Existing Platforms to Evaluate the Performance of Wireless Sensor Networks

Chen Haiming
Wireless Sensor Network Lab., ICT, CAS
chenhaiming@ict.ac.cn

Contents

1 Introduction 3

2 General Network Simulators 3
  2.1 NS-2 3
  2.2 OPNET 4
  2.3 OMNeT++ 5
  2.4 JavaSim/J-Sim 5
  2.5 COST 6
  2.6 Ptolomy II 6
  2.7 NCTUns 6
  2.8 JiST/SWANS 7
  2.9 Summary 7

3 Parallel and Distributed Simulation Platforms 7
  3.1 GloMoSim/Qualnet 7
  3.2 SSFNet/DaSSF 8
  3.3 GTNetS 8
  3.4 Summary 8

4 Sensor Network Simulators 9
  4.1 Prolwer 9
  4.2 SENS 9
  4.3 Sidh 9
  4.4 Shawn 10
  4.5 H-MAS 10
  4.6 GloNeMo 10
  4.7 Summary 10

5 Emulators 10
  5.1 TOSSIM 10
  5.2 COOJA 11
  5.3 ATEMU 11
  5.4 Avrora 11
  5.5 Emstar 12
6 Distributed Emulators
  6.1 VMNet ............................................................... 12
  6.2 DiSenS ............................................................. 12
  6.3 Summary ........................................................... 12

7 Testbeds
  7.1 GNOMES ............................................................ 13
  7.2 MoteLab ............................................................ 13
  7.3 EmuLab ............................................................. 13
  7.4 Kansci ............................................................... 13
  7.5 MiNT-m .............................................................. 13

8 Conclusion .......................................................... 14

9 Discussion and Future Work ...................................... 14

List of Tables

1 Milestones of the NS development ................................. 4
2 General Network Simulators and Their Extensions for Sensor Networks ........................................ 7
3 Parallel/Distributed Simulators and Their Extensions for Sensor Networks ..................................... 9
4 Sensor Network Simulators Developed by Universities ............................................................... 11
5 Sensor Emulators Developed by Universities ............................. 12
1 Introduction

With the fast development of embedded and networking technology, sensor networks become one of the hottest research topics in the world. At present, many funds are founded to support doing research on this advancing field. Many prototypes have been produced, as well as lots of simulation platforms have been developed. The increasing efforts put on this field has further accelerate the development of sensor networks, but a problem has also been rising.

The author of [1] named the problem as “fragment”, which means so many diverse prototypes and simulation tools developed by different institutions without compatibility. This problem has led to the difficulty of transplanting protocol modules and applications from one prototype to another, or from one simulation platform to another. In addition, this problem has resulted in the incomparability of the experimental results or simulation results.

We plan to establish a common simulation platform for evaluating the performance of sensor networks. Before diving into the project, we take a deep investigation on the currently existing tools and platforms. Based on the work done by the author of [2], we include more representative and latest developed tools into the report, and collect the developed testbeds for evaluating the performance of sensor network as well.

The rest of the report is organized as follows. The four most popular general network simulators, which are NS-2, OPNET, and OMNeT++, are introduced in Section 2. Section 4 presents several recently developed simulation tools especially for sensor networks. Some of them are based on the general network simulators introduced in Section 2, while the others are developed from the bottom. Next, four representative emulators for sensor networks are showed in Section 5. Additionally, we shows the three experimental sensor network testbeds developed by different universities in Section 7. A brief conclusion is made in Section 8. At last, we depict the future work to be done in Section 9.

2 General Network Simulators

2.1 NS-2

Evolving from the REAL (REalistic And Large) network simulator, which was developed by Keshav [3] in 1988 based on a modified version of the NEST (NEtwork Simulation Testbed) [4], the NS (Network Simulator) has been developed for almost 20 years.

NS is an object-oriented discrete event simulator. The original prototype (version 1.0a) [5] of NS was developed by the Network Research Group at the LBNL (Lawrence Berkeley National Laboratory) in 1995, mainly supporting for evaluating the performances of Transport Control Protocols and routing algorithms. It exploited C++ to implement the core set of high-performance simulation primitives and the Tcl [6] scripting language to express the definition, configuration, and control of the simulation. This C++/Tcl architecture makes the simulator easy to extend.

In 1996 NS development was supported by DARPA through the VINT (Virtual InterNet-Work Testbed) [7] project at LBNL, Xerox PARC (Palo Alto Research Center), UCB, and USC/ISI, which made the NS as a common simulation tool with features of abstraction and extensibility for network researchers to use in the design and deployment of new wide-area Internet protocols [8]. Since the VINT project made some fundamental changes in software architecture and defined the split-programming model based on OTcl (MIT’s Object Tcl) [9], the network simulator was named NS-2. The VINT project also extended the NS simulator in the aspects of emulation [10] and visualization [11].
In 1997, the CMU Monarch (MObile Networking ARCHitectures) project (now in Rice University) [12] extended the NS-2 to support for the simulation of wireless networks, which makes the NS-2 a simulation tool not only for wired networks but also for wireless and ad hoc networks. The milestones of the NS development is listed in the table 1.

Now NS-2 is maintained by the public as a project of open-source, and is accepted as one of the most popular simulation tools in the world. However, NS-2 has some drawbacks in scalability, customization and lack of an application model.

<table>
<thead>
<tr>
<th>Version</th>
<th>Events</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS 1.0a1</td>
<td>The first release</td>
<td>July 31, 1995</td>
</tr>
<tr>
<td>NS 2.0a1</td>
<td>Major architecture changes (based on OTcl)</td>
<td>November 6, 1996</td>
</tr>
<tr>
<td>NS 2.1b1</td>
<td>Incorporate tracing features for nam-1.0a2</td>
<td>November 11, 1997</td>
</tr>
<tr>
<td>NS 2.1b5</td>
<td>Incorporate the CMU wireless extension</td>
<td>March 16, 1999</td>
</tr>
</tbody>
</table>

With the occurrence of wireless sensor networks, some researchers tried to extend the NS-2 to meet the requirements of modeling the sensor networks. SensorSim [13] extends NS-2 in following three ways.

1. **A power model**, which takes into account each of the hardware components that would need battery power in order to operate.

2. **A sensor channel**, which includes sensing through both a geophone and a microphone.

3. **An interaction mechanism with external applications**, which allows for real sensed events to trigger reactions within the simulated environment.

Besides these improvements, SensorSim is featured by the SensorWare, which allows for dynamically managing nodes and provides a mechanism for distributed computation.

Taking into consideration the problem of scalability inherited from NS-2, the author of SensorSim did some updates in [14]. However, SensorSim has not yet addressed the problem of scalability very well. So this project has been stopped.

Another similar extension has been developed by the Naval Research Laboratory [15] to simulate the limited hardware and power of the sensor networks, and allow for external phenomena to trigger events as well.

### 2.2 OPNET

The OPNET network simulator [16] originated from early work on network modeling at MIT in the 1980s, and now is a commercial product of MIL3 company [17]. OPNET is another **discrete event, object-oriented, general purpose** network simulator. Models in the OPNET are organized in a hierarchical mode, which consists of network model, node model, and process model. These models are responsible for designing the network topology, defining the data flow and handling the control flow. Besides these models, a parameter editors is included.

OPNET has some strengths in customization, like modeling different sensor-specific hardware and defining custom packet format. However, it suffers from the same problem of scalability as NS-2. Additionally, OPNET is only available in commercial form, so the module extension in OPNET is not as prompt as NS-2.
2.3 OMNeT++

OMNeT++ [18] is a discrete event, component-based, general-purpose simulation environment written in C++, with strong GUI support and an embeddable simulation kernel.

The OMNeT++ is composed of modules, which are connected by messages to implement the simulator. The modules are organized in a hierarchical nested fashion, that is simple modules in the bottom, compound modules composed of simple modules, and system modules encompassing the compound modules. The system modules is referred to as the networks. The compound modules can be viewed as the hosts or nodes in the networks. The simple modules can be seen as the processing algorithms running on the host or node.

OMNeT++ exploits the C++ to implement the simple modules, and NED (NEtwork Description language) to describe the network topology. This two-language software architecture is similar with NS’s split-programming model. But the C++/NED architecture is more flexible than the C++/OTcl architecture, due to the difference between the NED and the OTcl. The NED is a description language that can be edited by graphical interface and exported into or imported from XML, while the OTcl is a scripting language that can only be interpreted by its shell.

In short, OMNeT++ is featured by its generic and flexible architecture. At present, it is an active public source software [19].

Based on the same architecture as SensorSim described in section 2.1, Mallanda [20] from Louisiana State University extended the OMNeT++ to support the simulation of sensor networks, which is named SensorSimulator. In accordance with the architecture of OMNeT++, echo node in the sensorSimulator is defined as a compound module, which includes protocol modules, hardware modules and coordinator modules. Besides that, it extended modules to represent target objects, sensor channels and wireless channels.

2.4 JavaSim/J-Sim

J-Sim, formerly named as JavaSim [21] is a component-based, real-time process driven simulator. It is based on the ACA (Autonomous Component Architecture) software architecture and a generalized packet-switched INET (internetworking) framework. Each element in the system is modeled as a component, which are assembled together at the system integration time by matching the port contracts of the participating components. Like OMNeT++, the INET framework organizes the components in a hierarchal manner, e.g., network, node, link and protocol instance.

Each node in the JavaSim is composed of a CSL (Core Server Layer) and several protocol modules. The CSL and protocol modules in the node are connected in a server-and-client mode. To extend the protocol, it is required to define the service provided by the CSL in terms of contracts, which are categorized into six categories: data, identity, routing table, interface/neighboring, multicast and packet filter configuration.

The architecture is somewhat like NS-2 and OMNeT++ in that they are all use two different languages to implement the protocol models and glue them together respectively. These two languages used by J-Sim are Java and Jacl.

J-Sim is a free software now, and updated by public these years. Current version of J-Sim includes some extended component to simulate the sensor networks [22]. The three major components integrated in J-Sim for simulating the sensor networks are the target node, the sensor node and the sink node, which are similar with the SensorSim extension to NS-2.

In view of the component-based architecture instead of object-oriented, J-Sim scales better than NS-2. But J-Sim inherits the assumed problem of inefficiencies from JAVA.
2.5 COST

COST (Component-Oriented Simulation Toolkit) \cite{23} is a general purpose discrete event driven simulator. It is based on the component-port model. Components are connected by the input ports and output ports. The COST is implemented by developed in CompC++, which is a component extension to C++ by defining following template classes:

1. Funtor: generalization of the function pointer.
2. Inport and Outport: interfaces to connect the components. The inports prescribe what functionalities a component can provide. The outports prescribe what functionalities may require from other components.
3. Simulation Time: time stamp of messages.
4. Port Index: extra argument to access the array of ports.
5. Timer: binding the time stamp argument and the index argument.

Since the COST only takes extensibility and reusability into considerations. scalability is still a problem for COST. In addition, inefficiency coming from the message exchange between components makes the problem more severe.

Based on the COST, a simulator called SENSE \cite{24} is developed for simulating the sensor networks. Each node component in the SENSE is made up of layered protocols, battery and power components, sensor component, mobility component, and wireless channel component. It inherits the merits of extensibility from the COST by using the component-port model. Besides that, it uses packet sharing model to partially solve the problem of scalability.

In addition, SENSE exploits simulation component classification to make not only the simulation models extensible, but also the simulation engines. The extensibility of simulation engines is embodied by the provision for the option to use parallel or sequential discrete event engines.

2.6 Ptolomy II

The Ptolemy project Ptolomy \cite{25} studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. In an approach of actor-oriented, hierarchical, heterogeneous modeling, each Module of Computation(MoC) is implemented as a domain.

PtolomyII \cite{26} is a software framework developed as part of the Ptolemy Project. It is a Java-based component assembly framework with a graphical user interface called Vergil. Vergil itself is a component assembly defined in Ptolemy II. Vergil stores models in ASCII files using an XML schema called MoML (Modeling Markup Language).

VisualSense \cite{27} is a part of Ptolomy II to simulate sensor networks. Viptos \cite{28} is an interface between Ptolomy II and TinyOS.

2.7 NCTUuns

NCTUuns \cite{29} is a high-fidelity and extensible network simulator capable of simulating both wired and wireless IP networks. It uses the real-life UNIX TCP/IP protocol stack, real-life network application programs, and real-life network utility programs to run simulations.
2.8 JiST/SWANS

JiST (Java in Simulation Time) [30] is a high-performance discrete event simulation engine that runs over a standard Java virtual machine. SWANS (Scalable Wireless Ad hoc Network Simulator) is a scalable wireless network simulator built atop the JiST platform.

Because JiST embeds simulation semantics directly into the Java execution model, it can execute the discrete event simulations both efficiently and transparently. In this approach, JiST out-performs the traditional system-based and language-based simulators in terms of time and memory consumption.

2.9 Summary

Table 2 gives a brief overview on the existing network simulators, and extensions to these simulators for supporting modeling the sensor networks. We see that most of these simulators have been extended, except the OPNET, NCTUns and JiST.

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Architecture</th>
<th>Language</th>
<th>Extension</th>
<th>Contributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS-2</td>
<td>Object-oriented</td>
<td>C++/OTcl</td>
<td>SensorSim</td>
<td>UCLA</td>
</tr>
<tr>
<td>OPNET</td>
<td>Object-oriented</td>
<td>C++</td>
<td>N/A</td>
<td>MIL3</td>
</tr>
<tr>
<td>OMNeT++</td>
<td>Component-based</td>
<td>C++/NED</td>
<td>SensorSimulator LSU</td>
<td></td>
</tr>
<tr>
<td>J-Sim</td>
<td>Component-based</td>
<td>Java/Jacl</td>
<td>J-Sim</td>
<td>OSU/UIUC</td>
</tr>
<tr>
<td>COST</td>
<td>Component-based</td>
<td>CompC++</td>
<td>SENSE</td>
<td>RPI</td>
</tr>
<tr>
<td>Ptolemy II</td>
<td>Actor-oriented</td>
<td>Java/XML</td>
<td>VisualSense</td>
<td>UCB</td>
</tr>
<tr>
<td>NCTUns</td>
<td>Unix Kernel-based</td>
<td>C</td>
<td>N/A</td>
<td>NCTU.TW</td>
</tr>
<tr>
<td>JiST/SWANS</td>
<td>JVM-based</td>
<td>Java</td>
<td>N/A</td>
<td>Cornell</td>
</tr>
</tbody>
</table>

3 Parallel and Distributed Simulation Platforms

3.1 GloMoSim/Qualnet

GloMoSim (Global Mobile System Simulator) [31] is a library-based sequential and parallel simulator for wireless networks. The library is written in PARSEC (PARallel Simulation Environment for Complex system) [32], which is an extension of C for parallel programming. Each node in the GloMoSim is modeled as a stack of protocols, which is radio/physical layer, MAC layer, IP/Routing layer, transport layer and application layer, from bottom to top in sequence. Propagation model and mobility model are also provided in the GloMoSim. It uses an object-oriented approach to organize all the models.

To address the problem of scalability, it partitions the nodes. Each object is responsible for running one layer in the protocol stack of every node for its given partition. However, GloMoSim cannot simulate the non-IP network, like sensor network effectively. It does not support the phenomenon occurring the outside of the simulation environment.

GloMoSim [33] began in 1998, but it stopped releasing updates in 2000. Instead, it is now updated as a commercial product called Qualnet [34].

The SensorSim, which is mentioned in section 2.1 as an extension of NS-2 to support modeling the sensor network, has been transplanted into the Qualnet. The extended Qualnet was named sQualent [35]. Following two improvements have been made in sQualent than SensorSim.
1. **Sensor physical layer**: an accurate diffusive sensing channel is modeled by solving the parabolic partial differential equations (PDEs).

2. **Battery model**: the current draw from the battery is modeled as a piece-wise linear current load profile.

### 3.2 SSFNet/DaSSF

Scalable Simulation Framework (SSF) [36] is developed as a common parallel simulation API suitable for but not exclusively for simulation of very large telecommunication systems.

SSFNet [37] is a collection of Java SSF-based components for modeling and simulation of Internet protocols and networks at and above the IP packet level of detail. Dartmouth SSF (DaSSF) [38] is a C++ implementation of SSF. Both SSFNet and DaSSF use Domain Modeling Language (DML) to describe network configuration.

SWAN (Simulation of Wireless and Ad-hoc Networks) [39] is an extension of DaSSF to support simulating wireless sensor network. It is comprised of inter-operating sub-models for terrain, plume dispersion, RF channel, and Node. Each node in SWAN consists of a wireless sensor model, a BBN WiroKit router model and operating system model. The operating system model is implemented with the DaSSF runtime kernel.

ToSSF [40] is an extension of SSFNet to support TinyOS native code.

### 3.3 GTNetS

The Georgia Tech Network Simulator (GTNetS) [41] is a C++ object-oriented simulator. It is designed in efficiency to support simulation of very large scale networks by reducing event list size, managing memory and reducing log file size [42].

GTNetS is designed specifically to allow creating distributed simulation platform easily. Unlike SSFNet/DaSSF, which are parallel simulators created from scratch, the parallel version of GTNeTs is created by interconnecting existing simulators based on a underlying runtime infrastructure (RTI) [43], which is identical to the architecture of PDNS (Parallel/Distributed Network Simulator) [44]. Each simulator is called a federate. To support distributed simulation, it used rlinks, IP address, and address masks to identify link end points, and an IP address and port number to identify a remote end host. GTNetS uses NIx-Vector routing to economize on memory required to store routing table information.

GTSNetS [45] is an extension to the GTNetS for sensor networks. Each wireless sensor node in GTSNets is composed of a computing unit, a sensing unit, a communication unit and a battery. Additional function is added to the sink node, which is keeping track of the life time of the sensor network. GTSNeTs also provides for extensive packet tracing. It features by its scalability, and is able to simulate sensor networks of several hundred thousand nodes while using less than 2 GB of memory. However, the modules in GTSNeTs is not so realistic, e.g. the battery is modeled as a reservoir of joules.

### 3.4 Summary

Table 3 gives a brief overview on the existing parallel/distributed simulators, and extensions to these simulators for supporting modeling the sensor networks. We see that all these simulators have been extended.
Table 3: Parallel/Distributed Simulators and Their Extensions for Sensor Networks

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Architecture</th>
<th>Language</th>
<th>Extension</th>
<th>Contributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloMoSim/Qualnet</td>
<td>Object-oriented</td>
<td>C++</td>
<td>sQualnet</td>
<td>UCLA</td>
</tr>
<tr>
<td>SSFNet/DaSSF</td>
<td>Object-oriented</td>
<td>C++/MDL</td>
<td>SWAN</td>
<td>Dartmouth</td>
</tr>
<tr>
<td>GTNetS</td>
<td>Object-oriented</td>
<td>C++</td>
<td>GTSNetS</td>
<td>GeTech</td>
</tr>
</tbody>
</table>

4 Sensor Network Simulators

4.1 Prolwer

Prolwer [46] is a MatLab-based, event-driven simulator for tuning the parameters of middleware (distributed system services) to provide an optimum for a given QoS metric.

4.2 SENS

SENS [47] is a component-based sensor network simulator. It is composed of following four components.

1. Application component: support importing realistic softwares;
2. Network component: protocol layer;
3. Physical component: power and sensing hardware;
4. Environment component: physical phenomena and layout.

The former three components make up the node module.

4.3 Sidh

Sidh [48] is a specific-purpose, component-based simulator for wireless networks. It is made up of a number of modules that interact with each other through events. The modules are divided into following categories.

1. Simulator module: base of Sidh;
2. Event module: communication between the other modules;
3. Node module: collection of hardware, network protocols and applications;
4. Environment module: physical phenomena, like target objects in SenseSim;
5. Medium and propagation module: wireless channel and sensor channel.

Sidh has some merits in extensibility, but may not perform well in view of efficiency. Modules provided by Sidh are very limited.
4.4 Shawn

Shawn [49] is a discrete event simulator for sensor networks with huge numbers of nodes. It is aimed at supporting the development cycle of sensor network protocols. Three major parts of Shawn are listed below.

1. **Sequencer**: the sequential simulation engine, which includes a *simulation controller* and an *event scheduler* responsible for scheduling the *simulation task*.

2. **Simulation environment**: the simulation objects, which are *world, nodes* and *processes*. These objects are organized in a hierarchical fashion, that is, the world contains several nodes and each node contains processes to process messages.

3. **Models**: foundation of the simulator, which includes the *communication model*, *edge model* and *transmission model*. The latter two models are based on the first model. Optional communication models are unit disk graphs, based on radio propagation physics and predefined connectivity.

Due to the simple models used by Shawn, it supports simulating large scale sensor networks with shorter run time and less memory consumption than NS-2. However, current available models for Shawn is very limited.

4.5 H-MAS

H-MAS [50] is a agent-based simulating environment using the *swarm* toolkit. It has only implemented a flood-based communication scheme and a non-contending media access method. Developed by University of Notre Dame. Another AI based simulation platform is presented in [51].

4.6 GloNeMo

GloNeMo [52] employs an approach of *formal modeling* to establish *global* models for the sensor networks. The models cover the hardware, protocol layers and application code for the nodes, and the physical environment as well.

The model formalist is made of *Communicating Input/Output Interpreted Automata*. Models for the components of the nodes are described in a functional-style language named *ReactiveML*, and the physical environmental model is implemented in a constraint-based language named *Lucky*. These models can be executed by the formal validation tools. In this way, the performance of the modeled sensor network can be evaluated.

4.7 Summary

Table 4 gives a brief overview on the existing simulators created specifically for sensor networks.

5 Emulators

5.1 TOSSIM

TOSSIM [53] is an emulator, which runs the actual TinyOS [54] applications built on MICA. It is featured by *scalability, completeness, fidelity* and *bridging*.
Table 4: Sensor Network Simulators Developed by Universities

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Architecture</th>
<th>Language</th>
<th>Contributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prowler</td>
<td>Matlab-based</td>
<td>C/Java</td>
<td>Vanderbilt</td>
</tr>
<tr>
<td>SENS</td>
<td>Component-based</td>
<td>C++</td>
<td>UIUC</td>
</tr>
<tr>
<td>Sidh</td>
<td>Component-based</td>
<td>Java</td>
<td>UMD</td>
</tr>
<tr>
<td>Shawh</td>
<td>Object-oriented</td>
<td>C++</td>
<td>TUB.Germany</td>
</tr>
<tr>
<td>H-MAS</td>
<td>Agent-based</td>
<td>Java</td>
<td>Univ. Notre Dame</td>
</tr>
<tr>
<td>GloNeMo</td>
<td>Formalism</td>
<td>ReactiveML</td>
<td>FranceTeleCom</td>
</tr>
</tbody>
</table>

It employs the identical structure with TinyOS and is generated by compiling the whole system into a discrete-event simulator. The TOSSIM architecture is mainly made up of following different components.

1. **Compiler support**: translates the TinyOS component graphs into a directed graph of bit error probability, and hardware interrupts into discrete events;

2. **Execution model**: discrete event queue;

3. **Abstract hardware**: ADC, clock and radio stack, etc.;

4. **Communication services**: command/event interface that allow PC applications to communicate with TOSSIM over TCP/IP.

   The probabilistic bit error model and translation of hardware interrupts into discrete events may lead to accuracy loss. Additionally, the phenomena is not simulated in TOSSIM.

   PowerTOSSIM \[55\] is an extension to TOSSIM for estimating per-node power consumption.

5.2 COOJA

COOJA \[56\] is a novel simulator for the Contiki operating system \[57\] that enables cross-level simulation: simultaneous simulation the network level, the operating system level, and the machine code instruction set level.

5.3 ATEMU

ATEMU \[58\] offers a more fine-grained AVR CPU emulator model, which uses a cycle-by-cycle strategy to run application code. So ATEMU is one of the most accurate sensor emulators available, but it has the problem of scalability.

ATEMU uses an XML configuration file to define the entire network in a hierarchical manner. It also provides a GUI interface, called XATDB, to debug the code.

5.4 Avrora

Avrora \[59\] is an emulator implemented in JAVA, and runs codes in an instruction-by-instruction fashion like ATEMU. It wants to make a tradeoff between TOSSIM and ATEMU, in other words, it want to gain the scalability of TOSSIM without reducing the accuracy of ATEMU so much. To obtain this aim, it employs two methods to reduce synchronization after every instruction.
5.5 Emstar

Emstar [60] is a Linux-based framework, which defines development cycle from pure simulation to actual deployment. The EmStar simulation model is component based, and provides an option to interact with actual hardware while running simulation.

Emstar uses a very simple environmental model and network media, but based on the interacting interface provided, it can make use of the actual sensors and communication channel. So Emstar is really a half-simulator and a half-emulator. Besides that, Emstar is featured by its support for developing software for Mica2 and iPAQ. Latest status of the project is reported in [61].

6 Distributed Emulators

6.1 VMNet

VMNet [62] is a distributed sensor network emulator based on EMPOWER [63], aiming at accurate emulation of a WSN for data-centric applications. Virtual motes (VM) running the real software are distributed in a LAN. These VMs are emulated to be connected logically by a virtual channel, and synchronized by a scheme of virtual time.

It is proved scalable and accurate, but it can only emulate one type of WSN now. Moreover, it does not take power consumption and mobility into consideration.

6.2 DiSenS

DiSens [64] is a full-system simulator, implemented especially for running in the distributed-memory parallel cluster systems. DiSens addressed the problem of extendibility, fidelity and scalability in simulating the sensor networks, by adopting a cycle accurate device emulator with pluggable models, a simple parallel synchronization protocol, and a sophisticated node partitioning algorithm.

The principle of DiSens is similar with VMNet. Both two platforms use a cluster of PCs or workstations to run the binary codes of the Motes, and communication among the Motes are simulated by sending messages via the LAN connecting the PCs or workstations. But DiSenS details more on how to solve the problem of synchronization and scalability.

6.3 Summary

Table 5 gives a brief overview on the existing emulators created specifically for sensor networks.

<table>
<thead>
<tr>
<th>Emulator</th>
<th>OS</th>
<th>Language</th>
<th>Contributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOSSIM</td>
<td>TinyOS</td>
<td>nesC</td>
<td>UCB</td>
</tr>
<tr>
<td>ATEMU</td>
<td>Any AVR</td>
<td>C</td>
<td>UMD</td>
</tr>
<tr>
<td>Arvora</td>
<td>TinyOS</td>
<td>nesC</td>
<td>UCLA</td>
</tr>
<tr>
<td>Emstar</td>
<td>Linux/TinyOS</td>
<td>nesC/C</td>
<td>UCLA</td>
</tr>
<tr>
<td>VMNet</td>
<td>TinyOS</td>
<td>nesC</td>
<td>UST.HK</td>
</tr>
<tr>
<td>DiSenS</td>
<td>TinyOS</td>
<td>nesC</td>
<td>UCSB</td>
</tr>
</tbody>
</table>
7 Testbeds

7.1 GNOMES

GNOMES [65] is a testbed for wireless heterogeneous sensor networks, which are refered to the networks composed of nodes sensing different things, developed by Rice University. Aiming at the longevity of each node, it employs dual-battery approach mixed with solar cell in order to recharge the battery when the node is in inactive. Besides that, the GNOMES node can function with either a 2.4GHz Bluetooth radio or a 900MHz radio module. The node can optionally integrate with a GPS module. As for the heart of the GNOMES node, it is a 16bit MPS430 microcontroller.

7.2 MoteLab

MoteLab [66] is a web-based sensor network testbed developed by Harvard University. Tens of Mica2 motes attached to MIB600 interface boards are connected by Ethernet to form a testbed. Each node in the testbed can be reprogrammed from a web interface or directly and the log data are permanently stored in a central database. So it allows the remote user to do experiment online or offline. Additionally, the job scheduler ensures the fair access.

Mirage [67] is a microeconomic resource allocation system using a combinatorial auction for MoteLab, aimed to address the testbed resource allocation problem.

7.3 EmuLab

Mobile EmuLab [68] is a robot-based mobile sensor network testbed developed by University of Utah. There are some resemblances between MoteLab and EmuLab, the main of which is both these two testbeds are web-based and can be accessed directly and remotely. The essential difference between them is that the former is for static sensor network, while the latter is for the mobile one. Motes in the MotLab are connected with a central computer via Ethernet, while Motes in the mobile EmuLab are mounted on the robots and communicate with the central computer through WIFI.

The Mobile EmuLab is built upon the EmuLab, and adds two important models for locating and moving the robots respectively.

7.4 Kansei

Kansei [69] is a heterogeneous sensor network testbed developed by the Ohio State University. It consists of a stationary array, a portable array and a mobile array. Besides its ability to provide a practical experimental environment, it supports hybrid simulation. When doing experiments, the motes run the real programs while the sensing readings are generated by sampling the recorded traces in the database. As for the hybrid simulation, it is accomodated by adapting the sensing and communication parts of TOSSIM. Like EmuLab and MoteLab, it also provides a web-based interface to configure and reconfigure the motes in the testbed.

7.5 MiNT-m

MiNT [70] is an autonomous mobile wireless experiment platform established in the Stony Brook University.
8 Conclusion

Though the testbed can guarantee high fidelity of the experimental results, it requires lots of time and energy to be putted. Especially when the time or finance is limited, it makes sense to evaluate the performance of sensor networks by simulation or emulation. So it is required to establish a extendable and scalable simulation/emulation platform modeling the sensor networks accurately.

In this paper, we gave a deep overview on the existing platforms to evaluate the performance of sensor networks. Some of them are for general purpose, while others are special for sensor networks. Concluding from the above descriptions of these platforms, either extension to the general-purpose simulators or implementation of a new simulator from script, following modules are supposed to be included in the simulators for sensor networks.

- **Physical nodes**: includes sensors, networking protocols and applications. The applications are mainly driven by the stimuli of the sensing readings.

- **Power consumption**: interacts with the hardware of the nodes in order to determine the its power consumption in different states. The primary hardware that should be taken into accounts are radio transceiver, CPU and sensor.

- **Environment**: simulates the target environment of sensors.

- **Channels**: includes wireless radio channel and sensing channel.

The complexity and architecture of these modules have deep effects on the performance of the simulators. Author of [71] have pointed out the three main modules comprising the simulators, namely radio channel, environment and energy consumption, and discussed their effects on the scalability of the simulators.

In short, the survey gives us a good reference to develop a extendable, scalable and accurate simulation platform for sensor networks.

9 Discussion and Future Work

We plan to develop a new platform for sensor networks in the near future. Before diving into it, several decisions should be made. First, which is appropriate to serve your purpose? Simulator or emulator? We think that the answer to this question is dependant on whether the platform is developed for evaluating the performance of the top level protocols or fine tuning the low-level algorithms? Second, in which way to build a simulator? Build on top of an existing general simulator or from the base to the top? We think that it is related to the time available and specific feature required, such as scalability and execution speed. Third, how to build? As for such a detailed question, following specific subquestions should be addressed.

- **Architecture**: component-base or object-oriented?

- **Language**: C++ or JAVA?

- **Simulation engine**: event driven or time driven?

- **Execution model**: sequential or parallel?

- **Features to be included**: modeling node and environment accurately, and provide the ability to interact with real nodes?
No matter what decisions made, the foremost important model to be implemented for simulating the sensor networks is the *power consumption model*, which is tightly associated with the *hardware model*. As for the hardware to be taken into considerations, it mainly includes *microcontroller*, *radio module* and *sensor/actuator*. When establishing the power consumption model for these hardware components, following aspects should be taken into accounts.

- **Performance parameters of the radio module**: What’s the mapping from radio power to consumed power? Receiver sensitivity? Maximum output power? Mapping from SINR to bit error rate? Available data rates? Number of available channels?

- **Modes of operations**: What is the power consumption for these components in different modes?

Besides the power consumption model, other necessary models must be established for sensor networks, like models of environment, radio and sensing channel. The accuracy of these modules also have deep influence on the results of performance evaluation. For mobile sensor networks, a model of mobility is required.

In the near future, we will firstly implement the typical protocol models for sensor networks, such as ZigBee(IEEE 802.15.4), B-MAC and Direct Diffusion, etc. Then we will implement the environment model, sensing channel model and sensor-driven application model. Additionally, the radio channel and mobility model will also be implemented if necessary.
References


[26] Ptolemy II. http://ptolemy.eecs.berkeley.edu/ptolemyII/.


