# A Comparative Study of Recent Wireless Sensor Network Simulators

IVAN MINAKOV and ROBERTO PASSERONE, University of Trento
ALESSANDRA RIZZARDI and SABRINA SICARI, Università degli Studi dell'Insubria

Over recent years, the continuous interest in *wireless sensor networks* (WSNs) has led to the appearance of new modeling methods and simulation environments for WSN applications. A broad variety of different simulation tools have been designed to explore and validate WSN systems before actual implementation and real-world deployment. These tools address different design aspects and offer various simulation abstractions to represent and model real-world behavior. In this article, we present a comprehensive comparative study of mainstream open-source simulation tools for WSNs. Two benchmark applications are designed to evaluate the frameworks with respect to the simulation runtime performance, network throughput, communication medium modeling, packet reception rate, network latency, and power consumption estimation accuracy. Such metrics are also evaluated against measurements on physical prototypes. Our experiments show that the tools produce equivalent results from a functional point of view and capacity to model communication phenomena, while the ability to model details of the execution platform significantly impacts the runtime simulation performance and the power estimation accuracy. The benchmark applications are also made available in the public domain for further studies.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems; I.6 [**Computing Methodologies**]: Simulation and Modeling; C.2 [**Computer Systems Organization**]: Computer-Communication Networks

General Terms: Design, Measurement, Performance, Verification

Additional Key Words and Phrases: WSN simulation tools and environments, system architectures, modeling, performance, power consumption estimation

**ACM Reference Format:**
Ivan Minakov, Roberto Passerone, Alessandra Rizzardi, and Sabrina Sicari. 2016. A comparative study of recent wireless sensor network simulators. ACM Trans. Sen. Netw. 12, 3, Article 20 (July 2016), 39 pages.
DOI: http://dx.doi.org/10.1145/2903144

## 1. INTRODUCTION

In recent years, we have witnessed a steadily increasing interest in the use of networked embedded systems in a large variety of applications. In particular, wireless sensor network (WSN) systems have great potential to bring new smart sensing environments and replace traditional wired sensor systems with more flexible and less expensive solutions. Despite the many research activities pursued at several academic and industrial centers, the area of WSN is still open, with many aspects related to architectures, protocols, and software layers not yet well defined and standardized [Mottola

Authors' addresses: I. Minakov and R. Passerone, Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, via Sommarive 9, 38123, Trento, Italy; emails: {ivan.minakov, roberto.passerone}@unitn.it; A. Rizzardi and S. Sicari, Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, via Mazzini 5, 21100, Varese, Italy; emails: {alessandra.rizzardi, sabrina.sicari}@uninsubria.it.

DOI: http://dx.doi.org/10.1145/2903144

et al. 2010]. In contrast to traditional networks (e.g., TCP/IP), there are no common design patterns, standard APIs, and hardware solutions to be applied for all possible applications. For instance, it is often the case that a sensor network system is designed essentially from the ground up, a situation that in turn requires an individual design approach in order to satisfy a range of application-specific demands. In addition to the design issues inherent to traditional networks, WSN development faces a new set of specific objectives, which were considered of secondary importance before. These include, for instance, the management of the limited hardware resources present in the network node, the limited battery power, and custom ad hoc communication infrastructure [Rossi et al. 2014]. In addition, the wide choice of available nonstandard MAC and routing protocols complicates the design process even more, making it difficult to choose the right solution that best fits a particular system [Langendoen and Meier 2010]. It is unlikely that this situation will change in the near future due to the very application-specific nature of the WSN systems. As a consequence, it is essential to be able to explore and validate the various aspects of a complex design space (induced by different design choices) before proceeding to the actual implementation and real-world deployment.

The most common approaches for the evaluation of WSN systems include modeling, simulation, and prototyping. Prototyping is a widely used method, which provides an efficient evaluation of the tested systems [Srinivasan et al. 2010; Li et al. 2013]. Despite its obvious benefits, prototyping has a number of crucial disadvantages in terms of high cost and time required to build, run, and debug experimental systems. When prototyping is impractical, and during the early phases of the design space exploration process, models can be used to analyze the system both analytically and through simulation [Pinto et al. 2006]. Analytical methods are commonly used to statically evaluate classes of techniques against some optimization criteria [Jung et al. 2009; Dietrich and Dressler 2009; Palopoli et al. 2011]. When the designer is interested in the dynamic behavior of the system, models are typically used in conjunction with simulation [He et al. 2006], which is currently one of the most widely used techniques to explore and evaluate the design of WSN applications. For this purpose, various simulation environments have been developed in recent years. These tools offer a diversified set of facilities to model, examine, and test aspects such as application algorithms, communication protocols, cooperative network behavior, and power-saving techniques. Each simulation tool is designed for a specific purpose and has a different level of accuracy and suitability depending on the target application. Because of this, it is essential to be aware of the strengths and weaknesses of the available simulators in order to use them properly according to the purpose they were designed for. Many studies and surveys on available WSN simulation environments have been recently presented in the literature [Singh et al. 2008; Khemapech et al. 2005; Egea-Lopez et al. 2005; Korkalainen et al. 2009; Kuorilehto et al. 2008]. These provide a general picture on the state of the art of WSN simulation toolkits. However, only a few of the published papers exhibit practical studies and real test cases to explore and compare the accuracy and the usability of the various tools [Lessmann et al. 2008b; Timm-Giel et al. 2008; Lee et al. 2007].

In this article, we present a practical comparative study of a number of recent open-source simulation tools for WSN systems. Our main objective is to evaluate and assess different mainstream simulation environments with respect to the general usability, runtime performance, model scalability, network channel model accuracy, network failure rate, MAC protocol performance, network latency, and accuracy of the power consumption estimation. In this article, we study the recent releases of the Castalia [Boulis 2007], MiXiM [Köpke et al. 2008], WSNet [Guillaume et al. 2007], TOSSIM [Levis et al. 2003], and COOJA [Osterlind et al. 2006] simulators, due to their popularity in the area, their completeness, and their level of current development support. In addition, we also evaluate the PASES simulation framework [Somov et al. 2009; Minakov and Passerone 2013], which we have recently extended with platform, protocol, and network simulation capabilities, offering an additional tradeoff point among accuracy, performance, and modeling abstraction. All these tools have much in common and offer similar functionality that allows one to compare them to each other. The simulations are also compared to small-scale real deployments to verify the results against realistic executions.

All evaluated metrics mentioned previously are obtained by means of two separate benchmark applications that were implemented natively and equally in each of the studied toolkits. The first test case is designed to assess and compare simulation performance, network channel models, and the MAC protocol functionality. In the second test case, we assess the accuracy of the power consumption simulation results by comparing them with reference values obtained on a real testbed. We consider these models as an additional contribution of this work, as they could be used as benchmarks for future comparisons and tool development. For this reason, all the test case applications studied in this article are open and available for download at http://eecs.disi.unitn.it/pases in the Download section.

This article is structured as follows. In Section 2, we first consider mainstream WSN simulation environments and overview related reports and practical studies of various simulation tools. Then, the benchmark test cases utilized in our study are described in Section 3. The details of the design process and the challenges of implementing a set of equivalent benchmark models in different tools are considered in Section 4. Section 5 presents and discusses the results obtained from the simulations. Finally, we conclude the article in Section 6.

## 2. RELATED WORK

In this section, we provide a broad overview of the available simulation environments for both generic networked and WSN applications. Further, we outline the results from published comparative evaluation studies and reports on different simulation tools.

### 2.1. Simulation Environments

The whole set of available simulators for WSNs might be roughly divided into three broad categories: the *general-purpose* (generic) network simulators, the *network-oriented* frameworks, and the *sensor-node-oriented* frameworks [Kuorilehto et al. 2008] specially designed for simulation of WSN applications. The generic network simulators are intended to model and evaluate conventional networks such as the IEEE 802 family of LAN/MAN systems. The WSN network-oriented tools focus on the network aspects of WSN systems and typically offer highly realistic and accurate models of the communication infrastructure. In general, these tools provide efficient simulation for large-scale network models while simplifying and abstracting the underlying hardware-software details of the real target systems. Another kind of simulator, referred to in the literature as either *sensor node simulators* or *emulators*, targets internal operations of sensor nodes while still providing simple lightweight communication models. These tools are generally intended to validate and test platform-specific software on top of the virtual model of the target hardware. Most of the emulators provide cycle-accurate execution models of the target binary files for various node architectures and offer different profiling facilities to track the application execution process. Despite the highly detailed simulation results of the internal behavior of the nodes, these tools are typically bound to a particular hardware architecture and are not capable of simulating others. The next subsection reviews the state of the art for all of these categories.

*2.1.1. Generic Network Simulators.* NS-2 (Network Simulator 2) [Downard 2004] is one of the most popular general-purpose open-source network simulators. It provides a broad variety of simulation models for widely used IP network protocols. These include TCP/IP, routing, and multicasting protocols for conventional wired and wireless networks. NS-2 has earned its popularity thanks to its highly extensible object-oriented architecture based on a discrete-event engine. This allows users to extend the simulation functionality by adding custom components and libraries. Hence, plenty of third-party implementations of communication protocols, models, and add-ons are available in NS-2.

The NS-3 tool (Network Simulator 3) [Henderson et al. 2008] is a recent version of the Network Simulator (NS) family that is intended to replace NS-2. However, NS-3 is still in the active development process. In spite of its youth, NS-3 is a powerful tool for network modeling and optimization. It includes models for the most popular protocols including socket API, TCP/IP, IPv6, MANET routing, IEEE802.11, WiMAX, and so forth. Like NS-2, NS-3 also has a modular object-oriented architecture combined with a discrete-event simulation engine written in C++. Due to the the relative novelty of the tool, there are not many extensions available, especially ones suitable for WSN applications, and it provides considerably fewer available models than its predecessor.

OMNet++ [Varga 2001] is a component-based C++ simulation library and framework that provides deep analysis of network activities at the packet layer. OMNet++ provides a GUI front-end for simulation setups and result interpretation analysis. It exploits modules and channels to implement and connect simulation components, where components are connected in a hierarchical fashion via generic interfaces. OMNet++ has become a popular simulator framework due to its plain and neat architecture and multiple extensions for conventional and WSN systems.

*2.1.2. Network-Oriented Simulators.* Most of the available WSN network-oriented tools are based on the generic network simulation environments that gained popularity in traditional network research and design.

Castalia [Boulis 2007] is a popular WSN simulator based on the OMNet++ framework. It provides a first-order analysis of algorithms and protocols before actual implementation on a specific node platform. Detailed radio and communication modeling is a strong feature of Castalia. It includes an adjustable and highly accurate radio physical (PHY) model along with an advanced communication channel model based on empirically measured data. To handle packet delivery, Castalia provides dynamic calculation of radio Signal Interference and Noise Ratio (SINR) based on modulation type, signal strength, and other parameters. The PHY model, in turn, includes multiple states with transition delays among them, different transmission levels and modulations, and configurable RSSI (Received Signal Strength Indicator) and CCA (Clear Channel Assessment). Besides, Castalia allows modeling of mobile network units. Additionally, Castalia includes customizable models of the most popular MAC and routing protocols for WSN applications. It also provides models of on-board sensors, including effects such as sensing device noise and bias. Castalia offers power-aware modeling and power consumption reporting for radio components. Besides, it can support simple modeling of clock drift for CPUs. In addition, the simulator is well documented and provides design examples for making it easy to adopt by the end-user. Despite its advantages, Castalia does not include power models for the many typical hardware peripheral components that constitute a node platform, such as timers, analog-to-digital converters, and so forth. Additionally, it does not provide any battery model and hence it does not offer lifetime estimation. However, power analysis is not one of the stated purposes of Castalia. From the point of view of usability, we point out the lack in Castalia of an efficient postprocessing tool with GUI support.

MiXiM (Mixed Simulator) [Köpke et al. 2008] is another mainstream simulation environment based on OMNet++. It is a joint project that combines various simulation tools designed for wireless and mobile simulations. As Castalia, MiXiM provides detailed models of the communication channel and the radio PHY layer. It offers various radio signal propagation models including a time-variant Signal-to-Noise Ratio (SNR) level model, a model for path loss, shadowing, and large- and small-scale fading models. The MiXiM's PHY layer provides a high level of customization in terms of modulation types, sensitivity, output signal power, and radio hardware operating states with power and timing parameters. Along with Castalia, MiXiM offers power-aware modeling and power reporting for radio modules. Additionally, mobile networks are also supported. In general, this tool has much in common with Castalia. As a relative disadvantage of MiXiM, it actually lacks models for hardware peripheral components typical of WSN nodes and a graphical representation tool for simulation results is still absent.

PAWiS [Weber et al. 2007] is another simulator built on top of the OMNet++ framework. The main focus of PAWiS is power analysis for a wide range of WSN applications. It provides facilities to simulate a whole sensor network along with accurate modeling of internal activity for each single node in the network. The model of the sensor node is represented as a hierarchical composition of HW/SW modules. Modules are interconnected via functional interfaces. Its architecture provides an extensible and flexible simulation infrastructure, where users can customize and add new simulation modules and components. PAWiS defines two types of simulation components. Software components reflect software tasks and application, routing, and MAC protocols. Hardware components represent hardware models of the sensor node, including the CPU, timers, Analog-to-Digital Converter (ADC), and radio transceiver. The tasks that simulate the work of the dedicated hardware are used to report the power consumption to the power supply module. The biggest disadvantages of PAWiS are (1) lack of support (most recent release was in 2009), (2) insufficient documentation, and (3) lack of clear design examples.

WSNet [Guillaume et al. 2007] is a standalone simulation tool written in C++ for Linux. WSNet offers a wide range of radio medium models including a basic ideal physical layer with no interference, no path loss, and a fixed radio range, as well as complex and precise ones exploiting Friis propagation formula [Rappaport 2002], Rayleigh fading [Rappaport 2002], multiple frequencies and complex correlation properties among frequencies, complex antenna radiation patterns, and so forth. The simulated nodes in WSNet are built as a composition of blocks that represent either a hardware or a software component of the node. The available model units include, among others, the following components and phenomena: mobility, energy source, application, routing protocols, MAC protocols, radio interface, and antenna. Additionally, WSNet provides an option to simulate dynamic physical phenomena (e.g., fire) and physical measures (e.g., temperature, humidity). This feature gives the opportunity to simulate sensor-driven applications and networks. The lack of a GUI environment for interpreting the simulation results might be considered a disadvantage of the WSNet toolkit.

DANSE [Baz and Pearce 2011] is a standalone Windows simulator with rich GUI support. This tool was originally designed for teaching purposes and hence it is optimized for ease of use. It provides an excellent GUI to configure various model options, run simulations, and study results. Results comprise, among others, simulation event logs, packet flow records, and visual representation of the nodes' PHY radio activity. DANSE offers a relatively simple communication channel model that captures the most relevant transmission aspects. It simulates packet passing based on the frame length, distance, path loss, bit error rate, and channel capacity. This toolkit also provides a set of the most popular MAC protocols. Additionally, DANSE includes experimentally based power models for radio PHY components. Despite its rich GUI features, DANSE does not provide any input interface for the user to define applications, MAC models, or new hardware options, which, in fact, limits the applicability of this tool. The only option available for the user is to tune the duty cycle and the packet rate of the available predefined models, which might be sufficient for the simplest kind of applications.

Another simulation tool with extensive GUI support is NetTopo [Shu et al. 2011], a Java-language-based open-source framework for simulation and visualization of large-scale WSNs. This framework provides a friendly GUI that makes it easy to use and study network models of interest. The important feature of the NetTopo tool is its efficient simulation scalability that allows it to handle extremely large-scale network models consisting of several thousand nodes. In addition, NetTopo provides a flexible framework for visualizing real sensor network testbeds and enables interactive communications among simulated sensor networks and visualized testbeds. Although NetTopo achieves a high level of scalability, the detailed and accurate representation of the node's hardware is penalized. This makes this tool not very well suited for the power consumption analysis and lifetime estimation of WSN applications.

PASES (Power-Aware Simulator for Embedded Systems) [Somov et al. 2009; Minakov and Passerone 2013] is a standalone Windows application written on top of the SystemC simulation engine [Grötker et al. 2002]. This toolkit was initially designed with the aim to perform accurate power consumption estimation for various WSN hardware platforms. It utilizes a flexible multilayer architecture that allows users to specify and assemble models of HW/SW platforms from a set of predefined components including CPUs, timers, ADCs, Flash, USART, radios, and other models. While PASES was initially oriented toward studying the sensor platform, several extensions have been developed to offer facilities to model and visually study the network communication behavior for both static and mobile nodes. The model of communication channel is based on the pass-log equation [Rappaport 2002] supplied with various interference models. This model is relatively simpler than the ones available in Castalia and MiXiM; however, it still provides a reliable communication environment. Additionally, the PASES component library includes a number of the most popular WSN MAC protocol models, which can be easily extended with new user-defined functionality. Simulation results in PASES consist of a number of log files for various network-related metrics along with energy-performance trace files for each individual peripheral component of every simulated node. The main benefit of the PASES toolkit is its plain and easy-to-use interface to assemble heterogeneous models of HW platforms and to organize them into a network. Additionally, it offers rich GUI support to interpret simulation results and perform lifetime analysis based on different battery models. One of the drawbacks is its relative complexity and dependency on external scripting tools such as Python, which provides a lot of the framework flexibility, but which might be considered an obstacle to get started with this framework. Additionally, because of the detailed hardware models, it suffers from lower runtime performance and scalability than other tools for large-scale network models.

Sense [Szymanski and Chen 2007] is a WSN simulator designed on top of COST [Chen et al. 2002b], a discrete-event simulation core. Sense implements each sensor node as a collection of components connected via unified input/output ports. Such an architecture provides a high level of extendibility of the simulation infrastructure. Sense includes models of the communication channel, duplex transceiver, and a wide range of MAC and routing protocols. Besides, it provides energy profiling for the radio transceiver and includes a linear battery model to perform simple lifetime analysis. However, Sense does not capture timing and power aspects of other hardware components present in the sensor node platform such as the CPU, timers, sensors, and so on.

Table I. WSN Simulation Tools: Comparative Table

| | **Castalia** | **MiXiM** | **PAWiS** |
|---|---|---|---|
| **Simulation engine** | OMNet++ | OMNet++ | OMNet++ |
| **Simulation input** | C++, ini files, nodes, protocols, application | C++, ini files, nodes, protocols, applications | C++, ini files, power models, nodes, protocols |
| **Scalability >100** | high | low | middle |
| **Communication channel model** | experimental SINR path loss model | path loss model, free space, log. shadowing | - |
| **Mobile nodes** | yes | yes | yes |
| **Supported HW platforms** | virtual platform | virtual platform | virtual platform |
| **RF PHY model** | cc2420, cc1100 | cc2420, cc1100 | cc2420 |
| **MAC** | TMAC, IEEE 802.15.4, custom MAC | IEEE802.15.4, BMAC, IEEE 802.11 | simple CSMA, IEEE 802.11, CSMA-MPS |
| **Routing** | - | flood, wise route | EADV |
| **Sensing channel** | yes | yes | yes |
| **Energy profiling** | simple radio | simple radio | for each HW component |
| **Battery/lifetime estimation** | no/no | linear/no | linear/no |

Table II. WSN Simulation Tools: Comparative Table

| | **WSNet** | **DANSE** | **PASES** |
|---|---|---|---|
| **Simulation engine** | custom | custom | SystemC |
| **Simulation input** | C++, XML, nodes, applications, protocols | C#, XML, nodes, apps, protocols | Python/C++, XML, power models, nodes, apps, protocols |
| **Scalability >100** | high | high | low |
| **Communication channel model** | free space, log. shadowing, Rayleigh fading | path loss model | path loss model |
| **Mobile nodes** | yes | yes | yes |
| **Supported HW platforms** | virtual platform | - | TelosB, Mica2, custom models |
| **RF PHY model** | cc2420, cc1100 | virtual PHY, custom | cc2420, custom |
| **MAC** | IEEE 802.11 | ALOHA, IEEE 802.11 | SMAC, IEEE 802.15.4 |
| **Routing** | greedy, geographic | flooding, AODV, Bellman Ford | flooding, AODV |
| **Sensing channel** | yes | no | yes |
| **Energy profiling** | simple radio | simple radio | each HW component |
| **Battery/lifetime estimation** | linear/yes | linear/yes | linear, relaxation/yes |

Table I and Table II provide a summary of some of the network-oriented simulators that we have discussed. In particular, Table I groups the simulators that are based on OMNet++, while those in Table II use either a custom simulation engine or SystemC. Despite the differences among the toolkits, these environments also share sufficient similarities, such as simulation input, available models for the wireless channel, physical layer and MAC level protocols, and so on, which make them amenable to a performance and accuracy comparison.

*2.1.3. Sensor Node Simulators.* We now present the characteristics of the sensor node simulators, which proved to be very useful when designing a network for future real deployment.

The TOSSIM simulation environment is included in the TinyOS [Levis et al. 2004] framework. It exploits the TinyOS component model and is fully integrated with it. This simulator provides code-level simulation of TinyOS applications that can be run on actual sensor network hardware. By replacing a small number of TinyOS components, TOSSIM simulates the behavior of the low-level hardware. It includes models for the CPU, ADCs, clocks, timers, and radio components. The TOSSIM simulation architecture provides a high level of scalability and execution speed for networks with a large number of sensor nodes. However, the abstract hardware model available in TOSSIM does not capture low-level details of timing and interrupts, which can be important for precise time-power analysis. The network simulation in TOSSIM is limited by the homogeneity of the execution model. It is able to run only the same program on each node in the network. Besides, the TOSSIM simulation model is not extensible and it supports only a single hardware platform model (Micaz [Polastre et al. 2005]). PowerTOSSIM [Perla et al. 2008] is a TOSSIM [Levis et al. 2003] extension that allows power analysis for TinyOS applications. This simulator provides code-level simulation of TinyOS applications that can be run on actual sensor network hardware. PowerTOSSIM equips the TOSSIM hardware model with a specific module, PowerState, to keep track of state activities for each simulated hardware component. Additionally, PowerTOSSIM provides CPU profiling support to report the number of cycles for each execution block. By combining the timing information with the energy model of the simulated hardware platform, PowerTOSSIM computes the total energy consumed by the node. However, PowerTOSSIM is included only in the first version of the TinyOS framework, which is not currently supported. Besides, simulation in TOSSIM and PowerTOSSIM is supported only for a single platform (Micaz [Polastre et al. 2005]).

AVRORA [Titzer et al. 2005] is one of the most widely used WSN emulation tools. It exploits cycle-accurate instruction simulation to run the code. AVRORA runs actual application binaries without the need to specially adapt them for simulation. It exploits an efficient method to handle large-scale networks with various events and interrupts happening. AVRORA is similar to TOSSIM in simulation runtime performance and scalability. Nevertheless, AVRORA solely supports AVR MCU cores [Myklebust 2004] and does not provide any extensions for other CPU architectures. Additionally, it includes only a single model for the radio transceiver component (CC1100). This significantly reduces flexibility and makes it impossible to simulate applications designed for non-AVR MCU-based platforms. AVRORA includes the AEON evaluating tool [Landsiedel et al. 2005] to estimate the power consumption of sensor nodes. AEON exploits the cycle-accurate execution model of AVRORA for precise timing and energy measurements. However, AEON supports energy modeling only for the Mica2 [Polastre et al. 2005] platform.

COOJA/MSPSiM [Eriksson et al. 2009] is a simulation framework for the Contiki [Dunkels et al. 2004] sensor node operating system. Both COOJA and MSPsim are written as standalone Java-based tools that together allow WSNs to be simulated at both the operating system (code) and machine instruction level. COOJA provides an application-level simulation for networks consisting of nodes running the Contiki OS. COOJA operates in a way similar to how TOSSIM works for TinyOs applications. The Contiki core, user processes, and special simulation proxy drivers are compiled into object code native to the simulator platform, and are then executed from COOJA. MSPSim [Eriksson et al. 2009], in turn, is a cycle-accurate emulator for WSN platforms based on the Texas Instruments MSP430 microcontroller [Texas Instruments Corporation 2006]. It combines cycle-accurate interpretation of CPU instructions with discrete event simulation of all other peripheral components on board, such as the transceiver, ADC, timers, Usart, and others. In this respect, MSPSim is similar to the AVRORA tool discussed earlier. COOJA and MSPSim are the best choice for development of Contiki-based applications running on the MSP430 hardware architecture. However, for other hardware platforms and target OSs, these tools do not provide any support.

VIPTOS (Visual Ptolemy and TinyOS) is a graphical development and simulation environment for TinyOS-based WSN applications [Cheong et al. 2005]. VIPTOS bridges together VisualSense [Baldwin et al. 2005], a Ptolemy II-based [Liu et al. 2001] graphical simulator, and the TinyOS framework. VisualSense exploits the actor-oriented computational model of Ptolemy II, a general modeling framework for heterogeneous embedded systems. VisualSense defines actor-oriented models for sensor node subsystems and communication channels. The interaction among different components occurs via timestamped events. VIPTOS combines the VisualSense execution semantic model and the TinyOS low-level hardware representation of the sensor node. Such a combination extends the capacity of TOSSIM to support simulation of heterogeneous networks [Cheong et al. 2005]. VIPTOS provides event-driven simulation of actual TinyOS programs along with packet-level network modeling. It allows one to use the available Ptolemy II models of computation to model different WSN subsystems. Besides, VIPTOS provides the capability to transform high-level representations of applications into actual TinyOS code. It allows designers to easily translate the algorithmic view of an application into a low-level implementation. However, VIPTOS does not provide any accurate hardware representation of a sensor node. Substantially, it focuses more on the algorithmic and application domains. Additionally, VIPTOS has been integrated only with the first version of TinyOS, which is not currently supported. Similar integrations have been provided for other general-purpose simulators. For instance, Rialto [Bonivento et al. 2006] supports a platform-based approach based on Metropolis [Davare et al. 2013]. Simulation, in this case, is used as part of the optimization process.

A similar federated approach has been recently proposed with the Wireless Cyber-Physical Simulator (WCPS) [Li et al. 2013]. This framework integrates TOSSIM with Simulink [2015] in order to provide a combined simulation of a wireless sensor monitoring system with a realistic representation of physical structures. Simulink is also used to model the controller in a closed-loop actuation network that typically runs outside the sensor nodes on a more powerful platform. Similar solutions have also been developed based on NS-2, such as the NCSWT modeling and simulation tool [Eyisi et al. 2012].

Table III provides a summary of the characteristics of some of the tools covered in this section.

### 2.2. Comparative and Evaluation Studies on Simulation Tools

A number of studies and evaluation reports on different network simulation environments have been published in the literature in recent years. For instance, Timm-Giel et al. [2008] compared the throughput and the transmission delay of WSN scenarios modeled in the OMNet++, NS-2, and OPNET [opnet 2014] tools. The simulation results obtained in this study revealed a close similarity between OPNET and NS-2 for the throughput and a close resemblance of OPNET to OMNet++ in end-to-end delay.

Lessmann et al. [2008b] presented a case study for the analysis and comparison of the J-Sim [Sobeih et al. 2007], OMNeT++, NS-2, and ShoX [Lessmann et al. 2008a] tools by implementing a simple network control algorithm called SPAN [Chen et al. 2002a]. The tools are evaluated with respect to the usability, the effort for installation and simulation implementation, the availability of usable models, and other aspects.

Table III. WSN Emulators: Comparative Table

| | **TOSSIM & PowerTOSSIM** | **AVRORA & AEON** | **COOJA** | **VIPTOS** |
|---|---|---|---|---|
| **Simulation engine** | custom C++ | custom in Java | custom in Java | custom C++ |
| **Simulation input** | C++/Python, AVR bin. files | Java, AVR bin. files | Java config, MSP bin. files, Contiki | XML, TinyOS apps |
| **Scalability >100** | high | high | middle | middle |
| **Communication channel model** | free space with directed graph of bit errors | free space | free space, log. shadowing | free space, log. shadowing, Rayleigh fading |
| **Mobile nodes** | no | no | no | no |
| **Supported HW platforms** | MicaZ | MicaZ, Mica2 | Tmote sky | TelosB, Tmote sky |
| **Extendability** | - | - | - | partly supported in C++ |
| **RF PHY model** | cc1100 | c2420, cc1100 | cc2420 | cc2420, CC1100 |
| **Sensing channel** | no | no | no | ADC interface file |
| **Energy profiling** | MicaZ platfrom | each HW component | each HW component | each HW component |
| **Battery/lifetime estimation** | no/no | no/no | no/no | no/no |
| **Accuracy timing/power** | high/high | high/high | high/high | high/high |

The authors concluded that there was no clear winner revealed in this study and each of the tools had its own strengths and weaknesses compared to the other candidates.

Weingartner et al. [2009] compared the performance of popular general-purpose network simulators including NS-2, NS-3, OMNet++, SimPy [Simpy 2013], and JiST [Barr et al. 2005]. The evaluation criteria consisted of the simulation runtime performance and the memory footprint for various network sizes. The JiST and NS-3 environments were found to be the most efficient tools for large-scale network models. At the same time, SimPy exhibited lower performance due to its Python-based simulation core.

Instead, Lee et al. [2007] focused on studying the accuracy of the wireless link model for several tools, including EmStar [Girod et al. 2004], TOSSIM, and NS-2. Their conclusion is that, while the empirical approaches available in most tools are able to recreate packet-level behavior to some extent, they are insufficient to take into account the effect of noise produced by neighboring networks. Therefore, it is not well captured by an additive white Gaussian component. Therefore, the author proposed various approaches to statistically characterize real noise traces and implement the corresponding algorithms in the TOSSIM framework, considerably improving the accuracy (at the expense, for one approach significantly, of simulation performance). One distinctive advantage of using traces is that the model is able to not only simulate packet loss but also to point at the *causes* of temporal variations in packet loss and connectivity, thereby adapting well to different deployments. Kamthe et al. [2013] improve on this work by using a multilevel statistical approach to model the short- as well as the long-scale behavior of links in wireless sensor networks, by modeling the correlations between successive packet receptions and failures. In our study, we explore the behavior of the channel model in the different simulators in terms of packet delivery ratio for comparison purposes. However, we do not present results in case of specific noise patterns due to other networks, which would require additional measurements. On the other hand, the study on noise models presented by Lee et al. and by Kamthe et al. is complementary to our comparison, and we refer the interested reader to the original papers for more details [Lee et al. 2007; Kamthe et al. 2013].

Finally, Boulis and Tselishchev, the designers of the Castalia framework, presented a series of experimental studies on Castalia performance [Pediaditakis et al. 2010] and effects on carrier sense modeling [Boulis and Tselishchev 2011]. These studies solely consider the Castalia toolkit and do not compare the obtained results with any other tool. However, in our evaluation, we partially rely on the experimental results obtained in these studies.

Summarizing, all of the studies discussed provide a varied level of analysis including pure analytical evaluation and a practical comparison of the runtime performance, memory footprint, and network simulation results produced by different tools. In our study, we decided to address most of the presented metrics. Moreover, we go even further and not only compare tools to each other but also evaluate the consistency of the simulation results with values obtained in real experiments. However, we do not consider the memory footprint taken by the different tools due to its not very practical value, considering the large amount of memory that is available today in even small-sized systems. To quantify, the maximum reported memory footprint for a large-scale network simulation was approximately only 150MB [Weingartner et al. 2009]. While this metric may reflect runtime efficiency of a particular tool, we believe it is more valuable to assess execution speed and accuracy of simulation.

## 3. COMPARATIVE STUDY

In this section, we discuss the benchmark methods and the application test cases that we have designed to compare different WSN simulation environments. We admit that it is a rather challenging and sometimes not even a feasible task to compare various tools designed for different purposes. However, in this study, we consider functionally similar environments and compare them by providing unified input models to produce consistent simulation results. In our study, we include six recent (January 2015) mainstream open-source WSN simulation toolkits. The latest releases of the Castalia (version 3.0), MiXiM (version 2.3), WSNet, PASES, TOSSIM, and COOJA tools are investigated. Our main objective for this study is to obtain and assess numerical performance values and figures on different simulation results. The following simulation metrics are collected and evaluated:

—Runtime simulation performance versus number of modeled units: it is the time taken by the tool to run a certain number of modeled units and events.
—Network throughput: it is the average rate of successfully delivered packets over a communication channel. Although this is a very application-specific parameter, we are interested in discovering the capacity of a certain tool to simulate the communication infrastructure under different conditions.
—Packet loss at the MAC layer: it is the number of packets discarded by the MAC layer due to the limited number of retransmissions and trials to send a packet while the channel is found busy or for other failures.
—Packet Delivery Ratio (PDR): it is the ratio between the packets that are successfully delivered to a destination and the total number of packets that have been sent. The PDR depends both on the communication environment and on the peer-to-peer link quality. In our PDR study, we incorporate two different communication media conditions. It includes an ideal model and a close-to-real indoor obstructed environment, in order to assess both the RF propagation and the radio reception models in the various simulation tools.

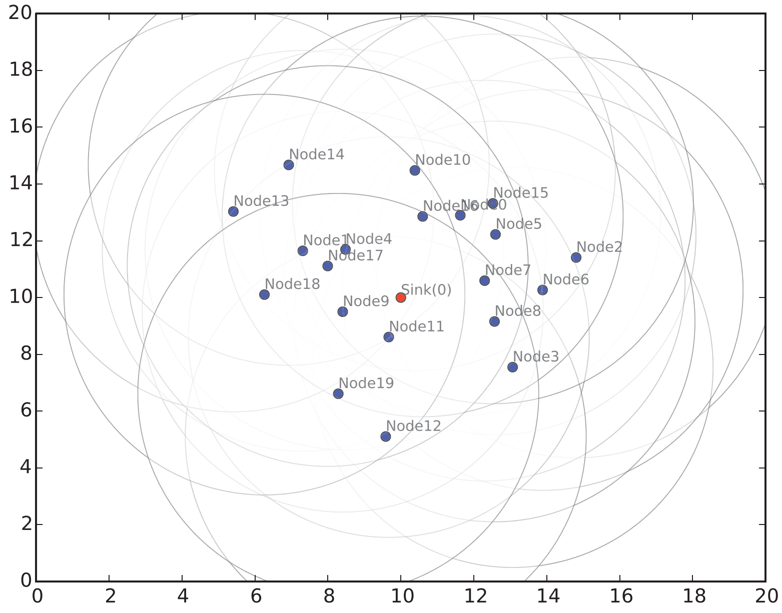


Fig. 1. Network topology consisting of 20 nodes.

—Network latency (delivery delay): it is defined as the average time delay between sending data frames from the application layer of the node to its reception at the destination node on the application layer. Along with network throughput, packet delivery depends on a number of parameters such as the application duty cycle, the MAC protocol implementation and options, the RF channel (e.g., obstacles), and so on.

—Accuracy of the power consumption estimation: it measures how results of power consumption estimation correlate with or resemble real-world values obtained by measurements on a testbed.

All presented metrics have been collected by means of two benchmark applications (Sections 3.1 and 3.2), which were identically implemented in each of the investigated toolkits. In any comparative study where different tools are evaluated, it is crucial to build equal models with identical behavior producing the same amount and rate of simulation events. Hence, the first and main challenge we were faced with was the equivalence of the models and the consistency of the simulation results. We started our study from the definition and implementation of uniform simulation setups, topologies, and application models.

### 3.1. Simulation Study I

The first simulation scenario is a WSN application intended to evaluate network-related metrics and simulation runtime performance. This scenario presents a data-centric network with a unidirectional data traffic flow. The network model consists of $N$ nodes, of which $N-1$ are senders and produce data transmitted to one sink node, which receives packets and reports delivery statistics. All senders are randomly arranged around the sink within its reception range in such a way that all the nodes are capable of hearing each other. Figure 1 shows an example of modeled topology consisting of 20 wireless nodes. The circles on the figure represent the node "communication disk

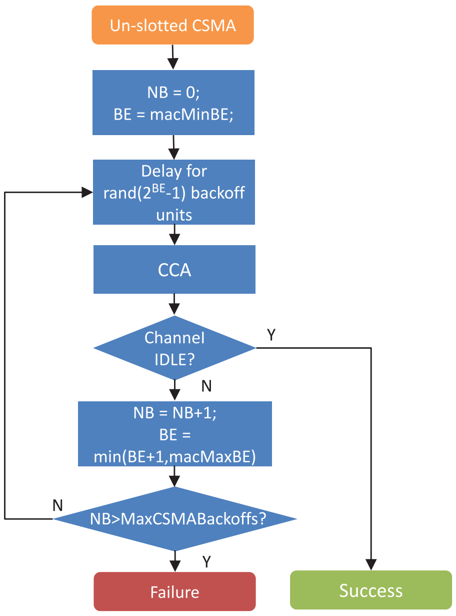


Fig. 2. Unslotted IEEE 802.15.4 CSMA/CA algorithm.

model," that is, the node communication range within which the transmitted signals are above the receiver sensitivity threshold.

Senders periodically broadcast packets with constant bit rate (CBR). This scenario represents the typical sensing or monitoring WSN application where broadcasting is the dominant approach for packet transmission. All network transmissions in this scenario rely on an unslotted IEEE 802.15.4 CSMA/CA algorithm [IEEE Standard Association 2006] (see Figure 2).

According to the MAC standard, each device operates with a set of variables that defines the boundaries for back-off operations. *NB* is the number of times the CSMA/CA algorithm is required to back off while attempting the current transmission; this value is initialized to zero before each new transmission attempt. *BE* is the back-off exponent, which is related to how many back-off periods a device shall wait before attempting to assess the availability of the channel [IEEE Standard Association 2006].

Before each transmission, *BE* is initialized to a defined value *macMinBE*. When a device wants to transmit data, it initializes *NB* and *BE* and proceeds to wait for a random number of back-off periods defined by *BE*. The duration of the back-off period (back-off unit) is defined by a constant value held in a variable *aUnitBackoff-Period*. Subsequently, after back-off, the node immediately performs a Clear Channel Assessment (CCA) to check whether the medium is idle. If so, the data is transmitted; otherwise, the device increases both *BE* (up to a maximum value) and *NB*, and backs off once again. This procedure is repeated until either CCA reports idle and the packet is transmitted or *NB* exceeds the maximum allowed number *(MaxCSMABackoffs)* of CSMA/CA back-offs. In the latter case, a MAC error occurs and the current packet is discarded from the node's transmission queue.

The model of this MAC protocol was initially available in the WSNet tool only. We have ported it to the other toolkits, ensuring that the behaviors of the different models were the same. Because the MAC algorithms used in the different tools are equivalent, we do not expect differences in the medium access behavior among the tools we have

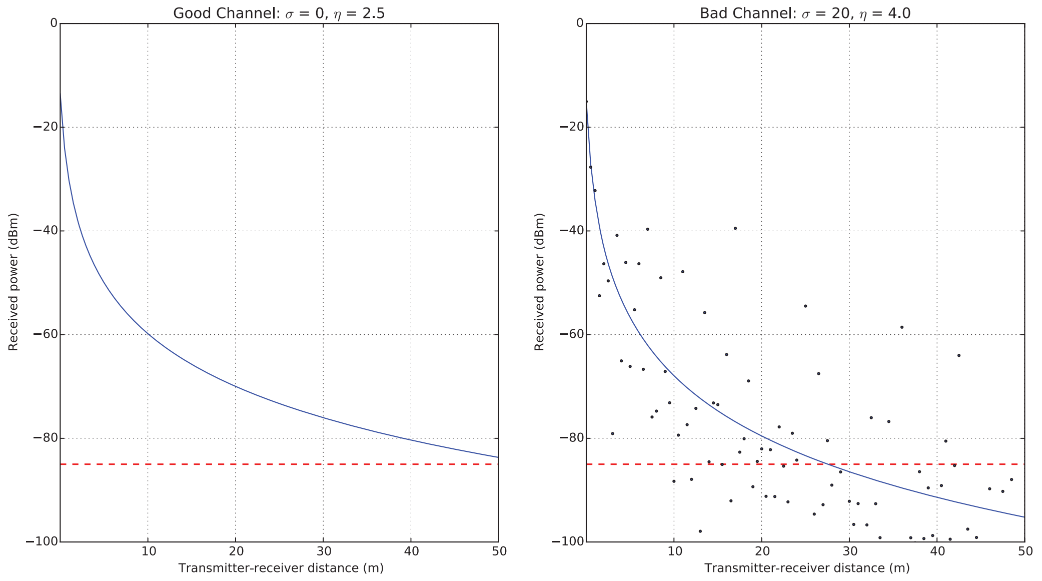


Fig. 3. Clear and log-normal shadowing channel conditions.

considered. The parameters for the MAC model have been set to the default values specified in the IEEE 802.15.4 standard; namely, *MaxCSMABackoffs* was set to 4; the *BE* boundaries values *macMinBE* and *macMaxBE* were set to 3 and 5, respectively; and *aUnitBackoffPeriod* was equal to 20 symbols (0.32 milliseconds). The acknowledgments and any handshaking algorithms (such as RTS-CTS) are not considered in our model due to the broadcast message-passing scenario. For all transmissions, the packet size was fixed to 64 bytes (16 bytes for the MAC header and the rest for the data payload). The sink node collects statistics on the total number of successfully received packets. Additionally, it computes the mean delivery latency by analyzing the timestamp included in each packet.

Along with the application and the MAC models, the settings of the radio propagation model have been carefully chosen in order to provide identical communication environments among the toolkits. Some of the tools provide a number of different communication channel models; however, for comparison reasons, we consider a propagation model based on the log-distance path loss equation [Rappaport 2002], which is available in all the tools out of the box:

$$PL_{dB}(d) = PL_{dB}(d_0) + 10\eta \log_{10}\left(\frac{d}{d_0}\right) + X_{\sigma,dB}, \tag{1}$$

where $d_0$ is the reference distance, $\eta$ is the path loss exponent that defines the rate at which the signal decays with respect to the distance $d$, and $X_{\sigma,dB}$ is a zero-mean Gaussian random variable (in dB) with standard deviation $\sigma$. The last term represents the signal shadowing effect that accounts for dependencies of all environmental factors, such as static and mobile obstacles and signal reflections. If there is no shadowing effect, then $\sigma$ is zero. In our study, we incorporate two propagation media conditions, shown in Figure 3. The first model represents an "ideal" network condition with an unobstructed clear path ($\eta = 2.5$) between the transmitter and the receiver. Conversely, the second model represents "close to real" link conditions with memory-less shadow fading distribution $\sigma = 20$ and path loss $\eta = 4.0$.

A simple interference model is chosen in all tools to handle signal collisions. In this model, packet collisions happen and transmissions fail (packets are marked as invalid) every time two or more nodes are concurrently transmitting within the receiver range.

The TOSSIM tool, in contrast to the other simulation environments, does not provide a log-normal shadowing interference model out of the box. Instead, it introduces a specific statistical approach, the Closest Pattern Matching (CPM) algorithm [Lee et al. 2007], for packet reception. In order to use it, the model requires a collection of empirical measured levels of noise in a specific environment. Hence, a noise trace has to be provided as an input from which a statistical model capturing different phenomena like interference is generated. Either the input file containing the noise trace has to be created or the user may rely on sample traces available in the public domain.

In order to model the shadowing effect in COOJA, we have used a specific radio propagation model provided by the tool, named Multipath Ray-tracer Medium (MRM). This model is packet based and uses a ray tracing technique, which allows us to specify the necessary parameters related to the radio medium (e.g., transmitted power, antenna gains, SNR threshold). This approach is able to approximate the signal strength attenuations, due to obstacles, among the simulated radios. Refraction, reflection, and diffraction coefficients are calculated and taken into account in order to provide a more realistic range for the transmitted signal. In this article, the receiver power is calculated by means of the Friis formula [Rappaport 2002]. When a radio antenna transmits a packet, the area of interference around the antenna is “busy” for a time that depends on the length of the packet itself; if the transmission is completed without any interference, then the packet is delivered; otherwise, no data are delivered. Note that, in addition to MRM, COOJA provides another three radio propagation models: the Unit Disk Graph Medium (UDGM) Constant Loss, in which the transmission range is an ideal disk, and where all (and only) the nodes within such a disk receive all the packets; the Unit Disk Graph Medium (UDGM) Distance Loss, which is an extension of the previous one, and which considers interferences, leading to the definition of a success transmission ratio and a success reception ratio probability; and finally the Directed Graph Radio Medium (DGRM), which allows also propagation delays to be defined. For the purpose of this work, we believe that MRM is the best choice with its capability of setting the proper parameters to simulate the shadowing channel conditions.

Finally, in order to make the communication even more realistic, we include properties of a real RF subsystem by assuming a CC2420 [Texas Instruments Corporation 2013] radio model for all simulated nodes. The RF systems are set to operate at a radio frequency of 2.4GHz with a data rate of 250kbps, -3dBm output power, and -85dBm signal sensitivity. Furthermore, for comparison reasons, we also annotated the RF model in all toolkits with state transitions and CCA operation timing delays (16 symbols) typical of the CC2420 transceiver.

In order to validate the simulation results, we deployed a small-scale wireless network composed of four TelosB nodes [Polastre et al. 2005]. TelosB is a popular open-source platform designed to enable experimentation with WSNs. The TelosB architecture includes a low-power 8MHz microcontroller (TI MSP430); the CC2420 RF transceiver, which is modeled in our simulations; and a set of sensors for ambient monitoring. Our test bench consisted of three nodes that generated data and one sink node connected to the PC for data logging. For each packet delivered to the sink, the information on RSSI level, time latency, and sender MAC state was collected. Time synchronization between nodes was performed by broadcasting reference timestamps from the sink side at the beginning of each experiment. All the settings of the nodes, the topology, the application, and the MAC functionality implemented on the test bench are fully compatible with the simulation scenario described earlier.

Table IV. CC2420 Current Consumption and Timing

| | Off(PD) | Idle | RX | TX | Off to Idle | Off to RX | RX to TX | TX to RX | RX to Off |
|---|---|---|---|---|---|---|---|---|---|
| Current (mA) | 0.02 | 0.42 | 18.8 | 17.5 | 0.55 | 2.0 | 18.5 | 18.5 | 7.0 |
| Time (us) | - | - | - | - | 600 | 1,920 | 200 | 200 | 800 |

## 3.2. Simulation Study II

The second study is intended to evaluate the accuracy of the power consumption estimation results in the considered tools. The simulation results, as in the first case study, are compared with respect to the values obtained on a real test bench. The application scenario for this study is a modification of the first test case. Here we consider a small three-node network that includes one sink and two senders. In order to preserve battery energy, the sender nodes exploit a simple power-saving technique that controls the operating states of the RF subsystem. Whenever the application has a packet to send, the RF component is switched from the idle to the active state. Upon the completion of the transmission, the radio antenna is turned back to the idle mode. Hence, the power consumption of the nodes is essentially determined by the transmission rate of the application that defines how often the radio antenna is required to be in the active state.

In order to model the node power consumption, each simulated node is provided with an energy model based on the TelosB platform [Polastre et al. 2005]. However, most of the tools in our study provide modeling facilities only for the RF subsystems and omit CPU and other peripheral components. Due to these limitations, it seems not to be possible to build the whole node functional and energy models without considerable design efforts and changes in the tools' architectures. Because of that, we focus solely on the RF component model, which is in any case the dominant power consumer. For comparison reasons, we consider a simplified energy model of the RF transceiver based on the CC2420 radio chip. Our model consists of three discrete operating states (idle, transmission, listening) and transition delays between them. Each state and transition take some power and time to be completed. Table IV gives values of current and time transitions related to the CC2420 transceiver.

In order to obtain the power consumption reference values with which simulation results can be compared, we have set up a test bench consisting of three TelosB nodes programmed according to the modeled scenario. The software applications and the MAC algorithm for both the sink and the sender nodes were implemented in TinyOS. Using a digital multimeter (Agilent 34411A), we measured the voltage drop across a 10 Ohm resistor connected between the board and two batteries wired in series (NiMH 1400mAh, 1.36V in fully charged conditions, each taken separately). The current load profile was obtained by logging the voltage drop with a 40-microsecond sample period and subsequently converting the voltage to the current value.

## 3.3. Simulation Studies Summary

Table V summarizes the global settings for the benchmark test cases presented in this article. Specific settings for the application, the MAC protocol, and the communication channel are given in Section 5, where we discuss the obtained results.

## 4. IMPLEMENTATION

In this section, we discuss the design of the models and the challenges we have faced during the implementation and installation of the benchmark simulation setups in different tools. All the tools in our study have many similarities regarding architecture, design principles, and simulation configuration. All the toolkits have an object-oriented architecture based on an event-driven simulator engine implemented in

Table V. Case Studies' Settings Summary

| | **Case Study I** | **Case Study II** |
|---|---|---|
| **Network area, m x m** | 40x40 | 10x10 |
| **Traffic type/rate (pkt/sec)** | CBR/1, 2, 5, 10 | CBR/1, 2, 5, 10, 20, 50, 100 |
| **Network size (*N*)** | 5, 10, 25, 50 | 3 |
| **Number of senders** | $N - 1$ | 2 |
| **Data packet size (bytes)** | 64 | 64 |
| **MAC/routing/PHY models** | IEEE802.15.4, none, CC2420 | IEEE802.15.4, none, CC2420 |
| **RF output power/ receiver sensitivity** | -3dBm/-85dBm | -3dBm/-85dBm |
| **Communication channel model** | log-normal shadowing $\eta$=2.5/$\eta$=4.0, $\sigma$=20 | log-distance path loss $\eta$=2.5, $\sigma$=0 |
| **Collected metrics** | simulation performance, throughput, packet loss, MAC performance | power consumption |
| **Sim time, sec** | 100 | 100 |

C++ or, for COOJA, in C and Java. Simulation configuration and network setups in all environments are made by means of external configuration and scripting files.

Castalia and MiXiM utilize OMNet++-based initialization files (.ini and .ned formats). Inside the .ned files, users may define the topology of simulation modules as an interconnection of simple components linked through channels and interfaces. Additionally, .ned files hold default values for the module external variables and constants. The .ini format files are used to configure simulation scenarios. They typically include the number of simulated nodes, node assigned locations, names of application and MAC modules, simulation time, and many others. Castalia also includes a set of plain-text files to configure power-timing properties of the RF section, while MiXiM uses .ned files for this purpose.

Both WSNet and PASES employ XML configuration files for the network and simulation setups. Additionally, PASES offers Python scripting facilities for MAC and application design as well as external Python files to hold energy-timing values for hardware models.

COOJA supports the creation of the desired simulations by first indicating the radio medium to be used and then by instantiating the nodes and by compiling their code (previously written in the C language). Once the network parameters have been defined (e.g., area, node number, topology), the simulation environment can be stored in a .csc file, in order to save it for future usage. Note that from the COOJA GUI, only the network configurations can be set, while the node behavior has to be defined at the Contiki code level; the file projec-conf.h contains all the defined configurations for a specific node.

The TOSSIM tool, being a part of standard TinyOS installation, requires some knowledge and experience with TinyOS applications, modules, and interfaces. The simulation scenario can be written either in Python or in C++ where all model properties (e.g., number of nodes, node start-up and simulation time, etc.) are defined. In our study, we use Python scripts for all simulation test cases.

Sections 4.1 and 4.2 provide a detailed report on challenges faced for the implementation of the first and second test cases, respectively.

Note that, at the application level, for both test cases we employed the “ThroughputTest” model available in Castalia. Porting of an application to the other tools is essentially similar to porting the MAC layer. Likewise, this process did not raise any challenge due to the very similar modularity and design principles applied in the tools. Since the tools provide different reporting facilities, we implemented simulation statistics gathering individually at the MAC and at the application layers. This included logging and reporting the number of received packets, the mean receive latency, the number of sent packets, the packet drops, and the mean back-off time at the MAC layer. Each tool provides built-in simulation performance information reporting out of the box, which we employed to evaluate their runtime performance.

### 4.1. Implementation I

In the first application test case, we aim at assessing tools with respect to the network communication behavior and utilization of the channel. Thus, given that different tools provide different and not compatible MAC models, we started our benchmark design from the implementation of functionally equivalent MAC algorithms in each tool. The core of the MAC model is the CSMA model that is easy to implement in any studied tool based on examples available. The model of the CSMA algorithm involves radio module CCA functionality and a single timer component to process back-off periods. All the tools in our study provide very similar simulation architectures and module partitioning with distinctive layers for application, routing, MAC, radio PHY, and communication channel. Besides, tools have almost equal event-driven semantics for simulation, where the functionality of the component is activated in response to the internal events and requests (e.g., CCA procedure completion or timer firing) generated by the upper and lower layers. Thus, porting of simulation models from one tool to another did not require considerable changes in the initial model structure and the execution flow. However, different simulation events, various APIs, and diversified interaction mechanisms among simulation components require one to individually adapt the initial model for each toolkit. In a similar way, TOSSIM requires new models to be compatible with TinyOS split-event command semantics. During porting, we revealed that TOSSIM does not fully support all functional modules and interfaces specific for real TinyOS applications. As a consequence, we had to manually adapt and modify the actual application code in order to compile it for simulation. In particular, TOSSIM does not support the *CC2420ActiveMessageC.RadioBackoff* interface that is the basis for the CSMA implementation in our test bench TinyOS application.

As soon as the communication algorithms and the application models were implemented, we started to verify their functional equivalence and consistency of simulation results with respect to the packet delivery and MAC behavior. For the sake of simplicity, the initial simulation setup consisted only of two nodes (one sink and one sender) that excluded the possibility for collisions. Simulation results for all studied metrics between the tools were almost equal. Some slight deviation was observed with respect to packet delivery latency that was in the range of 2%. This allowed us to assume that MAC and application models were functionally equal and simulation setups were correct. As mentioned in Section 3.1, for the tools Castalia, MiXim, WSnet, and PASES, we selected and set up the communication channel model based on the log-normal shadowing models available in each of these tools out of the box, while for COOJA the MRM propagation model is considered. Further analysis revealed that the differences in simulation results were caused mostly by the differences in the implementation of communication channel models in different tools. The results of the network throughput study are reported in Section 5.1.

The network topology specification in TOSSIM for its SNR-based communication model is conceptually different from the other simulators, where nodes are placed in the virtual space according to their coordinates. The topology in TOSSIM is not represented by the distances between the nodes, but rather by the signal gains. This approach supposes the knowledge of the gains between the nodes prior to the simulation run. Since the gains depend not only on the distances (path loss) but also on the characteristics of the communication channel, we profiled the signals for every modeled node in Castalia and then supplied this information into TOSSIM.

### 4.2. Implementation II

In the first application test case, the energy efficiency requirements for the nodes' operations are not considered. However, in the second test case, the nodes utilize a simple power management mechanism, which automatically turns the radio component on or off whenever transmission is required or completed. This simple state control was implemented at the MAC layer that, in turn, required some slight changes to be made with respect to the initial model. Nevertheless, no issues arose here due to the plain interfaces available in each of the toolkits to explicitly control radio operating states.

Certain challenges were encountered when we implemented the energy and timing models for the RF subsystem. For instance, Castalia and MiXIM provide plain interfaces in .txt or .ned files for energy-timing modeling, while WSNet does not allow any timing annotation for radio state transitions. In order to ensure that the delay due to the RF models among the tools was the same, we extended the WSNet's radio model with timing values for each operation and state transition. PASES offers an easy way to annotate any platform component with energy-timing values by means of external Python files. In COOJA, an extension, named Energest, was added in order to perform energy consumption measurements. In particular, Energest estimates the per-component energy consumption by measuring the amount of time spent by a certain component in a specific state. Between any two points in the source code we can calculate the value of these timing parameters, by logging their initial *p_init* and final *p_final* values. Then, the difference *p_final - p_init* gives the time spent in the desired interval. This result is given in *Rtimer* ticks (it is a CPU cycle counter and outputs 1 tick every $32\mu s$); therefore, it needs to be converted from time to energy by using the voltage and the current draw of the specific component (such data are available in the datasheet related to the Tmote Sky). TOSSIM is not present in this study due to the absence of an energy model. The results of the energy modeling study are presented in Section 5.2.

### 4.3. Ease of Use

A discussion about the ease of use of the analyzed tools is important for (1) making the best choice to minimize the development effort of porting new protocols on actual hardware and (2) mitigating the additional overhead required to engineer a real implementation. In this direction, Castalia presents a well-designed channel model (especially at the MAC layer) and allows one to simulate communication conditions that are closer to reality (e.g., shadowing, signal-to-noise ratio, packet reception probability, path losses). In addition, it provides various routing protocols. Moreover, Castalia provides a great documentation support and comprehensive set of examples. That makes Castalia an easy-to-get-started tool not requiring any prior experience with the OMNet++ framework. However, despite the high accuracy and flexibility in the network design aspects, the code produced by the Castalia simulator is not portable to an actual deployment. MiXiM, on the other hand, is more complex in terms of architecture, modules partitioning, and requires some basic experience with the OMNet++ environment and the Eclipse IDE. COOJA and its Contiki OS are designed for portability on many platforms, such as the Tmote Sky, ESB, and so on, since MSPSim is a sensor node emulator for MSP430-based nodes. Note that COOJA is able to execute native code by performing Java Native Interface calls from the Java environment to a compiled

Contiki code. COOJA makes available some interfaces that represent the hardware peripherals for the simulated nodes and enable the Java simulator to detect and trigger different kinds of events, such as incoming radio traffic and the lighting of an LED. Each Contiki system is compiled for a specific hardware platform, with proper drivers and communication modes to the actual hardware. This makes it possible to simulate and deploy the same code without any modifications, minimizing the delay between simulation and deployment. Additionally, COOJA also enables the simulation of TinyOS code, in the form of a binary file (e.g., .hex).

PASES is a tool specifically designed for fine-grained power consumption analysis of standalone and networked embedded systems. Thus, it offers a wide range of possible configurations and design options to explore for the target models, such as the kind of CPU core, a diverse set of hardware peripherals, clock frequencies, power supply, and so on. The utilization of PASES assumes an extensive use of Python scripting and XML configuration to set and refine different design options. In order to facilitate the simulation design process, PASES comes with a set of sample examples for various WSN applications and HW platforms, GUI utilities for power profiling, and network topology visualization. Custom module design and functional extension in PASES require some expertise in C++ and with the SystemC library.

The WSNet tool offers a wide set of examples, tutorials, and comprehensive documentation out of the box in the standard distribution. Its plain source code and clear partitioning of the functional modules and interfaces make WSNet an easy-to-start-with toolset.

TOSSIM, on the other hand, requires experience with the nesC and the TinyOS framework, involving application design, component development, and interface implementation. The TOSSIM simulation does not fully support the whole set of standard interfaces and modules present in a real TinyOS application. For this reason, apart from the simple examples, the TinyOS development process often requires that two separate versions of the same application are supported, one for the real-world deployment, the other for simulation.

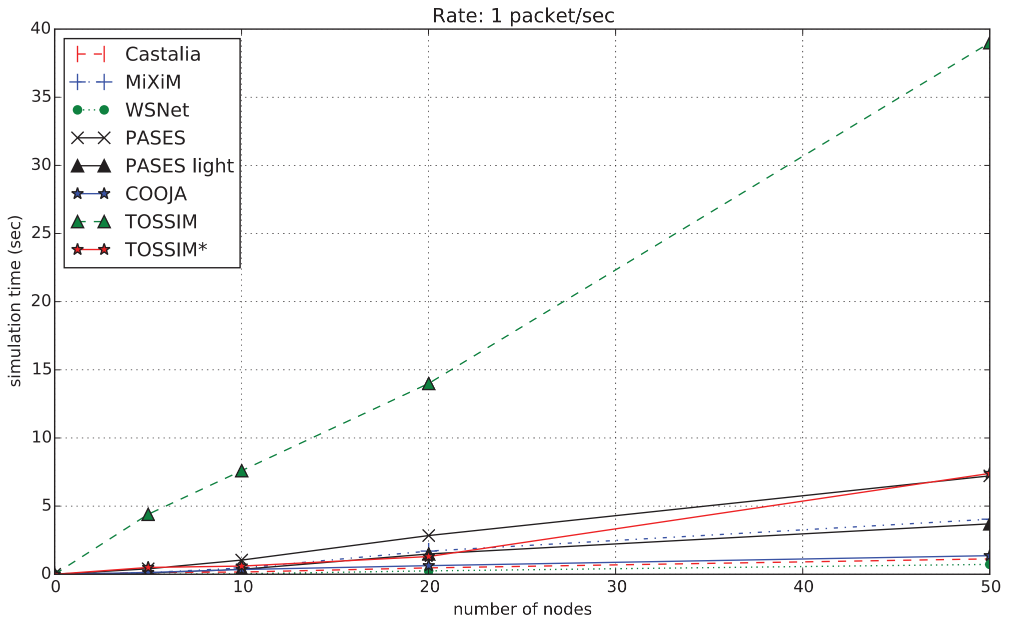
## 5. RESULTS

Our evaluation is based on a series of simulations that we have performed to compare toolkits with respect to the metrics discussed in Section 3. All simulations were conducted on a desktop PC equipped with an AMD Athlon 64 X2 5000+ 2.6GHz with 4GB of RAM running either Ubuntu 11.10 or the Windows 7 Professional operating system. The simulations were all set up to simulate 100 seconds of virtual operating time.

### 5.1. Simulation Results I

For the first test case, we carried out a set of simulations varying number of transmitting nodes and packet rates. The network size ranged from five to 50 nodes with four different packet rates for each network size (i.e., one, two, five, 10 packet/sec). All simulation results are averaged out of 10 execution sets for each case, to smooth out random differences.

Figure 4 and Figure 5 show the obtained runtime performance for various network sizes and data rates. As expected, the runtime of all tools grows almost linearly as both the network size and the packet rate increase. With this setup, in all cases, the performance of the simulators is faster than real time. The main outcome from this experiment is a noticeable difference in simulation performance among the tools. The performance of TOSSIM appears to be the worst among all the studied tool sets. This can be explained by the complexity of the TOSSIM SNR-based communication model. This model requires a precomputation phase during which a graph of connected nodes with SNR information for each link is created. The precomputation time contributes



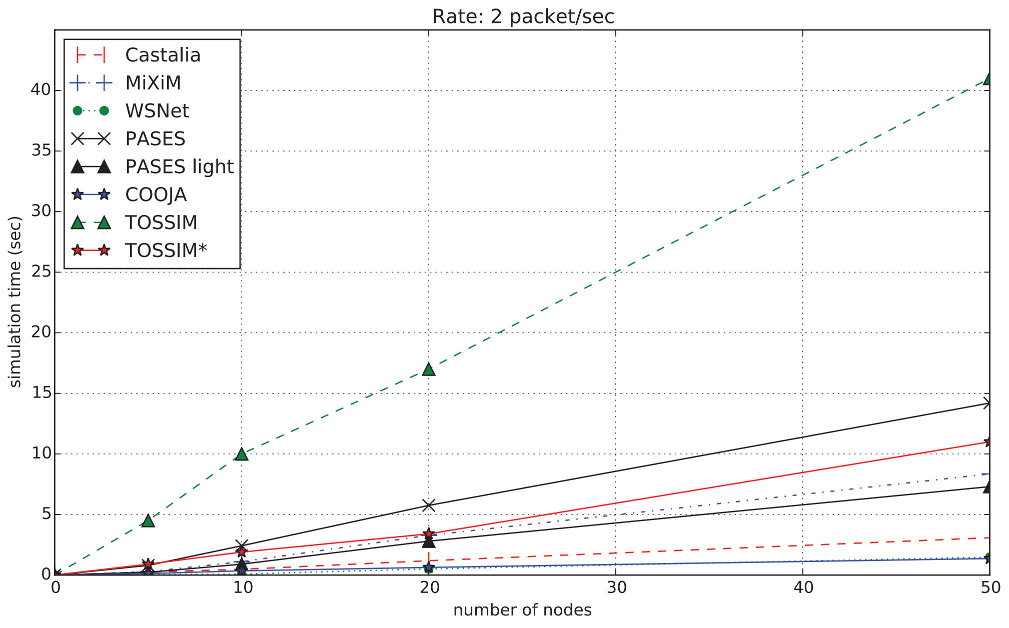
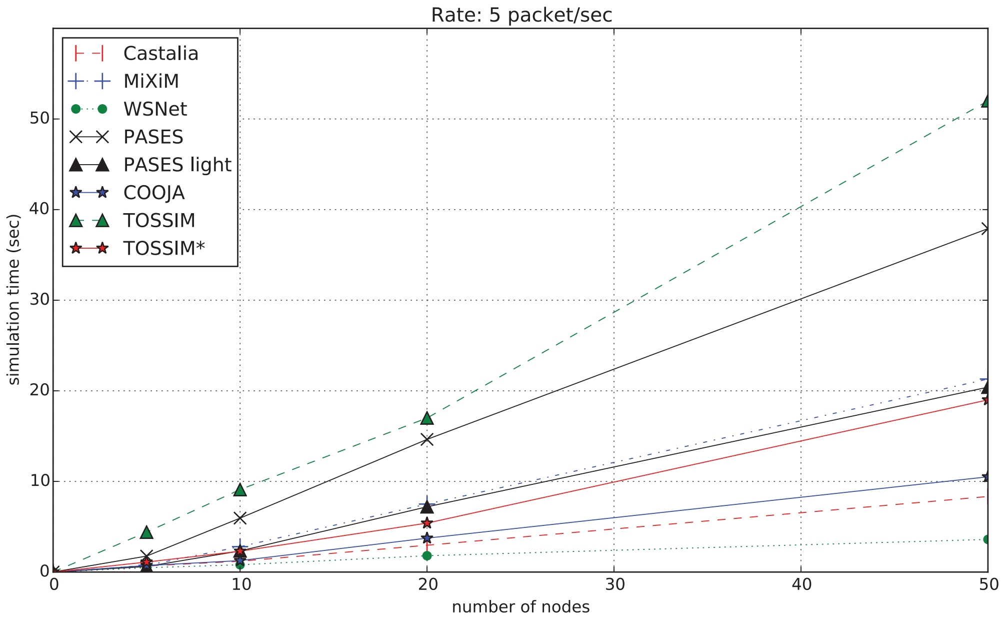


Fig. 4. Runtime performance for low packet rates.



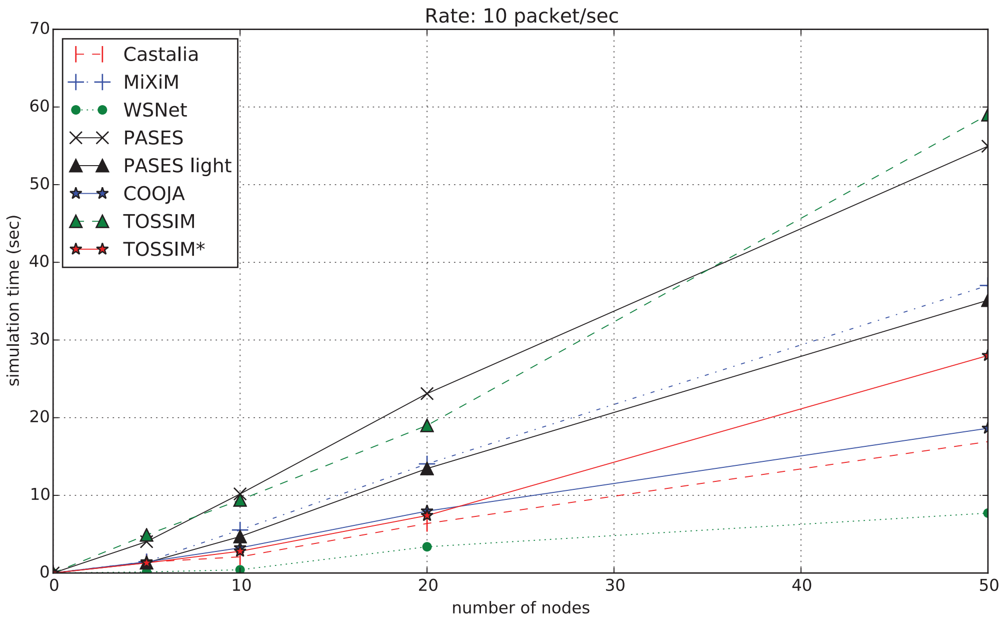


Fig. 5. Runtime performance for high packet rates.

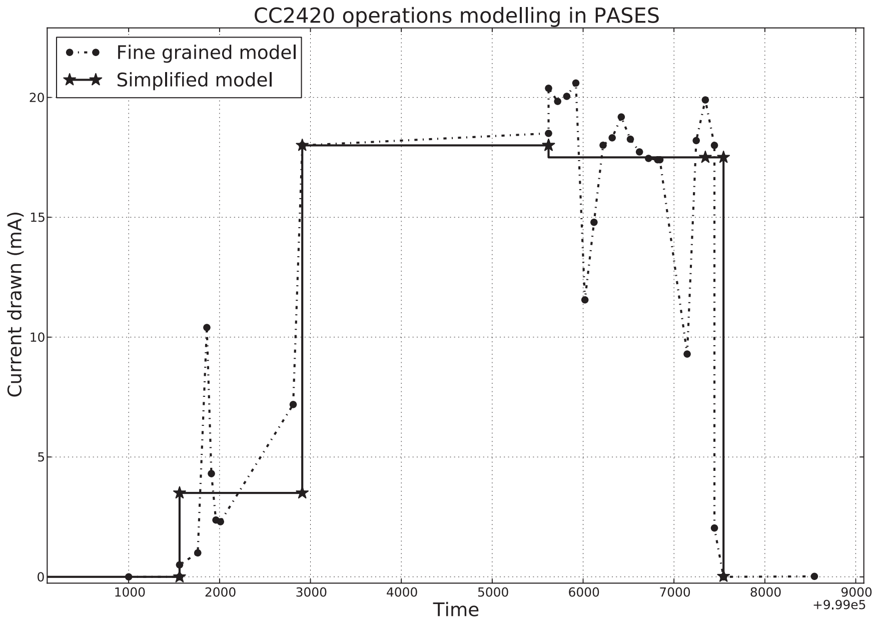
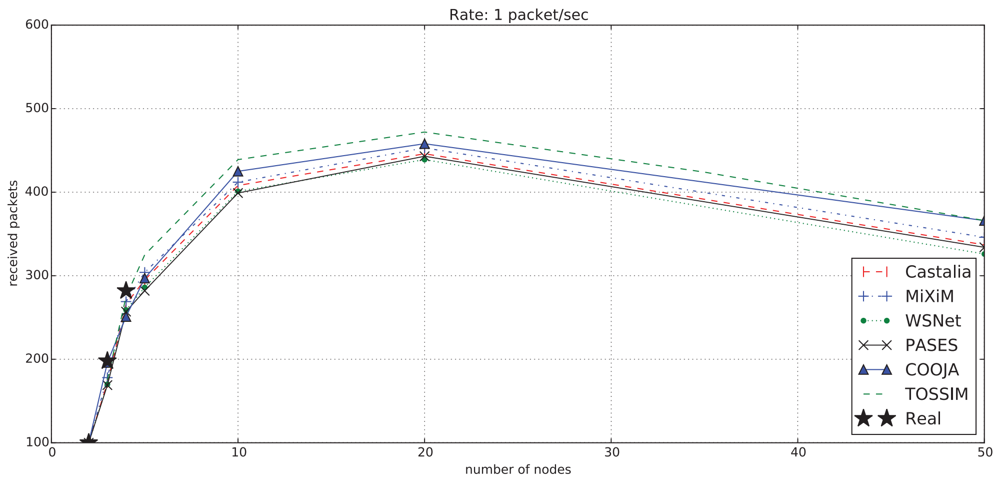


Fig. 6. Radio state transition waveforms in PASES.

up to 80% of the total runtime in TOSSIM. The actual runtime performance, excluding the noise model initialization, is comparable with other tools in our study, as shown by the TOSSIM* line on our performance plots (Figures 4 and 5). When the packet rate increases, the impact of the precomputation phase on the overall simulation is reduced in relative terms, making the performance of TOSSIM more in line with the other simulators. Note that all simulations in TOSSIM are done using Python scripts, which leads to a slight performance degradation in comparison with a C++ implementation.

The PASES framework also exhibits relatively low simulation scalability and poor runtime performance, being the second slowest tool in our experiments. The high computation demands shown by PASES are explained by its node-oriented architecture that captures the low-level features of the underlying hardware models. That, in turn, leads to an increased number of events and simulation overhead. Thus, as we shall see in Section 5, it provides better power consumption estimation. However, by reducing the details in the hardware models, the execution performance can be improved up to 40%, as shown in Figure 4 and Figure 5 by the data series denoted as "PASES light"; this happens without negatively affecting the power consumption estimation (see also the discussion in Section 5.2). The lightweight model is formed by averaging multiple intermediate transition states of the dedicated hardware components (CPU and RF) into a reduced set of states with the average amount of current drawn, shown in Figure 6. Performance can be further improved by simplifying the model even more; however, this will sacrifice the energy estimation capabilities of the tool.

The MiXiM tool also exhibits relatively low runtime performance that, however, is comparable to the runtime execution time of PASES running the lightweight model. We were unable to identify the actual reason for the poor performance in MiXiM, and additional experiments would be required to find performance bottlenecks in its simulation algorithm. In contrast to the previously discussed tools (TOSSIM, MiXiM, and PASES), the WSNet toolkit is found as the fastest and most scalable simulation



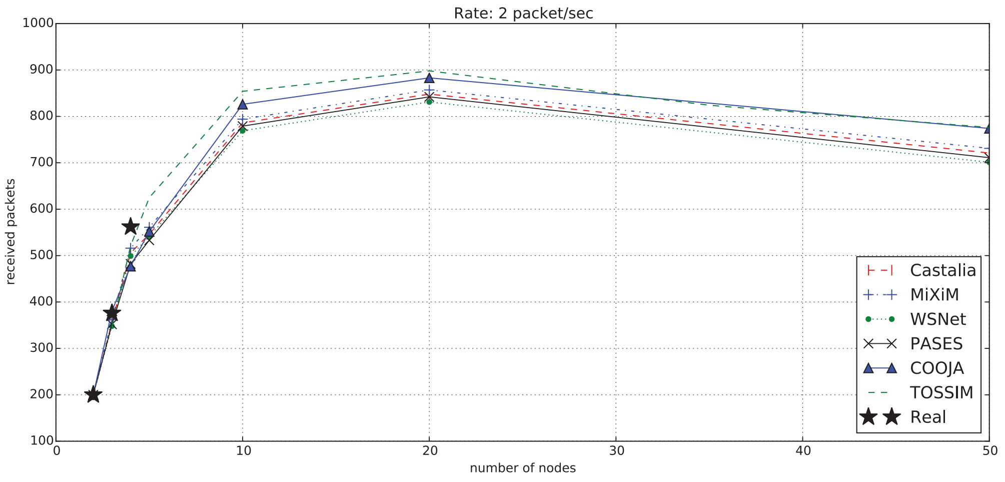
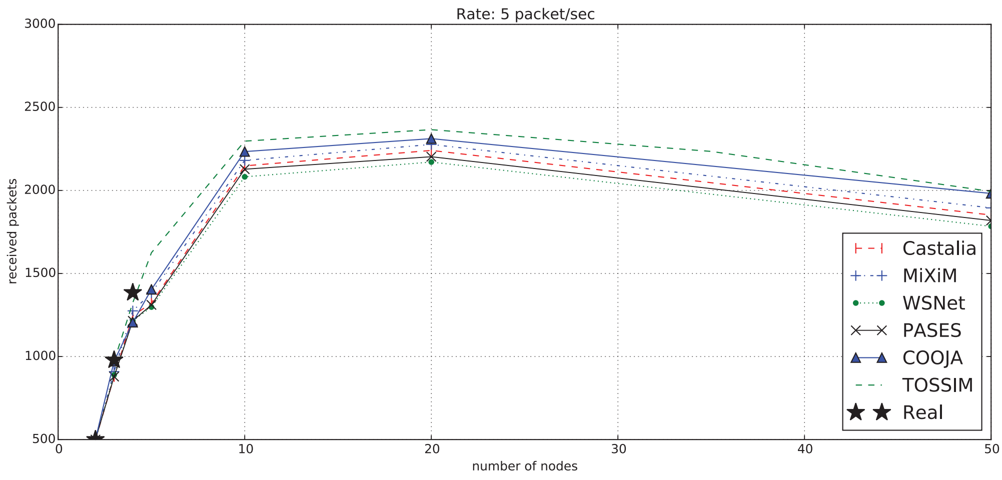


Fig. 7. Network throughput versus packet rate (base back-off value = 0.32ms).

environment among the ones compared. Castalia and COOJA show runtime results comparable to WSNet, making them also efficient tools for rapid WSN analysis.

The network throughput results are obtained in the same simulation series. The graphs in Figure 7 and Figure 8 show the total number of received packets (goodput) at the sink side for the various network sizes and transmission rates. The figures also include the results obtained on a real-world, small-scale (i.e., two, three, four, and five nodes) test bench deployment, which is functionally equivalent to the TinyOS application, as described at the end of Section 3.1. The purpose of this setup is to give reference values regarding the network behavior, which are useful to validate the simulation data. These, however, are not intended to be representative of larger-scale scenarios. As expected, the number of successfully delivered packets steadily increases with the network size and the packet rate. However, when the number of packets exceeds a particular threshold, the network throughput drops considerably, as shown by the simulations in all the tools. This effect is caused by the short value of the CSMA base back-off time (320 microseconds by default) that leads to a high level of collisions between contending nodes. By increasing the base back-off period,



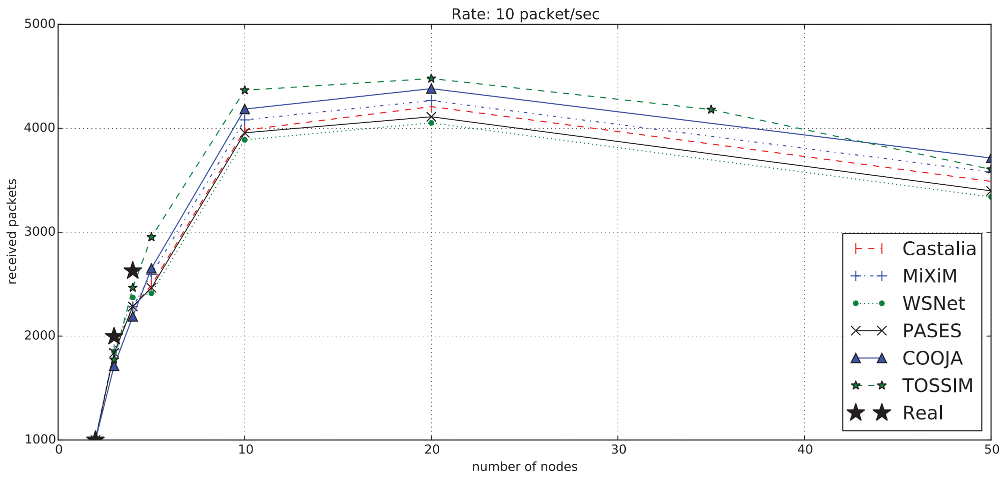


Fig. 8. Network throughput versus packet rate (base back-off value = 0.32ms).

we can improve the network throughput by letting fewer nodes contend in one time unit, consequently decreasing the probability of collisions. It also should be pointed out that the delivery latency depends linearly on the back-off time: the shorter the back-off time, the smaller the latency. Therefore, by increasing the base back-off value, we also increase the packet delivery delay.

Figure 9 and Figure 10 show the same simulation results in terms of network throughput when the base back-off value is set equal to 960 microseconds (three times more than before).

In this case, in all toolkits the total number of successfully delivered packets is higher than before and all curves steadily grow according to the packet rate. Simulation performance in this case slightly degrades compared to the previous case due to the increased number of packets that must be handled.

For both scenarios, all the tools produce qualitatively consistent results in terms of packet delivery amount and dynamics. Despite the equivalent simulation setups, the specific numbers output by the tools vary slightly, with an average deviation of 15%. As mentioned in Section 4, the deviation in simulation results can be explained by (1)

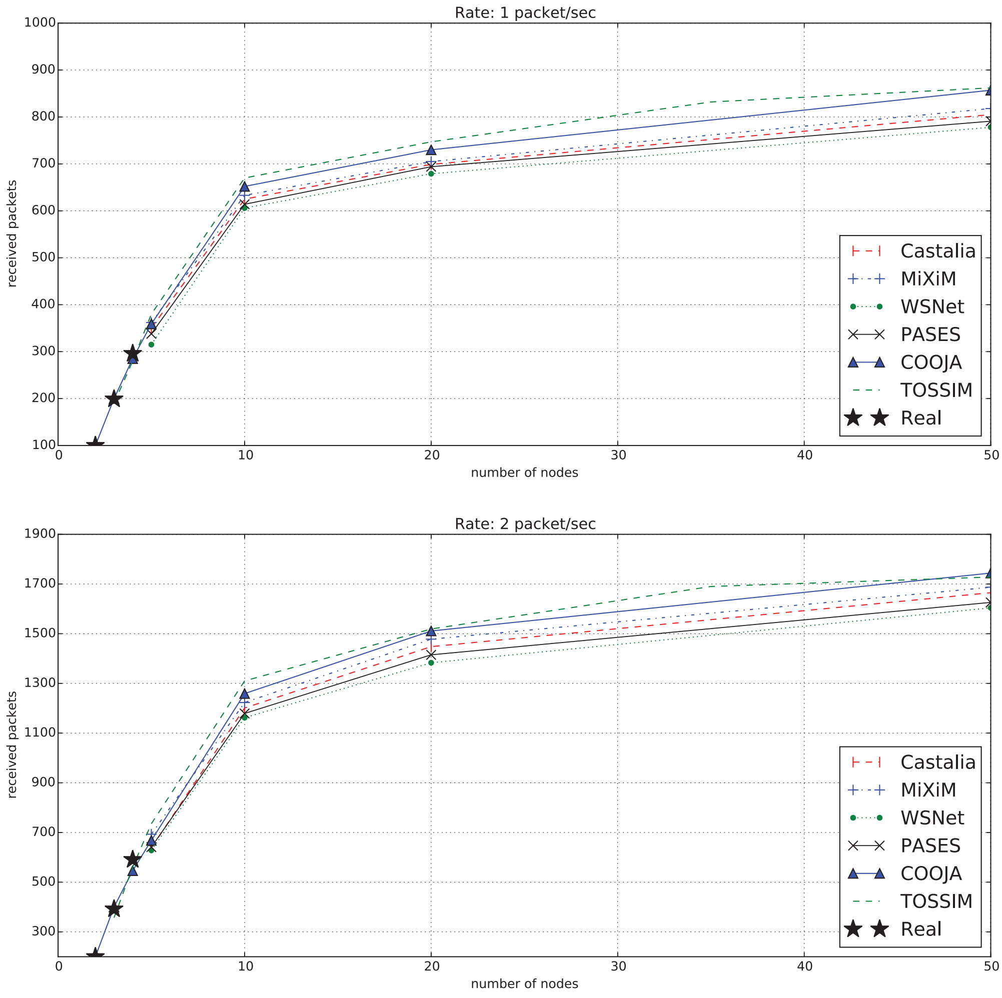
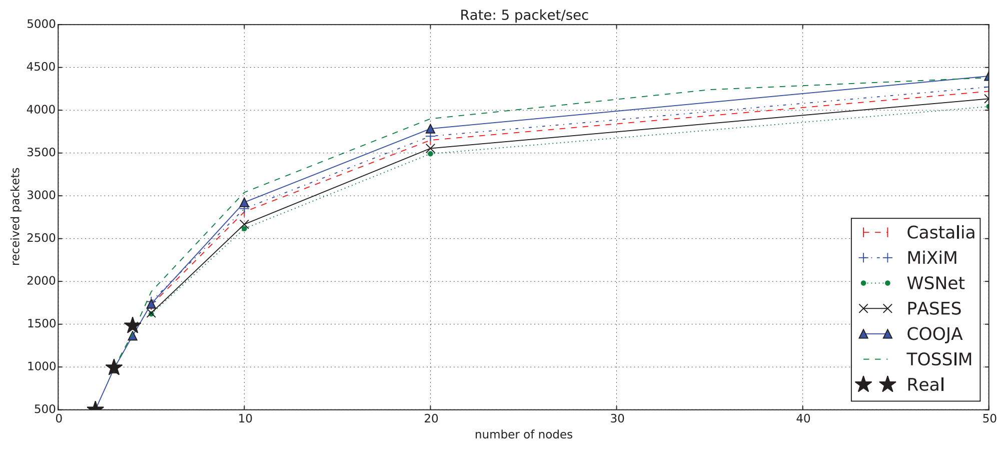


Fig. 9. Network throughput for low packet rates (base back-off value = 0.96ms).

the implementation and minor functional differences in communication channel models and (2) event and packet handling algorithms applied in the different tools. The noticeable outcome from this study is that the variation among tools strongly follows a precise pattern. In particular, WSNet always provides the lowest throughput in all simulation cases, while the TOSSIM toolkit shows the highest level of delivered packets. The remaining tools (Castalia, MiXiM, COOJA, and PASES), on the other hand, demonstrate similar results with a slight deviation of about 8% on average. Note that, compared to our small-scale networks, COOJA, along with TOSSIM, present results that are slightly better than the other tools and closer to the actual behavior. The TOSSIM highest throughput results are explained by its empirically based communication model that among other factors accounts for the radio "capture effect." The capture effect, also called cochannel interference tolerance, is the ability of some radios to receive a signal from one transmitter despite interference from another transmitter (even if the relative strengths of the two signals are almost the same) [Leentvaar and Flint 1976].



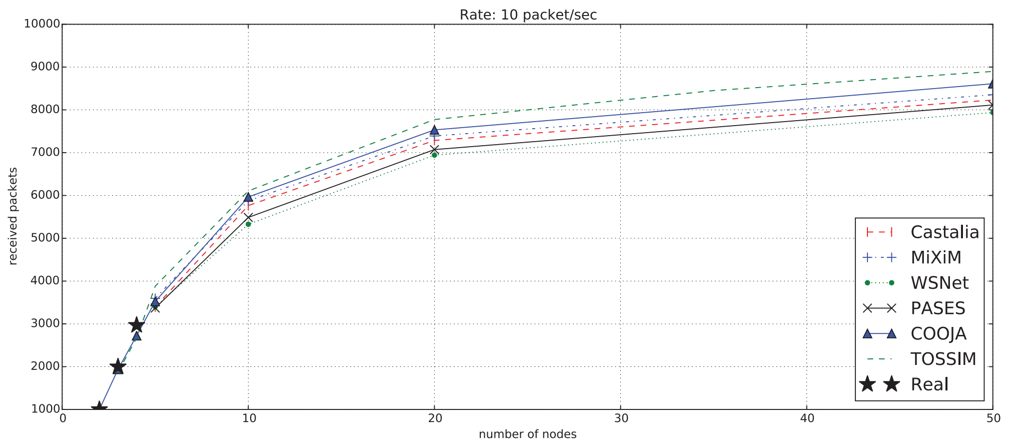


Fig. 10. Network throughput for high packet rates (base back-off value = 0.96ms).

Figure 11 demonstrates how different tools behave when the CSMA algorithm is disabled (the results are shown for higher packet rates, since those are the most interesting ones). In this case, all sender nodes run as follows: after the first back-off period, CCA always reports a clear channel condition and packets are transmitted. As for the network throughput with CSMA, TOSSIM delivers more packets in an amount that is close to the reference values obtained from the reference small-scale scenario. This demonstrates that the capture effect also takes place using the real nodes, which in fact are able to achieve a throughput in line with the TOSSIM estimation.

Along with the network throughput, we gather statistics on the total number of packets discarded at the MAC layer of the transmitting nodes. Packets are dropped when the maximum number of allowed CSMA back-off cycles (which is four by default) is exceeded, therefore, when the channel is found busy. Figure 12 shows the total number of dropped packets for both base CSMA back-off values and a data rate of 10 packets per second. In both cases, packet drop grows linearly with the packet rate. However, extended CSMA back-off leads to considerably fewer packet losses at the MAC layer in contrast to the initial back-off value. The obtained results are consistent with the throughput values. Packet drop is higher for the WSNet tool, while the lowest

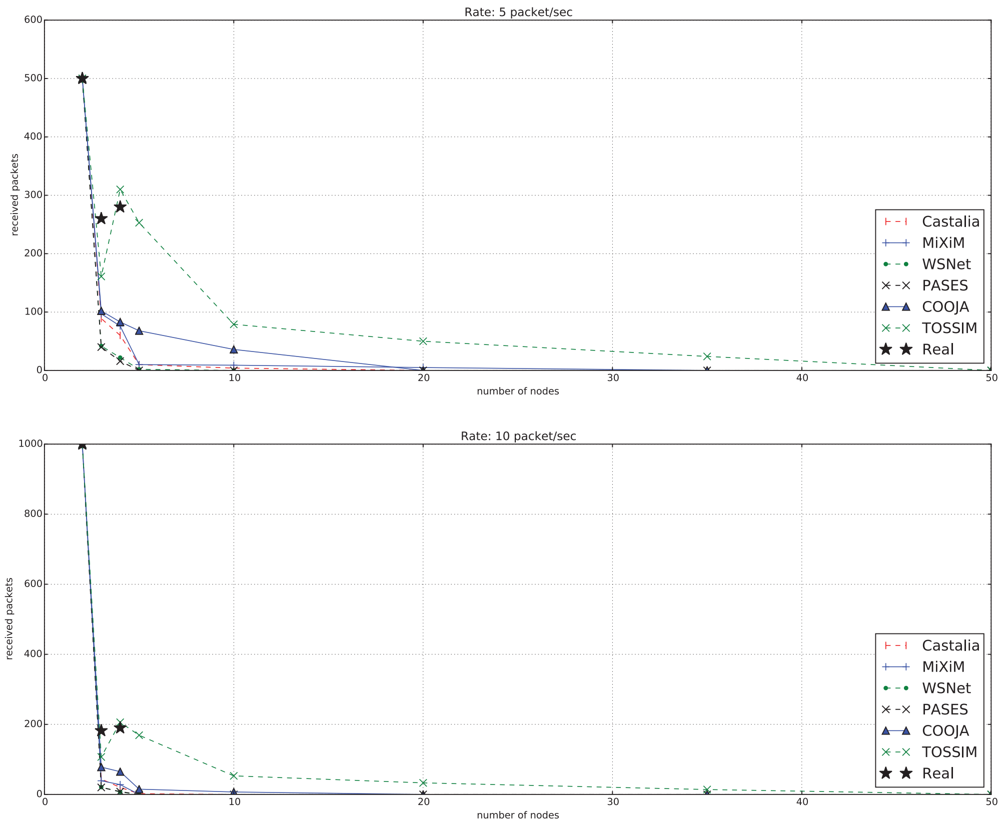


Fig. 11. Network throughput without CSMA.

level is shown by the TOSSIM toolkit, again consistently with the results obtained for throughput.

Furthermore, in this simulation case, the network packet delivery ratio (PDR) is studied for two different medium conditions, which model (1) an ideal situation with no shadowing and (2) a close-to-real obstructed indoor communication environment. As mentioned earlier, the tools provide different communication models that lead to differences in model settings, input data, and, consequently, simulation outcomes. In spite of that, we tried to provide unified model parameters and conditions that would make the communication models comparable. Figure 13 presents PDR results obtained for the log-normal shadowing communication model performed in the Castalia, MiXiM, WSNet, and PASES tools. All these tools exhibit very close results with an average variation of 7%. The plot shows the individual values obtained from the tools, while the line represents their average value. The figure also shows how the CSMA back-off settings affect the PDR at the various communication conditions. As expected, the extended back-off base value always produces a higher PDR. However, the PDR drops below 50% of successfully delivered packets at network sizes exceeding 10 nodes, even when an ideal communication medium is modeled. The PDR can be improved by extending the CSMA base back-off value even further, by increasing the number of trials to send a packet (*MaxCSMABackoffs* parameter), and by adjusting the node output power. For the obstructed network condition, the PDR curves start from a point below 50% due to the heavy fading conditions.

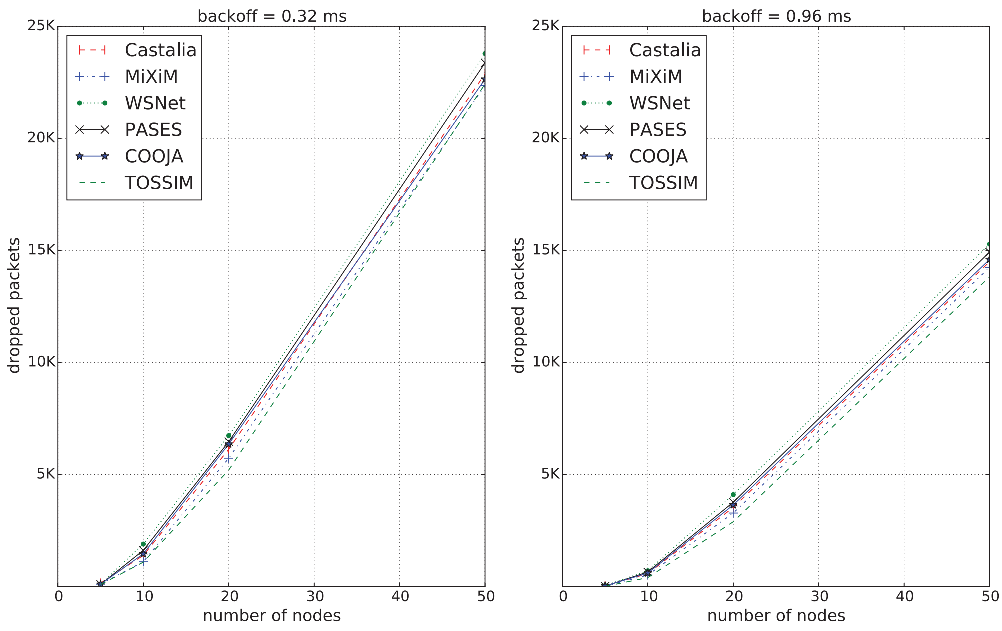
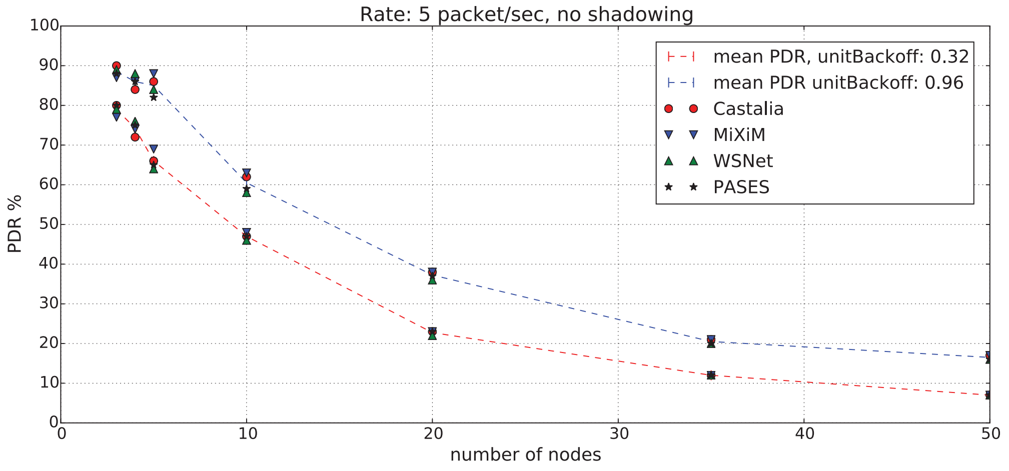


Fig. 12. Packet loss at MAC layer versus packet rate.

Figure 14 presents comprehensive results for all six tools running various communication medium models. The plot combines the curves presented on Figure 13 as well as PDR results for the COOJA and TOSSIM tools. It is noticeable that, in spite of the fact that different communication models were used, the obtained PDR results exhibit equal dynamics and partially comparable values. Nonetheless, for small network topologies, the difference in results from the various tools is quite evident. That in turn can again be explained by the capture effect that dominates on small topologies and decreases when the network size exceeds 10 nodes. The PDR results for larger topologies confirm our assumption that the specific settings for the noise floor for the TOSSIM SNR-based and COOJA MRM models, and the fading factor for the log-normal shadowing algorithm, produce comparable results. Such an outcome is also due to the absence of interfering networks in our setup, which may need to be accounted for in certain environments, and which contribute fundamentally to the noise levels. For more details on this aspect, we refer the reader to the original work by Lee et al. from which we have extracted the model [Lee et al. 2007].

Finally, Figure 15 and Figure 16 show the network latency for the higher packet rates (i.e., five and 10 packet/sec) and 320 and 960 microseconds base back-off time. The network latency is computed as the end-to-end packet delivery delay from the application layer of the sender node to the application layer of the destination node. More in detail, the latency can be divided in (1) the time required to process the packet from the application layer down to the radio layer and start the packet transmission; (2) the packet transmission time, given by the packet length and the bit rate, multiplied by the number of times the packet must be retransmitted; (3) the average back-off time, also multiplied by the number of retransmissions; and (4) the time to go back through the protocol stack up to the application layer, after a successful transmission. The results show a variation among the tools that is much more significant than the results obtained for throughput and packet delivery ratio. The reason has to do with both the nature of the metrics and the nature of the tools. For the first one, the number of times



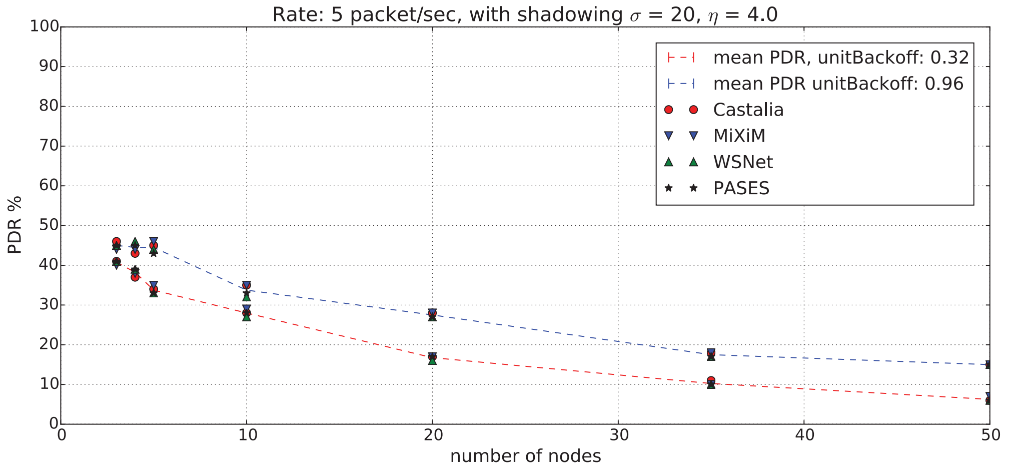
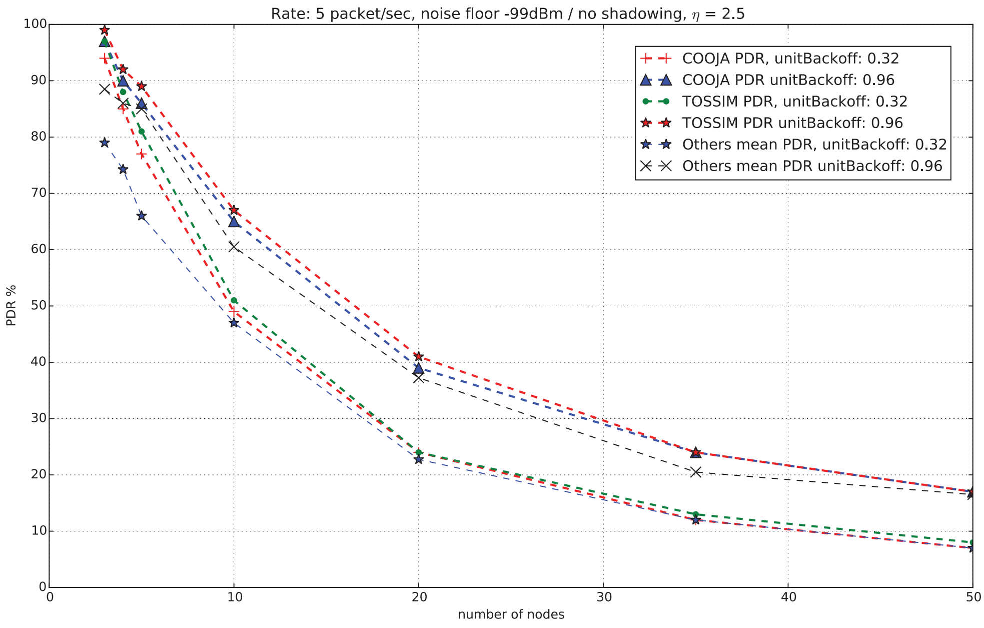


Fig. 13. PDR versus network size.

a packet must be retransmitted does not influence throughput and PDR, as long as the packet is eventually successfully delivered. Therefore, only differences in the *last* retransmission of each packet (whether it is successful or not) affect those metrics. On the other hand, latency is increased by *all* retransmissions of a successfully delivered packet. Hence, deviations in the way the channel is modeled have a more pronounced effect on latency, as shown in the figures. In particular, from the figures, it emerges that TOSSIM has the lowest average latency. This is to be expected, since TOSSIM employs a more accurate model that accounts for the already discussed "capture effect." In fact, it increases the efficiency of the channel and, therefore, decreases the number of retransmissions required to deliver a packet. For the small-scale networks (i.e., two, three, four, and five nodes), COOJA is actually closest to the real measurements on our deployment. This is because COOJA accounts for the MAC layer processing, whereas this time is not included in the TOSSIM simulation, which, therefore, shows a lower latency than the testbed. COOJA also has lower latency than the other tools on account of its more accurate channel model. Finally, the other tools stack up against each other consistently with the results obtained from the throughput analysis, and have higher latencies than the testbed. In particular, WSNet always shows the highest



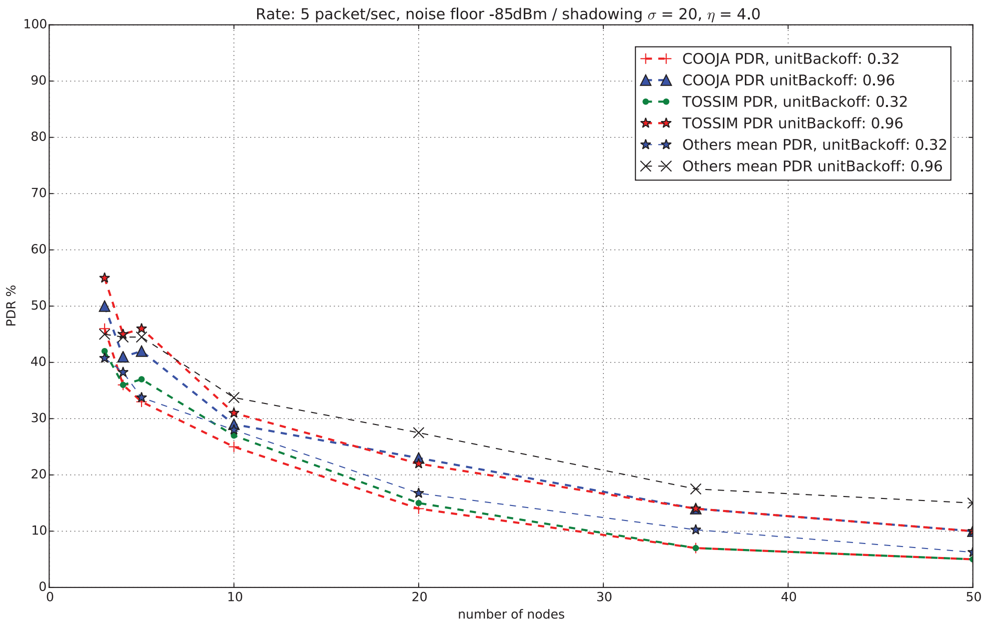


Fig. 14. PDR versus packet network size.

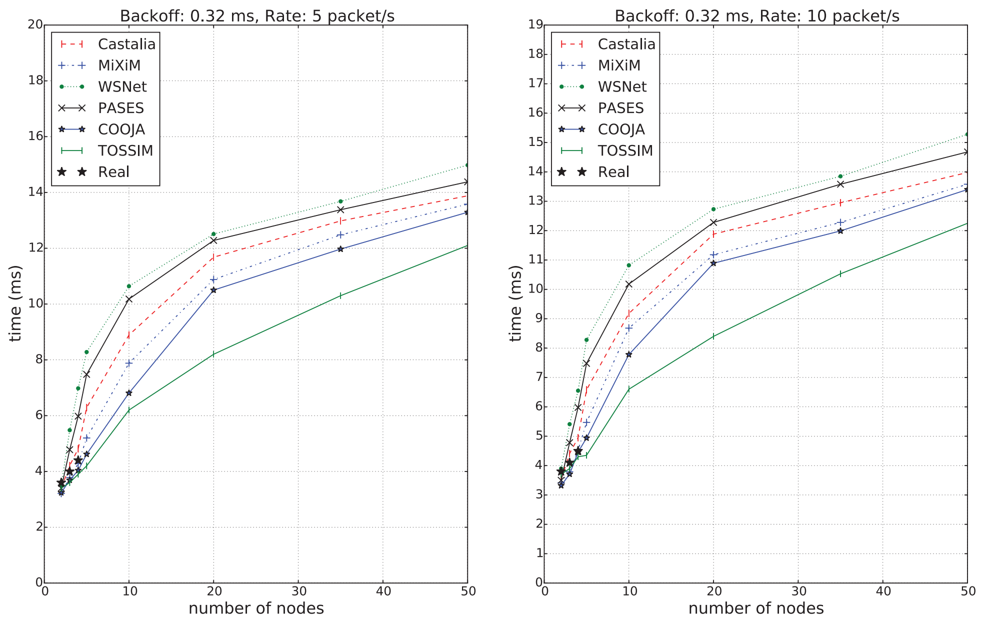


Fig. 15. Network latency for high packet rates (base back-off value = 0.32ms).

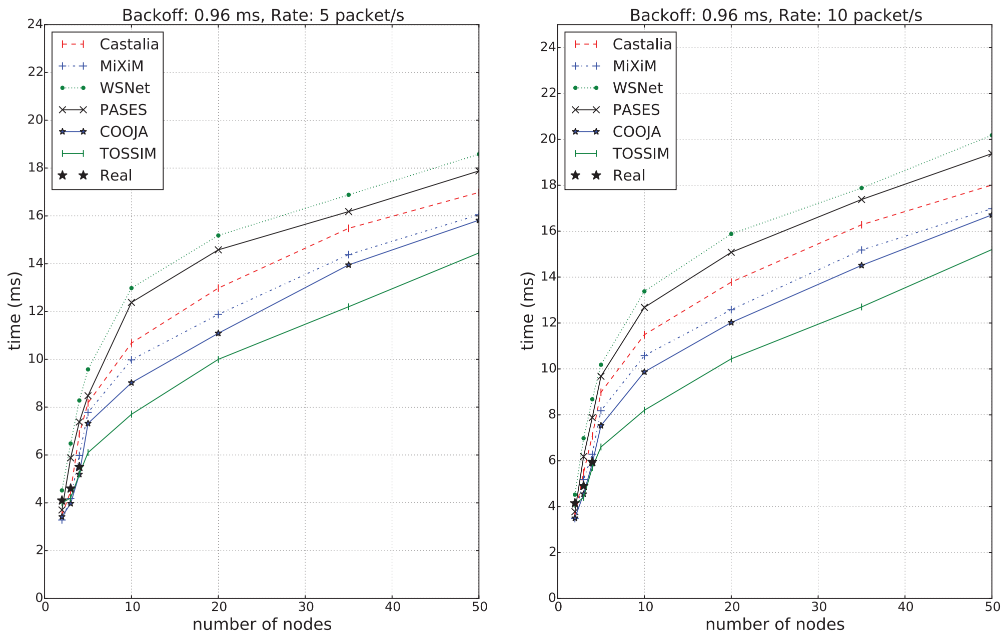


Fig. 16. Network latency for high packet rates (base back-off value = 0.96ms).

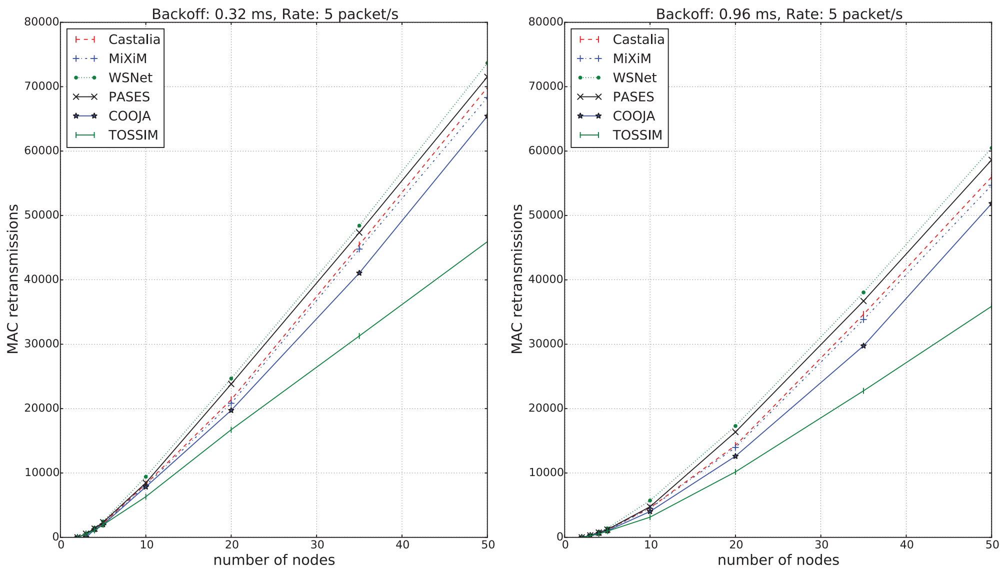


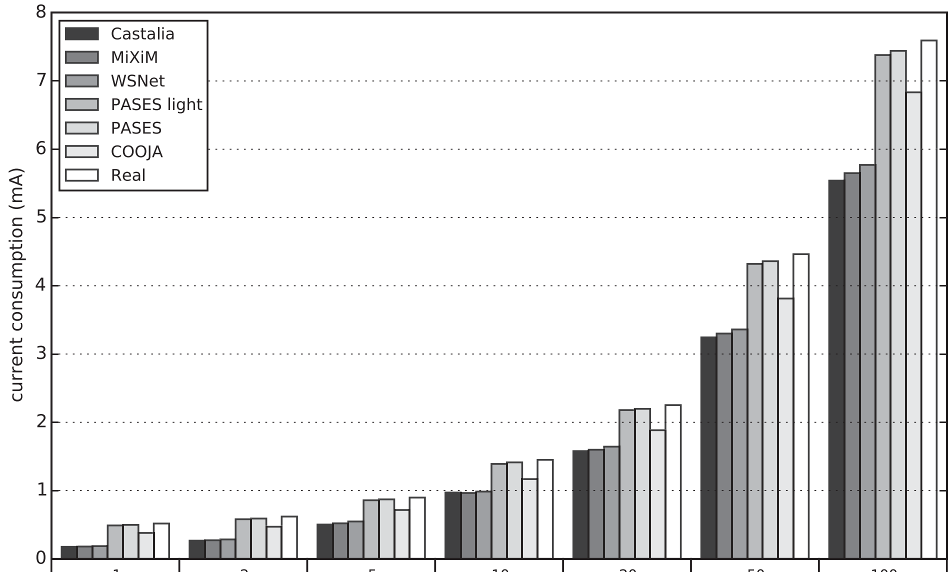
Fig. 17. MAC retransmissions.

levels of latency, closely followed by PASES. Castalia and MiXiM sit in between and close the gap with COOJA. To confirm our results, we also analyzed the total number of retransmissions for each of the tools, as shown in Figure 17, which are consistent with the network latency.

## 5.2. Simulation Results II

In the second study, we carried out simulations as well as actual experiments with varied transmission rates, ranging from one to 100 packets per second. The objective is to compare the power consumption estimation from simulation with the measured power consumption from a real WSN prototype (see Section 4 for a description of the experimental setup). It should be noticed that Castalia, MiXiM, and WSNet provide power estimation solely for the RF subsystem. Hence, the energy estimation results in these tools are expected to be lower than measurements, since they ignore the energy consumption of other peripheral components. Regarding COOJA, as described in Section 4, we have used the Energest extension, which allows us to separately estimate the energy consumption of reception, transmission, CPU execution, and low power mode. PASES, in turn, provides functional and experimentally based energy models for all hardware components typically present in a WSN platform. Therefore, we expect to get very accurate power estimation results by this tool. Figure 18 presents the power consumption simulation results and the reference experimental values for various transmission rates (power consumption is expressed as the average current drawn from the batteries). Based on the obtained results, we observe that different tools provide diverse levels of reliability for power consumption estimation. As expected, power consumption results in Castalia, MiXim, and WSNet are always lower than the reference values due to the lack of a complete hardware platform model in these toolkits. The outcomes of COOJA are more accurate with respect to those tools, since all four metrics previously introduced have been considered, thus obtaining a more precise model.

The accuracy of the power estimation that accounts only for RF operations grows proportionally to the transmission rate, as the RF part becomes a greater contributor



| | 1 | 2 | 5 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|---|
| Castalia | 0.176, -66.0% | 0.266, -57.0% | 0.502, -44.0% | 0.971, -33.0% | 1.578, -29.9% | 3.244, -27.3% | 5.540, -27.0% |
| MiXiM | 0.180, -65.2% | 0.272, -56.1% | 0.520, -42.0% | 0.964, -33.5% | 1.598, -29.0% | 3.300, -26.1% | 5.650, -25.6% |
| WSNet | 0.186, -64.0% | 0.284, -54.1% | 0.547, -39.0% | 0.985, -32.1% | 1.643, -27.0% | 3.360, -24.7% | 5.770, -24.0% |
| PASES light | 0.489, -5.4% | 0.580, -6.3% | 0.858, -4.3% | 1.390, -4.1% | 2.179, -3.2% | 4.320, -3.2% | 7.380, -2.8% |
| PASES | 0.497, -3.9% | 0.590, -4.7% | 0.871, -2.9% | 1.413, -2.5% | 2.197, -2.4% | 4.360, -2.3% | 7.441, -2.0% |
| COOJA | 0.379, -26.7% | 0.470, -24.1% | 0.715, -20.3% | 1.168, -19.4% | 1.884, -16.3% | 3.813, -14.6% | 6.834, -10.0% |
| Real | 0.517 | 0.619 | 0.897 | 1.450 | 2.252 | 4.463 | 7.593 |

Fig. 18. Power consumption estimation versus measurement.

to system energy consumption. At the lower transmission rates, the impact of peripheral components (e.g., CPU, sensors, memory) has a dominant effect on the power consumption that, in turn, leads to the poor estimation results. The breakdown of the component states and their relative contributions to the system current consumption at various network rates were computed using PASES and are presented in Figure 19. The results show how the current consumption proportion between different peripheral components and operating states changes with the packet rate. At the lowest network rate, the contribution of the CPU and Timer components taken together dominates (60%) the system consumption. With more frequent utilization of the RF subsystem, the relative contribution of the CPU and Timer components correspondingly decreases and drops below 20% at the highest rate.

PASES performs well with both the "detailed" and the "lightweight" model. The reason is that the data for the "lightweight" model are computed by averaging the values obtained by running the accurate system simulation. One may therefore question the benefits of a "detailed" model, as long as reasonable averages can be obtained. Nonetheless, average values fail to capture the dynamics of the current drawn from the battery, whose behavior is significantly affected by the peaks of the current consumption. Thus, while power consumption may be the same, lifetime estimation could give results that may deviate substantially from the experiments, depending on the kind of battery model used by the tool, such as a linear as opposed to a relaxation type. Since only

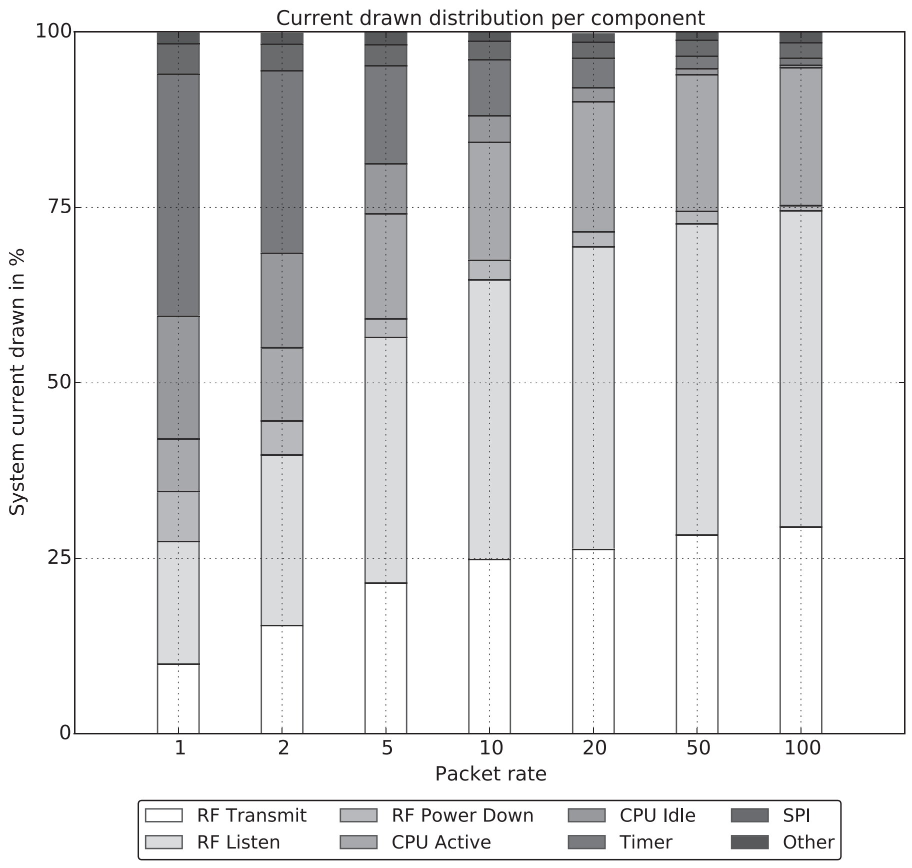


Fig. 19. Power consumption distribution per component.

PASES provides a relaxation model for the battery, we are unable to present a specific comparison on this aspect.

Summarizing, in this study, the Castalia, MiXiM, and WSNet toolkits show significant deviations from the actual measured values, with Castalia estimating the lowest level of energy spent by the nodes for all transmission rates. In contrast, PASES shows the most accurate results due to its lower abstraction level modeling capabilities and the experimentally obtained energy model for the studied hardware platform. On average, the error for the PASES "detailed" and "lightweight" energy models is computed as 2.7% and 3.9%, respectively. The other tools, which account solely for the RF operations, are still able to provide reasonable results. That at least might give designers an idea about the magnitude of system power consumption to expect, in relation to the degree of activity of the application. However, the use of energy-expensive components, such as extra sensors or actuators, must in these cases be accounted for separately. Interestingly, and as one may expect, the estimation error for these tools is roughly proportional to the percentage contribution of the extra components shown in Figure 19.

## 6. CONCLUSION

In this article, we have presented a comparative study of six open-source WSN simulation toolkits. Two simulation scenarios were designed and equally implemented in each of the tools to assess and compare them with respect to runtime performance, network throughput, packet loss at the MAC layer, packet delivery ratio, network latency, and accuracy of power consumption estimation. All the benchmark applications for each tool studied in this article are available for download at http://eecs.disi.unitn.it/pases in the Download section. Additionally, simulation results including throughput and power consumption estimation were compared with measured values obtained on a real test bench, for a small-scale network. Despite the differences in simulation analysis capabilities and available component models, the results show the correctness of the benchmark methods exploited in our study.

In the first test case, we found that WSNet, Castalia, and COOJA are very efficient tools for large-scale network modeling, while MiXiM, PASES, and TOSSIM require a larger amount of computation resources and time to run the same network setups. Nevertheless, all the tools present similar results in terms of network throughput and communication modeling for various network sizes and packet rates. Network latency results show a wider variation, as the effects of differences in modeling the channel have a larger impact. This analysis prompts the use of a more detailed model whenever latency has to be estimated more accurately, as in real-time control applications.

In the second case, due to their limited capability to model the full node architecture, Castalia, MiXiM, and WSNet showed lower simulated values than in the experiment. However, these tools still provide reasonable estimation that allows a designer to assess an order of expected energy consumption, at least in relation to the application network characteristics. PASES, in turn, being a tool specially designed to analyze power consumption of embedded systems, provides the most benefits in this area and produces the most accurate energy estimation results. COOJA offers a moderate level of power estimation accuracy that, combining with results obtained in runtime and throughput studies, makes this toolset a really good choice for WSN modeling. Additionally, COOJA enables execution and debugging of a real target application code, making transition from pure simulation into actual deployment easier and faster than any other tool presented in this article can offer. TOSSIM, being a part of a standard TinyOS distribution, is suitable only for applications designed in this framework. Nevertheless, in some practical case, TOSSIM does not fully support all standard interfaces and modules present in real code. That, in combination with absence of any power profiling facilities, makes TOSSIM a very specific and narrow tool for WSN applications. However, the TOSSIM SNR-based communication model is considered to be a very accurate network model, as proved by our experiments.

The general usability of the different tools cannot be directly evaluated since it is a matter of personal preference and experience with modeling tools. We point out that having the proper documentation and the availability of simulation examples may constitute one of the most significant factors in the choice of a particular tool over another. Additionally, the GUI support for simulation setup and postprocessing results interpretation is a big point for a modern state-of-the-art simulation environment. In such a way, Castalia and WSNet are the easiest tools to get started even without having any prior experience in the WSN field. PASES offers a wide set of examples and GUI utilities but lacks comprehensive documentation support. Both MiXiM and TOSSIM require some level of expertise in OMnet++ and the TinyOS frameworks, respectively. Finally, regarding COOJA/Contiki, the scientific community is very active in guaranteeing support for the developers who are interested in exploiting this tool; moreover, COOJA/Contiki provides many examples from which developers can start to modify in order to realize their own projects.

Our current work focuses on exploring the simulator capabilities when considering higher layers of the communication protocol [Minakov et al. 2016].

## REFERENCES

P. Baldwin, S. Kohli, A. L. Edward, X. Liu, and Y. Zhao. 2005. *VisualSense: Visual Modeling for Wireless and Sensor Network Systems*. Technical Memorandum UCB/ERL M05/25. University of California, Berkeley, CA.

R. Barr, J. Haas Zygmunt, and R. van Renesse. 2005. JiST: An efficient approach to simulation using virtual machines. *Software: Practice and Experience* 35, 6 (2005), 539–576.

M. Baz and D. A. J. Pearce. 2011. An introduction to DANSE. In *Proceedings of the 12th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*.

A. Bonivento, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. 2006. Platform based design for wireless sensor networks. *Mobile Networks and Applications* 11, 4 (August 2006), 469–485.

A. Boulis. 2007. Castalia: Revealing pitfalls in designing distributed algorithms in WSN. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys'07)*. 407–408.

A. Boulis and Y. Tselishchev. 2011. Effects of carrier sense modeling on wireless network simulation results. In *Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'11)*. 129–134. DOI: http://dx.doi.org/10.1145/2068897.2068921

B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. 2002a. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks* 8, 5 (2002), 481–494.

G. Chen, B. K. Szymanski, E. Yücesan, C. Chen, J. L. Snowdon, and J. M. Charnes. 2002b. Cost: A component-oriented discrete event simulator. In *Proceedings of the 34th Winter Simulation Conference: Exploring New Frontiers*, Vol. 1. ACM, 776–782.

E. Cheong, E. A. Lee, and Y. Zhao. 2005. Viptos: A graphical development and simulation environment for TinyOS-based wireless sensor networks. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*. 302–302.

A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu. 2013. METROII: A design environment for cyber-physical systems. *ACM Transactions on Embedded Computing Systems* 12, 1s (March 2013), 49:1–49:31.

I. Dietrich and F. Dressler. 2009. On the lifetime of wireless sensor networks. *ACM Transactions on Sensor Networking* 5, 1, Article 5 (February 2009), 39 pages. DOI: http://dx.doi.org/10.1145/1464420.1464425

I. T. Downard. 2004. *Simulating Sensor Networks in NS-2*. Technical Report. Naval Research Laboratory, Network and Communication Systems Information Technology Division, Washington, DC.

A. Dunkels, B. Gronvall, and T. Voigt. 2004. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th IEEE Conference on Local Computer Networks*. 455–462.

E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Marino, and J. Garcia-Haro. 2005. Simulation tools for wireless sensor networks. In *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*.

J. Eriksson, F. Osterlind, T. Voigt, N. Finne, S. Raza, N. Tsiftes, and A. Dunkels. 2009. Accurate power profiling of sensornets with the COOJA MSPSim simulator. In *Proceedings of the 6th IEEE Conference on Mobile Adhoc and Sensor Systems*. IEEE, 1060–1061.

E. Eyisi, J. Bai, D. Riley, J. Weng, Y. Wei, Y. Xue, X. D. Koutsoukos, and J. Sztipanovits. 2012. NCSWT: An integrated modeling and simulation tool for networked control systems. In *Proceedings of the 15th International Conference on Hybrid Systems: Computation and Control (HSCC'12)*.

L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. 2004. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM, New York, NY, 201–213. DOI: http://dx.doi.org/10.1145/1031495.1031519

T. Grötker, S. Liao, G. Martin, and S. Swan. 2002. *System Design with SystemC*. Springer.

C. Guillaume, F. Antoine, and E. Fleury. 2007. Worldsens: A fast and accurate development framework for sensor network applications. In *Proceedings of the 22nd ACM Symposium on Applied Computing*. 222–226.

T. He, B. Blum, Y. Pointurier, C. Lu, J. A. Stankovic, and S. Son. 2006. MAC layer abstraction for simulation scalability improvements in large-scale sensor networks. In *Proceedings of the 3rd International Conference on Networked Sensing Systems (INSS'06)*.

T. R. Henderson, M. Lacage, and G. F. Riley. 2008. Network simulations with the ns-3 simulator. In *Proceedings of the Special Interest Group on Data Communications*. 527.

IEEE Standard Association. 2006. *IEEE 802.15.4-2006 Standard for information technology*. IEEE Standard Association.

D. Jung, T. Teixeira, and A. Savvides. 2009. Sensor node lifetime analysis: Models and tools. *ACM Transactions Sensor Networking* 5, 1, Article 3 (February 2009), 33 pages. DOI:http://dx.doi.org/10.1145/1464420.1464423

A. Kamthe, M. Á Carreira-Perpinán, and A. E. Cerpa. 2013. Improving wireless link simulation using multilevel markov models. *ACM Transactions on Sensor Networks (TOSN)* 10, 1 (2013), 17.

I. Khemapech, A. Miller, I. Duncan, and N. Haugh. 2005. *Simulating Wireless Sensor Networks*. Technical Report. School of Computer Science, University of St Andrews.

A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. 2008. Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools'08)*. 71:1–71:8.

M. Korkalainen, M. Sallinen, N. Kärkkäinen, and P. Tukeva. 2009. Survey of wireless sensor networks simulation tools for demanding applications. In *Proceedings of the 5th International Conference on Networking and Services*. 102–106.

M. Kuorilehto, M. Kohvakka, J. Suhonen, P. Hamalainen, M. Hannikainen, and T. D. Hamalainen. 2008. *Ultra-Low Energy Wireless Sensor Networks in Practice. Theory, Realization and Deployment*. John Wiley & Sons Ltd, England.

O. Landsiedel, K. Wehrle, and S. Gotz. 2005. Accurate prediction of power consumption in sensor networks. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*. 37–44.

K. Langendoen and A. Meier. 2010. Analyzing MAC protocols for low data-rate applications. *ACM Transactions on Sensor Networking* 7, 2, Article 19 (Sept. 2010), 40 pages. DOI:http://dx.doi.org/10.1145/1824766.1824775

H. J. Lee, A. Cerpa, and P. Levis. 2007. Improving wireless simulation through noise modeling. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN'07)*. ACM, New York, NY, 21–30. DOI:http://dx.doi.org/10.1145/1236360.1236364

K. Leentvaar and J. Flint. 1976. The capture effect in FM receivers. *IEEE Transactions on Communications,* 24, 5 (May 1976), 531–539. DOI:http://dx.doi.org/10.1109/TCOM.1976.1093327

J. Lessmann, T. Heimfarth, and P. Janacik. 2008a. ShoX: An easy to use simulation platform for wireless networks. In *Proceedings of the 10th EUROS/UKSim International Conference on Computer Modelling and Simulation*. IEEE, 410–415.

J. Lessmann, P. Janacik, L. Lachev, and D. Orfanus. 2008b. Comparative study of wireless network simulators. In *Proceedings of the 7th International Conference on Networking*. 517–523.

P. Levis, N. Lee, M. Welsh, and D. Culler. 2003. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. 126–137.

P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. A. Brewer, and D. E. Culler. 2004. The emergence of networking abstractions and techniques in TinyOS. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*. 1–14.

B. Li, Z. Sun, K. Mechitov, G. Hackmann, C. Lu, S. Dyke, G. Agha, and B. Spencer. 2013. Realistic case studies of wireless structural control. In *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'13)*.

X. Liu, Y. Xiong, and E. A. Lee. 2001. The ptolemy II framework for visual languages. In *Proceedings of the International Symposium on Human-Centric Computing Languages and Environments*. 50.

I. Minakov and R. Passerone. 2013. PASES: An energy-aware design space exploration framework for wireless sensor networks. *Journal of Systems Architecture* 59, 8 (September 2013), 626–642. DOI:http://dx.doi.org/10.1109/JSEN.2013.2263962

I. Minakov, R. Passerone, A. Rizzardi, and S. Sicari. 2016. Routing behavior across WSN simulators: The AODV case study. In *Proceedings of the 12th IEEE World Conference on Factory Communication Systems (WFCS'16)*.

L. Mottola, G. P. Picco, M. Ceriotti, Ş. Gună, and A. L. Murphy. 2010. Not all wireless sensor networks are created equal: A comparative study on tunnels. *ACM Transactions on Sensor Networks* 7, 2, Article 15 (September 2010), 33 pages. DOI:http://dx.doi.org/10.1145/1824766.1824771

G. Myklebust. 2004. *The AVR Microcontroller and C Compiler Co-Design*. Technical Report. ATMEL Corporation, Trondheim, Norway.

opnet. 2014. OPNET. http://www.opnet.com/. (Accessed December 2014).

F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. 2006. Cross-level sensor network simulation with COOJA. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*. 641–648.

L. Palopoli, R. Passerone, and T. Rizano. 2011. Scalable off-line optimization of industrial wireless sensor networks. *IEEE Transactions on Industrial Informatics* 7, 2 (May 2011), 328–339.

D. Pediaditakis, Y. Tselishchev, and A. Boulis. 2010. Performance and scalability evaluation of the castalia wireless sensor network simulator. In *Proceedings of the 3rd International Conference on Simulation Tools and Techniques*. 53.

E. Perla, A. Ó Catháin, R. S. Carbajo, M. Huggard, and C. McGoldrick. 2008. PowerTOSSIM: Realistic energy modelling for wireless sensor network environments. In *Proceedings of the 3rd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM2HW2N'08)*. 35–42.

A. Pinto, A. Bonivento, A. L. Sangiovanni-Vincentelli, R. Passerone, and M. Sgroi. 2006. System level design paradigms: Platform-based design and communication synthesis. *ACM Transactions on Design Automation of Electronic Systems* 11, 3 (July 2006), 537–563.

J. Polastre, R. Szewczyk, and D. E. Culler. 2005. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*. 364–369.

T. S. Rappaport. 2002. *Wireless Communications: Principles and Practice* (2nd ed.). Prentice Hall PTR, NJ.

M. Rossi, L. Rizzon, M. Fait, R. Passerone, and D. Brunelli. 2014. Energy neutral wireless sensing for server farms monitoring. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 4, 3 (September 2014), 324–334.

L. Shu, M. Hauswirth, H.-C. Chao, M. Chen, and Y. Zhang. 2011. NetTopo: A framework of simulation and visualization for wireless sensor networks. *Ad Hoc Networks* 9, 5 (2011), 799–820.

Simpy. 2013. Simpy. http://simpy.sourceforge.net/. (Accessed April 2013).

Simulink. 2015. Simulink. http://it.mathworks.com/. (Accessed January 2015).

C. P. Singh, O. P. Vyas, and M. K. Tiwari. 2008. A survey of simulation in sensor networks. In *Proceedings of the International Conference on Computational Intelligence for Modelling Control Automation*. 867–872.

A. Sobeih, M. Viswanathan, D. Marinov, and J. C. Hou. 2007. J-sim: An integrated environment for simulation and model checking of network protocols. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*. 1–6.

A. Somov, I. Minakov, A. Simalatsar, G. Fontana, and R. Passerone. 2009. A methodology for power consumption evaluation of wireless sensor networks. In *Proceedings of the 12st IEEE International Conference on Emerging Technologies and Factory Automation*. 1–8.

K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. 2010. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks* 6, 2, Article 16 (March 2010), 49 pages. DOI:http://dx.doi.org/10.1145/1689239.1689246

B. K. Szymanski and G. G. Chen. 2007. *Handbook of Dynamic System Modeling*. CRC/Taylor and Francis Publishing.

Texas Instruments Corporation. 2006. *MSP430x1xx Family. User's Guide, SLAU094F*. Texas Instruments Corporation.

Texas Instruments Corporation. 2013. *SWRS041c - 2.4 GHz IEEE 802.15.4 ZigBee-ready RF Transceiver* (revision c ed.). Texas Instruments Corporation.

A. Timm-Giel, K. Murray, M. Becker, C. Lynch, C. Gorg, and D. Pesch. 2008. Comparative simulations of WSN. In *Proceedings of ICT Mobile and Wireless Communications Summit*.

B. L. Titzer, D. K. Lee, and J. Palsberg. 2005. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*. 477–482.

A. Varga. 2001. The OMNeT++ discrete event simulation system. In *Proceedings of the 15th European Simulation Multiconference*. The Society for Computer Simulation, International, Prague, Czech Republic, 319–324.

D. Weber, J. Glaser, and S. Mahlknecht. 2007. Discrete event simulation framework for power aware wireless sensor networks. In *Proceedings of the 5th IEEE International Conference on Industrial Informatics*. 335–340.

E. Weingartner, H. vom Lehn, and K. Wehrle. 2009. A performance comparison of recent network simulators. In *Proceedings of the 5th IEEE International Conference on Communications*. 1–5.

Received February 2015; revised November 2015; accepted March 2016