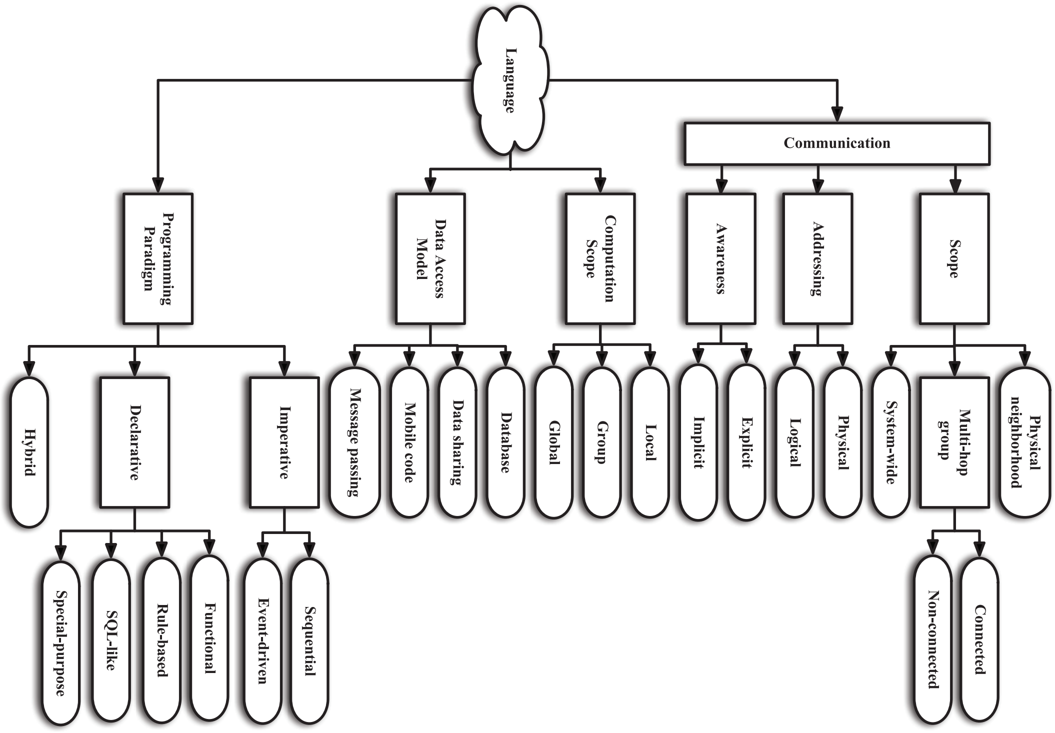


Figure 4: A taxonomy of language aspects in WSN programming abstractions (from[29]).

programmer, through the available constructs, is made aware of communication itself. In systems where awareness is *explicit*, the programmers deal directly with issues like message buffering, serialization, parsing, and sometimes even the scheduling of transmissions. When *implicit* communication is provided, instead, programmers are shielded from all these details, and in some cases cannot even discern precisely when and how data is exchanged among nodes. Active Messages is a representative of explicit communication, while Abstract Regions[36] is an example of implicit communication.

Computation goes hand-in-hand with communication, and WSNs are no exception. The *computation scope* defines the set of nodes directly affected by the execution of a single instruction in the program. If the scope is *local*, as in nesC,[37] an instruction involves only the node where it is executed. At the other extreme, an instruction that can influence the entire network is said to have a *global* scope, as in the case of TinyDB. Finally, in systems that determine the scope of computation based on the notion of *group*, the execution of a single instruction involves only a subset of the WSN nodes, as in the case of Regiment.[38]

Existing solutions provide different abstractions embodying a *data access model*. The related choices heavily influence the way programmers deal with both communication and computation, and therefore impacts significantly the development process. Some systems rely on a *database* model, where the entire network is treated as a relational database and programmers interact with it using an SQL-like language. TinyDB is arguably the most popular in this category. In *data sharing* systems, instead, programmers can access the information shared by the remote nodes by means of dedicated constructs that read and write data elements—variables as in the case of Kairos[30] or tuples as in the case of TeenyLIME.[39] In systems that rely on *mobile code*, data is accessed locally in a node by moving specific instructions to the remote node where data is avalaible, as in Agilla.[40] DSWare[41] is, instead, a representative of systems relying on *message passing*, where data is accessed through messages exchanged among the nodes involved.

Finally, the *programming paradigm* determines the abstractions used to represent the individual elements of a program. In *imperative* languages, the programmer explicitly defines, through the provided constructs, how the program state changes. This is by far the most common case, and it can be further classified into *sequential* (e.g., Kairos) and *event-driven* (e.g., nesC). *Declarative* languages, instead, define the goal of an application task without defining how it should be accomplished, as in TinyDB. ATaG[42] is, instead, a *hybrid* where the local behavior is described with an imperative language, while a declarative, graphical one is used to define the interaction among nodes.

4.1.2 Architectural aspects

The taxonomy of architectural aspects is shown in Figure 5. The dimension of *programming support* is concerned with whether a programming abstraction is designed to be used alone or, on the contrary, together with other solutions. TinyDB is an example of the former class of systems providing *holistic* programming support. Instead, generic role assignment (GRA)[43,44] and Hood[45] are examples of the latter class of *building block* approaches.

Instead, the *layer focus* is concerned with which architectural layers are the main focus of the system at hand. If the latter can be used across layers of the stack (e.g., to implement routing, time synchronization, localization, and possibly even MAC protocols) then the focus is *vertical*. This is the case for TeenyLIME and Hood. Instead, TinyDB is a representative of systems that are designed to be used only with an *application* focus.

In WSNs, cross-layer interactions often play a key role. However, these require the ability to access *low-level configuration* details of the runtime from the language constructs. Unfortunately, most systems provide only a *fixed* configuration that can be changed only at compile time. MiLAN[46] is one of the few systems that explicitly provide an *interface* empowering programmers with access to the lowest layers at runtime.

Finally, WSN implementations are generally difficult to port. Therefore, programmers must often consider carefully the hardware platforms

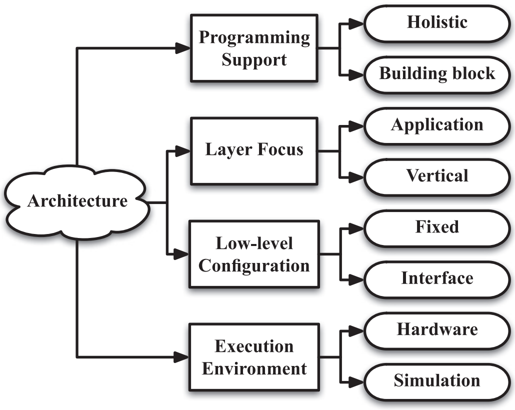


Figure 5. A taxonomy of architectural aspects in WSN programming abstractions (from[29]).

explicitly supported by the programming platform. Interestingly, of the 28 systems surveyed by Mottola and Picco,[29] only 13 have actually been implemented on WSN *hardware*: the others have been evaluated only through *simulation*. Moreover, to the best of our knowledge, TeenyLIME is the only system that has been used in a real-world deployment.[47]

4.2 *makeSense macroprogramming language*

The purpose of the makeSense programming language[48] is not to propose another macroprogramming language. Rather, it is to provide a framework where the abstractions contributing to the language are decoupled, leverage on existing implementations, and can be changed or extended easily to suit specific application needs.

This goal influenced the entire language design. To properly identify the units of functionality, reuse, and extensions makeSense defines the notion of *meta-abstraction*, implemented through different "concrete" abstractions, as described later. Abstractions provide the key concepts enabling interaction with the WSN. However, their composition can be achieved by using common control flow statements, provided by a core language that serves as the "glue" among macroprogramming abstractions. The core language is, in this case, a stripped-down version of Java tailored for WSNs.

Figure 6 shows a UML meta-model for the meta-abstractions provided by the makeSense macroprogramming language. It focuses on the notion of *action*, a task executed by one or more WSN nodes. Actions are separated into *local*, whose effect is limited to the node where the action is invoked (e.g., acquiring a reading from the onboard temperature sensor), and *distributed*, whose effect instead spans multiple nodes.

Distributed actions are further divided into *tell*, *report*, and *collective* actions. The former two represent the one-to-many and many-to-one interaction patterns commonly used in WSNs to enable communication between the node (the "one") issuing the action and a set of nodes (the "many") where the latter is executed. A tell action enables a node to request the execution of a set of actions on other nodes, e.g., to issue actuation commands or to trigger reconfiguration of system parameters such as the sampling rate. A report action enables a node to gather data from other nodes. Event-based abstractions and periodic, continuous queries both fall in this category. Data acquisition occurring on each target node is specified by a local action given as input to the report action. The output of the local action is returned to the report one. Collective actions, in contrast to tell and report ones, do *not* focus on a special node where the action starts or ends. They enable a global network behavior and are executed cooperatively by the entire WSN through many-to-many communication. An example are distributed assertions,[49] where programmers specify a (global) property monitored collectively by the WSN nodes.

Distributed actions may optionally have *modifiers* associated with them, "customizing" their behavior. The makeSense language defines two modifiers, target and data operator. For example, an automatic ventilation system for buildings requires both CO_2 and presence sensors. Programmers must be able to map actions to the set of nodes of interest. A *target* identifies a set of nodes

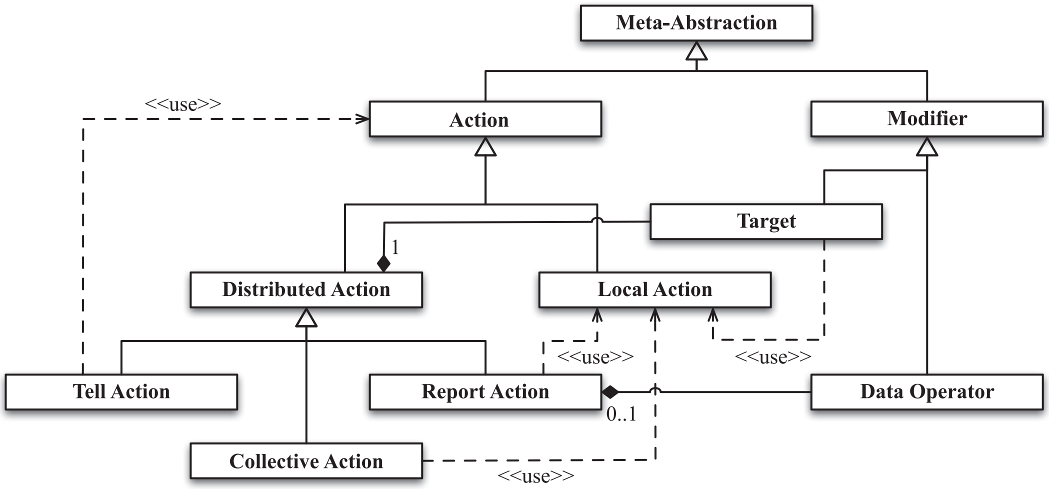


Figure 6: A model for the meta-abstractions of the makeSense macroprogramming language.

satisfying application constraints, and gives the ability to apply a distributed action to the nodes in this set. Instead, a report action may have a *data operator*, specifying processing performed on the results after gathering and before they are returned to the caller, e.g., to filter or aggregate the data.

To create an instance of a meta-abstraction, a class implementing its interface must be defined in the core language. As abstraction implementations, typically, closely interact with the operating system, methods of abstraction classes are implemented in C using a native code interface provided by the core language. Some abstractions require extensive configuration, for example, a target needs to define a set of nodes based on their properties.[35] To simplify such configuration, the core language supports the concept of *embedded languages*, code snippets formulated in the declarative configuration language provided by an abstraction. These are compiled by appropriate compiler plugins, instead of being interpreted at runtime.

5 Virtualization

Traditionally, resource-constrained devices have been programmed using low-level languages. Applications execute in a run-time environment that provides a simple set of system services, and there are essentially no restriction placed on individual applications on how they can access and modify system resources. For instance, Contiki applications are written in C, and operate on devices that do not provide memory protection. TinyOS applications are written in nesC, which provides some extensions to C for the management of components and event-based programming, but the applications still have the ability to access arbitrary parts of the system as in Contiki.

The need to ameliorate such problems has spurred the development of *virtual machines* and other related solutions for *software-based runtime control of applications*. A virtual machine can provide support for high-level programming languages and safe execution environments at the cost of slower execution speed. Applications that depend on intensive calculations (e.g., an encryption algorithm) may experience severe performance problems if executed in a virtual machine. WSN applications, however, are typically event-driven (i.e., sleeping most of the time and waking up momentarily to process incoming packets, send queued packets, or gather sensor samples); therefore, they are feasible to execute in a virtual machine.

One of the earliest virtual machines for WSN devices is Maté,[50] which has a core instruction set of the most common operations. For individual applications, certain high-level operations that require many instructions may be expressed more succinctly if the virtual machine can be augmented with instructions for these operations. Hence, Maté enables developers to add such instructions that are implemented natively in the virtual machine. Unlike Maté, which does not support a high-level language, Darjeeling[51] and TakaTuka[52] are virtual machines that support different subsets of the Java programming language. Figure 7 shows an example of a virtual machine architecture, which provides programming abstractions on a higher level than the underlying OS, and run-time support for convenient features such as garbage collection and preemptive threads.

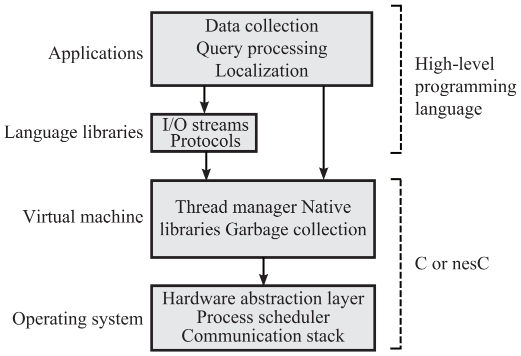


Figure 7: An example virtual machine architecture for WSN devices.

The t-kernel[53] protects systems from malfunctioning applications, and extend the run-time services by providing virtual memory and preemptive threading. Unlike the aforementioned virtual machines, the t-kernel executes applications in native machine code, but it has to be translated into binary before execution. This binary translation inserts additional instructions, which force the application to jump into the t-kernel for control whenever it is executing code that can potentially cause harm, e.g., writing to a memory address. Whilst the execution overhead of this method is 50%–200% compared to a native application, the CPU is still idle more than 92% of the time in the applications tested. This further emphasizes that WSN applications are in general not CPU-intensive, and can indeed draw advantage from the features and safety provided by virtual machines.

6 IP-Based Networking

To include WSN in CPS, there is need to connect heterogeneous devices and technologies: sensors, actuators, and the controller. To this end, an appealing approach is to rely on IP, the protocol suite of the Internet. Doing so, devices can interoperate, and be connected to existing infrastructures and to the Internet. Through a common network-layer protocol, IP-based CPS enables interoperability at the levels: (i) among heterogeneous link-layer technologies such as IEEE 802.11, IEEE 802.15.4, Bluetooth or Powerline Communication (PLC); and (ii) among heterogeneous applications such as web-based configuration systems, publish-subscribe protocols, etc.,. This section reviews the different layers involved in IP-based low-power WSN, as well as research related to this area.

6.1 *The 6LoWPAN adaptation layer*

The number of Internet-connected devices is continuously growing, and already exceeds the number of possible IPv4 addresses. To facilitate network management and scale to dozens or hundreds or billions of connected devices, IPv6 is commonly considered as an appealing option. To make IPv6 suitable for IEEE 802.15.4 networks, the IETF[g] has standardized a network adaptation layer, 6*LoWPAN*.[54] It defines compression and fragmentation of IPv6 packets so that they can fit into the IEEE 802.15.4 frames (of size 127 bytes). 6LoWPAN is currently supported in the *two* mainstream operating systems for WSN: *Contiki* with *μIPv6*[55] and *TinyOS* with *Berkeley Low-power IP stack* (BLIP).[56]

The operating principle of 6LoWPAN is to compress header fields in a flexible manner, assuming commonly used values. For instance, the IPv6 addresses that, normally, take up 16 bytes can be either 'carried inline' (uncompressed) with their prefix elided (if the prefix is among the set of common prefixes), and/or have their interface identifier elided (if it can be derived from the MAC address). In the best case, the IPv6 header can be compressed from 40 bytes down to 2 bytes.[54]

6LoWPAN and IP assume wireless communication links to be *always-on*. In low-power WSN, it is typically achieved through asynchronous radio duty cycling, such as low-power listening (sender-initiated, e.g., X-MAC[57] or ContikiMAC[58]) or low-power probing (receiver-initiated, e.g., RI-MAC[59] or A-MAC[60]). These solutions share the same design goal: spend most of the time with the radio chip idle (typically more than 99% of the time), and wake up periodically to check for pending communication. This results in significant reduction in energy consumption, but increases the end-to-end latency and link capacity. Recent results have shown that standard TCP traffic can be carried over a multi-hop duty cycled IEEE 802.15.4 network in a reliable manner with an end-to-end goodput of approximately 6 kB/s[61] by forwarding data in large chunks at the MAC layer ('Burst Forwarding').

6.2 *The RPL routing protocol*

The IETF ROLL working group has recently focused on developing standards for interoperable routing over low-power IPv6 networks. In March 2012, it standardized RPL: IPv6 Routing Protocol for Low-power and Lossy Networks.[62]

The design of RPL is largely based on the Collection Tree Protocol (CTP),[63] the reference data collection protocol for WSNs. The RPL topology is a Destination-Oriented Directed Acyclic Graph (DODAG) built in direction of the root, typically an access point to the Internet. RPL is capable of routing IPv6 traffic between any node pair in the network, and in both directions between individual nodes and Internet hosts.

An RPL topology is built using periodic beaconing, where nodes advertise their logical distance to the sink. Nodes join the DODAG upon receiving such beacons, and configure their global prefix so that they can communicate with external networks. The same topology is used in the up direction (routing towards the root) as in the down direction (routing away from the sink).

RPL provides the following two modes of operation.

a. *Storing mode*: In this mode, each node maintains a routing table to reach all nodes in their sub-DODAG. Any-to-any traffic is supported by first routing upwards until a common ancestor of the destination and the source is found, and then downwards by following the nodes' routing table (Figure 8).
b. *Non-storing mode*: To reduce the memory requirements on RPL nodes, it also provides a non-storing mode in which nodes do not store a routing table. Instead, they always route upwards, towards the root. The root gathers global knowledge of the network topology, and performs source routing (by including the multi-hop path in the packet header).

RPL is a generic and extensible protocol, and can be used with various so-called *Objective functions* that define the criteria for the routing topology. RPL also allows to run multiple logical instances on the same physical network, which makes it possible to maintain topologies that maximize different metrics and follow different constraints. A typical example would be an energy-efficient topology for background traffic along with a low-latency topology for delay-sensitive alarms.

[g] IETF: Internet Engineering Task Force.

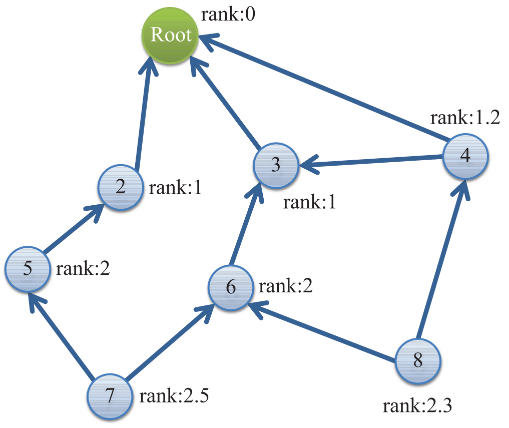


Figure 8: In RPL, the network topology (a DODAG) is anchored at the network root. Nodes are placed in the topology according to their rank, which reflects a logical distance to the root. In storing mode, all traffic is routed first upwards until reaching a common ancestor, and then downwards to the destination. For instance, node #2 can send to node #8 through the path: #2, Root, #3, #6, #8.

6.3 *RESTful interaction*

The primary benefit of IP-based WSN and CPS is interoperability among heterogeneous devices. At the application layer, an interesting solution is to build a distributed system of services by using a *RESTful architecture*. In a RESTful system, servers provide a set of resources that can be easily composed and browsed through hypermedia (i.e., a generalization of hypertext). Each resource is accessed using the traditional RESTful *verbs*: GET, PUT, POST and DELETE. Doing so, operations are explicitly classified as *safe* (GET, with no side effect) or *unsafe* (PUT, POST, or DELETE that affect the server-side state); and make it possible to cache content and scale to larger networks.

In RESTful WSN, every node runs a *tiny web server*[64,65] that exposes the device's basic sensing and actuation capabilities in an application-agnostic way. Applications can be built on top of such a RESTful architecture by simply combining the resources of various devices. For instance, a node may give access to its temperature sensor through:

```
GET /sensors/temperature
```

and to its door lock actuator through:

```
PUT /sensors/doorlock [lock|unlock]
```

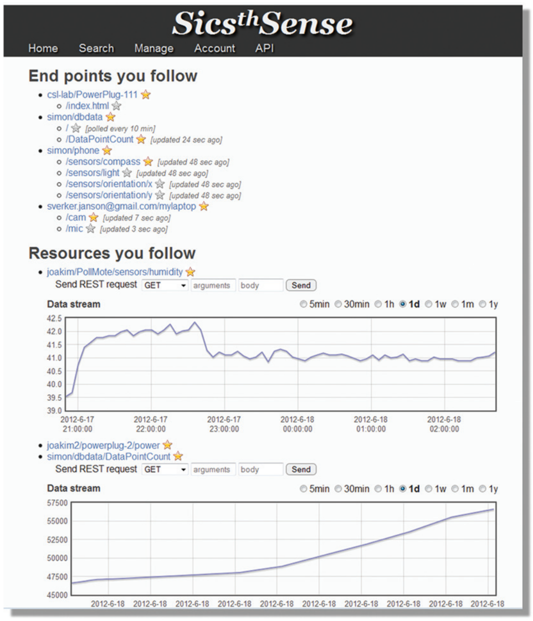
This approach makes it easy both to use the system in an interactive way, or to script interactions.

To bring RESTful resources to the most constrained devices, the IETF CoRE working group is currently defining the CoAP.[66] In contrast to HTTP, CoAP uses UDP as the transport layer, which results in lightweight and efficient implementation of embedded RESTful servers.[67] CoAP also supports *natively push notification* and *multicast communication—two* operation that are notoriously difficult to support with HTTP over TCP.

6.4 *Connecting to the cloud*

Cloud computing refers to Internet servers that provide services in a scalable way. In this cloud-centric CPS/IoT vision, all devices are connected to the cloud where most of the application logic resides. IP-connected devices and CPS/IoT can benefit in interoperability, flexibility, and efficiency from Cloud connectivity. This makes it easy to use heterogeneous devices for the same application, update the application logic, and also allows offloading of heavy computation from constrained devices to elastic cloud platforms.

A number of cloud platforms following this architecture are being developed. Cosm (https://cosm.com/), Evrythng (http://www.evrythng.com/), Sense (http://open.sen.se/), and SicsthSense (http://sense.sics.se) are a few such examples. These platforms provide storage space for sensor readings, possibility to set triggers, share resources, and visualize data. Figure 9 shows an example of the data feed visualization in SicsthSense.



End points you follow

- csl-lab/PowerPlug-111 ☆
  - /index.html ☆
- simon/dbdata ☆
  - / ☆ [polled every 10 min]
  - /DataPointCount ☆ [updated 24 sec ago]
- simon/phone ☆
  - /sensors/compass ☆ [updated 48 sec ago]
  - /sensors/light ☆ [updated 48 sec ago]
  - /sensors/orientation/x ☆ [updated 48 sec ago]
  - /sensors/orientation/y ☆ [updated 48 sec ago]
- sverker.janson@gmail.com/mylaptop ☆
  - /cam ☆ [updated 7 sec ago]
  - /mic ☆ [updated 3 sec ago]

Resources you follow

- joakim/PollMote/sensors/humidity ☆

Send REST request GET arguments body Send

Data stream ○ 5min ○ 30min ○ 1h ● 1d ○ 1w ○ 1m ○ 1y

- joakim2/powerplug-2/power ☆
- simon/dbdata/DataPointCount ☆

Send REST request GET arguments body Send

Data stream ○ 5min ○ 30min ○ 1h ● 1d ○ 1w ○ 1m ○ 1y

Figure 9: SicsthSense offers a centralized place to manage and share RESTful resources, gather and visualize data, and script actuation.

On the cloud side, it is comparatively easy to develop and maintain flexible applications. For example, Actinium[68] is a RESTful engine that runs Javascript programs on an application server. It provides a simple application program interface (API) to interact with CoAP resources that make it possible to get data from a device, interpret and make decisions, and send actuation commands. This logic resides on the server and is implemented as a script that makes it easy to maintain and share, and is open to interactions with other networks and external services.

7 IP-Based Security

Providing security in WSNs is challenging as sensor nodes are mostly resource-constrained, are deployed in unattended environments, the communication links are lossy, and there is an extremely diverse set of potential application scenarios. In case of the IP-based WSNs (as part of IoT), shown in Figure 10, the problem of enabling security is even more challenging because of global connectivity of sensing devices and of potential applications where humans are directly or indirectly an integral part of the system. However, unlike typical WSN where security is mostly ignored, security is one of the main requirements in the IoT.[69–71] Furthermore, in order to provide interoperability with the IP-networks, we require compatibility to existing Internet standards.

Table 1 shows a protocol stack of IoT devices inside a 6LoWPAN network and the corresponding standard-based security solutions at each level. There are multiple options to secure IoT devices and each of these has its own pros and cons. One can choose an option based on the application's security requirements: *end-to-end* (E2E) between source and destination, or *per-hop* between two neighboring devices.

7.1 *End-to-end security*

In E2E security, confidentiality and integrity protection and authenticity of messages is provided between the source and destination devices. E2E security is achieved by enabling security at the application, transport, or network layer. There are no standard based application layer security protocols that are applicable in multiple applications scenarios. For example, there exists Secure/MIME (S/MINE) but can be used for MIME messages only.[72] The typical security solution at the application layer is the use of username and password which does not cryptographically protect the communication, but rather provides a supplement to authenticate and grant access to resources. Commonly used and standardized E2E security protocols for the UDP/IP networks (and hence for the IoT) are IPsec[18] and DTLS[h].[73,74] Here, by IPsec we only mean the IPsec in *transport* mode as the tunnel mode is unfeasible for constrained devices. On one hand, IPSec is mandatory in IPv6 protocol; which means that all IPv6 ready devices (by default) should have IPsec support that may be enabled at any time. This is a flexible solution for the IoT as it can work with any transport protocol, such as TCP or UDP, and it can protect the IP header as well. On the other hand, IPsec security services are shared among all applications running on a particular machine. In case IKEv2[75] is not used, it is cumbersome for a

[h] TLS is most common in the TCP/IP networks.

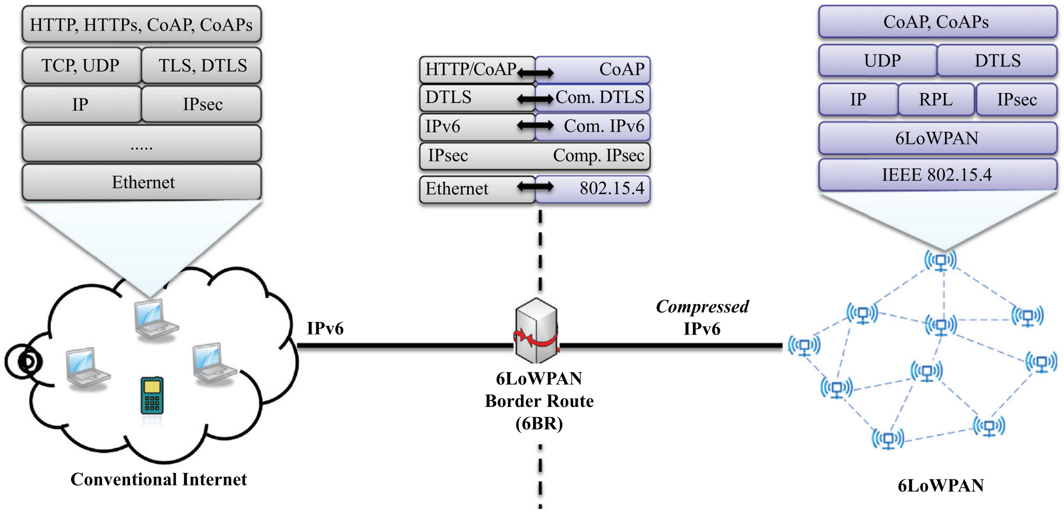


Figure 10: A secure IoT setup showing interconnection of a 6LoWPAN network and conventional Internet with novel IoT and traditional Internet technologies.

Table 1: IoT stack with standardized security solutions.

IoT Layer	IoT Protocol	Security Protocol	Scope
Application	CoAP, HTTP	User-defined	E2E
Transport	UDP, TCP	DTLS, TLS	E2E
Network	IP	IPsec	E2E
Routing	RPL	RPL security	Per-hop
6LoWPAN	6LoWPAN	None	None
Data-link	IEEE 802.15.4	802.15.4 security	Per-hop

end user to enable multiple sessions and manually set the IPsec keys and security policies. However, we believe that IPsec is one of the most applicable security options for the IoT as mostly only one single application runs on a tiny device and the default security policies are enough for such scenarios. Furthermore, IPsec is provisioned at the network layer, and therefore, application developers require little effort to enable it. In our previous work, we have designed, implemented and evaluated a lightweight 6LoWPAN compressed IPsec for the IoT.[76]

Although, IPsec can be used in the IoT, it is not primarily designed for *web* protocols such as HTTP or CoAP. For web protocols, TLS or DTLS are common security solutions. However, since CoAP is the protocol envisioned to enable the Web-of-Things on top of IoT, DTLS is used in the CoAP-enabled IoT. DTLS guarantees E2E security of different applications on a single machine by operating between the transport and application layers. A secure version of CoAP (CoAPs) that uses DTLS as the underlying security protocol is already being standardized.[74] A web resource on an IoT device can then be accessed securely via CoAPs protocol as: **coaps**://myIPv6 Address:port/MyResource.xml

7.2 *Per-hop security*

Hop-by-hop, or rather End-to-Middle security can be used to protect a communication link between two neighboring devices, and protect the network against unauthorized access and resource misuse.[77] For the IoT, the standard based RPL security or IEEE 802.15.4 security can be used to provide hop-by-hop security. RPL's security feature only protects RPL-based routing messages and not the application data, whereas IEEE 802.15.4 security is flexible enough to protect and work with any network, transport, and application layer protocol. The per-hop security helps to protect against data modification attacks on each hop unlike E2E where modified packets traverse the entire path up to the destination to be detected. In our previous work, we have implemented and evaluated IEEE 802.15.4 security and compared its overhead with IPsec.[76]

In both, RPL and IEEE 802.15.4 security, all devices in a 6LoWPAN network share a common security key. In case an attacker compromises one device in the network, it gains access to the key; and hence, the security of the whole network is compromised. Therefore, in the case of the IoT we recommend that E2E security should be enabled between a device in a 6LoWPAN network and a host on the Internet. In addition, per-hop security with at least integrity protection should be used to prevent unauthorized access to network's resources and to defend against effortless attacks launched to waste constrained resources. In case of CoAP-based web applications in the IoT, we recommend to use CoAPs rather than IPsec as it enables seamless secure access to web resources. Also, small-scale sensor devices (as used in WSN) are easy to capture and clone to recover their secret contents. Therefore, in addition to communication security, the IoT should be protected with an intrusion detection system (IDS),[78] and stored secrets inside the sensor devices[79] should be protected by encryption.

8 Time and Location

Time and location play an important role in sensor network applications, since they provide valuable context (*'when and where it happened ?'*) in interpreting sensed data (*'what happened ?'*).

8.1 *Time*

The clock synchronization problem is to maintain a *common notion of time* among the constituent parts (i.e., nodes) of a distributed system (i.e., network of nodes).[80] Having such a notion of time is important for (temporal) message ordering, data fusion (to assemble distributed information in a coherent manner), intra-network coordination among sensor nodes for data consistency, security and communication protocols, localization, etc.,.

Each node has a (hardware) clocking system that consist of a counter that counts time steps, and is incremented at a rate provided by an oscillator. It is said to be linear if the clock rate remains constant with time; however, they experience *variable drift* due to changes in supply voltage, temperature, etc.,. WSN devices usually contain non-expensive oscillators that, typically, drift in the range of 10–100 ppm.[i] Nodes must, therefore, synchronize their drifting hardware clocks by

[i] ppm: parts-per-million, 10^{-6}. A clock drift of 100 ppm corresponds to a drift of 100 μs in one second.

exchanging information about their current state. Nevertheless, there is also a *variable delay* in this operation of sensing and receiving messages.

It is assumed that a node can only read its hardware clock (and not modify it). Therefore, a logical (software) clock is maintained in each node whose value depends on its hardware clock and the information received from its neighboring nodes. When clocks are linear, they can be described by their skew (rate) and offset. A *clock synchronization* algorithm, therefore, *adapts the logical clock value in a optimal manner that minimizes the clock skew between any two nodes in the network, regardless of their relative distance*.

There is a large literature on clock synchronization in distributed systems, which mostly focus on bounding the internode clock skews. Some of the prominent algorithms (ordered by publication date) are listed in Table 2 along with their respective synchronization accuracies. We refer our readers to the review articles by Romer et al.[85] and Lenzen et al.[80] for more detailed information on their underlying assumptions and implementation techniques. Glossy has reported the lowest time synchronization error in WSN. It was *primarily* designed for fast and reliable (> 99.99%) network flooding for which it exploits constructive interference of IEEE 802.15.4 symbols. Its synchronization accuracy of 0.5 μs is, infact, necessary to make concurrent transmissions of the same packet interfere constructively. Glossy is now a part of the Contiki codebase.

8.2 *Location*

The localization problem is to determine the *current location of the nodes in the network within a given coordinate system*. Location information assists in network management tasks, such as routing (for geographical algorithms),[86,87] network connectivity adjustment (i.e., topology control) for energy saving[88] and security.[89] Besides, the location itself is often the data that needs to be sensed, and is extremely useful for the self-localization of the sensor nodes, if it is accurate enough.[90]

Table 2: Accuracy of time synchronization algorithms for WSN.

Algorithm	Avg. Accuracy (μs) (Single Hop)
RBS[81]	29.10
TPSN[82]	16.90
FTSP[83]	1.48
Glossy[84]	0.50

Reference Broadcast Synchronization (RBS)
Timing-Sync Protocol for Sensor Networks (TPSN)
Flooding Time Synchronization Protocol (FTSP)

Localization consists of two-phases: first, *measurement* that estimates inter-node location specific signal metrics, and second, *positioning* that utilizes the measured information to compute their (absolute or relative) location coordinates. An adjunct *calibration* step is included in the measurement phase to compensate for errors due to manufacturing differences in the hardware or changing environmental conditions. Depending on the sensed modality, measurement techniques can be classified into range-based, motion dynamics, and range-free.

Range-based techniques predominately depend on probing with a physical signal (of known characteristics) to estimate a ranging parameter. For example, *received signal strength* (RSS) is estimated by measuring the attenuation of the physical signal (transmitted at a known power level) at the receiver node, and is used with a path-loss and shadowing model to compute the distance. Distance measurements of higher accuracy can be obtained by estimating the *travel time* of the physical signal (RF/acoustic), and this forms the basis of time-of-arrival (TOA), time-difference-of-arrival (TDOA) (and angle-of-arrival AOA for angle measurements), round-trip time (RTT), elapsed time between two time-of-arrivals (ETOA). RSS estimators exhibit large uncertainty due to shadowing and multipath reflections from the environment and ground. Time delay-based signal metrics are susceptible to errors due to obstructions between the transmitter-receiver node pair leading to non-line-of-sight (NLOS) conditions, noise, interference, multipath, clock drifts, etc.,. *Acoustic signals deliver higher accuracy*[91,92] *than RF* due to better compensation of timing errors derived from their slower propagation speed; but have limited range and coverage.[93] They are, moreover, affected by other physical factors such as air density variations caused by thermal effects leading to differences in sound speed, and propagation effects caused by non-uniformity in the atmosphere (due to wind and turbulence).

The challenges faced by range-based techniques can be overcome by measuring the *motion dynamics* (such as velocity, acceleration and orientation) using inertial and magnetic units (IMU). While this mode of measurement is obvious for a network of mobile sensors, motion dynamics in static cases can also be recorded by deployment personnel carrying embedded devices with IMU. IMUs are immune to ambient noise, and are free of LOS requirements However, they suffer from drift since errors in measured dynamics accumulate when integrated over time. While inertial units remain unaffected by electromagnetic interference, the performance of magnetic units (e.g., magnetic compass) is severely affected by stray magnetic fields caused by various electrical equipment or ferrous metals in the environment, typical of indoor environments.

Range-free techniques, on the other hand, provide course location estimates by utilizing proximity (with respect to one-hop anchors)[94] or connectivity information (with neighboring objects to build hop-based virtual distance)[95] by measuring the RSS. Hence, they are affected by the error sources for RSS where the measurement uncertainty further intensifies when the communication range is large or the connectivity of the network is low.

Measured data can be of various types, but different measurement methods can deduce similar information. For example: RSS can be translated into distance, proximity or connectivity status; time-delay and step-counter (from IMU-accelerometer) can be transformed into distance, while time-difference and orientation (from IMU-gyroscope/compass) result in angle. *Every measurement modality has a different set of error sources that are mostly orthogonal to each other, and have different performance levels.* Therefore, combining several such measurement methods can benefit the final position result.[96] Based on the algorithm that generates the location coordinates in the positioning phase, the process of location estimation can be categorized into: anchor-based and anchor-free, and can be executed in an incremental or concurrent manner.[97]

The *anchor-based* method requires provisioning a set of (fixed/mobile) reference points (also referred to as anchors or landmarks) with known location coordinates. The measured distances (and/or angles) to the target node from these anchors and the anchor locations are used to obtain the target position within the selected coordinate system using geographical calculations/mathematical computations such as triangulation, multilateration, etc.,. If the anchors are moving, they should follow a predefined trajectory so that their coordinates can be determined at any given instance.

In the *anchor-free* mechanism, the location of the target nodes are determined by using only the knowledge of their separation distances without the support of anchors. The underlying algorithm searches the defined coordinate space to find the optimal coordinates for the nodes that satisfy the measured distance constraints. However, the uniqueness of the solution cannot be guaranteed since the coordinate assignment continues to be valid under translation and rotation[97] without violating any of the distance constraints. Therefore, this method of localization, eventually, becomes difficult due to the absence of anchors.

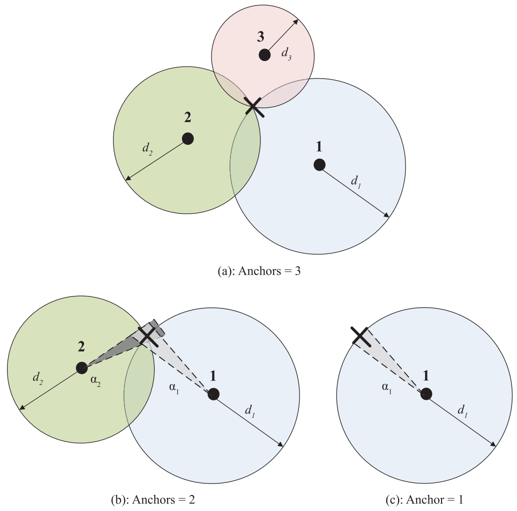
Assuming that the target nodes have to be localized in a two-dimensional plane with anchors, their positions (consisting of two unknown variables *x* and *y*) can be unambiguously determined by solving a system of two linearly independent equations for each target node. Relying on distances only, the necessary equation system requires three distance measurements from *three anchors* to determine the position of a target node (Figure 11(a)). The angle information between the anchors and the target node benefits in obtaining an unambiguous solution to the problem with only *two anchors* (Figure 11(b)). Nevertheless, its position can also be uniquely determined by solving a single equation that combines a single distance and angle measurement to a *single anchor node* (Figure 11(c)).

Therefore, the minimum number of anchors that are required to determine the position of the target node can be limited (i.e., from three to one) depending on the measurement capabilities (i.e., distance/angle) of the nodes. However, measurements to additional anchor nodes with multiple ranging parameters improve the positioning accuracy.

In the broader context of (higher capability) devices in IoT, there exist a wide range of sensing modalities (such as microphone, camera, GPS, accelerometer, gyro, digital compass, and/or infra-red) and wireless communication interfaces (such as 3G, GSM, WiFi, Bluetooth) that can be used for location sensing (Table 3). Many of the core WSN concepts covered in the section are also valid for ranging and positioning with these devices. Despite significant progress in this field, the resources required for signal detection are a deciding factor for the cost, power, size and weight of the sensing platform, and this essentially strikes a *trade-off between localization accuracy/coverage range and energy efficiency*. In regards to closing this gap, the emerging technology of *sparse representation/compressive sensing based location systems* are being investigated.[98,99]

9 Network Monitoring and Management

Deploying and managing a WSN is a tedious task. Resource constrained embedded devices, typically used in WSNs, can experience problems during (and after) deployment due to hardware, software or environment conditions. These anomalies coupled with the ambitious targets of long term deployment (over several years), large scale



(a): Anchors = 3

(b): Anchors = 2

(c): Anchor = 1

Figure 11: Depending on the measurement capabilities of the WSN nodes, the minimal anchor count can be reduced from *three* to *one*. *d* and α, respectively, represent distance and angle measurements.

Table 3: Accuracy of positioning technologies.

Technology	Avg. Accuracy
GPS	15 m
DGPS	5 m
Cellular ID	50 m–300 m
Wi-Fi	2 m–100 m
Bluetooth	2 m–10 m
Infrared	5 cm–10 m
RFID	5 cm–5 m
Vision	1 cm–1 m
Ultrawideband	10 cm–15 cm
Sound	1 cm–10 cm

Global Positioning System (GPS)
Differential GPS (DGPS)
Wireless-Fidelity (Wi-Fi)
Radio Frequency Identification (RFID)

(comprising of several hundreds of nodes), and performance guarantees make it necessary to have network monitoring and management services[100] for network health checks, fault-detection and debugging, etc.,.

Several tools have been developed for this purpose, both for testbeds and live deployments of WSNs. For live networks, systems such as Sympathy[101] are able to reason about node failures, SNMS[102] and Memento[103] provide state inspection facilities, while EnviroLog[104] assists in logging function calls that can later be replayed to replicate the node behavior. A recent addition to this set of WSN monitoring and management tools is the 6PANview,[105] a SNMP[j]-based system for 6LoWPAN/RPL networks. There also exist tools such as LiveNet[106] and Multihop Network Tomography (MNT)[107] that are able to reconstruct and recover the network dynamics (such as routing paths, network topology, bandwidth usage, etc.,.) by simply observing the packets.

10 Making WSNs Dependable

Albeit significant advancements in the field of WSN (as narrated in the previous sections), its widespread adoption by the industry[k] for supporting CPS applications of safety-critical nature[110] is still not apparent. Among different factors, this reluctance stems primarily from the current *"best effort"* communication nature of WSN protocols that cannot provide guarantees[1] in *reliability*, *latency* and *capacity*. Therefore, the established designs of dependable distributed systems cannot be applied to CPS.

The Internet based on the IP communication stack is a best effort network architecture. While such an architecture suits the needs of many WSN applications, CPS often need dependable assurances. In this regard, both the MAC and the transport layers play an important role. In the Internet, TCP is the standard transport-layer protocol for reliable communication. Although, TCP is known to perform sub-optimally in multi-hop wireless scenarios, it has been shown recently to be suitable for reliable, energy-efficient communication in IP-based WSN[61] when running atop a reliable, asynchronous duty-cycled MAC. Other options, higher in the protocol stack, include the CoAP protocol and its confirmable requests that implement *reliability* on top of UDP.

Providing *latency* and *capacity* guarantees is non-trivial to achieve in WSN due to the lossy nature of wireless links, and the complexity of multi-hop communication. Assuming planned communication patterns (e.g., network traffic with constant interval and payload), scheduled MAC layers offer predictable performance. Low-Power Wireless Bus (LWB)[111] is a recent best-effort communication layer; which performs synchronous transmissions across the network, and exploits constructive interference to achieve robust and efficient communication with any traffic pattern. The same team of researchers also

[j] SNMP: Simple Network Management Protocol.

[k] WirelessHART[108] and ISA100[109] are two existing IEEE 802.15.4 standards that were introduced for promoting the adoption of WSNs in industrial control and automation.

introduced VIRTUS,[112] a virtually-synchronous messaging layer for resource-constrained WSN devices in CPS applications. VIRTUS provides atomic multi-cast and view management with a combination of dedicated techniques that build on LWB. Also, the recent IEEE 802.15.4e standard includes a time-scheduled channel-hoping MAC (TSCH) that aims to deliver more predictable performance in planned communication scenarios. It is currently being investigated by the IP-based WSN community at the IETF with the 6TSCH working group (currently under definition). 6TSCH focuses on running the 6LoWPAN network stack on top of IEEE 802.15.4e TSCH, and is aimed towards deterministic IPv6 communication in WSN. RELYonIT[113] is another recent initiative in this direction. It aims to provide a systematic framework and toolchain that would account for all relevant environmental properties, and their impact on WSN platforms and protocols to enable dependable CPS/IoT applications.

11 Application: A Case-Study of WSN-Based CPS for Smart Buildings

There are many motivating factors behind making both new and existing buildings 'smarter' of which the important ones are energy efficiency, damage prevention, and increased comfort and control.

Energy efficiency. It is a key aspect that can break the rising trend of energy usage. A first step in this direction is to *raise awareness about household energy consumption* at the consumer level. Such a practice can be easily developed with the help of modern smart (power) meters and monitoring tools that record current consumption of individual appliances. Households and buildings can be further instrumented with a network of sensors (i.e., WSN) for monitoring temperature and light. Therefore, combining the above physical parameters can allow more fine-grained control over the energy consumption of electrical appliances for heating, ventilation and light.

While energy billing has been done with fixed energy fees for extended periods of time, we believe that flexible contracts will become more common in the future (with moving tariffs) following the fluctuations of the energy markets. This would create stronger incitements to adjust the energy consumption locally; for example, by shifting time independent consumption to outside of peak hours, and moving non-critical tasks to periods when the energy price is the lowest. This would lead to better load balancing at the power grid.

Damage prevention. The damage of buildings can be caused by many environmental/human factors. Dampness is one such cause, which is reported to cost close to 600 M€ annually in Sweden[114] for building repair. Building damage from these factors can be easily prevented by monitoring them effectively. For example, damp sensors that report divergent moisture levels can help detect dampness or leakages before it becomes a problem.

Increased comfort and control. Several manufacturers of heat pumps already offer products to remotely control them by SMS. This allows house owners to heat up the building in advance before returning to the property. There also exist solutions for remotely controlled alarms, and many custom made home control systems developed by enthusiasts. They can all offer accessibility services and increased comfort for the end users. However, in the current form, they are either provided as separate non-interoperable services, prohibitively expensive monolithic systems, or system requiring special installation and maintenance knowledge.

With the help of a WSN-driven CPS architecture for smart buildings/houses (Figure 12) and demo smart household (Figure 13), we show how these demands can be addressed by CPS and some of the concepts presented earlier in this article. For our demo household, the CPS consists of: (i) a *WSN* to monitor temperature, humidity and illumination intensity inside the house, and *power meters* to record the current consumption of individual electrical appliances such as a fan, a room heater, and a light bulb; (ii) *actuators* (each dedicated to an appliance) to control the power supply and moderate the physical space inside the household, and (iii) a *controller* in the form of an energy gateway. The aggregated electricity consumption of the household is forwarded to the electricity provider/supplier, which reports back the current tariff. In the home, the *energy gateway* is the control point. It connects all monitored and/or controllable household devices, receives pricing updates and tariff reports from the electricity provider/supplier, and regulates the household energy consumption according to the (consumer configurable) power policy, target energy saving and comfort level.

By using *low power WSN devices*, together with *efficient radio duty cycling*, resource consumption is kept minimal. This ensures that the benefit of an increased energy efficiency is not pulled back by increased energy spending. WSN devices follow standard *IP-based* communication to connect and exchange information with the gateway. The collected data is forwarded to the service provider

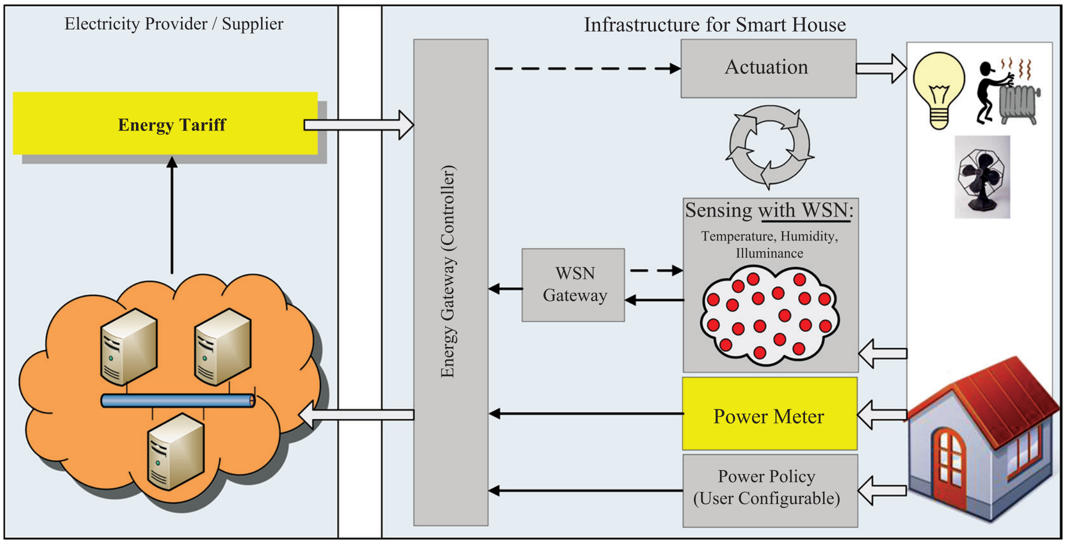

Electricity Provider / Supplier
Energy Tariff
Infrastructure for Smart House
Actuation
Energy Gateway (Controller)
Sensing with WSN:
Temperature, Humidity, Illuminance
WSN Gateway
Power Meter
Power Policy (User Configurable)

Figure 12: Functional components of a WSN-based CPS architecture for application in Smart Building/House.

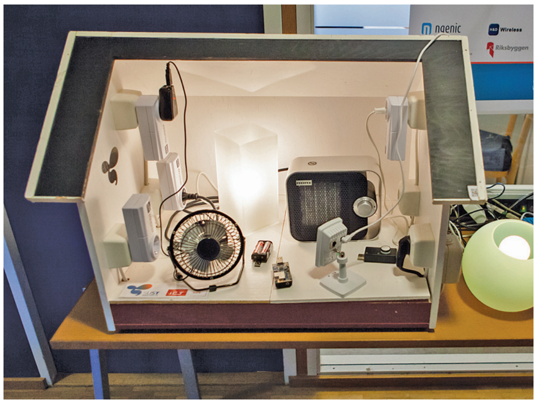


Figure 13: A small scale model of a *Smart House* based on the architecture shown in Figure 12.

through a data layer that handles *subscriptions*. It, therefore, ensures that only requested data is forwarded in order to minimize network traffic. A trusted security service is required for the purpose of granting remote device access to building infrastructure services or individual household appliances. With an *IPsec* end-to-end solution in place, user authentication for access control and privacy of the generated data can be enforced.

This architecture and prototype illustrated a mechanism to efficiently use and manage energy. The damage prevention aspect with smart buildings was not demonstrated, but can be easily implemented by configuring a dampness sensor and defining safety rules for triggering a warning. However, in order to use the solutions from different vendors, there is a need to follow defined standards for creating a system that can be upgraded or extended with heterogeneous devices (as the need arises). For a manufacturer, it is tempting to stick to closed proprietary protocols as it allows them to retain control of the generated data. This limits the customers' choices, and forces them to use the proprietary services only. There are a number of ongoing projects that are investigating as to how different metering devices could be made to cooperate.[115] Besides the hardware related interoperability issues, standard compliance is also needed to build higher level services. This applies both to services for monitoring individual buildings; and to services for monitoring, control and optimization of heterogeneous power grids where multiple small and medium sized energy providers need to be incorporated.

With the readily available hardware and software technologies, the described CPS for smart buildings/households supported by WSN can be used both for new constructions, and also, for cost efficient retrofitting of existing infrastructure.

12 A Final Note

The article outlined the basic building blocks of a cyber-physical system, and showed how a wireless sensor network could potentially fit within its scope to support CPS applications. It reflected on the existing requirements and newer demands of WSN; and subsequently, reported the latest developments in software and services (operating system, simulator, emulator, programming abstraction, visualization, IP-based communication and security, time and location, network monitoring and management, and dependable assurances) for designing WSNs that are in-line with the goals of CPS/IoT. Finally, the article revisited some of the important concepts with a case-study of WSN-driven CPS for smart buildings.

Acknowledgements

This work has been supported by the European Commission through the makeSense, Calipso and RELYonIT projects; and also by SSF, VINNOVA and the Swedish Energy Authority. P. Misra carried out his work on this article during the tenure of an ERCIM 'Alain Bensoussan' Fellowship Programme funded by the European Union Seventh Framework Programme (FP7/2007–2013) under grant agreement *no.* 246016.

Received xx xxxx 2013.

References

1. M. Ceriotti, M. Corra, L. D'Orazio, R. Doriguzzi, D. Facchin, S.T. Guna, G.P. Jesi, R. Lo Cigno, L. Mottola, A.L. Murphy, M. Pescalli, G.P. Picco, D. Pregnolato, and C. Torghele. Is there light at the ends of the tunnel? wireless sensor networks for adaptive lighting in road tunnels. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks*, IPSN '11, pages 187–198, 2011.
2. K.-D. Kim and P.R. Kumar. Cyber-physical systems: A perspective at the centennial. *Proceedings of the IEEE*, 100(Special Centennial Issue): 1287–1308, 2012.
3. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA, 2010. ACM.
4. K.-J. Park, R. Zheng, and X. Liu. Editorial: Cyber-physical systems: Milestones and research challenges. *Computer Communication*, 36(1): 1–7, December 2012.
5. M. Conti, S.K. Das, C. Bisdikian, M. Kumar, L.M. Ni, A. Passarella, G. Roussos, G. Tröster, G. Tsudik, and F. Zambonelli. Fast track article: Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber-physical convergence. *Pervasive Mobile Computing*, 8(1): 2–21, February 2012.
6. I.F. Akyildiz, S. Weilian, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8): 102–114, 2002.
7. J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.
8. J.A. Stankovic and T. He. Energy management in sensor networks. *Philosophical Transactions of the Royal Society A*, 370(1958): 52–67, 2012.
9. A. Dunkels, B. Gronvall, and T. Voigt. Contiki—a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.
10. P. Levis. Experiences from a decade of TinyOS development. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 207–220, Berkeley, CA, USA, 2012. USENIX Association.
11. A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 29–42, New York, NY, USA, 2006. ACM.
12. A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 85–98, New York, NY, USA, 2003. ACM.
13. M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making sensor networks IPv6 ready. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 421–422, New York, NY, USA, 2008. ACM.
14. A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 335–349, New York, NY, USA, 2007. ACM.
15. A. Dunkels, N. Finne, J. Eriksson, and T. Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 15–28, New York, NY, USA, 2006. ACM.
16. N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling large-scale storage in sensor networks with the Coffee file system. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 349–360, Washington, DC, USA, 2009. IEEE Computer Society.
17. A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 28–32, New York, NY, USA, 2007. ACM.
18. S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing communication in 6LoWPAN with Compressed IPsec. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems*, DCOSS '11, pages 1–8. IEEE, 2011.
19. N. Tsiftes and A. Dunkels. A database in every sensor. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 316–332, New York, NY, USA, 2011. ACM.
20. Contiki wiki. https://github.com/contiki-os/contiki/wiki.

21. A. Varga. The OMNET++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, pages 319–324, Prague, Czech Republic, June 2001. SCS—European Publishing House.

22. T.R. Henderson, M. Lacage, G.F. Riley, C. Dowell, and J.B. Kopena. Demo: Network simulations with the NS-3 simulator. In *Proceedings of the 2008 ACM Conference on Data Communication*, SIGCOMM '08, New York, NY, USA, 2008. ACM.

23. Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA, 2003. ACM.

24. F. Österlind. *Improving Low-Power Wireless Protocols with Timing-Accurate Simulation*. PhD thesis, Uppsala University, 2011.

25. J. Eriksson, F. Österlind, N. Finne, A. Dunkels, N. Tsiftes, and T. Voigt. Accurate network-scale power profiling for sensor network simulators. In *Proceedings of the 6th European Conference on Wireless Sensor Networks*, EWSN '09, pages 312–326, Berlin, Heidelberg, 2009. Springer-Verlag.

26. J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P.J. Marrón. COOJA/MSPSim: interoperability testing for wireless sensor networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 27: 1–27: 7, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

27. J.G. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler. Beyond interoperability: pushing the performance of sensor network IP stacks. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 1–11, New York, NY, USA, 2011. ACM.

28. J. Ko, N. Tsiftes, A. Dunkels, and A. Terzis. Pragmatic low-power interoperability: ContikiMAC vs TinyOS LPL. In *Proceedings of the 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, SECON '12, pages 94–96, 2012.

29. L. Mottola and G.P. Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Computing Surveys*, 43(3): 19: 1–19: 51, April 2011.

30. R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *Proceedings of the First IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS'05, pages 126–140, Berlin, Heidelberg, 2005. Springer-Verlag.

31. S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1): 122–173, March 2005.

32. D.E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. In *Proceedings of the First International Workshop on Embedded Software*, EMSOFT '01, pages 114–130, London, UK, UK, 2001. Springer-Verlag.

33. L. Luo, T.F. Abdelzaher, T. He, and J.A. Stankovic. EnviroSuite: An environmentally immersive programming framework for sensor networks. *ACM Transactions on Embedded Computing Systems*, 5(3): 543–576, August 2006.

34. L. Mottola and G.P. Picco. Logical neighborhoods: a programming abstraction for wireless sensor networks. In *Proceedings of the Second IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS'06, pages 150–168, Berlin, Heidelberg, 2006. Springer-Verlag.

35. L. Mottola and G.P. Picco. Programming wireless sensor networks with logical neighborhoods. In *Proceedings of the First International Conference on Integrated Internet Ad Hoc and Sensor Networks*, InterSense '06, New York, NY, USA, 2006. ACM.

36. M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation—Volume 1*, NSDI'04, pages 3–3, Berkeley, CA, USA, 2004. USENIX Association.

37. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pages 1–11, New York, NY, USA, 2003. ACM.

38. R. Newton, G. Morrisett, and M. Welsh. The regiment macroprogramming system. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, IPSN '07, pages 489–498, New York, NY, USA, 2007. ACM.

39. P. Costa, L. Mottola, A.L. Murphy, and G.P. Picco. TeenyLIME: transiently shared tuple space middleware for wireless sensor networks. In *Proceedings of the International Workshop on Middleware for Sensor Networks*, MidSens '06, pages 43–48, New York, NY, USA, 2006. ACM.

40. C.-L. Fok, G.-C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, ICDCS '05, pages 653–662, Washington, DC, USA, 2005. IEEE Computer Society.

41. S. Li, S.H. Son, and J.A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In *Proceedings of the 2nd International Conference on Information Processing in Sensor Networks*, IPSN '03, pages 502–517, Berlin, Heidelberg, 2003. Springer-Verlag.

42. A. Bakshi, V.K. Prasanna, J. Reich, and D. Larner. The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In *Proceedings of the 2005 Workshop on End-to-End, Sense-and-Respond Systems, Applications and Services*, EESR '05, pages 19–24, Berkeley, CA, USA, 2005. USENIX Association.

43. C. Frank and K. Römer. Algorithms for generic role assignment in wireless sensor networks. In *Proceedings of*

the 3rd International Conference on Embedded Networked Sensor Systems, SenSys '05, pages 230–242, New York, NY, USA, 2005. ACM.

44. C. Frank and K. Römer. Solving generic role assignment exactly. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS '06, pages 177–177, Washington, DC, USA, 2006. IEEE Computer Society.
45. K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, pages 99–110, New York, NY, USA, 2004. ACM.
46. W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, and M.A. Perillo. Middleware linking applications and networks. *IEEE Network*, 18, 2004.
47. M. Ceriotti, L. Mottola, G.P. Picco, A.L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 277–288, Washington, DC, USA, 2009. IEEE Computer Society.
48. F. Casati, F. Daniel, G. Dantchev, J. Eriksson, N. Finne, S. Karnouskos, P.M. Montero, L. Mottola, F.J. Oppermann, G.P. Picco, A. Quartulli, K. Römer, P. Spiess, S. Tranquillini, and T. Voigt. Towards business processes orchestrating the physical enterprise with wireless sensor networks. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1357–1360, Piscataway, NJ, USA, 2012. IEEE Press.
49. K. Romer and J. Ma. PDA: Passive distributed assertions for sensor networks. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 337–348, Washington, DC, USA, 2009. IEEE Computer Society.
50. P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. *SIGOPS Operating Systems Review*, 36(5):85–95, October 2002.
51. N. Brouwers, K. Langendoen, and P. Corke. Darjeeling, a feature-rich VM for the resource poor. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 169–182, New York, NY, USA, 2009. ACM.
52. F. Aslam, L. Fennell, C. Schindelhauer, P. Thiemann, G. Ernst, E. Haussmann, S. Rührup, and Z.A. Uzmi. Optimized java binary and virtual machine for tiny motes. In *Proceedings of the 6th IEEE international Conference on Distributed Computing in Sensor Systems*, DCOSS'10, pages 15–30, Berlin, Heidelberg, 2010. Springer-Verlag.
53. L. Gu and J.A. Stankovic. t-kernel: providing reliable OS support to wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 1–14, New York, NY, USA, 2006. ACM.
54. G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Internet proposed standard RFC 4944, September 2007.
55. M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making sensor networks IPv6 ready. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 421–422, New York, NY, USA, 2008. ACM.
56. J.W. Hui and D.E. Culler. IP is dead, long live IP for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 15–28, New York, NY, USA, 2008. ACM.
57. M. Buettner, G.V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 307–320, New York, NY, USA, 2006. ACM.
58. A. Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, Swedish Institute of Computer Science, December 2011.
59. Y. Sun, O. Gurewitz, and David B. Johnson. RI-MAC: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 1–14, New York, NY, USA, 2008. ACM.
60. 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 *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 1–14, New York, NY, USA, 2010. ACM.
61. S. Duquennoy, F. Österlind, and A. Dunkels. Lossy links, low power, high throughput. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 12–25, New York, NY, USA, 2011. ACM.
62. T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 routing protocol for low power and lossy networks, March 2012. RFC 6550.
63. O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 1–14, New York, NY, USA, 2009. ACM.
64. S. Duquennoy, G. Grimaud, and J.-J. Vandewalle. Serving embedded content via web applications: model, design and experimentation. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, EMSOFT '09, pages 117–126, New York, NY, USA, 2009. ACM.
65. D. Yazar and A. Dunkels. Efficient application integration in IP-based sensor networks. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '09, pages 43–48, New York, NY, USA, 2009. ACM.

66. Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP) draft-ietf-core-coap-13, December 2012.
67. M. Kovatsch, S. Duquennoy, and A. Dunkels. A low-power CoAP for Contiki. In *Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, MASS '11, pages 855–860, Washington, DC, USA, 2011. IEEE Computer Society.
68. M. Kovatsch, M. Lanter, and S. Duquennoy. Actinium: A RESTful runtime container for scriptable Internet-of-Things applications. In *Proceedings of the 3rd International Conference on the Internet of Things*, IOT '12, pages 135–142, 2012.
69. M.S. Familiar, J.F. Martínez, I. Corredor, and C. García-Rubio. Building service-oriented smart infrastructures over wireless ad hoc sensor networks: A middleware perspective. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 56(4): 1303–1328, March 2012.
70. O. Garcia-Morchon, S.L. Keoh, S. Kumar, P. Moreno-Sanchez, F. Vidal-Meca, and J.H. Ziegeldorf. Securing the IP-based Internet-of-Things with HIP and DTLS. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 119–124, New York, NY, USA, 2013. ACM.
71. O. Garcia-Morchon, S. Keoh, S. Kumar, R. Hummen, and R. Struik. Security Considerations in the IP-based Internet-of-Things, March 2013. http://tools.ietf.org/html/draft-garcia-core-security-05
72. B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751 (Proposed Standard), January 2010. http://www.ietf.org/rfc/rfc5751.txt
73. E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, January 2012. http://www.ietf.org/rfc/rfc6347.txt
74. S. Raza, H. Shafagh, K. Hewage, H. Hummen, and T. Voigt. Lithe: Lightweight Secure CoAP for the Internet of Things. *IEEE Sensors Journal*, 2013.
75. C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), September 2010. http://www.ietf.org/rfc/rfc5996.txt
76. S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt. Secure communication for the Internet of Things—a comparison of link-layer security and IPsec for 6LoWPAN. *Security and Communication Networks*, 2012.
77. T. Heer. *Direct End-to-Middle Authentication in Cooperative Networks*, volume 3 of *Reports on Communications and Distributed Systems*. Shaker, 2012.
78. S. Raza, L. Wallgren, and T. Voigt. SVELTE: Real-time Intrusion Detection in the Internet of Things. *Ad Hoc Networks, Elsevier*, 2013.
79. I.E. Bagci, S. Raza, T. Chung, U. Roedig, and T. Voigt. Combined Secure Storage and Communication for the Internet of Things. In *10th IEEE International Conference on Sensing, Communication, and Networking (SECON'13)*, pages 1–9, New Orleans, USA, June 2013.
80. C. Lenzen, T. Locher, P. Sommer, and R. Wattenhofer. Clock synchronization: Open problems in theory and practice. In *36th International Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM '10, January 2010.
81. J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, OSDI '02, pages 147–163, New York, NY, USA, 2002. ACM.
82. S. Ganeriwal, R. Kumar, and M.B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 138–149, New York, NY, USA, 2003. ACM.
83. M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 39–49, New York, NY, USA, 2004. ACM.
84. F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '11, pages 73–84. ACM/IEEE, April 2011.
85. K. Romer, P. Blum, and L. Meier. *Handbook of Sensor Networks: Algorithms and Architectures*, chapter Time Synchronization and Calibration in Wireless Sensor Networks, pages 199–238. Wiley-Interscience, 2005.
86. Y. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation—Volume 2*, NSDI '05, pages 217–230, Berkeley, CA, USA, 2005. USENIX Association.
87. M. Mauve, A. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, 15(6): 30–39, 2001.
88. Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, pages 70–84, New York, NY, USA, 2001. ACM.
89. L. Lazos and R. Poovendran. SeRLoc: Robust localization for wireless sensor networks. *ACM Transactions on Sensor Networks*, 1(1): 73–100, August 2005.
90. K.D. Frampton. Acoustic self-localization in a distributed sensor network. *IEEE Sensors Journal*, 6(1): 166–72, 2006.
91. P. Misra, D. Ostry, N. Kottege, and S. Jha. TWEET: an envelope detection based broadband ultrasonic ranging system. In *Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '11, pages 409–416, New York, NY, USA, 2011. ACM.

92. P. Misra, N. Kottege, B. Kusy, D. Ostry, and S. Jha. Acoustical ranging techniques in embedded wireless sensor networked devices. *ACM Transactiosn on Sensor Networks (to appear)*.

93. P. Misra, D. Ostry, and S. Jha. Improving the coverage range of ultrasound-based localization systems. In *Wireless Communications and Networking Conference*, WCNC '11, pages 605–610. IEEE, 2011.

94. T. He, C. Huang, B.M. Blum, J.A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, MobiCom '03, pages 81–95, New York, NY, USA, 2003. ACM.

95. Y. Shang, W. Ruml, Y. Zhang, and M.P.J. Fromherz. Localization from mere connectivity. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, MobiHoc '03, pages 201–212, New York, NY, USA, 2003. ACM.

96. N. Wirström, P. Misra, and T. Voigt. Poster abstract: Spray, embracing multimodality. In *The 10th European Conference on Wireless Sensor Networks*, EWSN '13, 2013.

97. N.B. Priyantha. *The Cricket Indoor Location System*. PhD thesis, Massachusetts Institute Of Technology, 2005.

98. P. Misra, W. Hu, M. Yang, and S. Jha. Efficient cross-correlation via sparse representation in sensor networks. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*, IPSN '12, pages 13–24, New York, NY, USA, 2012. ACM.

99. G.S. Sidhu, A. Kamboj, P. Misra, S. Kanhere, and S. Jha. Poster abstract: u-BeepBeep: Low power acoustic ranging on mobile devices. In *The 10th European Conference on Wireless Sensor Networks*, EWSN '13, 2013.

100. R. Jurdak, X.R. Wang, O. Obst, and P. Valencia. Wireless sensor network anomalies: Diagnosis and detection strategies. In Andreas Tolk and Lakhmi Jain, editors, *Intelligence-Based Systems Engineering*. Springer, 2011.

101. N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, pages 255–267, New York, NY, USA, 2005. ACM.

102. G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceeedings of the 2nd European Workshop on Wireless Sensor Networks*, EWSN '05, pages 121–132, 2005.

103. S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *Proceedings of the 3th Annual IEEE Communication Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, SECON '06, pages 575–584, Reston, VA, September 2006.

104. L. Luo, T. He, G. Zhou, L. Gu, T.F. Abdelzaher, and J.A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor net-works with envirolog. In *Proceeedings of the 25th IEEE International Conference on Computer Communications*, INFOCOM '06, pages 1–14, 2006.

105. A.R. Bhadriraju, S. Bhaumik, Y.S. Lohith, M.C. Brinda, A. Svr, and M. Hegde. 6PANview: Application performance conscious network monitoring for 6lowpan based wsns. In *Proceeedings of the 18th National Conference on Communications*, NCC '12, pages 1–5, 2012.

106. B.-R. Chen, G. Peterson, G. Mainland, and M. Welsh. LiveNet: Using passive monitoring to reconstruct sensor network dynamics. In *Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS '08, pages 79–98, Berlin, Heidelberg, 2008. Springer-Verlag.

107. M. Keller, J. Beutel, and L. Thiele. How was your journey?: uncovering routing dynamics in deployed sensor networks with multi-hop network tomography. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 15–28, New York, NY, USA, 2012. ACM.

108. WirelessHART Overview. http://www.hartcomm.org/protocol/wihart/wireless overview.html

109. ISA100, Wireless Systems for Automation. www.isa.org/ISA100.

110. J.A. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and obligations for physical computing systems. *Computer*, 38(11): 23–31, November 2005.

111. F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 1–14, New York, NY, USA, 2012. ACM.

112. F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Virtual synchrony guarantees for cyber-physical systems. In *Proceedings of the 32nd IEEE International Symposium on Reliable Distributed Systems*, SRDS '13, 2013. To appear.

113. RELYonIT. http://www.relyonit.eu/

114. Sofie Jansson. Fuktskada i hus eller bostad? bostadsjuristerna. http://tinyurl.com/cvzjd8q, May 2013.

115. Tomas Augustsson. Forskarna kopplar upp och ihop. http://tinyurl.com/cekjphk, April 2013.

Prasant Misra is a postdoctoral fellow at the Swedish Institute of Computer Science. He received his Ph.D. from the University of New South Wales (Australia) in 2012. His current research interests include energy efficient location sensing and information processing techniques with the focus on sensornets for Cyber-physical Systems. His professional and research contributions have been recognized by numerous awards, of which the more prestigious are: the ERCIM Alain Bensoussan (Marie Curie) Fellowship 2012; the Google CSE Ph.D. Travel Prize 2011; the Best International Indian Student Academic Achievement Award (ICA, UNSW) 2010; the AusAID Australia Awards (Leadership Program) 2008; and (team member) the Best Project Award Q4–06, Keane Inc. 2007. He has published papers in premier sensornet forums such as ACM/IEEE IPSN and ACM TOSN. Prasant has many years of experience in technology development for: Keane Inc. (now a unit of NTT Data Corporation), India; CSIRO ICT Centre, Australia; and Red Lotus Technologies, USA. He was the Editor of the CIRCUIT newsletter and the Vice-chair of the GOLD Affinity group of the IEEE NSW Section 2012; and the Chair of the IEEE UNSW Student Branch 2010–11.

Luca Mottola is an assistant professor at Politecnico di Milano (Italy) and a senior researcher at the Swedish Institute of Computer Science. Previously, he was a research scholar at the University of Southern California (USA). He completed his Ph.D. at Politecnico di Milano (Italy) in 2008, after obtaining a M.Sc. in Computer Science from the University of Illinois at Chicago (USA) and a M.Sc. in Computer Engineering from Politecnico di Milano (Italy). His research interests focus on modern networked embedded systems, with the current focus on the "Internet of Things" and Cyberphysical Systems. Out of his research, he was listed amongst Postscapes "Internet of Things Top 100 Thinkers", and has received: the Cor Baayen Award for the most promising young researcher in computer science and applied mathematics; the EWSN/CONET European Best Ph.D. Thesis Award; the Best Paper Award at ACM/IEEE IPSN 2011; the Best Paper Award at ACM/IEEE IPSN 2009; and the Best Demo Award at ACM SenSys 2007.

Shahid Raza is a senior researcher at the Swedish Institute of Computer Science (SICS). He completed his Ph.D. at Mälardalen University Västeras and SICS Swedish ICT, Sweden in 2013. He holds a Technology of Licentiate degree (Ph.Lic.) from Mälardalen University Västeras. He also received a Master of Science degree from KTH, The Royal Institute of Technology, Stockholm. Raza received his Bachelors degree from UAAR Pakistan, where he was awarded with the *Gold Medal* for his extraordinary performance. His research interests include security issues in wireless sensor networks in general and Internet of Things (IoT) in particular. [www.ShahidRaza.info]

Simon Duquennoy is a senior researcher at SICS, the Swedish Institute of Computer Science. He obtained his Ph.D. from Universite de Lille 1 (INRIA, CNRS, France) in 2010. From October 2010 to September 2011, he was the holder of an ERCIM Alain Bensoussan post-doctoral fellowship, and completed his postdoc at SICS. His research interests are at the intersection between operating systems and networking for constrained embedded devices, with a focus on IP-based sensor networks and the Internet of Things. He is the author or contributor of a number of open-source projects in this area, including Contiki, Smews, and SicsthSense. Over the last years, he has published papers in the two most prestigious conferences in the sensor network community, namely ACM/IEEE IPSN and ACM SenSys.

Nicolas Tsiftes is a researcher in the Networked Embedded Systems group of the Swedish Institute of Computer Science. He received the M.S. degree in computer science from Stockholm University, Stockholm, Sweden, and is currently working towards the Ph.D. degree at Uppsala University. His main research interest is system design for resource-constrained embedded devices. He has published numerous papers concerning this topic in conferences such as ACM SenSys and ACM/IEEE IPSN. Beside conducting research, he is a member of the development team of the Contiki operating system project, and has made several open-source software contributions, including the Coffee file system, the Antelope database system, and ContikiRPL.

Joel Höglund is a researcher in the Networked Embedded Systems group of the Swedish Institute of Computer Science. He made his master of science at the Royal Institute of Technology in Stockholm, Sweden. His main research interests lies in self configuration, routing and adaptation of standards. He has been working with using wireless sensors real world application areas such as Smart Grid monitoring and control, and structural health monitoring.

Thiemo Voigt manages the Networked Embedded Systems group at the Swedish Institute of Computer Science. He shares his time between SICS and Uppsala University where he is Professor for Wireless Sensor Networks. He is the director of the Uppsala VINN Excellence Center for Wireless Sensor Networks WISENET, a ten year multi-disciplinary research center. His main interests are in networking and system issues in wireless sensor networks. He has published papers at flagship sensor networking conferences such as ACM SenSys and IEEE/ACM IPSN and received awards for several of these publications. He has also been TPC co-chair for IEEE/ACM IPSN and EWSN, the European Conference on Wireless Sensor Networks.